# Kernel Matching Pursuit

**Pascal Vincent and Yoshua Bengio**
*Dept. IRO, Université de Montréal*
C.P. 6128, Montreal, Qc, H3C 3J7, Canada
{vincentp,bengioy}@iro.umontreal.ca

## Abstract

We show how Matching Pursuit can be used to build kernel-based solutions to machine-learning problems while keeping control of the sparsity of the solution, and how it can be extended to use non-squared error loss functions. We also derive MDL motivated generalization bounds for this type of algorithm. Finally, links to boosting algorithms and RBF training procedures, as well as extensive experimental comparison with SVMs are given, showing comparable results with typically sparser models.

## 1 Quick overview of Matching Pursuit

### 1.1 Basic Matching Pursuit

Matching Pursuit was introduced in the signal-processing community as an algorithm "*that decomposes any signal into a linear expansion of waveforms that are selected from a redundant dictionary of functions.*" [8]. It is a general, greedy sparse-approximation scheme. Given a finite dictionary $\mathcal{D}$ of functions in a Hilbert space $\mathcal{H}$ and a target function $f \in \mathcal{H}$ we are interested in approximations of $f$ that are expansions of the form $\tilde{f}_n = \sum_{k=1}^{n} \alpha_k g_k$ where $(\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$ and $\{g_1, \ldots, g_n\} \subset \mathcal{D}$ are chosen to minimize the squared norm of the residue $\|R_n\|^2 = \|f - \tilde{f}_n\|^2$.

However finding the optimal basis $\{g_1, \ldots, g_n\}$ for a given $n$ is in general NP-complete. So the algorithm proceeds in a constructive, greedy fashion, starting at stage 0 with $\tilde{f}_0 = 0$, and recursively appending functions to an initially empty basis. Given $\tilde{f}_n$ we build $\tilde{f}_{n+1} = \tilde{f}_n + \alpha_{n+1} g_{n+1}$ by searching for $g_{n+1} \in \mathcal{D}$ and for $\alpha_{n+1} \in \mathbb{R}$ that minimize the squared norm of the residue $\|R_{n+1}\|^2 = \|R_n - \alpha_{n+1} g_{n+1}\|^2$

$$(g_{n+1}, \alpha_{n+1}) = \underset{(g \in \mathcal{D}, \alpha \in \mathbb{R})}{\arg\min} \| \underbrace{(\sum_{k=1}^{n} \alpha_k g_k)}_{\tilde{f}_n} + \alpha g - f \|^2 \tag{1}$$

The $g_{n+1}$ that minimizes this expression is the one that maximizes $|\frac{\langle g_{n+1}, R_n \rangle}{\|g_{n+1}\|}|$ and the corresponding $\alpha_{n+1}$ is $\alpha_{n+1} = \frac{\langle g_{n+1}, R_n \rangle}{\|g_{n+1}\|^2}$.

### 1.2 Matching Pursuit with backfitting

An improvement over the basic algorithm, while still choosing $g_{n+1}$ as previously, consists in re-estimating all the coefficients $\alpha_{1..n+1}$ at each step instead of only the last $\alpha_{n+1}$:

$$\alpha_{1..n+1}^{(n+1)} = \underset{(\alpha_{1..n+1} \in \mathbb{R}^{n+1})}{\arg\min} \|(\sum_{k=1}^{n+1} \alpha_k g_k) - f\|^2 \tag{2}$$

This step is called backfitting or back-projection.

There is a further possible refinement: rather than finding $g_{n+1}$ using (1) and only then optimizing (2), which means we might be picking a dictionary function other than the one that would give the best fit, we may want to directly find

$$(g_{n+1}, \alpha_{1..n+1}^{(n+1)}) = \underset{(g \in \mathcal{D}, \alpha_{1..n+1} \in \mathbb{R}^{n+1})}{\arg\min} \|(\sum_{k=1}^{n} \alpha_k g_k) + \alpha_{n+1} g - f\|^2 \tag{3}$$

We shall call the latter procedure *pre-backfitting* and the former *post-backfitting* (as backfitting is done only *after* the choice of $g_{n+1}$).

When doing least-squares fit in a finite-dimensional vector space, both backfitting versions can be carried very efficiently through some form of orthogonal least squares procedures. The resulting algorithms are often referred to as Orthogonal Matching Pursuit [11, 3]. Let $\mathcal{B}_n = span(g_1, \ldots, g_n)$. Our implementation maintains at each step a representation of the target and every dictionary vector $g$ as the sum of two components: component $g_{\mathcal{B}_n} \in \mathcal{B}_n$ is expressed as a linear combination of current basis vectors, while component $g_{\mathcal{B}_n^\perp}$ lies in $\mathcal{B}_n$'s orthogonal complement and is expressed in the original vector space coordinates. Pre-backfitting is then achieved easily by considering only the components in $\mathcal{B}_n^\perp$: we choose $g_{n+1}$ as the $g \in \mathcal{D}$ whose $g_{\mathcal{B}_n^\perp}$ is most collinear with $R_n \in \mathcal{B}_n^\perp$. This procedure requires, at every step, only two passes through the dictionary (searching $g_{n+1}$, then updating the representation) where basic matching pursuit requires one.

## 2 Kernel Matching Pursuit and links with other paradigms

Kernel Matching Pursuit (KMP) is simply Matching Pursuit with a kernel-based dictionary, applied to problems in machine-learning. Let $\mathcal{L} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$ a training set containing $l$ (input,output) pairs sampled i.i.d. from an unknown distribution $P(X, Y)$ with $X \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ for regression problems, or $Y \in \{-1, +1\}$ for binary classification (our main focus). Then, given a kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, we use as our dictionary the kernels centered on the training points: $\mathcal{D} = \{K(\cdot, x_i) | i = 1..l\}$. Optionally, the constant function can also be included in the dictionary, which accounts for a bias term $b$: the functional form of approximation $\tilde{f}_n$ then becomes

$$\tilde{f}_n(x) = b + \sum_{i=1}^{n} \alpha_i K(x, x_{\gamma_i}) \tag{4}$$

where the $\gamma_{1..n}$ are the indexes of the "support points" selected by the algorithm. During training we only consider the values of the dictionary functions at the training points, so that it amounts to doing Matching Pursuit in a finite-dimensional function-space: a vector-space of dimension $l$. Consequently, efficient backfitting procedures can be used.

The functional form (4) is very similar to the one obtained with the *Support Vector Machine* (SVM) algorithm [1], the main difference being that SVMs impose further constraints on $\alpha_{1..n}$. However the quantity optimized by the SVM algorithm is quite different from the KMP greedy optimization, which for now[1] is a simple least-squares fit. Consequently the

[1] We discuss extension of KMP to other loss functions in section 3.1

support vectors and coefficients found by the two types of algorithms are usually different (see experimental results). One other important difference is that in KMP, capacity control is achieved by *directly* controlling the sparsity of the solution, i.e. the number $n$ of support vectors, whereas the capacity of SVMs is controlled through the box-constraint parameter $C$, which has an indirect and hardly controllable influence on sparsity.

Squared-error KMP with a Gaussian kernel and pre-backfitting is identical to *Orthogonal Least Squares Radial Basis Functions* [2] (OLS-RBF). In [13] SVMs were compared to "classical RBFs", where the RBF centers were chosen by unsupervised k-means clustering, and SVMs gave better results. To our knowledge, however, there has been no experimental comparison between OLS-RBF and SVMs, although their resulting functional forms are very much alike. This is one of the contributions of this paper.

KMP in its basic form is also very similar to boosting algorithms [4, 6], in which the chosen class of weak-learners would be the set of kernels centered on the training points. These algorithms differ mainly in the loss function they optimize, which we discuss in the second part of the next section.

## 3 Extension to non-squared error loss

### 3.1 Gradient descent in function space

It has been noticed that boosting algorithms are performing a form of gradient descent in function space with respect to particular loss functions [12, 9]. Following [5], the technique can be adapted to extend KMP to optimize arbitrary differentiable loss functions, instead of doing least-squares fitting. Given a loss function $L(y_i, \tilde{f}_n(x_i))$ that computes the cost of predicting a value of $\tilde{f}_n(x_i)$ when the true target was $y_i$, we use an alternative residue $\tilde{R}_n$ rather than the usual $R_n = f - \tilde{f}_n$ when searching for the next dictionary element to append to the basis at each step. $\tilde{R}_n$ is the direction of steepest descent (the gradient) in function space (evaluated at the data points) with respect to $L$:

$$\tilde{R}_n = \left( -\frac{\partial L(y_1, \tilde{f}_n(x_1))}{\partial \tilde{f}_n(x_1)}, \ldots, -\frac{\partial L(y_n, \tilde{f}_n(x_n))}{\partial \tilde{f}_n(x_n)} \right) \tag{5}$$

i.e. $g_{n+1}$ is chosen such that it is most collinear with this gradient. A line-minimization procedure can then be used to find the corresponding coefficient

$$\alpha_{n+1} = \arg\min_{\alpha \in \mathbb{R}} \sum_{i=1}^{m} L(f(x_i), \tilde{f}_n(x_i) + \alpha g_{n+1}(x_i))$$

This corresponds to basic matching pursuit. It is also possible to do post-backfitting, by re-optimizing all $\alpha_{1..n+1}$ to minimize the target cost (with a conjugate gradient optimizer for instance). But as this can be time-consuming, it may be desirable to do it every few steps instead of every single step.

### 3.2 Margin loss functions versus traditional loss functions for classification

While the original notion of margin in classification problems comes from the geometrically inspired hard-margin of SVMs (the smallest Euclidean distance between the decision surface and the training points), a slightly different perspective has emerged in the boosting community along with the notion of margin loss function. The margin quantity $m = y\tilde{f}(x)$ of an individual data point $(x, y)$, with $y \in \{-1, +1\}$ can be understood as a confidence measure of its classification by function $\tilde{f}$, while the class decided for is given by sign($\tilde{f}(x)$). The loss functions that boosting algorithms optimize are typically expressed as functions of $m$. Thus AdaBoost [12] uses an exponential ($e^{-m}$) margin loss function,

LogitBoost [6] uses the negative binomial log-likelihood $\log_2(1+e^{-2m})$, whose shape is similar to a smoothed version of the soft-margin SVM loss function $(1-Cx)_+$, and Doom II [9] approximates a theoretically motivated margin loss with $1-\tanh(m)$. As can be seen in Figure 1 (left), all these functions encourage large positive margins, and differ mainly in how they penalize large negative ones. In particular $1-\tanh(x)$ is expected to be more robust, as it won't penalize outliers to excess.

It is enlightening to compare these with the more traditional loss functions that have been used for neural networks in classification tasks, when we express them as functions of $m$.
Squared loss: $(\tilde{f}(x)-y)^2=(1-m)^2$
Squared loss after $\tanh$ with modified target: $(\tanh(\tilde{f}(x))-0.65y)^2=(0.65-\tanh(m))^2$

Both are illustrated on figure 1 (right). Notice how the squared loss after $\tanh$ appears similar to the margin loss function used in Doom II, except that it slightly increases for large positive margins, which is why it behaves well and does not saturate even with unconstrained weights (boosting and SVM algorithms impose further constraints).

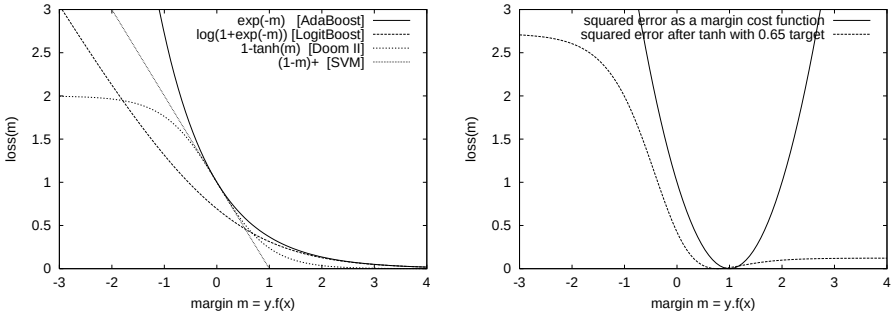


Figure 1: *Boosting and SVM margin loss functions (left) vs. traditional loss functions (right) viewed as functions of the margin. Interestingly the last-born of the margin motivated loss functions (used in Doom II) is similar to the traditional squared error after* $\tanh$.

## 4 Bounds on generalization error

The results of Vapnik on the Minimum Description Length [14] provide a framework for establishing bounds on expected generalization error for KMP algorithms. We will show that, essentially, the bound depends *linearly on the number of support vectors* and *logarithmically on the total number of training examples*

Vapnik's result states that the expected generalization error, $E_{gen}$, for binary classification, when training with $l$ examples, is less than $2\mathcal{C}\log(2)-2\log(\eta)/l$ with probability $1-\eta$, where $\mathcal{C}$ is the "compression rate": the number of bits to transfer the compressed conditional value of the training target classes (given the training input points) divided by the number of bits required to transmit them without compression, i.e., $l$. The compression is due to the representation learned by the training algorithm. Here we take advantage of the sparse representation of the learned function in terms of only $n$ "support points". To obtain a rough bound we will encode the target outputs using three sets of bits, corresponding to three terms for $\mathcal{C}$. The first one is the total empirical classification error $E_{emp}$. The second term is required to encode the identity of the support points: $\log_2(\binom{l}{n})<n\log_2 l$ bits. The third term is to encode the quantized weights $\alpha_k$ associated with each support point, which will cost $np$ bits, where $p$ is the number of bits of precision to quantize the weights, and it can be chosen as the smallest number that allows to obtain with the discretized $\alpha$'s the same classes on the training set as the undiscretized $\alpha$'s. To summarize, for KMP, we have $E_{gen}<\frac{E_{emp}}{l}+\frac{n\log l}{l}+\frac{np}{l}$. In contrast, one can obtain an expectation bound [14] for

SVMs that is $E[E_{gen}] < E[\frac{n}{l}]$, where $E$ is expectation over training sets (note that for SVMs, $n$ is random because it depends on the training set).

## 5 Experimental results on binary classification

Throughout this section, any mention of KMP without further specification of the loss function means least-squares KMP (also sometimes written *KMP-mse*) as opposed to *KMP-tanh* which refers to KMP using squared error after a hyperbolic tangent with modified targets as described earlier in section 3.2.

### 5.1 2D experiments

Figure 2 shows a 2D binary classification problem with the decision surface found by the three versions of KMP-mse and SVMs, when using a Gaussian kernel.

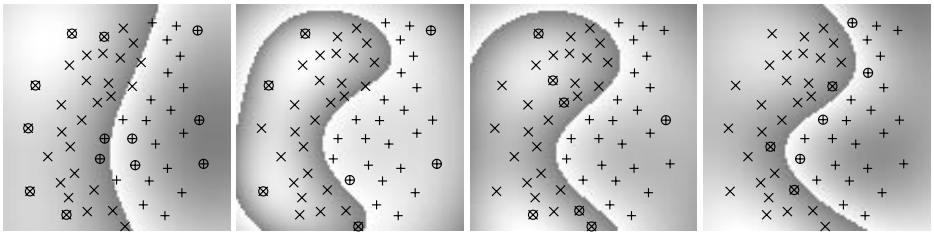

Figure 2: *From left to right: 100 iterations of basic KMP, 7 iterations of KMP post-backfitting, 7 iterations of KMP pre-backfitting, and SVM. Classes are $+$ and $\times$. Support vectors are circled. Pre-backfitting KMP and SVM appear to find equally reasonable solutions, though using different support vectors. Only SVM chooses its support vectors close to the decision surface. Post-backfitting chooses yet another support set, and its decision surface appears to have a slightly worse margin. As for basic KMP, after 100 iterations during which it mostly cycled back to previously chosen support points to improve their weights, it appears to use more support vectors than the others while still being unable to separate the data points, and is thus a bad choice if we want sparse solutions.*

All further mentions of KMP will refer to backfitting KMP, that is pre-backfitting for least-squares KMP, and post-backfitting for KMP-tanh.

Figure 3, where we used a simple dot-product kernel (i.e. linear decision surfaces), illustrates a problem that can arise when using least-squares fit: since the squared error penalizes large positive margins, the decision surface is drawn towards the cluster on the lower right, at the expense of a few misclassified points. The use of a $\tanh$ corrects this problem.

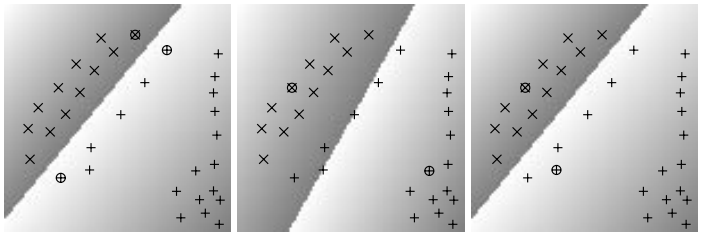

Figure 3: *Problem with least squares fit that leads KMP-mse (center) to misclassify points, but does not affect SVMs (left), and is successfully treated by KMP-tanh (right).*

### 5.2 US Postal Service Database

The purpose of this experiment was to complement the results of [13] with those obtained using KMP-mse (which, as already mentioned, is equivalent to orthogonal least squares RBF [2]). We used the same Gaussian Kernel and the same training set (7300 patterns) and independent test set (2007 patterns) of preprocessed handwritten digits. Table 1 gives the number of errors obtained by the various algorithms on the tasks consisting of discriminating each digit versus all the others (see [13] for more details). As can be seen, results obtained with KMP are comparable to those obtained for SVMs, contrarily to those obtained with "classical" RBFs.

Table 1: *USPS Results: number of errors on the test set (2007 patterns), when using the same number of support vectors as found by SVM (except last row which uses half #sv). The first half of this table was taken from [13].*

| Digit class | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| #sv | 274 | 104 | 377 | 361 | 334 | 388 | 236 | 235 | 342 | 263 |
| SVM | 16 | 8 | 25 | 19 | 29 | 23 | 14 | 12 | 25 | 16 |
| classical RBF | 20 | 16 | 43 | 38 | 46 | 31 | 15 | 18 | 37 | 26 |
| KMP with same #sv | 15 | 15 | 26 | 17 | 30 | 23 | 14 | 14 | 25 | 13 |
| KMP with half #sv | 16 | 15 | 29 | 27 | 29 | 24 | 17 | 16 | 28 | 18 |

### 5.3 Benchmark datasets

We did some further experiments, on 5 well-known datasets from the the UCI Machine Learning Databases, using Gaussian kernels. A first series of experiments used the Delve [7] system on the Mushrooms dataset. Hyper-parameters (the $\sigma$ of the kernel, the bound constraint C for SVM and the number of support points for KMP) were chosen automatically using K-fold cross validation. The results for varying sizes of the training set are summarized in table 2.

Table 2: *Results obtained on the mushrooms data set with the Delve [7] system. KMP require less support vectors, while none of the differences in error rates are significant.*

| size of train | KMP error | SVM error | p-value (t-test) | KMP #s.v. | SVM #s.v. |
|---|---|---|---|---|---|
| 64 | 6.28% | 4.54% | 0.24 | 17 | 63 |
| 128 | 2.51% | 2.61% | 0.82 | 28 | 105 |
| 256 | 1.09% | 1.14% | 0.81 | 41 | 244 |
| 512 | 0.20% | 0.30% | 0.35 | 70 | 443 |
| 1024 | 0.05% | 0.07% | 0.39 | 127 | 483 |

For Wisconsin Breast Cancer, Sonar, Pima Indians Diabetes and Ionosphere, we used a slightly different procedure: each dataset was randomly split into three equal-sized subsets for training, validation and testing. SVM, KMP-mse and KMP-tanh were then trained on the training set while the validation set was used to choose the optimal C for SVMs, and to do early stopping (decide on the number of s.v.) for KMP. This procedure was repeated 50 times to give confidence measures. Table 5.3 reports the average error rate measured on the test sets, and the rounded average number of support vectors found by each algorithm.

As can be seen from these experiments, the error rates obtained are comparable, but the KMP versions appear to require fewer support vectors. On these datasets, however, KMP-tanh did not seem to give any significant improvement over KMP-mse. Even in other experiments with added label noise, KMP-tanh didn't seem any better.

Table 3: *Results on 4 UCI-MLDB datasets. Again, error rates are not significantly different (values in parentheses are the p-values for the difference with SVMs, as given by the resampled t-test [10]), but KMPs require fewer support vectors.*

| Dataset | SVM error | KMP-mse error | KMP-tanh error | SVM #s.v. | KMP-mse #s.v. | KMP-tanh #s.v. |
|---|---|---|---|---|---|---|
| Wisc. Cancer | 3.41% | 3.40% (0.49) | 3.49% (0.45) | 42 | 7 | 21 |
| Sonar | 20.6% | 21.0% (0.45) | 26.6% (0.16) | 46 | 39 | 14 |
| Pima Indians | 24.1% | 23.9% (0.44) | 24.0% (0.49) | 146 | 7 | 27 |
| Ionosphere | 6.51% | 6.87% (0.41) | 6.85% (0.40) | 68 | 50 | 41 |

## 6 Conclusion

We have shown how Matching Pursuit provides a flexible framework to build and study alternative kernel-based methods, how it can be extended to use arbitrary differentiable loss functions, and how it relates to RBFs and boosting methods. We have also provided experimental evidence that such greedy algorithms can perform as well as SVMs, while allowing a better control of sparsity, and thus often lead to much sparser solutions. It should also be mentioned that the use of a dictionary gives additional flexibility: it could be used, for instance, to mix several kernel shapes to choose from, or to include other non-kernel functions based on prior knowledge, which opens the way to further research.

## References


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

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

[3] G. Davis, S. Mallat, and Z. Zhang. Adaptive time-frequency decompositions. *Optical Engineering*, 33(7):2183–2191, July 1994.

[4] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 148–156, 1996.

[5] J. Friedman. Greedy function approximation: a gradient boosting machine. Technical report, Dept. of Statistics, Stanford University, 1999.

[6] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University, 1998.

[7] G. Hinton, R. Neal, and R. Tibshirani. Assessing learning procedures using delve. Technical report, University of Toronto, Department of Computer Science, http://www.cs.utoronto.ca/neuron/delve/delve.html., 1995.

[8] S. Mallat and Z. Zhang. Matching pursuit with time-frequency dictionaries. *IEEE Trans. Signal Proc.*, 41(12):3397–3415, Dec. 1993.

[9] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In S. A. Solla, T. K. Leen, and K-R. Mller, editors, *Advances in Neural Information Processing Systems 12*. The MIT Press, 2000. Accepted for Publication.

[10] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. In *Advances in Neural Information Processing Systems 12*, page to appear. MIT Press, 2000.

[11] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, Nov. 1993.

[12] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 1998.

[13] B. Schoelkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45:2758–2765, 1997.

[14] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New-York, 1995.