Embedded Methods

Thomas Navin Lal[1], Olivier Chapelle[1], Jason Weston[2], and André Elisseeff[3]

[1] Max Planck Institute for Biological Cybernetics, Tübingen, Germany
{navin.lal, olivier.chapelle}@tuebingen.mpg.de
[2] NEC Research, Princeton, U.S.A. jasonw@nec-labs.com
[3] IBM Research, Zürich, Switzerland ael@zurich.ibm.com

1 Introduction

Although many embedded feature selection methods have been introduced during the last few years, a unifying theoretical framework has not been developed to date. We start this chapter by defining such a framework which we think is general enough to cover many embedded methods. We will then discuss embedded methods based on *how* they solve the feature selection problem.
Embedded methods differ from other feature selection methods in the way feature selection and learning interact. Filter methods do not incorporate learning. Wrapper methods use a learning machine to measure the quality of subsets of features without incorporating knowledge about the specific structure of the classification or regression function, and can therefore be combined with any learning machine. In contrast to filter and wrapper approaches, in embedded methods the learning part and the feature selection part can not be separated - the structure of the class of functions under consideration plays a crucial role. For example, Weston et al. [2000] measure the importance of a feature using a bound that is valid for Support Vector Machines only (Section 3.1) - thus it is not possible to use this method with, for example, decision trees.
Feature selection can be understood as finding the feature subset of a certain size that leads to the largest possible generalization or equivalently to minimal risk. Every subset of features is modeled by a vector $\boldsymbol{\sigma} \in \{0,1\}^n$ of indicator variables, $\sigma_i := 1$ indicating that a feature is present in a subset and $\sigma_i := 0$ indicating that that feature is absent ($i = 1, \cdots, n$). Given a parameterized family of classification or regression functions[4] $f : \Lambda \times \mathbb{R}^n \to \mathbb{R}, (\boldsymbol{\alpha}, \mathbf{x}) \mapsto f(\boldsymbol{\alpha}, \mathbf{x})$ we try to find a vector of indicator variables $\boldsymbol{\sigma}^* \in \{0,1\}^n$ and an $\boldsymbol{\alpha}^* \in \Lambda$ that minimize the expected risk

$$R(\boldsymbol{\alpha}, \boldsymbol{\sigma}) = \int L[f(\boldsymbol{\alpha}, \boldsymbol{\sigma} * \mathbf{x}), y] \, dP(\mathbf{x}, y), \tag{1}$$

where $*$ denotes the pointwise product, L is a loss function and P is a measure on the domain of the training data (X, Y). In some cases we will also have the additional constraint $s(\boldsymbol{\sigma}) \leq \sigma_0$, where $s : [0,1]^n \to \mathbb{R}^+$ measures the sparsity of a given indicator variable $\boldsymbol{\sigma}$. For example, s could be defined to be $s(\boldsymbol{\sigma}) := l_0(\boldsymbol{\sigma}) \leq \sigma_0$, that is to bound the zero "norm" $l_0(\boldsymbol{\sigma})$ - which counts the number of nonzero entries in $\boldsymbol{\sigma}$ - from above by some number σ_0.[5]

The wrapper approach to approximate minimizers of (1) can be formulated in the following way

$$\inf_{\boldsymbol{\sigma} \in \{0,1\}^n} G(f^*, \boldsymbol{\sigma}, X, Y) \quad \text{s.t.} \quad \begin{cases} s(\boldsymbol{\sigma}) \leq \sigma_0 \\ f^* = \widetilde{T}(\mathcal{F}, \boldsymbol{\sigma}, X, Y), \end{cases} \tag{2}$$

where $\mathcal{F} \subset \mathbb{R}^{\mathbb{R}^n}$ denotes the family of classifying functions. Given such a family $\mathcal{F}$, a fixed indicator vector $\boldsymbol{\sigma}$ and the training data (X, Y) the output of the function $\widetilde{T}$ is a classifying or regression function f^* trained

[4] For example, in the case of Support Vector Machines, the vector $\boldsymbol{\alpha}$ codes the weight vector $\mathbf{w} \in \mathbb{R}^n$ and the offset $b \in \mathbb{R}$ of the hyperplane.
[5] Please note that $l_0(\cdot)$ is not a norm, since $l_0(c\boldsymbol{\sigma}) \neq |c| \; l_0(\boldsymbol{\sigma})$ for $\boldsymbol{\sigma} \neq 0, |c| \notin \{0,1\}$.

on the data X using the feature subset defined by $\boldsymbol{\sigma}$. The function G measures the performance of a trained classifier $f^*(\boldsymbol{\sigma})$ for a given $\boldsymbol{\sigma}$. It is very important to understand that although we write $G(f^*, \cdot, \cdot, \cdot)$ to denote that G depends on the classifying or regression function f^*, the function G does not depend on the structure of f^*; G can only access f^* as a black box, for example in a cross-validation scheme. Moreover, G does not depend on the specific learner $\widetilde{T}$. In other words $\widetilde{T}$ could be any off-the-shelf classification algorithm and G guides the search through the space of feature subsets.[6]
If we allow G to depend on the learner $\widetilde{T}$ and on parameters of f^* we get the following formulation:

$$\inf_{\boldsymbol{\sigma}\in\{0,1\}^n} G(\boldsymbol{\alpha}^*, \widetilde{T}, \boldsymbol{\sigma}, X, Y) \quad \text{s.t.} \quad \begin{cases} s(\boldsymbol{\sigma}) \leq \sigma_0 \\ \boldsymbol{\alpha}^* = \widetilde{T}(\boldsymbol{\sigma}, X, Y). \end{cases} \tag{3}$$

To emphasize that G can access the structure of the classifying or regression function we use the notation $f(\alpha^*, \cdot)$ instead of f^*. In this formulation the function G can use information about the learner and the class of functions on which it operates. Thus, G could evaluate, e.g., a bound on the expected risk valid for the specific choice of $\widetilde{T}$ and the classifying function $f(\alpha^*, \cdot)$ (see Section 3.1). To simplify the notation we will also write $G(\boldsymbol{\alpha}^*, \boldsymbol{\sigma}, X, Y)$ whenever $\widetilde{T}$ and $\mathcal{F}(\Lambda)$ are defined in the context. We define methods of type (3) as embedded methods.
Some embedded methods do not make use of a model selection criterion to evaluate a specific subset of features. Instead, they directly use the learner $\widetilde{T}$. Assuming that many learning methods $\widetilde{T}$ can be formulated as an optimization problem[7]:

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\Lambda} T(\boldsymbol{\alpha}, \boldsymbol{\sigma}, X, Y) = \widetilde{T}(\boldsymbol{\sigma}, X, Y)$$

we can rewrite the minimization problem (3) for the special case of $G = T$ as

$$\inf_{\boldsymbol{\alpha}\in\Lambda, \boldsymbol{\sigma}\in\{0,1\}^n} T(\boldsymbol{\alpha}, \boldsymbol{\sigma}, X, Y) \quad \text{s.t.} \quad s(\boldsymbol{\sigma}) \leq \sigma_0. \tag{4}$$

Unfortunately, both minimization problems (3) and (4) are hard to solve. Existing embedded methods approximate solutions of the minimization problem. In this chapter we discuss embedded methods according to *how* they solve problem (3) or (4):

1. Methods that iteratively add or remove features from the data to greedily approximate a solution of minimization problem (3) or (4) are discussed in Section 2.
2. Methods of the second type relax the restriction of $\boldsymbol{\sigma} \in \{0,1\}^n$ and minimize G over the compact set $[0,1]^n$. In this case we refer to $\boldsymbol{\sigma} \in [0,1]^n$ as scaling factors instead of indicator variables [Chapelle, 2002]. Section 3 is devoted to these methods.
3. If T and s are convex functions and if we assume $\boldsymbol{\sigma} \in [0,1]^n$, problem (4) can be converted into a problem of the form

$$\min_{\boldsymbol{\alpha}\in\Lambda, \boldsymbol{\sigma}\in[0,1]^n} T(\boldsymbol{\alpha}, \boldsymbol{\sigma}, X, Y) + \lambda s(\boldsymbol{\sigma}). \tag{5}$$

   More specifically, let T and s be strictly convex functions and let $(\boldsymbol{\alpha}^*, \boldsymbol{\sigma}^*)$ be a solution of problem (4) for a given $\sigma_0 > 0$. Then there exists a unique $\lambda > 0$ such that $(\boldsymbol{\alpha}^*, \boldsymbol{\sigma}^*)$ solves (5). Furthermore if $(\boldsymbol{\alpha}^*, \boldsymbol{\sigma}^*)$ solves (5) for a given $\lambda > 0$ then there exists one and only one σ_0 such that $(\boldsymbol{\alpha}^*, \boldsymbol{\sigma}^*)$ solves (4). The focus of Section 4 is on methods that can be formulated as a minimization problem of type (5).

[6] A function is no more than a complete collection of all input-output pairs. Thus one could argue that having f^* as a black box is equivalent to having f^* as a analytical expression. That is true in theory, but for many applications it does not hold true, for example for reasons of feasibility.

[7] For better readability, we assume that the function has one and only one minimum. However, this is not the case for all algorithms.

2 Forward-Backward Methods

In this section we discuss methods that iteratively add or remove features from the data to greedily approximate a solution of the minimization problem (3). These methods can be grouped into three categories. *Forward selection methods:* these methods start with one or a few features selected according to a method-specific selection criteria. More features are iteratively added until a stopping criterion is met. *Backward elimination methods:* methods of this type start with all features and iteratively remove one feature or bunches of features. *Nested methods:* during an iteration features can be added as well as removed from the data.

2.1 Sensitivity of the Output

Several methods that rank features according to their influence on the regression or discriminant function f have been proposed. These methods make use of a variety of criteria. For instance, one can measure the sensitivity of the output on the point $\mathbf{x}_k$ with respect to the i-th feature by[8]

$$t_{i,k} := \left.\frac{\partial f}{\partial x^i}\right|_{\mathbf{x}=\mathbf{x}_k}, \tag{6}$$

and the final criterion for that feature could be the ℓ_1 or ℓ_2 norm of the vector $\mathbf{t}_i$.
The underlying assumption of methods based on derivatives of the classifying function is the following: if changing the value of a feature does not result in a major change of the regression or classification function, the feature is not important. This approach can be used for a wide range of regression or classification functions. For example, the squared derivative of a linear model $f_{w,b}(x) = b + \mathbf{x} \cdot \mathbf{w}$ w.r.t. feature i yields $t_{i,k} = w_i^2,\ \forall k$. This quantity is used as a ranking criterion in Recursive Feature Elimination (RFE) [Guyon et al., 2003]. The sensitivity of the output has also been used for neural networks. A good introduction can be found in Leray and Gallinari [1999]. The authors also discuss methods that include second derivatives of the regression function (second order methods) as well as methods based on the regression function itself (zero-order methods). The sensitivity criterion can be used by methods of all three categories: forward, backward and nested.

In the following paragraph we develop a link between the minimization problem (4) of embedded methods and methods that use the sensitivity of the classifying function as a measure for feature relevance. The optimization of (4) is done over Λ and $\{0,1\}^n$. Since this minimization problem is difficult to solve, many methods fix $\boldsymbol{\sigma}$ and optimize over Λ, fix $\boldsymbol{\alpha}$ and optimize over Σ and so on. In many cases, like RFE and Optimal Brain Damage, the optimization over Λ is a standard learning method, like SVM (RFE) or neural networks (OBD). Oftentimes, the full calculation of $\min_{\boldsymbol{\sigma}\in\{0,1\}^n} G(\boldsymbol{\sigma}, \boldsymbol{\alpha})$ (for a fixed $\boldsymbol{\alpha}$) is too expensive since it might involve setting one σ_i at a turn to zero, training the learner and evaluating the result. Instead, methods that are based on the sensitivity of the classifying (or regression) function or weight based methods (Section 2.3) use the gradient of G with respect to an indicator variable to approximate the expensive calculation of the minimum.
Suppose that we want to learn a parameterized function $f(\boldsymbol{\alpha}, \mathbf{x})$. The vector $\boldsymbol{\alpha}$ can, for example, be the weights of a neural network. Generally, one finds the weight vector $\boldsymbol{\alpha}$ which minimizes a regularized functional

$$T(\boldsymbol{\alpha}) = \sum_{k=1}^{m} L(f(\boldsymbol{\alpha}, \mathbf{x}_k), y_k) + \Omega(\boldsymbol{\alpha}). \tag{7}$$

Here L is a loss function and Ω is a regularization term such as $\lambda||\boldsymbol{\alpha}||^2$.
Let us introduce a scaling factor $\sigma_i \in [0,1]$ for every input component i. The functional T now also depends on the vector of scaling factors $\boldsymbol{\sigma}$ and we write it as $T(\boldsymbol{\alpha}, \boldsymbol{\sigma})$. Let G be defined as

[8] Notation: $\mathbf{x}_k$ denotes the k^{th} training point, $x_{k,i}$ the i^{th} feature of training point $\mathbf{x}_k$, and x^i denotes the i^{th} feature of training point $\mathbf{x}$.

$$G(\boldsymbol{\sigma}) = \min_{\boldsymbol{\alpha}} T(\boldsymbol{\alpha}, \boldsymbol{\sigma}) = \min_{\boldsymbol{\alpha}} \sum_{k=1}^{m} L(f(\boldsymbol{\alpha}, \boldsymbol{\sigma} * \mathbf{x}_k), y_k) + \Omega(\boldsymbol{\alpha}).$$

The removal of feature p corresponds to changing the p-th component of the scaling vector $\boldsymbol{\sigma} := (1, \cdots, 1)$ to 0. We will refer to this vector as $\mathbf{1}_0(p)$. A greedy backward method would start with all features, i.e. $\boldsymbol{\sigma} = \mathbf{1}$. As said earlier, for every $p \in \{1, ..., n\}$ it would have to calculate $G(\mathbf{1}_0(p))$, remove the feature p that minimizes this value and continue with the remaining features. However, the value $G(\mathbf{1}_0(p))$ can be approximated by the value of $G(\mathbf{1})$ and the gradient of T at the minimal point:

$$G(\mathbf{1}_0(p)) \approx G(\mathbf{1}) - \left.\frac{\partial G(\boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\sigma}=\mathbf{1}} \qquad (p = 1, ..., n). \tag{8}$$

Let us write $G(\boldsymbol{\sigma}) = T(\boldsymbol{\alpha}^*(\boldsymbol{\sigma}), \boldsymbol{\sigma})$, where $\boldsymbol{\alpha}^*(\boldsymbol{\sigma})$ denotes the parameter vector which minimizes T (for a given $\boldsymbol{\sigma}$). Then,

$$\frac{\partial G(\boldsymbol{\sigma})}{\partial \sigma_p} = \sum_j \frac{\partial \alpha_j^*(\boldsymbol{\sigma})}{\partial \sigma_p} \overbrace{\left.\frac{\partial T(\boldsymbol{\alpha}, \boldsymbol{\sigma})}{\partial \alpha_j}\right|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}^*(\boldsymbol{\sigma})}}^{=0} + \left.\frac{\partial T(\boldsymbol{\alpha}, \boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}^*(\boldsymbol{\sigma})} \qquad (p = 1, ..., n) \tag{9}$$

and with equation (8) we obtain

$$G(\mathbf{1}_0(p)) \approx G(\mathbf{1}) - \left.\frac{\partial T(\boldsymbol{\alpha}, \boldsymbol{\sigma})}{\partial \sigma_p}\right|_{(\boldsymbol{\alpha},\boldsymbol{\sigma})=(\boldsymbol{\alpha}^*(\mathbf{1}),\mathbf{1})} \qquad (p = 1, ..., n). \tag{10}$$

For every $p \in \{1, ..., n\}$ the gradient of T can be expressed as

$$\begin{aligned}
\left[\left.\frac{\partial T(\boldsymbol{\alpha}, \boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\sigma}=\mathbf{1}}\right]^2 &\overset{\text{(eq.7)}}{=} \left[\sum_{k=1}^{m} L'(f(\boldsymbol{\alpha}, \mathbf{x}_k), y_k)\ \mathbf{x}_{k,p}\ \frac{\partial f}{\partial x^p}(\boldsymbol{\alpha}, \mathbf{x}_k)\right]^2 \\
&\overset{\text{(Cauchy-Schwarz)}}{\leq} \sum_{k=1}^{m} [L'(f(\boldsymbol{\alpha}, \mathbf{x}_k), y_k)\ \mathbf{x}_{k,p}]^2 \sum_{k=1}^{m} \left[\frac{\partial f}{\partial x^p}(\boldsymbol{\alpha}, \mathbf{x}_k)\right]^2 \\
&\leq \max_{k=1,...,m} L'(f(\boldsymbol{\alpha}, \mathbf{x}_k), y_k)^2 \ \sum_{k=1}^{m} (\mathbf{x}_{k,p})^2 \ \sum_{k=1}^{m} \left[\frac{\partial f}{\partial x^p}(\boldsymbol{\alpha}, \mathbf{x}_k)\right]^2 \\
&\overset{(*)}{=} C \sum_{k=1}^{m} \left[\frac{\partial f}{\partial x^p}(\boldsymbol{\alpha}, \mathbf{x}_k)\right]^2 \propto ||\mathbf{t}_p||_2^2,
\end{aligned}$$

where $L'(\cdot, \cdot)$ denotes the derivative of L with respect to its first variable, C is a constant independent of p and $\mathbf{t}_p$ has been defined in (6). For equality $(*)$ to hold, we assume that each input has zero mean and unit variance. This inequality and the approximation (10) link the definition (4) to methods based on the sensitivity of the output: The removal of a feature does not change the value of the objective G significantly if the classifying function f is not sensitive to that feature.

2.2 Forward Selection

In this section we describe the forward selection methods *Gram-Schmidt Orthogonalization*, *decision trees* and the *Grafting*-method. The three approaches iteratively increase the number of features to construct a target function.

Forward Selection with Least Squares

If S is a set of features, let X_S denotes the submatrix of the design matrix where only the features in S are included: $X_S := (\dots \mathbf{x}_i \dots)_{i \in S}$. When doing standard regression with the subset S, the residuals on the training points are given by

$$P_S Y \quad \text{with} \quad P_S := I - X_S^\top (X_S X_S^\top)^{-1} X_S. \tag{11}$$

Note that P_S is a projection matrix, i.e. $P_S^2 = P_S$.
The classical forward selection for least square finds greedily the components which minimize the residual errors:

1. Start with $Y = (y_1 \dots y_m)^\top$ and $S = \emptyset$.
2. Find the component i such that $||P_{\{i\}} Y||^2 = Y^\top P_{\{i\}} Y$ is minimal.
3. Add i to S
4. Recompute the residuals Y with (11)
5. Stop or go back to 2

More recent algorithms based on this idea include Gram-Schmidt Orthogonalization described below and *Least Angle Regression* (LARS) [Efron et al., 2004].

Grafting

For fixed $\lambda_0, \lambda_1, \lambda_2 > 0$, Perkins et al. [2003] suggested minimizing the function

$$C(\boldsymbol{\alpha}) = \frac{1}{m} \sum_{k=1}^{m} L(f(\boldsymbol{\alpha}, \mathbf{x}_k), y_k) + \lambda_2 \|\boldsymbol{\alpha}\|_2 + \lambda_1 \|\boldsymbol{\alpha}\|_1 + \lambda_0 \; l_0(\boldsymbol{\alpha}) \tag{12}$$

over the set of parameters Λ that defines the family of regression or classification functions $\mathcal{F} = \{f(\alpha, \cdot) \mid \boldsymbol{\alpha} \in \Lambda\}$.[9] We restrict our analysis to linear models, i.e. $\mathcal{F} := \{f : \mathbb{R}^{n+1} \times \mathbb{R}^n \to \mathbb{R} \mid f(\boldsymbol{\alpha}, \mathbf{x}) = \boldsymbol{\alpha} \cdot \mathbf{x} + \alpha_{n+1},\ \boldsymbol{\alpha} \in \mathbb{R}^{n+1}, \mathbf{x} \in \mathbb{R}^n\}$. In this case α_i is associated with feature i $(i = 1, \cdots, n)$. Therefore requiring the l_0 as well as the l_1 norm of $\boldsymbol{\alpha}$ to be small results in reducing the number of features used. Note that for families of non-linear functions this approach does not necessarily result in a feature selection method. Since the objective C has the structure of minimizing a loss function plus a sparsity term it will also be mentioned in Section 4.
Perkins et al. [2003] solve problem (12) in a greedy forward way. In every iteration the working set of parameters is extended by one α_i and the newly obtained objective function is minimized over the enlarged working set. The selection criterion for new parameters is $|\partial C / \partial \alpha_i|$. Thus, the α_i is selected which yields the maximum decrease of the objective function C after one gradient step. Furthermore, the authors discuss possible stopping criteria.
Note that Grafting is an embedded method, because the structure of the target functions f (namely linear functions) is important for the feature selection part.

Decision Trees

Decision trees can be used to learn discrete valued functions. The most prominent approaches include CART [Breiman et al., 1984], ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993]. Decision trees are iteratively build by splitting the data depending on the value of a specific feature. The ''splitting'' feature is chosen according to its importance for the classification task. A widely used criterion for the importance of a feature is the mutual information between feature i and the outputs Y [Duda et al., 2001, Section 8.3.2]:

$$MI(X^i, Y) = H(Y) - H(Y|X^i),$$

where H is the entropy [Cover and Thomas, 1991, Chapter 2] and $X^i := (x_{1,i}, \cdots, x_{m,i})$. In case of a continuous input space, the components of vector X_i are replaced by binary values corresponding to decisions of the type $x_{k,i} \leq$ *threshold*. Another choice is to split the data in such a way that the resulting tree has minimal empirical error.
In many cases, only a subset of the features is included into a decision tree. Thus, feature selection is implicitly built into the algorithm and therefore decision trees can be understood as an embedded method.

[9] The original optimization problem is slightly more complex. It includes weighted Minkowski norms.

Gram-Schmidt Orthogonalization

Gram-Schmidt Orthogonalization [e.g. Chen et al., 1989] uses the angle of a feature to the target as an evaluation criterion to measure the importance of a feature for classification or regression purposes. In an iterative procedure the angle of every feature to the target is computed and the feature maximizing this quantity is selected. The remaining features are mapped to the null subspace of the previously selected features and the next iteration starts. During each iteration the least-squares solution of a linear-in-its-parameters model is computed based on the already ranked features. Thus the selection method is an embedded method for a linear least-square predictor. It is similar in spirit to the partial least squares (PLS) algorithm, but instead of constructing features that are linear combinations of the input variables, the Gram-Schmidt variable selection method selects variables.
Rivals and Personnaz [2003] suggested using polynomials as baseline classifiers. In a first step all monomials of degree smaller than an a priori fixed number d are ranked using Gram-Schmidt Orthogonalization. In a second step the highest ranked monomials are used to construct a polynomial. To reduce the complexity of the constructed polynomial one monomial at a time is removed from the polynomial in an iterative procedure until a stopping criterion is met. For large data sets the authors use the Fisher test as a stopping criterion; for small data sets the leave-one-out error is used since the statistical requirements for the hypothesis test are not satisfied. Both criteria are based on the mean squared error. The paper also describes how the ranked monomials can be used to select a subset of features as inputs to a neural network. Since this approach has more the flavor of a filter method, we do not discuss the details here.
Stoppiglia et al. [2003] introduced a stopping criterion for the iteration during Gram-Schmidt Orthogonalization by adding a random feature to the set of features. The idea is that features ranked higher than the random feature carry information about the target concept whereas features ranked lower than the random feature are irrelevant. See also Chapter **??** for more details on Gram-Schmidt Orthogonalization, probe methods and the Fisher test.

2.3 Backward Elimination

Starting with all features, backward elimination methods iteratively remove features from the data according to a selection criterion until a stopping criterion is met. In this section we first discuss methods that derive the selection criterion from the analysis of the weight vector of the classifying or regression function. In the second part we briefly mention a method that uses a learning bound as a selection criterion.

Weight Based Analysis

In this section we review algorithms which choose features according to the weights given to those features by a classifier. The motivation for this is based on the idea that the value of a feature is measured by the change in expected value of error when removing it. Given this assumption, these methods approximate this goal by using the weights given by the classifier itself.

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is a recently proposed feature selection algorithm described in Guyon et al. [2003]. The method, given that one wishes to employ only $\sigma_0 < n$ input dimensions in the final decision rule, attempts to find the best subset of size σ_0 by a kind of greedy backward selection. It operates by trying to choose the σ_0 features which lead to the largest margin of class separation, using an SVM classifier (see Chapter **??**). This combinatorial problem is solved in a greedy fashion at each iteration of training by removing the input dimension that decreases the margin the least until only σ_0 input dimensions remain.
The algorithm can be accelerated by removing more than one feature in step 2. RFE has shown good performance on problems of gene selection for microarray data [Guyon et al., 2003, Weston et al., 2003, Rakotomamonjy, 2003]. In such data there are thousands of features, and the authors usually remove half of the features in each step.

Algorithm 1 Recursive Feature Elimination (RFE) in the linear case

1: **repeat**
2: Find $\mathbf{w}$ and b by training a linear SVM.
3: Remove the feature with the smallest value $|w_i|$.
4: **until** σ_0 features remain.

One can also generalize the algorithm to the nonlinear case [Guyon et al., 2003]. For SVMs the margin is inversely proportional to the value $W^2(\alpha) := \sum \alpha_k \alpha_l y_k y_l k(\mathbf{x}_k, \mathbf{x}_l) (= \|\mathbf{w}\|^2)$. The algorithm thus tries to remove features which keep this quantity small[10]. This leads to the following iterative procedure:

Algorithm 2 Recursive Feature Elimination (RFE) in the nonlinear case.

1: **repeat**
2: Train a SVM resulting in a vector α and scalar b.
3: Given the solution α, calculate for each feature p:

$$W^2_{(-p)}(\alpha) = \sum \alpha_k \alpha_l y_k y_l k(\mathbf{x}_k^{-p}, \mathbf{x}_l^{-p})$$

(where $\mathbf{x}_k^{-p}$ means training point k with feature p removed).
4: Remove the feature with smallest value of $|W^2(\alpha) - W^2_{(-p)}(\alpha)|$.
5: **until** σ_0 features remain.

If the classifier is a linear one, this algorithm corresponds to removing the smallest corresponding value of $|w_i|$ in each iteration. The nonlinear formulation helps to understand how RFE actually works. Given the criterion of small W^2 (large margin) as a measure of generalization ability of a classifier, one could at this point simply choose one of the usual search procedures (hill climbing, backward elimination, etc.). However, even if we were to choose backward elimination, we would still have to train $\frac{1}{2}(n^2 + n - \sigma_0^2 + \sigma_0)$ SVMs (i.e. on each iteration we have to train as many SVMs as there are features) which could be prohibitive for problems of high input dimension. Instead, the trick in RFE is to estimate in each iteration the change in W^2 by only considering the change in the kernel k resulting from removing one feature, and assuming the vector $\boldsymbol{\alpha}$ stays fixed.

Note that RFE has been designed for two-class problems although a multi-class version can be derived easily for a one-against-the-rest approach [Weston et al., 2003]. The idea is then to remove the features that lead to the smallest value of $\sum_{c=1}^{Q} |W_c^2(\alpha^c) - W^2_{c,(-p)}(\alpha^c)|$ where $W_c^2(\alpha^c)$ is the corresponding margin based value for the machine discriminating class c from all the others. This would lead to removing the *same* features from all the hyperplanes that build up the multi-class classifier, thus coupling the classifiers. This is useful if the same features are relevant for discriminating between different classes, if this is not the case it may not be a good idea to couple the hyperplanes in this way.

Clearly, it is easy to generalize RFE to other domains as well, such as other classifications algorithms or regression. In Zhu and Hastie [2003] the authors show how to adapt RFE to penalized logistic regression to give performance similar to SVMs, but with the added benefit of being able to output probability of correctness, rather than just predictions. They demonstrate that RFE gives performance superior to a univariate correlation score feature selector using this classifier.

Finally, in Lal et al. [2004] the authors show how to treat groups of features with RFE in the domain of Brain Computer Interfaces (BCI). In this problem there are several features for each EEG channel, and one is interested in finding which channels are relevant for a particular task. Therefore the authors modify RFE to remove the least important *channel* on each iteration by measuring the importance of a channel with $\sum_{i \in C_j} |w_i|$ for each channel (set of features) C_j.

[10] Although the margin has been shown to be related to the generalization ability by theoretical bounds for SVMs [Vapnik, 1998], if you change the feature space by removing features, bounds on the test error usually also involve another term controlling this change, e.g. the bound $R^2\|\mathbf{w}\|^2$ in equation (16) also includes the radius R of the sphere around the data in feature space. However, if the data is suitably normalized, it appears that RFE works well.

RFE-Perceptron

In Gentile [2004], the authors propose an algorithm similar to RFE but for (approximate) large margin perceptrons using the p-norm [Grove et al., 1997] (ALMA$_p$). For $p = 2$ the algorithm is similar to an SVM and for $p = \infty$ the algorithm behaves more like an multiplicative update algorithm such as Winnow [Kivinen and Warmuth, 1995]. They adapt the p-norm according to the number of features so when the feature size is shrunk sufficiently their algorithm behaves more like an SVM (they report that using the full feature set Winnow generally outperforms an SVM). The authors then propose a feature selection method that removes the features with the smallest weights until w fails to satisfy: $\sum_i |w_i| \geq 1 - \alpha(1-\alpha)\gamma^{p/(p-1)}$ where $\alpha \in (0, 1]$ is a hyperparameter controlling the degree of accuracy and γ is the margin obtained from running ALMA$_p$. This differentiates it from RFE which requires setting the number of features to remove in advance (see Chapter **??** for statistical tests and cross-validation). The authors report this automatic choice results in a small number of chosen features, and generalization ability often close to the optimum number of features for RFE chosen using the test set.

Optimal Brain Damage

Optimal Brain Damage (OBD) [LeCun et al., 1990] is a technique for pruning weights in a Neural Network. The method works by, after having chosen a reasonable network architecture and training that network, first computing the second derivatives h_{ii} for each weight w_i. It then sorts the weights by saliency, defined as $h_{ii}w_i^2/2$ and deletes r weights with low saliency, and then repeats. Essentially, this measures the change in error when a weight is decreased. Thus OBD removes weights which do not appear to influence the training error. The authors demonstrate good performance on an Optical Character Recognition (OCR) problem. Starting with a highly constrained and sparsely connected network with 2,578 weights, trained on 9300 examples, they were able to delete more than 1000 weights without reducing training or testing error.

Motivation for Weight Based Approaches

In this section we motivate weight-based approaches based on the minimization problem (4) for embedded methods. Suppose that the function f is a typical neural network:

$$f(\mathbf{w}, \mathbf{x}) = \tilde{f}(\ldots, w_{iq}x^i, \ldots, \tilde{\mathbf{w}}),$$

where the vector $\mathbf{w}$ of weights has been split between the weights w_{iq} of the first layer and the rest of the parameters $\tilde{\mathbf{w}}$. The connection of the i-th neuron of the input layer and the q-th neuron of the hidden layer is weighted by w_{iq} ($i \in \{1, \cdots, n\}, q \in \{1, \cdots, Q\}, Q \in \mathbb{N}$).
For a fixed $\boldsymbol{\sigma} = \mathbf{1}$, let $\mathbf{w}^*$ be a minimizer of $T(\mathbf{w}, \boldsymbol{\sigma}) = \sum_k L(f(\mathbf{w}, \boldsymbol{\sigma} * \mathbf{x}_k), y_k) + \Omega(\mathbf{w})$. With equation (9) we get for $p = 1, ..., n$:

$$\begin{aligned}\left.\frac{\partial G(\boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\sigma}=\mathbf{1}} &= \left.\frac{\partial T(\mathbf{w}, \boldsymbol{\sigma})}{\partial \sigma_p}\right|_{(\mathbf{w},\boldsymbol{\sigma})=(\mathbf{w}^*,\mathbf{1})} \\ &= \sum_{k=1}^{m} L'(f(\mathbf{w}^*, \mathbf{x}_k), y_k)\; x_k^p \sum_{q=1}^{Q} w_{pq}^* \left.\frac{\partial \tilde{f}(\ldots, t_{pq}, \ldots, \tilde{\mathbf{w}^*})}{\partial t_{pq}}\right|_{t_{pq}=w_{pq}^* x_p}. \end{aligned} \tag{13}$$

On the other hand, since the weight vector $\mathbf{w}^*$ is a minimizer of $T(\mathbf{w}, \mathbf{1})$, we also have

$$\begin{aligned} 0 &= \left.\frac{\partial T(\mathbf{w}, \boldsymbol{\sigma})}{\partial w_{pq}}\right|_{(\mathbf{w},\boldsymbol{\sigma})=(\mathbf{w}^*,\mathbf{1})} \\ &= \sum_{k=1}^{m} L'(f(\mathbf{w}^*, \mathbf{x}_k), y_k)\; x_k^p \left.\frac{\partial \tilde{f}(\ldots, t_{pq}, \ldots, \tilde{\mathbf{w}^*})}{\partial t_{pq}}\right|_{t_{pq}=w_{pq}^* x_p} + \left.\frac{\partial \Omega(\mathbf{w})}{\partial w_{pq}}\right|_{\mathbf{w}=\mathbf{w}^*}. \end{aligned}$$

By multiplying this last equation by w_{pq}, summing over q and combining it with (13), we finally get

$$\left.\frac{\partial G(\boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\sigma}=\mathbf{1}} = -\sum_{q=1}^{Q} w_{pq}^* \left.\frac{\partial \Omega(\mathbf{w})}{\partial w_{pq}}\right|_{\mathbf{w}=\mathbf{w}^*}.$$

If the regularization term Ω is quadratic, this last equation yields

$$\left.\frac{\partial G(\boldsymbol{\sigma})}{\partial \sigma_p}\right|_{\boldsymbol{\sigma}=\mathbf{1}} = -2\sum_{q=1}^{Q} \left(w_{pq}^*\right)^2.$$

This equation suggests to remove the component such that the l_2 norm of the weights associated with this input is small. More general, weight based methods remove the component with the smallest influence on G after one gradient step.

Bounds for Support Vector Machines

Rakotomamonjy [2003] reports a greedy backward feature selection method similar to RFE. Two bounds on the leave-one-out error LOO of a trained *hard margin* SVM classifier are used as ranking criterion: the radius margin bound [Vapnik, 1998]:

$$LOO \leq 4R^2\|\mathbf{w}\|^2, \tag{14}$$

where R denotes the radius of the smallest sphere that contains the training data[11], and the span-estimate [Vapnik and Chapelle, 2000]:

$$LOO \leq \sum_k \boldsymbol{\alpha}_k^* S_k^2, \qquad S_k^2 = \frac{1}{(\tilde{\mathbf{K}}^{-1})_{kk}}, \qquad \tilde{\mathbf{K}} = \begin{pmatrix} \mathbf{K}_{SV} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{pmatrix},$$

where $\tilde{\mathbf{K}}$ is the "augmented" kernel matrix computed only on the support vectors. Such SVM having kernel matrix $\mathbf{K}$ is trained with squared slack variables. Based on these bounds, two types of ranking criteria for features are introduced:

(i) Feature i is removed from the training data. An SVM is trained and the leave-one-out error LOO^{-i} is computed. The same procedure is repeated for the remaining features. The features are ranked according to the values of LOO^{-i} (the feature i with the highest value of LOO^{-i} is ranked first).
This approach might be computationally expensive but it can be approximated in the following way: An SVM is trained using all features. Let $\boldsymbol{\alpha}^*$ be the solution. To measure the importance of feature i, the i-th component is removed from the kernel matrix. The bound $\widetilde{LOO}^{-i}$ is calculated using the reduced kernel matrix and $\boldsymbol{\alpha}^*$. The features are ranked according to the values of $\widetilde{LOO}^{-i}$. The assumption that would justify this approach is that the solution vector $\boldsymbol{\alpha}_{-i}^*$ of an SVM trained on the training data where the i-th feature was removed can be approximated by the solution $\boldsymbol{\alpha}^*$.

(ii) Instead of ranking features according to how their removal changes the value of the bound, the author also suggested to use the sensitivity of the bound with respect to a feature as a ranking criterion. In other words, a feature i is ranked according to the absolute value of the derivative of the bound with respect to this feature (the feature yielding the highest value is ranked first).

Like RFE, the method starts with a trained SVM that uses all features. The features are ranked using one of the two approaches (i) or (ii). The least ranked feature is removed from the data and the next iteration starts. This method is closely related to the work of Weston et al. [2000]: Instead of iteratively removing features, scaling factors are introduced and optimized by a gradient descent (see also Section 3.1).

The method described by Rakotomamonjy [2003] can be interpreted as an approximation to the optimization problem (3). The functional $\widetilde{T}$ is the Support Vector Machine and the functional G is either the radius/margin bound or the leave-one-out estimate. The minimization over Σ is approximated in a greedy backward way without the guarantee of finding a global minimum. In (i), the criterion G is explicitly computed after a tentative feature has been removed, whereas in (ii), the gradient $\partial G/\partial \sigma_i$ is calculated to measure the importance of that feature.

[11] Please note that R was previously also used for the risk.

3 Optimization of Scaling Factors

3.1 Scaling Factors for SVM

One method of using SVMs for feature selection is choosing the scaling factors which minimize a bound, as proposed by the authors of Weston et al. [2000]. The idea is the following. Feature selection is performed by scaling the input parameters by a vector $\boldsymbol{\sigma} \in [0,1]^n$. Larger values of σ_i indicate more useful features. Thus the problem is now one of choosing the best kernel of the form:

$$k_\sigma(\mathbf{x}, \mathbf{x}') = k(\boldsymbol{\sigma} * \mathbf{x}, \boldsymbol{\sigma} * \mathbf{x}'), \tag{15}$$

where $*$ is element-wise multiplication, i.e. we wish to find the optimal parameters $\boldsymbol{\sigma}$. This can be achieved by choosing these hyperparameters via a generalization bound.

Gradient Descent on the R^2w^2 Bound

Taking the expectation on both sides of inequality (14) and using the fact that the leave-one-out error is an unbiased estimator of the generalization error [Luntz and Brailovsky, 1996] one gets the following bound for hard margin SVMs,[12]

$$EP_{err} \leq \frac{4}{m} E\left\{R^2 \|\mathbf{w}\|^2\right\}, \tag{16}$$

if the training data of size m belong to a sphere of size R and are separable with margin M (both in the feature space) Here, the expectation is taken over sets of training data of size m. Although other bounds exist, this one is probably the simplest to implement. A study of using other bounds for the same purpose is conducted in [Chapelle et al., 2002]. The values of $\boldsymbol{\sigma}$ can be found by minimizing such a bound by gradient descent, providing the kernel itself is differentiable. This can be solved by iteratively training an SVM, updating $\boldsymbol{\sigma}$ (the kernel) according to the gradient, and retraining until convergence. An implementation is available at Spider [2004]. This method has been successfully used for visual classification (face detection, pedestrian detection) [Weston et al., 2000] and for gene selection for microarray [Weston et al., 2000, 2003]. After optimization of the scaling factors, most of them will hopefully be small and one can explicitly discard some components and optimize the scaling factors for the remaining components. There are 3 different ways of discarding irrelevant components:

- The most straightforward is to remove the components which have small scaling factors. This is somehow similar to the RFE algorithm (see Section 2.3, as well as the end of this section).
- A more expensive strategy is to explicitly remove each feature at a turn, retrain an SVM and see by how much the radius margin bound increases (Chapelle [2002], Rakotomamonjy [2003] and Section 2.3).
- To avoid such expensive retrainings, Chapelle [2002] suggest to use a second order information and, similarly to OBD [LeCun et al., 1990], remove the features p which have a small value of

$$\sigma_p^2 \frac{\partial R^2 \mathbf{w}^2}{\partial \sigma_p^2}.$$

One could easily use the same trick for classifiers other than SVMs if a bound for that classifier is readily available. Alternatively one could use a validation set instead, this route is explored in Chapelle et al. [2002].

Other Criteria

In the previous section, the hyperparameters $\boldsymbol{\sigma}$ were optimized by gradient descent on $R^2\mathbf{w}^2$. Please note that other model selection criteria are possible such as the span bound [Vapnik and Chapelle, 2000] or a validation error. Experimental results of such approaches are presented in Chapelle et al. [2002], but from a

[12] For soft margin SVMs, the simplest way to still make this apply is to add a ridge to kernel matrix [Chapelle et al., 2002].

practical point of view, these quantities are more difficult to optimize because they are highly non-convex. Also, in equation (16), one can use the variance instead of the radius, i.e.

$$R^2 \approx \frac{1}{m}\sum_{k=1}^{m}\left(\Phi(\mathbf{x}_k) - \frac{1}{m}\sum_{l=1}^{m}\Phi(\mathbf{x}_l)\right)^2 = \frac{1}{m}\sum_{k=1}^{m}k(\mathbf{x}_k,\mathbf{x}_k) - \frac{1}{m^2}\sum_{k,l=1}^{m}k(\mathbf{x}_k,\mathbf{x}_l). \tag{17}$$

The first advantage in using the variance instead of the radius is that it is easier to compute. But a second advantage is that it is less sensitive to outliers and that it is also theoretically sound [Bousquet, 2002].
A similar approach to the one described in the previous section is to consider the scaling factors $\boldsymbol{\sigma}$ not anymore as hyperparameters but as parameters of the learning algorithm [Grandvalet and Canu, 2003]. A scalar parameter σ_0 controls the norm of $\boldsymbol{\sigma} \geq 0$:

$$\min_{\sigma}\max_{\alpha}\sum_{k}\alpha_k - \frac{1}{2}\sum_{k,l}\alpha_k\alpha_l y_k y_l k_\sigma(\mathbf{x}_k,\mathbf{x}_l)$$

under constraints

$$0 \leq \alpha_k \leq C, \quad \sum \alpha_k y_k = 0 \quad \text{and} \quad ||\boldsymbol{\sigma}||_p = \sigma_0, \tag{18}$$

with k_σ defined in (15). The closer the hyperparameter p is to 0, the sparser the solution, but also the more difficult the optimization.

Linear SVMs

We now study the special case of optimizing the scaling factors for a linear kernel,

$$k_\sigma(\mathbf{x}_k,\mathbf{x}_l) = \sum_{i=1}^{n}\sigma_i^2 x_{k,i}x_{l,i}.$$

Assume that the data are centered and that each component is rescaled such that it has unit variance. Then the variance criterion (17), as an approximation of the radius, gives

$$R^2 = \frac{1}{m}\sum_{i=1}^{n}\sum_{k=1}^{m}\sigma_i^2(x_{k,i})^2 = \sum_{i=1}^{n}\sigma_i^2.$$

Since for hard-margin SVMs the maximum of the dual is equal to $\mathbf{w}^2/2$, the optimization of the radius-margin criterion (16) can be rewritten as the maximization of the margin under constant radius[13],

$$\min_{\sigma}\max_{\alpha}\sum_{k}\alpha_k - \frac{1}{2}\sum_{k,l=1}^{m}\alpha_k\alpha_l y_k y_l \sum_{i=1}^{n}\sigma_i^2 x_{k,i}x_{l,i} \tag{19}$$

under constraints

$$\alpha_k \geq 0, \quad \sum \alpha_k y_k = 0 \quad \text{and} \quad \sum \sigma_i^2 = 1.$$

Note that in the linear case, we recover the optimization procedure proposed in Grandvalet and Canu [2003] with $p = 2$ in (18).
As explained in the beginning of this section, it is possible to find the optimal scaling factors $\boldsymbol{\sigma}$ by gradient descent. Moreover, it turns out that (19) is a convex optimization problem in σ_k^2 (as a pointwise maximum of linear functions) and is thus easy to solve. However, when the number of variables is larger than the number of training points, it might be advantageous to take the dual of (19) and solve the following optimization problem:

$$\max_{\alpha}\sum_{k=1}^{m}\alpha_k$$

[13] It is indeed possible to fix the radius since multiplying the vector $\boldsymbol{\sigma}$ by a scalar λ would result in a multiplication of R^2 by λ^2 and a weight vector $\widetilde{w}$ with norm $\|w\|/\lambda$ and thus would not affect the bound.

under constraints

$$\alpha_k \geq 0, \quad \sum \alpha_k y_k = 0$$

$$-1 \leq \sum_{k=1}^{m} \alpha_k y_k x_{k,i} \leq 1, \quad 1 \leq i \leq n.$$

Let μ_i^+ and μ_i^- be the Lagrange multipliers associated with the two sets of constraints above. Then, one can recover the scaling parameters σ_i by $\sigma_i^2 = \mu_i^+ + \mu_i^-$, which is typically a sparse vector. A similar approach was proposed by Peleg and Meir [2004].
Please note that even if this kind of approach is appealing, this might lead to overfitting in the case where there are more dimensions than training samples.

Link with RFE

When the input components are normalized to have variance 1, the derivative of the radius/margin bound with respect to the square of the scaling factors is

$$\frac{\partial R^2 \mathbf{w}^2}{\partial \sigma_p^2} = -R^2 w_p^2 + \mathbf{w}^2.$$

From this point of view, RFE amounts to making one gradient step and removing the components with the smallest scaling factors.

3.2 Automatic Relevance Determination

Automatic Relevance Determination has first been introduced in the context of Neural Networks [MacKay, 1994, Neal, 1996]. In this section, we follow the Sparse Bayesian learning framework described in Tipping [2001]. Further details as well as application to gene expression data can be found in Li et al. [2002].
In a probabilistic framework, a model of the likelihood of the data is chosen $P(\mathbf{y}|\mathbf{w})$ as well as a prior on the weight vector, $P(\mathbf{w})$. To predict the output of a test point $\mathbf{x}$, the average of $f_{\mathbf{w}}(\mathbf{x})$ over the posterior distribution $P(\mathbf{w}|\mathbf{y})$ is computed. If this integral is too complicated to estimate, a standard solution is to predict using the function $f_{\mathbf{w}_{\text{MAP}}}$, where $\mathbf{w}_{\text{MAP}}$ is the vector of parameters called the Maximum a Posteriori (MAP), i.e.

$$\mathbf{w}_{\text{MAP}} = \underset{\mathbf{w}}{\text{argmax}}\, P(\mathbf{w}|\mathbf{y}) = \underset{\mathbf{w}}{\text{argmin}}\; -\log P(\mathbf{y}|\mathbf{w}) - \log P(\mathbf{w}). \tag{20}$$

The right hand side of the previous equation can be interpreted as the minimization of a regularized risk, the first term being the empirical risk and the second term being the regularization term.
One way to get a sparse vector $\mathbf{w}_{\text{MAP}}$ is to introduce several hyperparameters β_i controlling the variance of the w_i [Tipping, 2001],

$$P(\mathbf{w}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \sqrt{\frac{\beta_i}{2\pi}} \exp(-\beta_i w_i^2/2).$$

The vector $\boldsymbol{\beta}$ is learned by *maximum likelihood type II*, i.e. by finding the vector $\boldsymbol{\beta}$ which maximizes (assuming (improper) flat hyperprior on $\boldsymbol{\beta}$),

$$P(\mathbf{y}|\boldsymbol{\beta}) = \int P(\mathbf{y}|\mathbf{w}) P(\mathbf{w}|\boldsymbol{\beta}) d\mathbf{w}. \tag{21}$$

Using a Laplace approximation to compute the above integral and setting the derivative of (21) to 0, the vector $\boldsymbol{\beta}$ satisfies at the optimum [MacKay, 1994, Tipping, 2001]

$$\beta_i = \frac{1 - \beta_i H_{ii}^{-1}}{(w_i)_{\text{MAP}}^2}, \tag{22}$$

where H is the Hessian of the log posterior around its maximum, $H = -\nabla^2 \log P(\mathbf{w}|\mathbf{y})|_{\mathbf{w}_{\text{MAP}}}$.
An iterative scheme is adopted: compute $\mathbf{w}_{\text{MAP}}$ with (20) and update $\boldsymbol{\beta}$ using (22).

- When a component is not useful for the learning task, the maximization of the marginalized likelihood (21) over the corresponding β_i will result in $\beta_i \to \infty$ and thus $w_i \to 0$, effectively pruning the weight w_i.
- From this point of view, $\beta_i^{-1/2}$ is the Bayesian equivalent of the scaling factors σ_i introduced in Section 3.1.
- The prior on the weights w_i can be computed as

$$P(w_i) = \int P(w_i|\beta_i)P(\beta_i)d\beta_i.$$

When the hyperprior on β_i is a Gamma distribution, this gives a Student-t distribution on $P(w_i)$ [Tipping, 2001]: it is sharply peaked around 0 and thus tends to set the weights w_i at 0, leading to a sparse solution.

3.3 Variable Scaling: Extension to Maximum Entropy Discrimination

The Maximum Entropy Discrimination (MED) framework has been introduced in Jaakkola et al. [1999]. It is a probabilistic model in which one does not learn parameters of a model, but distributions over them. Those distributions are found by minimizing the KL divergence with a prior distribution while taking into account constraints given by the labeled examples. For classification, it turns out that the optimization problem solved in Jaakkola et al. [1999] is very similar to the SVM one.
Feature selection can be easily integrated in this framework [Jebara and Jaakkola, 2000]. For this purpose, one has to specify a prior probability p_0 that a feature is active.
If w_i would be the weight associated with a given feature for a linear model (as found by a linear SVM for instance), then the expectation of this weight taking into account this sparse prior is modified as follows Jebara and Jaakkola [2000],

$$\frac{w_i}{1 + \frac{1-p_0}{p_0}\exp(-w_i^2)}.$$

This has the effect of discarding the components for which

$$w_i^2 \ll \log \frac{1-p_0}{p_0}.$$

For this reason, even though the feature selection is done in a complete different framework than a standard SVM, this algorithm turns out to be similar to RFE in the sense that it ignores features whose weights are smaller than a threshold. The MED framework was recently extended to the case where multiple inter-related learning tasks are jointly solved [Jebara, 2004].

3.4 Joint Classifier and Feature Optimization (JCFO)

The Joint Classifier and Feature Optimization (JCFO) algorithm [Krishnapuram et al., 2004] is similar to sparse Gaussian Process for classification (see also Seeger [2000], Williams and Barber [1998]) with an ARD prior (Section 3.2). It considers classifiers of the form:

$$\sum_{k=1}^{m} \alpha_k k(\boldsymbol{\sigma} * \mathbf{x}, \boldsymbol{\sigma} * \mathbf{x}_k) + b.$$

The aim of this classifier is to find a function with a sparse vector $\boldsymbol{\alpha}$ *and* a sparse vector of scaling factors $\boldsymbol{\sigma}$. This is achieved by using a Laplacian prior on $\boldsymbol{\alpha}$ and $\boldsymbol{\sigma}$. The inference in this probabilistic model is done by a type of EM algorithm as well as by conjugate gradient descent on $\boldsymbol{\sigma}$. Note that, as in Section 3.1, this gradient descent can be avoided in the case of a linear kernel.

4 Sparsity Term

In the case of linear models, indicator variables are not necessary as feature selection can be enforced on the parameters of the model directly. This is generally done by adding a *sparsity term* to the objective function that the model minimizes. To link minimization problem (4) of the introductory section to methods that use a sparsity term for feature selection purposes, we consider linear decision functions of the form $f(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$, indicator variables $\boldsymbol{\sigma} \in \{0, 1\}^n$ and we make use of the following lemma,

Lemma: *If the learning criterion T can be expressed as $T(\mathbf{w}, \boldsymbol{\sigma}) = \sum L(f(\mathbf{w} * \boldsymbol{\sigma}, \mathbf{x}_k), y_k) + \Omega(\mathbf{w})$ and Ω is component-wise minimized at 0, then*

$$\min_{\mathbf{w},\ l_0(\boldsymbol{\sigma})=\sigma_0} T(\mathbf{w}, \boldsymbol{\sigma}) = \min_{l_0(\mathbf{w})=\sigma_0} T(\mathbf{w}, \mathbf{1}), \tag{23}$$

where the zero "norm" l_0 is defined as $l_0(\boldsymbol{\sigma}) :=$ cardinality$(\{i \in \{1, \cdots, n\} : \sigma_i \neq 0\})$.
Proof: Let $\mathbf{w}^*$ and $\boldsymbol{\sigma}^*$ minimize the left hand side of (23). Then $T(\mathbf{w}^*, \boldsymbol{\sigma}^*) \geq T(\mathbf{w}^* * \boldsymbol{\sigma}^*, \mathbf{1})$ because setting one of the component of $\mathbf{w}$ to 0 will decrease Ω by assumption. Thus, the left hand side of (23) is larger than its right hand side. On the other hand, if $\mathbf{w}^*$ is the minimizer of the right hand side, defining $\sigma_i = 1_{w_i^* \neq 0}$ shows the other inequality.

In other words, in the linear case, one can ignore the scaling factors $\boldsymbol{\sigma}$ and directly find a sparse vector $\mathbf{w}$. The following section presents in more detail how this is implemented for several algorithms. For the sake of simplicity, we will assume that the target is in $\{-1, 1\}$. When generalizations to more complex problems (regression or multi-class) are possible, it will be stated clearly.

4.1 Feature Selection as an Optimization Problem

Most linear models that we consider can be understood as the result of the following minimization:

$$\min_{\mathbf{w},b} \frac{1}{m} \sum_{k=1}^{m} L\left(\mathbf{w} \cdot \mathbf{x}_k + b, y_k\right) + C\Omega(\mathbf{w}),$$

where $L(f(\mathbf{x}_k), y_k)$ measures the loss of function $f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x} + b)$ on the training point $(\mathbf{x}_k, y_k)$, $\Omega(\mathbf{w}) : \mathbb{R}^n \to \mathbb{R}_+$ is a penalizing term and C is a trade-off coefficient balancing the empirical error with this penalizing term. Examples of empirical errors are:

1. *The ℓ_1 hinge loss*:
$$\ell_{\text{hinge}}\left(\mathbf{w} \cdot \mathbf{x} + b, y\right) := \left|1 - y\left(\mathbf{w} \cdot \mathbf{x} + b\right)\right|_+,$$
where $|z|_+ = z$ if $z > 0$ and $|z|_+ = 0$ otherwise.
2. *The ℓ_2 loss*:
$$\ell_2\left(\mathbf{w} \cdot \mathbf{x} + b, y\right) := \left(\mathbf{w} \cdot \mathbf{x} + b - y\right)^2.$$
3. *The Logistic loss*:
$$\ell_{\text{Logistic}}\left(\mathbf{w} \cdot \mathbf{x} + b, y\right) := \log(1 + e^{-y(\mathbf{w} \cdot \mathbf{x} + b)}).$$
This loss, usually used in logistic regression, is based on the following model: $\log\left(\frac{P(y=1|\mathbf{w},b)}{1-P(y=1|\mathbf{w},b)}\right) = \mathbf{w} \cdot \mathbf{x} + b$.

The penalizing terms that we will consider here will be of two types:

1. *The ℓ_0 norm*:
$$\Omega(\mathbf{w}) = \ell_0(\mathbf{w})$$
representing the number of non-zero coordinates of $\mathbf{w}$.
2. *The ℓ_1 norm*:
$$\Omega(\mathbf{w}) = \sum_{i=1}^{n} |w_i|.$$

Table 1 introduces the possible combinations of loss and sparsity along with the corresponding algorithms.

Loss \ Sparsity	ℓ_0	ℓ_1
hinge	FSV [4.3]	ℓ_1 SVM [4.2]
ℓ_2	multiplicative update [4.3]	LASSO [4.4]
Logistic	×	Generalized Lasso [4.4]

Table 1. Methods that have been designed to enforce feature selection during the training of a linear model. A cross means that such combination has not been considered so far. The number in brackets refers to the section number.

4.2 ℓ_1 Support Vector Machine

The ℓ_1 Support Vector Machine (ℓ_1-SVM) of Bradley and Mangasarian [1998] for classification solves the following optimization problem:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} |w_i| + C \sum_{k=1}^{m} \xi_k$$

subject to: $\xi_k \geq 0$ and $y_k \left(\mathbf{w} \cdot \mathbf{x}_k + b\right) \geq 1 - \xi_k$. The main difference compared to a classical SVM is the replacement of the quadratic regularization term $\|\mathbf{w}\|_2^2$ by the ℓ_1 norm $\sum_i |w_i|$. This slight change in the regularization term induces a big difference in the final outcome of the optimization. This is due to the strict convexity shape of the quadratic norm. Assume that two linear models parameterized by $\mathbf{w}_1$ and $\mathbf{w}_2$ are consistent on the training set. Assume furthermore that $\mathbf{w}_1$ uses only the first half of the features and $\mathbf{w}_2$ the second half (the input space is built from redundant features). Then any linear combination: $\mathbf{w} = (1-\lambda)\mathbf{w}_1 + \lambda\mathbf{w}_2$ ($\lambda \in (0,1)$) will have a smaller ℓ_2 norm than $\mathbf{w}_1$ or $\mathbf{w}_2$. This implies - in this particular case - that choosing a vector $\mathbf{w}$ with more features induces a strictly smaller ℓ_2 norm. This shows that the SVM tends to return a model that uses many redundant features. It is by the way one of the strengths of the SVM to distribute the classification decision among many redundant features, making it more robust to the overall noise.[14] In the context of feature selection, such a property might be a drawback. The introduction of the ℓ_1 norm tends to remove this property by giving the same value of the regularization term to all the $\mathbf{w} \in [\mathbf{w}_1, \mathbf{w}_2]$. This lets other factors (like the minimization of the empirical error) choose the right model and hence choose a sparse model if it decreases the empirical error. Note that the trade-off parameter C can be used to balance the amount of sparsity relative to the empirical error. A small C will lead to a sparse linear model but whose empirical error is not as controlled as with a large C.
The ℓ_1 SVM was applied as a feature selection method by different authors. Bradley and Mangasarian [1998] introduced this version of the SVM more like a general classification technique but they noted the ability of this method to return sparse linear models. Fung and Mangasarian [2003] exploited this property to perform feature selection introducing a new optimization technique. Note that the ℓ_1 SVM can be defined for regression as well. Bi et al. [2003] use this approach in the context of drug design.

4.3 Concave Minimization

In the case of linear models, feature selection can be understood as the following optimization problem:

$$\min_{\mathbf{w},b} \ell_0(\mathbf{w})$$

subject to: $y_k \left(\mathbf{w} \cdot \mathbf{x}_k + b\right) \geq 0$. Said differently, feature selection is interpreted as finding a $\mathbf{w}$ with as few non zero coordinates as possible such that the derived linear model is consistent on the training set. This problem is known to be NP-hard [Amaldi and Kann, 1998] and hence cannot be solved directly. In this section, we present two attempts to approximately optimize it. Both approximations are based on replacing ℓ_0 by a smooth function whose gradient can be computed and can be used to perform a gradient descent. Note that the above problem does not match the goal of feature selection in machine learning. The latter is indeed interested in improving generalization error. In the current set-up, we are just interested in finding the smallest number of features consistent with the training set. This can obviously lead to overfitting when

[14] If each redundant feature is perturbed by the same zero mean random noise, adding these features would reduce the overall influence of this noise to the output of the SVM.

the number of training samples is much smaller than the number of dimensions.[15] This supports the idea that exact minimization of the ℓ_0 norm is not desired and it motivates approximations that push the solution towards low capacity systems for which the generalization error is better controlled. Such systems might be, for instance, large margin classifiers. The iterative nature of the methods that we describe below make it possible as well to use early stopping. The latter technique computes an estimate of the generalization error on a validation set and stops the procedure when the error estimate is too high.

***F**eature **S**election Concav**e** (FSV)*

Bradley and Mangasarian [1998] propose to approximate the function $\ell_0(\mathbf{w})$ as:

$$\ell_0(\mathbf{w}) \approx \sum_{i=1}^{n} 1 - \exp(-\alpha|w_i|).$$

The coefficient α controls the steepness of the function and its closeness to $\ell_0(\mathbf{w})$. In their paper, Bradley and Mangasarian suggest to take $\alpha = 5$ as a first guess. Note that this function is not differentiable directly but a constrained gradient descent can be applied without any difficulty. Algorithm 3 presents the method. Although this algorithm is presented here for separable datasets, it is described in the original paper for non-separable datasets. The errors are then computed using the hinge loss.

Algorithm 3 Feature **S**election Concav**e** (FSV)

Require: α: controls the steepness of the objective function
1: Start with $\mathbf{v}^0$
2: cont=true; t=0;
3: **while** (cont==true) **do**
4: Let $\mathbf{v}^*$ be the solution of the Linear Program: {Find the descent direction}

$$\min_{\mathbf{v}} \sum_{k=1}^{n} \alpha e^{-\alpha v_k^t} \left(v_k - v_k^t\right)$$

subject to: $y_k \left(\mathbf{w} \cdot \mathbf{x}_k + b\right) \geq 1$, $-v_k \leq w_k \leq v_k$
5: $\mathbf{v}^{t+1} = \mathbf{v}^*$;
6: **if** $(\mathbf{v}^{t+1} == \mathbf{v}^t)$ **then**
7: cont=false; {If nothing changes, stop}
8: **end if**
9: t=t+1;
10: **end while**

Multiplicative Update

Weston et al. [2003] use a slightly different function. They replace the ℓ_0 norm by:

$$\ell_0(\mathbf{w}) \leftrightarrow \sum_{i=1}^{n} \log\left(\epsilon + |w_i|\right).$$

Although not a good approximation of ℓ_0, Weston *et al.* argue that its minimum is close to the minimum of the ℓ_0 norm. The interest of taking such an approximation is that it leads directly to an iterative scheme whose basic step is a classical SVM optimization problem. The multiplicative update can therefore be implemented very quickly from any SVM optimization code. Algorithm 4 describes the approach.
As for FSV, the multiplicative update can be extended to non separable datasets. The errors are then measured with a quadratic loss. Both FSV and multiplicative update can be generalized to regression problems. We refer the reader to the original papers for more details.

[15] Having a small number of samples increases the chance of having a noisy feature completely correlated with the output target.

Algorithm 4 Multiplicative update

Start with $\sigma^0 = (1, ..., 1) \in \mathbb{R}^n$
cont=true; t=0;
while (cont==true) **do**
  Let $\mathbf{w}^*$ be the minimum of: {SVM optimization}
  $\min_{\mathbf{w}} \|\mathbf{w}\|_2^2$
  subject to: $y_k \left(\mathbf{w} \cdot (\boldsymbol{\sigma}^t * \mathbf{x}_k) + b\right) \geq 1$
  $\boldsymbol{\sigma}^{t+1} = \boldsymbol{\sigma}^t * \mathbf{w}^*$; {$*$ is the component-wise multiplication}
  if $(\boldsymbol{\sigma}^{t+1} == \boldsymbol{\sigma}^t)$ **then**
    cont=false; {If nothing changes, stop}
  end if
  t=t+1;
end while

4.4 LASSO

The LASSO technique (Least Absolute Shrinkage and Selection Operator) [Tibshirani, 1996] is very similar in its spirit to the ℓ_1 SVM. It minimizes the following problem:

$$\min_{\mathbf{w},b} \sum_{k=1}^{m} (\mathbf{w} \cdot \mathbf{x}_k - y_k)^2$$

subject to: $\sum_{i=1}^{n} |w_i| \leq \sigma_0$. The use of the ℓ_1 norm constraint on the parameter leads to a sparse model as in the case of the ℓ_1 SVM. It can be used for regression and classification. Recently, the LASSO technique has been generalized to handle classification problem with a more adequate loss. Roth [2003] defined what is called *generalized LASSO* as:

$$\min_{\mathbf{w},b} \sum_{k=1}^{m} \log(1 + e^{-y_k(\mathbf{w} \cdot \mathbf{x}_k + b)})$$

subject to: $\sum_{i=1}^{n} |w_i| \leq \sigma_0$. Although the problem is convex, it is not quadratic nor linear. To solve it, Roth suggests to use an Iterative Reweighed Least Square scheme. The generalized LASSO has the advantage of producing sparse models whose outputs can be interpreted as probabilities.

4.5 Other Methods

This section is by no means an exhaustive survey of all machine learning techniques that can be understood as minimizing an empirical error plus a sparsity term. Such an interpretation encompasses too many methods to discuss them in this chapter. We have instead presented examples of which we believe that they cover a wide range of approaches. Other methods, like sparse kernel Fisher discriminant [Mika et al., 2000], the grafting method [Perkins et al., 2003] (see also Section 2.2) or the Potential Support Vector Machine [Hochreiter and Obermayer, 2004] can be understood - in a certain sense - as minimizing an empirical error plus a sparsity term. In fact, any machine learning technique involving a linear model can be extended to implement feature selection by adding a sparsity term. We have shown two of those sparsity terms in this section (namely the ℓ_1 and the ℓ_0 norm). We believe they can be combined with most objective functions and optimized using the methods described, or variants of them.

5 Discussions and Conclusions

The introduction of this chapter provides a theoretical framework which unifies many embedded feature selection methods that were introduced during the last few years. This framework is built upon the concept of scaling factors. We discussed embedded methods along *how* they approximate the proposed optimization problem:

- Explicit removal or addition of features - the scaling factors are optimized over the discrete set $\{0,1\}^n$ in a greedy iteration,
- Optimization of scaling factors over the compact interval $[0,1]^n$, and
- Linear approaches, that directly enforce sparsity of the model parameters.

From the literature that covers embedded methods it is neither possible to infer a ranking that reflects the relative abilities of the methods nor is it possible to state which methods work best in which scenario. These conclusions could be drawn only from a systematic comparison - which clearly is beyond the scope of this chapter.
Every family of feature selection methods (filter, wrapper and embedded) has its own advantages and drawbacks. In general, filter methods are fast, since they do not incorporate learning. Most wrapper methods are slower than filter methods, since they typically need to evaluate a cross-validation scheme at every iteration. Whenever the function that measures the quality of a scaling factor can be evaluated faster than a cross-validation error estimation procedure, we expect embedded methods to be faster than wrapper approaches. Embedded methods tend to have higher capacity than filter methods and are therefore more likely to overfit. We thus expect filter methods to perform better if only small amounts of training data are available. Embedded methods will eventually outperform filter methods as the number of training points increase.

Acknowledgements

We would like to thank Bernhard Schölkopf, N. Jeremey Hill and Michael Schröder for their help with this work. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. T.N.L. was supported by a grant from the Studienstiftung des deutschen Volkes.

References

E. Amaldi and V. Kann. On the Approximability of Minimizing non zero Variables or Unsatisfied Relations in Linear Systems. *Theoretical Computer Science*, 209:237–260, 1998.

J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality Reduction via Sparse Support Vector Machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003.

O. Bousquet. *Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms*. PhD thesis, École Polytechnique, 2002.

P. S. Bradley and O. L. Mangasarian. Feature Selection via Concave Minimization and Support Vector Machines. In *Proc. 15th International Conf. on Machine Learning*, pages 82–90. Morgan Kaufmann, San Francisco, CA, 1998.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

O. Chapelle. *Support Vector Machines: Induction Principles, Adaptive Tuning and Prior Knowledge*. PhD thesis, LIP6, Paris, 2002.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing Multiple Parameters for Support Vector Machines. *Machine Learning*, 46(1-3):131–159, 2002.

S. Chen, S.A. Billings, and W. Luo. Orthogonal Least Squares and Their Application to Non-linear System Identification. *International Journal of Control*, 50:1873–1896, 1989.

T. Cover and J. Thomas. *Elements of Information Theory*. Wiley and Sons, USA, 1991.

R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, New York, USA, second edition, 2001.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

G. Fung and O. L. Mangasarian. A Feature Selection Newton Method for Support Vector Machine Classification. *Computational Optimization and Aplications*, pages 1–18, 2003.

C. Gentile. Fast Feature Selection from Microarray Expression Data via Multiplicative Large Margin Algorithms. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Y. Grandvalet and S. Canu. Adaptive Scaling for Feature Selection in SVMs. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, Cambridge, MA, USA, 2003. MIT Press.

A. J. Grove, N. Littlestone, and D. Schuurmans. General Convergence Results for Linear Discriminant Updates. In *Computational Learing Theory*, pages 171–183, 1997.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene Selection for Cancer Classification using Support Vector Machines. *Journal of Machine Learning Research*, 3:1439–1461, March 2003.

S. Hochreiter and K. Obermayer. Gene Selection for Microarray Data. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*. MIT Press, Cambridge, Massachusetts, 2004.

T. Jaakkola, M. Meila, and T. Jebara. Maximum Entropy Discrimination. Technical Report AITR-1668, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1999.

T. Jebara. Multi-Task Feature and Kernel Selection For SVMs. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

T. Jebara and T. Jaakkola. Feature Selection and Dualities in Maximum Entropy Discrimination. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

J. Kivinen and M. Warmuth. The Perceptron Algorithm vs. Winnow: Linear vs. Logarithmic Mistake Bounds when few Input Variables are Relevant. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 289–296, New York, USA, 1995. ACM Press.

B. Krishnapuram, L. Carin, and A. Hartemink. Gene Expression Analysis: Joint Feature Selection and Classifier Design. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*. MIT Press, Cambridge, MA, 2004.

T.N. Lal, M. Schröder, T. Hinterberger, J. Weston, M. Bogdan, N. Birbaumer, and B. Schölkopf. Support Vector Channel Selection in BCI. *IEEE Transactions on Biomedical Engineering. Special Issue on Brain-Computer Interfaces*, 51(6):1003–1010, June 2004.

Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal Brain Damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, San Mateo, CA, 1990. Morgan Kauffman.

P. Leray and P. Gallinari. Feature Selection with Neural Networks. *Behaviormetrika*, 26(1), 1999.

Y. Li, C. Campbell, and M. Tipping. Bayesian Automatic Relevance Determination Algorithms for Classifying Gene Expression Data. *Bioinformatics*, 18(10):1332–1339, 2002.

A. Luntz and V. Brailovsky. On the Estimation of Characters Obtained in Statistical Procedure of Recognition. *Technicheskaya Kibernetica*, 1996.

D. J. C. MacKay. Bayesian non-linear modelling for the prediction competition. *ASHRAE Transactions*, 100(2): 1053–1062, 1994.

S. Mika, G. Rätsch, and K.-R. Müller. A Mathematical Programming Approach to the Kernel Fisher Algorithm. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, pages 591–597, Cambridge, MA, USA, 2000. MIT Press.

R. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996.

D. Peleg and R. Meir. A feature selection algorithm based on the global minimization of a generalization error bound. In *NIPS 18*, 2004.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space. *Journal of Machine Learning Research*, 3:1333–1356, 2003.

J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

A. Rakotomamonjy. Variable Selection Using SVM-based Criteria. *Journal of Machine Learning Research*, 3:1357–1370, 2003.

I. Rivals and L. Personnaz. MLPs (Mono-Layer Polynomials and Multi-Layer Perceptrons) for Nonlinear Modeling. *Journal of Machine Learning Research*, 3:1383–1398, 2003.

V. Roth. The Generalized LASSO. *IEEE Transactions on Neural Networks*, 2003.

M. Seeger. Bayesian Model Selection for Support Vector Machines, Gaussian Processes and Other Kernel Classifiers. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, Cambridge, MA, USA, 2000. MIT Press.

Spider. Machine Learning Toolbox http://www.kyb.tuebingen.mpg.de/bs/people/spider/, 2004.

H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar. Ranking a Random Feature for Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1399–1414, 2003.

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

M. E. Tipping. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

V. Vapnik and O. Chapelle. Bounds on Error Expectation for Support Vector Machines. *Neural Computation*, 12(9), 2000.

V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, USA, 1998.

J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping. Use of the Zero-Norm with Linear Models and Kernel Methods. *Journal of Machine Learning Research*, 3:1439–1461, March 2003.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature Selection for SVMs. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 526–532, Cambridge, MA, USA, 2000. MIT Press.

C. Williams and D. Barber. Bayesian Classification with Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(20), 1998.

J. Zhu and T. Hastie. Classification of Gene Microarrays by Penalized Logistic Regression. *Biostatistics*, 5(3):427–443, 2003.