Proceedings of the IASTED International Conference
ARTIFICIAL INTELLIGENCE AND APPLICATIONS
February 16-18, 2004, Innsbruck, Austria

Adaptive Kernel Matching Pursuit for Pattern Classification

Vlad Popovici
Signal Processing Institute
Swiss Federal Institute of Technology
Lausanne, CH-1015, Switzerland
email: Vlad.Popovici@epfl.ch

Jean-Philippe Thiran
Signal Processing Institute
Swiss Federal Institute of Technology
Lausanne, CH-1015, Switzerland
email: JP.Thiran@epfl.ch

ABSTRACT
[1] A sparse classifier is guaranteed to generalize better than a denser one, given they perform identical on the training set. However, methods like Support Vector Machine, even if they produce relatively sparse models, are known to scale linearly as the number of training examples increases. A recent proposed method, the Kernel Matching Pursuit, presents a number of advantages over the SVM, like sparser solutions and faster training. In this paper we present an extension of the KMP in which we prove that adapting the dictionary to the data results in improved performances. We discuss different techniques for dictionary adaptation and present some results on standard datasets.

KEY WORDS
pattern recognition, kernel matching pursuit, sparse classifiers

1 Introduction

Recently there was a revived interest in kernel–based classification methods. Even if the research in the field dates back from seventies, it is only in the last decade that practical tools, like Support Vector Machines (SVM) [2, 18], became available. The basic idea behind SVM is to find a large margin separation boundary between classes in some feature space generated by a Mercer kernel function. Even if SVMs are quite successful in generating relatively sparse solutions, both theoretical and practical reasons justify the effort of searching for methods that produce even sparser models.

From a theoretical point of view, it has been proven [19] that the expected generalization error rate, for a binary classification, is less than $2C\log 2 - 2l^{-1}\log\eta$, with probability $1-\eta$, where l is the number of training examples and C is the compression rate. This means that a sparser model has better generalization capabilities than a denser model, given the same performance on the training set. From a practical perspective, besides generalization performances, having a sparser model means not only less storage required to hold the model, but also faster classification decisions.

A large number of techniques have been proposed to produce sparse non-linear classifiers. They range from post-processing the models produced by an SVM, like in [5], to different greedy approximations [15, 8, 17, 13, 6]. An interesting approach is pursued in [20] where the classical Matching Pursuit algorithm is extended and adapted to classification framework.

In this paper we propose an adaptive version of it that tries to improve the performances of Kernel Matching Pursuit (KMP) by tuning the dictionary to better fit data. To this end we exploit the flexibility one has in choosing the dictionary functions and we show how an adapted dictionary can be built from a larger (possibly infinite) family of functions. The structure of the paper is as follows: after a brief overview of the problem and of KMP given in section 2, an adaptive approach to KMP is presented in section 3. Finally, a number of experiments are reported in section 4 and conclusions are drawn in the last section.

2 Matching Pursuit for Pattern Classification

2.1 Supervised Learning Problem

The problem of learning classification functions from examples can be formally stated as an estimation problem of a function $\hat{f}: X \subseteq \mathbb{R}^p \to Y = \{-1, 1\}$ using the training set $Z_l = \{z_i = (\mathbf{x}_i, y_i) | i = 1, \ldots, l\} \subset Z = X \times Y$ generated by some unknown function f, such that $\hat{f}$ will correctly classify unseen examples $z = (\mathbf{x}, y)$, i.e. $\hat{f}(\mathbf{x}) = y$ for examples z that are drawn from the same underlying probability distribution $P(Z)$ as the training data.

Let $\mathcal{L}: Y \times Y \to \mathbb{R}$ be a *loss function*, i.e. a function that measures the discrepancies between the true class y_i and the predicted class $\hat{f}(\mathbf{x}_i)$ for a given set of examples and a given/estimated classification function $\hat{f}$. Without any restrictions on the class of functions $\hat{f}$ can be chosen from, even if the performances on the training set are good (e.g. $\hat{f}(\mathbf{x}_i) = y_i, \forall i = 1, \ldots, l$) it does not mean $\hat{f}$ generalizes well to unseen examples. Hence, only minimizing the training error (empirical risk)

$$R_{emp}[\hat{f}] = \frac{1}{l}\sum_{i=1}^{l}\mathcal{L}(y_i, \hat{f}(\mathbf{x}_i)) \qquad (1)$$

does not imply low test error (risk) averaged over the test

[1] Work performed with the financial support of the Swiss OFES and with the support of the IM2–NNCR of the Swiss NFS.

examples drawn from the underlying distribution $P(Z)$:

$$R[\hat{f}] = \int \mathcal{L}(y, \hat{f}(\mathbf{x}))dP(\mathbf{x}, y). \quad (2)$$

Common approaches for improving the generalization capabilities of the learned function include controlling the capacity of the function class (as in Support Vector Machines), imposing some regularization constraints or assuming some functional form for $\hat{f}$.

2.2 Matching Pursuit for Pattern Classification

Let $\mathcal{H}$ be a Hilbert space of functions equipped with an inner product $\langle \cdot, \cdot \rangle$ and a norm $\|f\| = \sqrt{\langle f, f \rangle}$. A subset $\mathcal{D} \subset \mathcal{H}$ is called *dictionary* if for each $g \in \mathcal{D}$ we have $\|g\| = 1$ and its linear span is dense in $\mathcal{H}$.

Matching Pursuit (MP) is a technique commonly used in signal processing for obtaining sparse approximations for given signals. Generally, MP is a greedy algorithm for approximating a function f with a $n-$term linear combination of some *basis functions* selected from a dictionary $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$:

$$\hat{f}_n = \sum_{k=1}^{n} \alpha_k g_{i_k} \quad (3)$$

where indices $i_k \in \{1, \ldots, N\}$.

In the case of approximating a known function f there are some conditions that guarantee that a greedy algorithm like MP will generate an optimal solution [16]. However, in the case of pattern classification the optimality is given by the generalization capabilities of the approximation, i.e. we have to minimize a criterion like (2) when the only information available is the training set Z_l.

Let now $\Sigma_n(\mathcal{D})$ be the set of all functions from $\mathcal{H}$ that can be expressed as a linear combination of at most n elements from $\mathcal{D}$. Thus,

$$\forall h \in \Sigma_n(\mathcal{D}): \qquad h = \sum_{k \leq n} \alpha_k g_{i_k}, \quad \alpha_i \in \mathbb{R} \quad (4)$$

The best achievable error by a $n-$term approximation is

$$^nR^* \triangleq \inf_{h \in \Sigma_n(\mathcal{D})} {}^nR[h] \quad (5)$$

$$= \inf_{h \in \Sigma_n(\mathcal{D})} \int \mathcal{L}(y, \sum_{i \in I} \alpha_i g_i(\mathbf{x}))dP(\mathbf{x}, y). \quad (6)$$

However, as the only information available is the training data, one has to use the empirical counterpart $^nR_{emp}[h]$ instead, as the objective function.

In its basic form, MP builds a series of approximations $\{\hat{f}_n | \hat{f}_n \in \Sigma_n(\mathcal{D}), n \geq 1\}$ by adding at each step n a term $\alpha_n g_{i_n}$ to the current approximation, where $\alpha_n \in \mathbb{R}$ and $g_{i_n} \in \mathcal{D}$ are chosen to minimize the empirical risk:

$$(\alpha_n, g_{i_n}) = \arg \min_{\substack{\alpha \in \mathbb{R} \\ g \in \mathcal{D}}} {}^nR_{emp}[\hat{f}_{n-1} + \alpha g] \quad (7)$$

where $\hat{f}_{n-1}$ is the current approximation. This amounts to performing an optimization in the function space, so using a steepest-descent approach we have

$$g_{i_n} = \arg \min_{g \in \mathcal{D}} |\langle g, -\nabla ({}^nR_{emp}) \rangle|, \quad (8)$$

in other words, g_{i_n} is the function from $\mathcal{D}$ that is most collinear with the gradient of the error function.

Finally, α_n can be chosen by a line-search minimization procedure:

$$\alpha_n = \arg \min_{\alpha \in \mathbb{R}} {}^nR_{emp}[\hat{f}_{n-1} + \alpha g_{i_n}]. \quad (9)$$

This algorithm is termed *basic MP* in [20], where two other variants are presented, aimed at globally optimizing the coefficients α_n. Indeed, as one may notice, at step n there is only one coefficient that is optimized. The previously chosen coefficients $\alpha_1, \ldots, \alpha_{n-1}$ are not optimal with respect to the currently chosen function g_{i_n}. The first proposed variant, *MP with backfitting*, re–adjusts all the coefficients previously selected with respect to the current set of basis functions. Noticing that with a proper decomposition of the function space in two complementary subspaces, one may efficiently optimize

$$(\alpha^{(n)}_{1,\ldots,n}, g_{i_n}) = \arg \min_{\substack{\alpha_{1,\ldots,n} \in \mathbb{R}^n \\ g \in \mathcal{D}}} {}^nR_{emp}[\hat{f}_{n-1} + \alpha_n g], \quad (10)$$

a second variant, called *MP with prefitting (MP-P)* is proposed in [20]. A modified version of MP-P will be used throughout all the experiments reported here and a schematic description of the algorithm is given in Appendix A.

3 Adaptive Kernel Matching Pursuit

Let $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ be a function used to generate a dictionary $\mathcal{D} = \{g_i(\cdot) = K(\cdot, \mathbf{x}_i) | i = 1, \ldots, l\}$. Then

$$\Sigma_n(\mathcal{D}) = \left\{ h(\mathbf{x}) = \sum_{k=1}^{n} \alpha_k K(\mathbf{x}, \mathbf{x}_{i_k}) | i_k \in \{1, \ldots, l\} \right\} \quad (11)$$

and this functional form closely resembles the one used in kernel machines methods (e.g. Support Vector Machines), hence the name of *Kernel MP* (KMP)[20]. Note however, that K is not necessarily a kernel function in Mercer sense, but simply any function of the above form.

Another important observation is that the dictionary $\mathcal{D}$ may contain a heterogeneous collection of functions, allowing a larger class of approximation functions to be implemented. However, one still has to control the capacity of the dictionary in order to obtain classification functions that generalize well.

Generally, in kernel machines algorithms the kernel functions depend on a hyper–parameter θ (be it a scalar or a vector, for example the degree of a polynomial kernel or γ for a Gaussian kernel – see the experimental section) that is

usually tuned by a validation procedure. Here we propose a different approach, where we try to adapt the parameter θ before starting the greedy procedure. We investigate three different strategies and we will show in the experiments section that they have the capability of generating sparser models with lower error rates than KMP or SVM.

Let now consider a slightly different form of the kernel function, namely

$$g_i(\mathbf{x};\theta_i) \triangleq K(\mathbf{x},\mathbf{x}_i;\theta_i) \tag{12}$$

where θ_i is an element of some parameter space. In contrast with previous KMP approach we let each function have its own parameter θ_i. Depending on the nature of the parameter space, we may obtain a potentially infinite dictionary, and allowing each basis function to have its own parameter increases the flexibility of the dictionary. However, only some of these functions are relevant for a given classification task. The problem is how to select an appropriate θ_i for each g_i. Naturally, we would like to have such a θ_i that will better support the building of approximation function $\hat{f}_n$. Let us consider the case when $n = 1$, so only one basis function is selected in the approximation and let this function be g_i. Then, we would like to maximize the correct classification with respect to this particular choice, i.e. we would like to minimize the loss function:

$$\theta_i = \arg\min_{\substack{\theta \\ j\in J_i}} \mathcal{L}(y_j, g_i(\mathbf{x}_j;\theta)) \tag{13}$$

$$= \arg\min_{\substack{\theta \\ j\in J_i}} \mathcal{L}(y_j, K(\mathbf{x}_j,\mathbf{x}_i;\theta)). \tag{14}$$

The set of indices J_i defines a neighborhood for each $\mathbf{x}_i$ and different choices of this neighborhood lead to different strategies for optimizing θ_i and consequently different Adaptive KMP) algorithms:

- *global optimization* (AKMP-PG): we let $J_i = \{1,\dots,l\}\backslash\{i\}$ and solve (14) for all $i = 1,\dots,l$. This means that for each point in the training set we perform a global optimization of the parameter θ. However, such a global approach may be too costly for some problems (especially if θ is a vector) and may lead to overfitting, as the dictionary obtained will be too specialized.

- *local optimization* (AKMP-PL): Let $J_i = \{j | d(\mathbf{x}_i,\mathbf{x}_j) \leq T\}$, so J_i will contain the indices of those points that are closer that T to $\mathbf{x}_i$, according to some metric d. This is the opposite strategy of the one above, and tries to avoid a possible overfitting by limiting the optimization only to a neighborhood of each point. In practice however, it may happen that a neighborhood of an example contain only examples from the same class, making the optimization pointless. In such cases we just set the corresponding parameter to a predefined value.

- *stochastic optimization* (AKMP-PS): For every point $\mathbf{x}_i$ we generate a series of random sets $J_i^b \subset \{1,\dots,l\}, b = 1,\dots,B$ by sampling without replacement from $\{1,\dots,l\}$. Then we solve B times the optimization (14) for each point $\mathbf{x}_i$, obtaining θ_i^b. The final value for the parameter is taken as the sample mean:

$$\theta_i = \frac{1}{B}\sum_{b=1}^{B} \theta_i^b. \tag{15}$$

We have tried also to use the median of $\{\theta_i^b\}$, but the results did not change significantly. While this procedure may be more computationally intensive, it also avoids overfitting the training set. In practice we noticed that a number of 20–30 samples, each containing 10%–25% of whole training data is sufficient for a good adaptation of the dictionary.

4 Experiments

In this section we will describe a number of experiments that have been performed in order to assess the performance of the algorithms presented and to study their behavior on various datasets.

In all experiments we have used a square loss function

$$\mathcal{L}_{se}(y, \hat{f}(\mathbf{x})) = \left(y - \hat{f}(\mathbf{x})\right)^2 \tag{16}$$

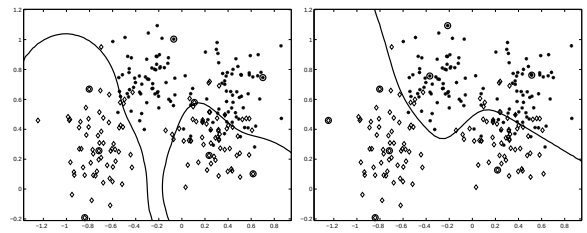
and the dictionary was based on Radial Basis Functions family:

$$g_i(\mathbf{x}) = K(\mathbf{x},\mathbf{x}_i) = \exp\left(-\gamma_i \|\mathbf{x} - \mathbf{x}_i\|^2\right) \tag{17}$$

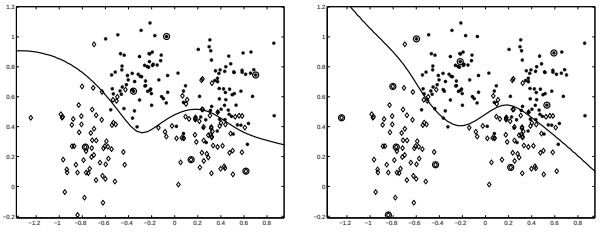
Besides the AKMP algorithm, we used for comparison the Relevance Vector Machine (RVM) [17], a Support Vector Machine (SVM) [2, 18], and the KMP with prefitting. Some datasets were already splitted into training and testing part (e.g. Ripley's synthetic data) while others were not. In the first case we used the training set both for training and validation (by splitting it in two independent subsets) while in the latter we split the full sets in three parts. All experiments have been repeated 50 times and the average classification error is reported.

The first experiment tested AKMP method on Ripley's synthetic data [14]. This set consists of 250 training patterns and 1000 testing examples, generated from a Gaussian mixture, and for which the Bayes error rate is about 8%. The best performer was AKMP-PS (8.3% error rate), the KMP with stochastically optimized dictionary, that outperformed all the other methods, while providing a very sparse solution (6 support vectors against 7 for RVM or 73 for SVM). [2]

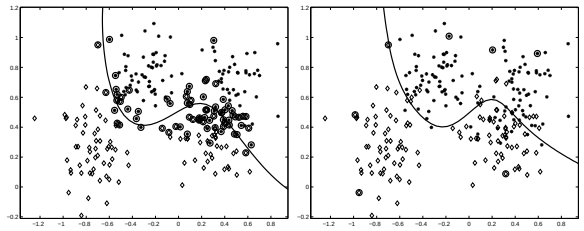
The second set of tests was carried out on a number of datasets from UCI repository [1]. On the Pima and Ionosphere datasets we used a similar testing procedure as the one in [20] and the results obtained there are given in the table 1, and for the Titanic dataset we used the same settings



(a) AKMP-PG (8.8%) (b) AKMP-PL (8.7%)



(c) AKMP-PS (8.3%) (d) KMP-P (9.4%)



(e) SVM (9.7%) (f) RVM (9.9%)

Figure 1. Different classification boundaries obtained with AKMP (Fig. 1(a)–1(c)) and with KMP (1(d)), SVM (1(e)) and RVM (1(f)), respectively. All methods, except SVM generated very sparse solutions. Note, however, the tendency to overfitting of the AKMP method with global optimization of the dictionary. For each classifier, its average error rate on the testing set is indicated between parentheses.

	Ionosphere	Pima	Titanic
AKMP-PG	6.16% (17)	26.01% (15)	22.38% (19)
AKMP-PL	5.95% (25)	25.07% (18)	22.51% (25)
AKMP-PS	5.87% (25)	23.52% (17)	22.37% (21)
KMP [20]	6.87% (50)	23.9% (7)	N/A
SVM [20, 17]	6.51% (68)	24.1% (146)	22.1% (94)
RVM [17]	N/A	19.6% (4)	23.0% (65)

Table 1. Comparative results on different datasets. Between parentheses is indicated the number of dictionary functions/support vectors used.

as in [17]. As can be noted from table 1, generally AKMP family of methods produces sparser models while improving also the classification rates. The only exception is the Pima dataset on which the model contains more functions than in the original KMP.

Another interesting observation is that, unlike SVM and like RVM, all KMP-based algorithms choose as "support vectors" points that do not necessarily lie close to the classification boundary.

5 Conclusions

In this paper we have presented an extension of Kernel Matching Pursuit method that has the potential of producing even sparser models than the original KMP at comparable or better classification rates. Relying on the fact that the dictionary functions do not have to obey very strict rules (like the Mercer kernels in SVM) one can adapt the parameters of each function individually, in such a way that it better fits the problem at hand. We have discussed three alternative techniques that adapt the dictionary using either the full training set or a subset of it and we have empirically analyzed their properties. Among them, the stochastic approach provided the most reliable results and it also avoided the overfitting in all our experiments. On the other hand it required more computational effort than the other two approaches.

The study presented here opens interesting perspectives on finding problem-dependent (or data-driven) dictionaries that would generate sparse models. It should be noted that the dictionary adaptation has been done in a rather agnostic manner: the same method has been used for data comming from various sources and we didn't incorporate any a priori knowledge. Currently, we study other means of adapting the dictionary by, for example, adapting the distance used in radial basis functions of the dictionary to incorporate different invariant properties that are more related to a specific problem.

Adaptive KMP with pre-fitting algorithm

In this section we will briefly present the algorithm for AKMP-Px that has been used throughout all the experiments reported here. This algorithm is an adaptation of the algorithm described in [20] and represents an efficient means for directly optimizing

$$\left(\alpha_{1,\ldots,n}^{(n)}, g_{i_n}\right) = \arg \min_{\substack{\alpha_{1,\ldots,n}\in\mathbb{R}^n \\ g\in\mathcal{D}}} {}^n R_{emp}[\hat{f}_{n-1} + \alpha_n g]. \tag{18}$$

Here $\alpha_{1,\ldots,n}^{(n)}$ is the n–dimensional vector of coefficients α_i obtained at iteration n. Note that in the optimization problem above, the first $n-1$ α's are absorbed in $\hat{f}_{n-1}$. Basi-

[2]The results for RVM are slightly different from those in [17] as we have trained it on the full set of 250 points and not only on a subset of 100 points as in [17].

cally, equation (18) states that we are searching for a function $g \in \mathcal{D}$ and *a set* of coefficients α_i such that they minimize the empirical loss. The method described in [20] relies on maintaining a decomposition of the dictionary functions as well as the target on two orthogonal components, given by the projections onto the space generated by the currently selected functions and its complement with respect to the full dictionary. Before starting we introduce some notations: let $D = [D(i,j)]$ be the dictionary matrix, i.e. $D(i,j) = g_j(\mathbf{x}_i), \forall g_j \in \mathcal{D}, i = 1,\ldots,l$. By $D(:,j)$ we will denote D's j–th column, i.e. the l-dimensional vector obtained by applying the function g_j to all points of the training set. Finally, I will be the vector of indices of functions selected so far, and D^I and D^C will denote the projection on the space of selected functions and its complement, respectively.

Algorithm 1 Adaptive KMP with pre-fitting in the case $\mathcal{L}(y_i, \hat{f}(x_i)) = (y_i - \hat{f}(x_i))^2$

Input: $Z_l = \{(\mathbf{x}_i, y_i) | i = 1,\ldots,l\}$
$N > 0$ – the number of terms in approximation
Output: $\hat{f}_N = \sum_{n=1}^{N} \alpha_n g_{i_n}$
Step 1: Build dictionary matrix D by tuning the kernels using any of the methods described in section 3.
*Step 2:*Initialize: $\hat{f}_0 \leftarrow 0$, $I \leftarrow []$, $D^I \leftarrow []$, $D^C \leftarrow D$, $R \leftarrow -\nabla R_{emp}[\hat{f}_0] = [y_1,\ldots,y_n]^t$
Step 3:
for $n = 1,\ldots,N$ **do**
  $i_n \leftarrow \arg\max_i |\langle D^C(:,i), R\rangle|$
  $I \leftarrow [I, i_n]$
  $\alpha_n \leftarrow \langle D^C(:,i_n), R\rangle$
  $R \leftarrow R - \alpha_n D^C(:,i_n)$
  $\alpha_{1,\ldots,n} \leftarrow \alpha_{1,\ldots,n} - \alpha_n D^I(:,i_n)$
  for $i = 1,\ldots,l$ **do**
    $\beta_i \leftarrow \langle D^C(:,i_n), D^C(:,i)\rangle$
    $D^C(:,i) \leftarrow D^C(:,i) - \beta_i D^C(:,i_n)$
    $D^I(:,i) \leftarrow D^I(:,i) - \beta_i D^I(:,i_n)$
  end for
  $D^C(:,i_n) \leftarrow 0$
  $D^C(:,i) \leftarrow \frac{D^C(:,i)}{\|D^C(:,i)\|}, \quad i = 1,\ldots,n$ {for $\|D^C(:,i)\| \neq 0$}
  $D^I(:,i_n) \leftarrow 0$
  $\beta_{i_n} \leftarrow 1$
  $D^I \leftarrow [D^I; (\beta_1,\ldots,\beta_l)]$ {add β as a new row}
end for

Note that, according to the definition of the dictionary we used, the functions g_i in the above algorithm are considered to have the norm equal to 1. Otherwise, one has to scale all the above equations involving inner products with a relevant factor.

References

[1] C. Blake and C. Merz. UCI repository of machine learning databases.

[2] B. Boser, I. Guyon, and V. Vapnik. An algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[3] S. Chen, C. Cowan, and P. Grant. Orthogonal least squares learning algorithm for radial basis function network. *IEEE Transactions on Neural Networks*, 2(2):302–309, March 1991.

[4] N. Cristianini and J. Shawe-Taylor. *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 200.

[5] T. Downs, K. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, December 2001.

[6] M. A. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1150–1159, September 2003.

[7] S. Floyd and M. Warmuth. Sample compression, learnability, and the vapnik-chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.

[8] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[9] T. Graepel, R. Herbrich, and J. Shawe-Taylor. Generalization error bounds for sparse linear classifiers. In *Proceedings of the Thirteenth Annual Conference on Computational Theory*, pages 298–303. Morgan Kaufmann, 2000.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning. Data Mining, Inference and Prediction*. Springer–Verlag, 2001.

[11] S. Mannor, R. Meir, and T. Zhang. The consistency of greedy algorithms for classification. In *Proc. of the Fifteenth Annual Conference on Computational Learning Theory*, pages 319–333, 2002.

[12] L. Mason, J. Baxter, P. Bartlett, and M. Frean. *Functional Gradient Techniques for Combining Hypotheses*, volume Advances in Large Margin Classifiers, chapter 3, pages 221–246. The MIT Press, 1999.

[13] P. B. Nair, A. Choudhury, and A. J. Keane. Some greedy learning algorithms for sparse regression and classification with Mercer kernels. *Journal of Machine Learning Research*, 3:781–801, 2002.

[14] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[15] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML2000)*, pages 911–918. Morgan Kaufmann, 2000.

[16] V. Temlyakov. Greedy algorithms and m-term approximation with regard to redundant dictionaries. Technical Report 97.10, Department of Mathematics, University of South Carolina, 1997.

[17] M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, June 2001.

[18] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[19] V. Vapnik. *Statistical Learning Theory*, volume 454 of *Lecture Notes in Economics and Mathematical Systems*. Wiley, 1998.

[20] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1):165–187, 2002.