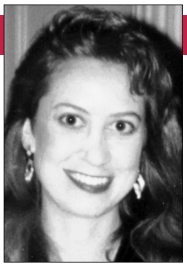


Support vector machines

By Marti A. Hearst
University of California, Berkeley
hearst@sims.berkeley.edu

My first exposure to Support Vector Machines came this spring when I heard Sue Dumais present impressive results on text categorization using this analysis technique. This issue's collection of essays should help familiarize our readers with this interesting new racehorse in the Machine Learning stable. Bernhard Schölkopf, in an introductory overview, points out that a particular advantage of SVMs over other learning algorithms is that it can be analyzed theoretically using concepts from computational learning theory, and at the same time can achieve good performance when applied to real problems. Examples of these real-world applications are provided by Sue Dumais, who describes the aforementioned text-categorization problem, yielding the best results to date on the Reuters collection, and Edgar Osuna, who presents strong results on application to face detection. Our fourth author, John Platt, gives us a practical guide and a new technique for implementing the algorithm efficiently.

–Marti Hearst

SVMs—a practical consequence of learning theory

Bernhard Schölkopf, GMD First

Is there anything worthwhile to learn about the new SVM algorithm, or does it fall into the category of "yet-another-algorithm," in which case readers should stop here and save their time for something more useful? In this short overview, I will try to argue that studying support-vector learning is very useful in two respects. First, it is quite satisfying from a theoretical point of view: SV learning is based on some beautifully simple ideas and provides a clear intuition of what learning from examples is about. Second, it can lead to high performances in practical applications.

In the following sense can the SV algorithm be considered as lying at the intersection of learning theory and practice: for certain simple types of algorithms, statistical learning theory can identify rather precisely the factors that need to be taken into account to learn successfully. Real-world applications, however, often mandate the use of more complex models and algorithms—such as neural networks—that are much harder to analyze theoretically. The SV algorithm achieves both. It constructs models that are complex enough: it contains a large class of neural nets, radial basis function (RBF) nets, and polynomial classifiers as special cases. Yet it is simple enough to be analyzed mathematically, because it can be shown to correspond to a *linear* method in a high-dimensional *feature space* nonlinearly related to input space. Moreover, even though we can think of it as a linear algorithm in a high-dimensional space, in practice, it does not involve any computations in that high-dimensional space. By the use of *kernels*, all necessary computations are performed directly in input space. This is the characteristic twist of SV methods—we are dealing with complex algorithms for nonlinear pattern recognition,[1] regression,[2] or feature extraction,[3] but for the sake of analysis and algorithmics, we can pretend that we are working with a simple linear algorithm.

I will explain the gist of SV methods by describing their roots in learning theory, the optimal hyperplane algorithm, the kernel trick, and SV function estimation. For details and further references, see Vladimir Vapnik's authoritative treatment,[2] the collection my colleagues and I have put together,[4] and the SV Web page at *http://svm.first.gmd.de*.

Learning pattern recognition from examples

For pattern recognition, we try to estimate a function $f:\mathcal{R}^N \rightarrow \{\pm 1\}$ using training data—that is, N-dimensional patterns $\mathbf{x}_i$ and class labels y_i,

$$(\mathrm{x}_1, y_1), \ldots, (\mathrm{x}_\ell, y_\ell) \in \mathcal{R}^N \times \{\pm 1\}, \qquad (1)$$

such that f will correctly classify new examples $(\mathbf{x}, y)$—that is, $f(\mathbf{x}) = y$ for examples $(\mathbf{x}, y)$, which were generated from the same underlying probability distribution $P(\mathbf{x}, y)$ as the training data. If we put no restriction on the class of functions that we choose our estimate f from, however, even a function that does well on the training data—for example by satisfying $f(\mathbf{x}_i) = y_i$ (here and below, the index i is understood to run over $1, \ldots, \ell$)—need not generalize well to unseen examples. Suppose we know nothing additional about f (for example, about its smoothness). Then the values on the training patterns carry no information whatsoever about values on novel patterns. Hence learning is impossible, and minimizing the training error does not imply a small expected test error.

Statistical learning theory,[2] or VC (Vapnik-Chervonenkis) theory, shows that it is crucial to restrict the class of functions that the learning machine can implement to one with a *capacity* that is suitable for the amount of available training data.

Hyperplane classifiers

To design learning algorithms, we thus must come up with a class of functions whose capacity can be computed. SV classifiers are based on the class of hyperplanes

$$(\mathrm{w} \cdot \mathrm{x}) + b = 0 \quad \mathrm{w} \in \mathcal{R}^N, b \in R, \qquad (2)$$

corresponding to decision functions

$$f(\mathrm{x}) = \mathrm{sign}((\mathrm{w} \cdot \mathrm{x}) + b). \qquad (3)$$

We can show that the *optimal hyperplane*, defined as the one with the maximal margin of separation between the two classes (see Figure 1), has the lowest ca-

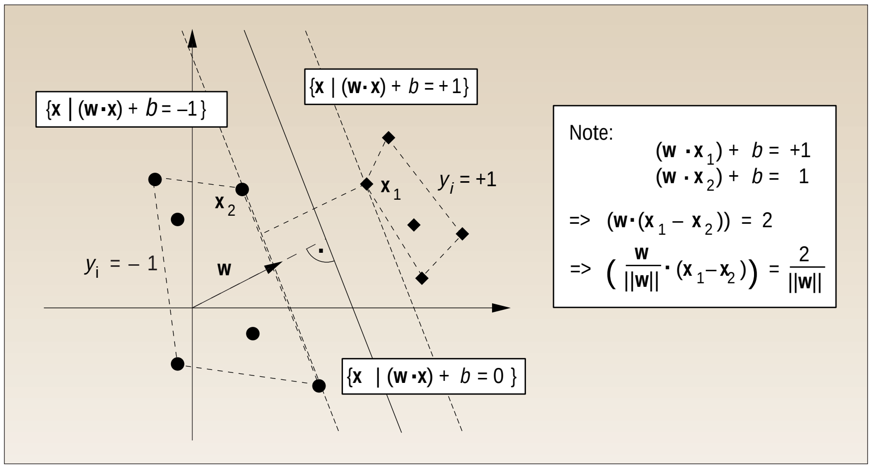


Figure 1. A separable classification toy problem: separate balls from diamonds. The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half way. There is a weight vector **w** and a threshold *b* such that $y_i \cdot ((\mathbf{w}\cdot\mathbf{x}_i) + b) > 0$. Rescaling **w** and *b* such that the point(s) closest to the hyperplane satisfy $|(\mathbf{w}\cdot\mathbf{x}_i) + b| = 1$, we obtain a form $(\mathbf{w},b)$ of the hyperplane with $y_i((\mathbf{w}\cdot x_i) + b) \geq 1$. Note that the margin, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$. To maximize the margin, we thus have to minimize $|\mathbf{w}|$ subject to $y_i((\mathbf{w}\cdot\mathbf{x}_i) + b) \geq 1$.

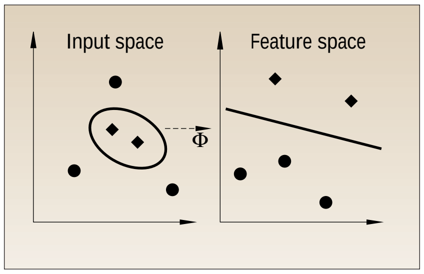


Figure 2. The idea of SV machines: map the training data nonlinearly into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

pacity. It can be uniquely constructed by solving a constrained quadratic optimization problem whose solution **w** has an expansion $\mathbf{w} = \sum_i v_i \mathbf{x}_i$ in terms of a subset of training patterns that lie on the margin (see Figure 1). These training patterns, called support vectors, carry all relevant information about the classification problem.

Omitting the details of the calculations, there is just one crucial property of the algorithm that we need to emphasize: both the quadratic programming problem and the final decision function $f(\mathbf{x}) = \mathrm{sign}(\sum_i v_i(\mathbf{x}\cdot\mathbf{x}_i) + b)$ depend only on dot products between patterns. This is precisely what lets us generalize to the nonlinear case.

Feature spaces and kernels

Figure 2 shows the basic idea of SV machines, which is to map the data into some other dot product space (called the *feature space*) *F* via a nonlinear map

$$\Phi: \mathcal{R}^N \to F, \tag{4}$$

and perform the above linear algorithm in *F*. As I've noted, this only requires the evaluation of dot products.

$$k(\mathbf{x},\mathbf{y}) := (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})). \tag{5}$$

Clearly, if *F* is high-dimensional, the right-hand side of Equation 5 will be very expensive to compute. In some cases, however, there is a simple *kernel k* that can be evaluated efficiently. For instance, the polynomial kernel

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d \tag{6}$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly *d* dimensions of $\mathcal{R}^N$. For *d*=2 and **x**, **y**∈ $\mathcal{R}^2$, for example, we have

$$\begin{aligned}(\mathbf{x}\cdot\mathbf{y})^2 &= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\cdot\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right)^2 \\ &= \left(\begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}\cdot\begin{pmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{pmatrix}\right) \\ &= (\Phi(\mathbf{x})\cdot\Phi(\mathbf{y})),\end{aligned} \tag{7}$$

defining $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. More generally, we can prove that for every kernel that gives rise to a positive matrix $(k(\mathbf{x}_i,\mathbf{x}_j))_{ij}$, we can construct a map Φ such that Equation 5 holds.

Besides Equation 6, SV practitioners use radial basis function (RBF) kernels such as

$$k(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2 / (2\sigma^2)), \tag{8}$$

and sigmoid kernels (with gain κ and offset Θ)

$$k(\mathbf{x},\mathbf{y}) = \tanh(\kappa(\mathbf{x}\cdot\mathbf{y}) + \Theta). \tag{9}$$

SVMs

We now have all the tools to construct nonlinear classifiers (see Figure 2). To this end, we substitute $\Phi(\mathbf{x}_i)$ for each training example $\mathbf{x}_i$, and perform the optimal hyperplane algorithm in *F*. Because we are using kernels, we will thus end up with nonlinear decision function of the form

$$f(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{\ell} v_i \cdot k(\mathbf{x},\mathbf{x}_i) + b\right). \tag{10}$$

The parameters v_i are computed as the solution of a quadratic programming problem.

In input space, the hyperplane corresponds to a nonlinear decision function whose form is determined by the kernel (see Figures 3 and 4).

The algorithm I've described thus far has a number of astonishing properties:

- It is based on statistical learning theory,
- It is practical (as it reduces to a quadratic programming problem with a unique solution), and
- It contains a number of more or less heuristic algorithms as special cases: by the choice of different kernel functions, we obtain different architectures (Figure 4), such as polynomial classifiers (Equation 6), RBF classifiers (Equation 8 and Figure 3), and three-layer neural nets (Equation 9).

The most important restriction up to now has been that we were only considering the case of classification. However, a generalization to regression estimation—that is, to *y*∈ $\mathcal{R}$, can be given. In this case, the algorithm tries to construct a linear function in the feature space such that the training points lie within a distance ε > 0. Similar to the pattern-recognition case, we can write this as a quadratic programming problem in terms of kernels. The nonlinear regression estimate takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} v_i \cdot k(\mathbf{x}_i,\mathbf{x}) + b \tag{11}$$

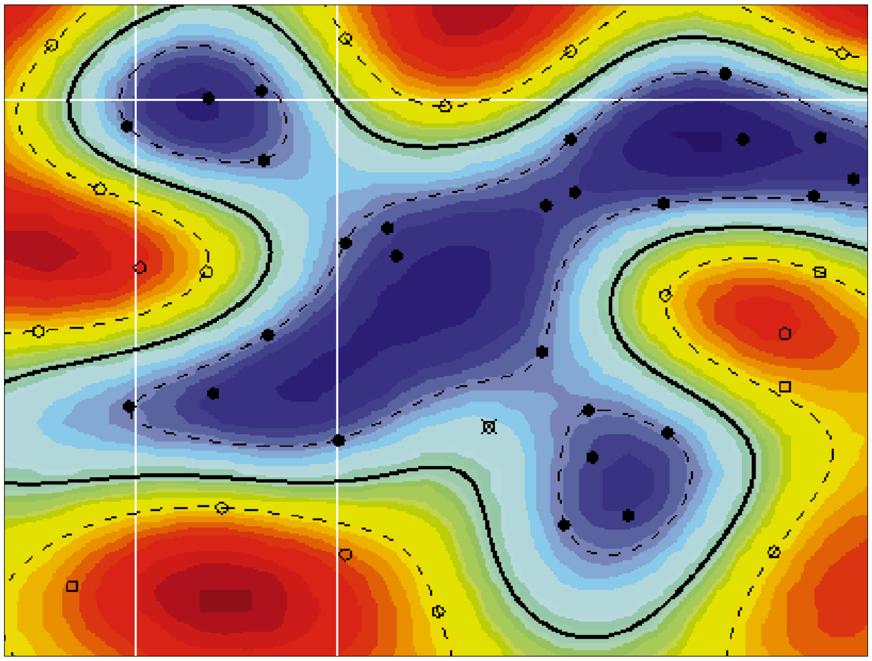
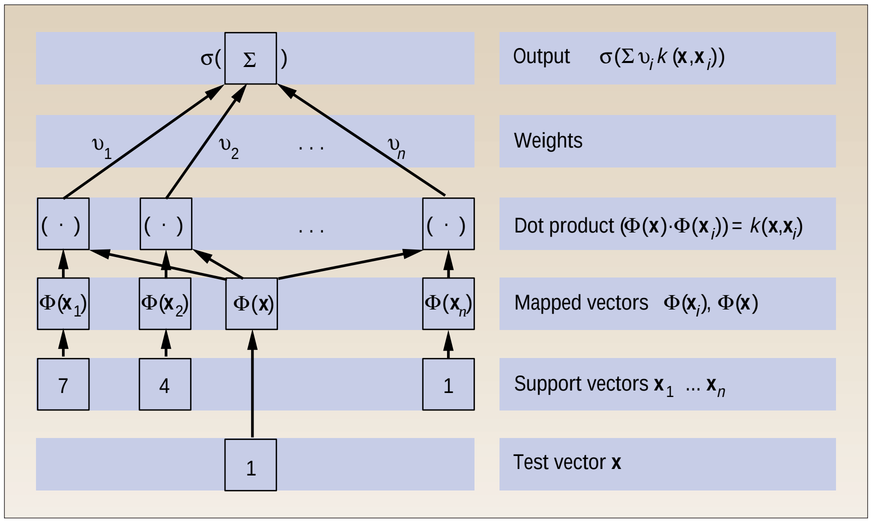


Figure 3. Example of an SV classifier found by using a radial basis function kernel (Equation 8). Circles and disks are two classes of training examples; the solid line is the decision surface; the support vectors found by the algorithm lie on, or between, the dashed lines. Colors code the modulus of the argument $\sum_i v_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$ of the decision function in Equation 10.



σ(Σ)
Output σ(Σ υ_i k (x,x_i))
υ_1
υ_2
. . .
υ_n
Weights
(·)
(·)
. . .
(·)
Dot product (Φ(x)·Φ(x_i)) = k(x,x_i)
Φ(x_1)
Φ(x_2)
Φ(x)
Φ(x_n)
Mapped vectors Φ(x_i), Φ(x)
7
4
1
Support vectors x_1 ... x_n
1
Test vector x

Figure 4. Architecture of SV methods. The input $\mathbf{x}$ and the support vectors $\mathbf{x}_i$ (in this example: digits) are nonlinearly mapped (by Φ) into a feature space F, where dot products are computed. By the use of the kernel k, these two layers are in practice computed in one single step. The results are linearly combined by weights ν_i, found by solving a quadratic program (in pattern recognition, $\nu_i = y_i \alpha_i$; in regression estimation, $\nu_i = \alpha^*_i - \alpha_i)^2$ or an eigenvalue problem (in kernel PCA[3]). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \mathrm{sign}(x + b)$; in regression stimation, $\sigma(x) = x + b$; in kernel PCA, $\sigma(x) = x$.

To apply the algorithm, we either specify ε a priori, or we specify an upper bound on the fraction of training points allowed to lie outside of a distance ε from the regression estimate (asymptotically, the number of SVs) and the corresponding ε is computed automatically.[5]

Current developments and open issues

Chances are that those readers who are still with me might be interested to hear how researchers have built on the above, applied the algorithm to real-world problems, and developed extensions. In this respect, several fields have emerged.

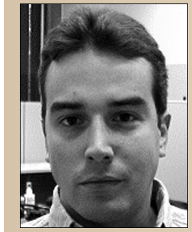
- Training methods for speeding up the quadratic program, such as the one described later in this installment of Trends & Controversies by John Platt.
- Speeding up the evaluation of the decision function is of interest in a variety of applications, such as optical-character recognition.[6]
- The choice of kernel functions, and hence of the feature space to work in, is of both theoretical and practical interest. It determines both the functional form of the estimate and, via the objective function of the quadratic program, the type of regularization that is used to constrain the estimate.[7,8] However, even though different kernels lead to different types of learning machines, the choice of kernel seems to be less crucial than it may appear at first sight. In OCR applications, the kernels (Equations 6, 9, and 8) lead to very similar performance and to strongly overlapping sets of support vectors.
- Although the use of SV methods in applications has only recently begun, application developers have already reported state-of-the-art performances in a variety of applications in pattern recognition, regression estimation, and time series prediction. However, it is probably fair to say that we are still missing an application where SV methods significantly outperform any other available algorithm or solve a problem that has so far been impossible to tackle. For the latter, SV methods for solving inverse problems are a promising candidate.[9] Sue Dumais and Edgar Osuna describe promising applications in this discussion.
- Using kernels for other algorithms emerges as an exciting opportunity for developing new learning techniques. The kernel method for computing dot products in feature spaces is not restricted to SV machines. We can use it to derive nonlinear generalizations of any algorithm that can be cast in terms of dot products. As a mere start, we decided to apply this idea to one of the most widely used algorithms for data analysis, *principal component analysis*. This leads to kernel PCA,[3] an algorithm that performs nonlinear PCA by carrying out linear PCA in feature space. The

method consists of solving a linear eigenvalue problem for a matrix whose elements are computed using the kernel function. The resulting feature extractors have the same architecture as SV machines (see Figure 4). A number of researchers have since started to "kernelize" various other linear algorithms.

References

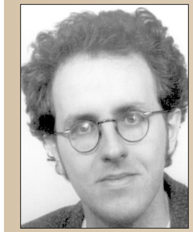
1. B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," *Proc. Fifth Ann. Workshop Computational Learning Theory*, ACM Press, New York, 1992, pp. 144–152.
2. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.
3. B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation*, Vol. 10, 1998, pp. 1299–1319.
4. B. Schölkopf, C.J.C. Burges, and A.J. Smola, *Advances in Kernel Methods—Support Vector Learning*, to appear, MIT Press, Cambridge, Mass., 1998.
5. B. Schölkopf et al., "Support Vector Regression with Automatic Accuracy Control," to be published in *Proc. Eighth Int'l Conf. Artificial Neural Networks, Perspectives in Neural Computing*, Springer-Verlag, Berlin, 1998.
6. C.J.C. Burges, "Simplified Support Vector Decision Rules," *Proc. 13th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1996, pp. 71–77.
7. A. Smola and B. Schölkopf, "From Regularization Operators to Support Vector Kernels," *Advances in Neural Information Processing Systems 10*, M. Jordan, M. Kearns, and S. Solla, eds., MIT Press, 1998.
8. F. Girosi, *An Equivalence between Sparse Approximation and Support Vector Machines*, AI Memo No. 1606, MIT, Cambridge, Mass., 1997.
9. J. Weston et al., *Density Estimation Using Support Vector Machines*, Tech. Report CSD-TR-97-23, Royal Holloway, Univ. of London, 1997.

Susan T. Dumais is a senior researcher in the Decision Theory and Adaptive Systems Group at Microsoft Research. Her research interests include algorithms and interfaces for improved information retrieval and classification, human-computer interaction, combining search and navigation, user modeling, individual differences, collaborative filtering, and organizational impacts of new technology. She received a BA in mathematics and psychology from Bates College and a PhD in cognitive psychology from Indiana University. She is a member of the ACM, the ASIS, the Human Factors and Ergonomic Society, and the Psychonomic Society. Contact her at Microsoft Research, 1 Microsoft Way, Redmond, WA 98052; sdumais@microsoft.com; http://research.microsoft.com/~sdumais.



Edgar Osuna has just returned to his native Venezuela after receiving his PhD in operations research from the Massachusetts Institute of Technology. His research interests include the study of different aspects and properties of Vapnik's SVM. He received his BS in computer engineering from the Universidad Simon Bolivar, Caracas, Venezuela. Contact him at IESA, POBA International #646, PO Box 02-5255, Miami, FL 33102-5255; eosuna@usb.ve.

John Platt is a senior researcher in the Signal Processing Group at Microsoft Research. Recently, he has concentrated on fast general-purpose machine-learning algorithms for use in processing signals. More generally, his research interests include neural networks, machine learning, computer vision, and signal processing. He received his PhD in computer science at Caltech in 1989, where he studied computer graphics and neural networks. His received his BS in chemistry from California State University at Long Beach. Contact him at Microsoft Research, 1 Microsoft Way, Redmond, WA 98005; jplatt@microsoft.com; http://research.microsoft.com/~jplatt.



Bernhard Schölkopf is a researcher at GMD First, Berlin. His research interests are in machine learning and perception, in particular using SVMs and Kernel PCA. He has an MSc in mathematics from the University of London, and a Diploma in physics and a PhD in computer science, both from the Max Planck Institute. Contact him at GMD-First, Rm. 208, Rudower Chaussee 5, D-12489 Berlin; bs@first.gmd.de; http://www.first.gmd.de/~bs.

Using SVMs for text categorization

Susan Dumais, Decision Theory and Adaptive Systems Group, Microsoft Research

As the volume of electronic information increases, there is growing interest in developing tools to help people better find, filter, and manage these resources. *Text categorization*—the assignment of natural-language texts to one or more predefined categories based on their content—is an important component in many information organization and management tasks. Machine-learning methods, including SVMs, have tremendous potential for helping people more effectively organize electronic resources.

Today, most text categorization is done by people. We all save hundreds of files, e-mail messages, and URLs in folders every day. We are often asked to choose keywords from an approved set of indexing terms for describing our technical publications. On a much larger scale, trained specialists assign new items to categories in large taxonomies such as the Dewey Decimal or Library of Congress subject headings, Medical Subject Headings (MeSH), or Yahoo!'s Internet directory. Between these two extremes, people organize objects into categories to support a wide variety of information-management tasks, including information routing/filtering/push, identification of objectionable materials or junk mail, structured search and browsing, and topic identification for topic-specific processing operations.

Human categorization is very time-consuming and costly, thus limiting its applicability—especially for large or rapidly changing collections. Consequently, interest is growing in developing technologies for (semi)automatic text categorization. Rule-based approaches similar to those employed in expert systems have been used, but they generally require manual construction of the rules, make rigid binary decisions about category membership, and are typically difficult to modify. Another strategy is to use *inductive-learning* techniques to automatically construct classifiers using labeled training data. Researchers have applied a growing number of learning techniques to text categorization, including multivariate regression, nearest-neighbor classifiers, probabilistic Bayesian models, decision trees, and neural networks.[1,2] Recently, my colleagues and I and others have used SVMs for text categorization with very promising results.[3,4] In this essay, I briefly describe the results of experiments in which we use SVMs to classify newswire stories from Reuters.[4]