A biologist's introduction to support vector machines

William Stafford Noble
Department of Genome Sciences
Department of Computer Science and Engineering
University of Washington
Seattle, WA, USA

November 1, 2006

Abstract

The support vector machine (SVM) is a pattern recognition algorithm that has been used to analyze an increasing variety of complex biological data sets, including microarray expression profiles, DNA and protein sequences, protein-protein interaction networks, tandem mass spectra, etc. This tutorial describes the algorithm in a non-technical fashion, using as an example a leukemia microarray expression data set. Four components of the SVM are described in turn: the separating hyperplane, the maximum margin hyperplane, the soft margin and the kernel function. The aim of the tutorial is to allow the non-specialist to determine whether an SVM would be appropriate for a given analysis task and to provide them with sufficient intuitions to apply existing SVM software to the task.

Introduction

The support vector machine, or SVM, is a computer algorithm that, despite its odd-sounding name, is enjoying increasing popularity for many biological applications. Pubmed includes 171 papers published within the last 12 months whose abstracts contain the phrase "support vector machine," and 475 such papers in the last five years. This tutorial aims to provide an intuitive understanding of how the SVM works and to enable a biologist to determine whether an SVM might be appropriate for a given analysis problem. In addition, I briefly describe how the SVM compares to other, similar algorithms, and I provide pointers to the technical literature and to existing software implementations.

The SVM algorithm learns by example to assign labels to objects. For instance, an SVM can learn to recognize fraudulent credit card activity by examing hundreds or thousands of fraudulent and non-fraudulent credit card activity reports. Alternatively, an SVM can learn to recognize handwritten digits by examining a large collection of scanned images of handwritten zeroes, ones, etc. For credit card companies and for the United States Postal Service, the ability to automatically assign labels to objects—credit card transaction histories or handrwritten ZIP codes—is of obvious value.

Table 1: **Selected examples of SVM applications.**

Protein homology detection	[Jaakkola et al., 1999]
Microarray gene expression classification	[Brown et al., 2000]
Splice site detection	[Degroeve et al., 2002]
Secondary structure prediction	[Hua and Sun, 2001]
Peptide identification from tandem mass spectrometry	[Anderson et al., 2003]

SVMs have been successfully applied to an increasingly wide variety of biological applications. For example, a common biomedical application of support vector machines is the automatic classification of microarray gene expression profiles. Theoretically, an SVM can examine the gene expression profile derived from a tumor sample or from peripheral fluid and arrive at a diagnosis or prognosis. Throughout this tutorial, I will use as a motivating example a seminal study of acute leukemia expression profiles [Golub et al., 1999]. Table 1 provides a small sampling of biological applications, which involve classifying objects as diverse as protein and DNA sequences, microarray expression profiles and mass spectra. At least two reviews of SVM applications in biology exist [Byvatov and Schneider, 2003, Noble, 2004], though neither is exhaustive.

In essence, an SVM is a mathematical entity, an algorithm (or recipe) for maximizing a particular mathematical function with respect to a given collection of data. However, I aim to make this article as accessible as possible to non-mathematicians. Consequently, I will avoid using any mathematical notation, and I will attempt to frame the exposition as concretely as possible. You should be able to understand the basic ideas behind the SVM algorithm without ever reading an equation.

Indeed, I claim that, in order to understand the essence of SVM classification, you only need to grasp four basic concepts: (1) the *separating hyperplane*, (2) the *maximum margin hyperplane*, (3) the *soft margin* and (4) the *kernel function*. I will explain each of these concepts in the order listed above, giving geometric interpretations for each. I have taught this topic to many undergraduate and graduate students in biology and computer science. In my experience, the fourth concept—the kernel function—is the most abstract and hence the most difficult to understand.

Before describing the SVM, though, let's return to the problem of classifying cancer gene expression profiles. The Affymetrix microarrays employed by Golub *et al.* contained probes for 6817 human genes. For a given bone marrow sample, the microarray assay returns 6817 values, each of which represents the quantitave mRNA expression level of a given gene. Golub *et al.* performed this assay on 38 bone marrow samples, from 27 individuals with acute lymphoblastic leukemia (ALL) and 11 individuals with acute myeloid leukemia (AML). The subsequent SVM learning task, depicted in Figure 1, is to learn to tell the difference between ALL and AML expression profiles. If the learning is successful, then the SVM will be able to successfully diagnose a new patient as AML or ALL based upon their bone marrow expression profile.

In order to allow an easy, geometric interpretation of the data, I am going to drastically simplify this problem by pretending, temporarily, that the microarray contained probes for only two genes. Hence, our gene expression profiles now consist of two numbers, which can be easily plotted in a two-dimensional Cartesian grid, as shown in Figure 2. Based upon

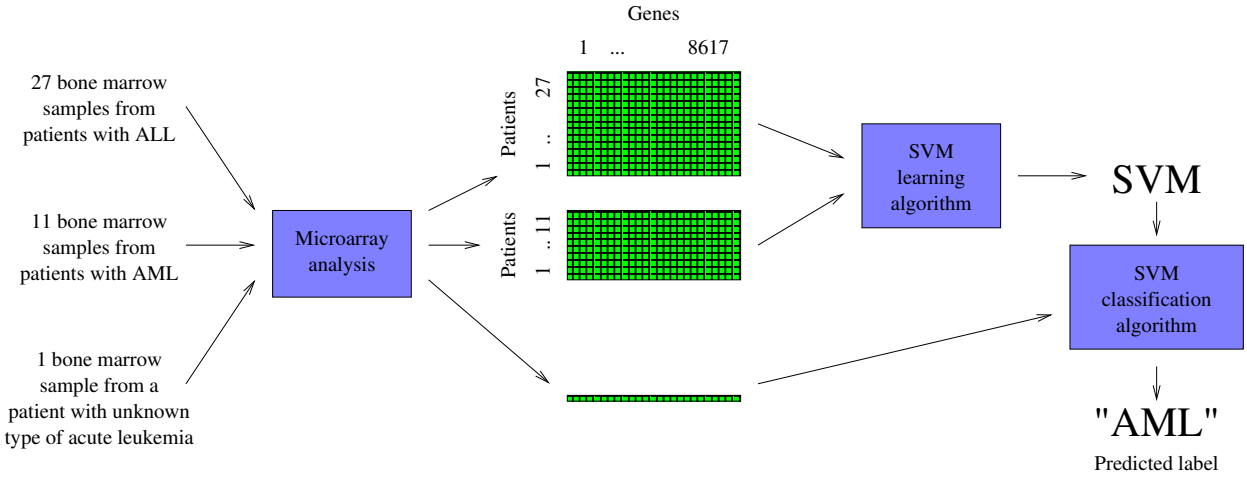


Figure 1: **Learning to discriminate between acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML) gene expression profiles.** The SVM learning algorithm produces an SVM classifier that can be used subsequently to predict whether a given gene expression profile is derived from an ALL or AML bone marrow sample.

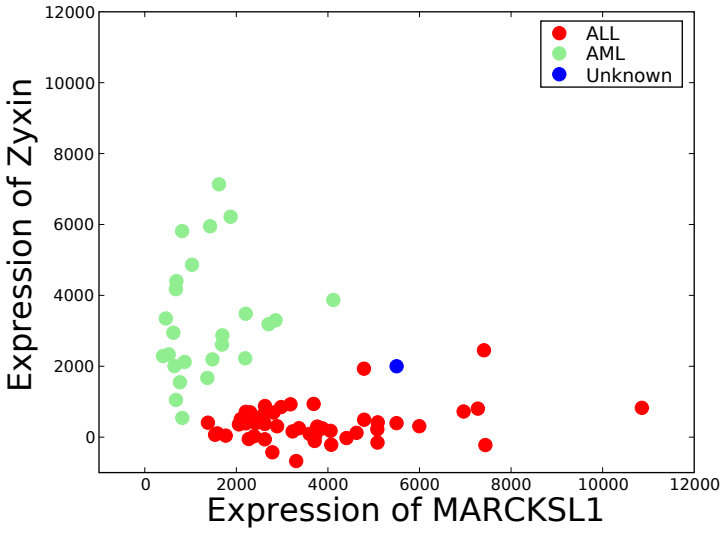


Figure 2: **Two-dimensional ALL and AML expression profiles.** Each dimension corresponds to the measured mRNA expression level of a given gene. The SVM's task is to assign a label to the gene expression profile labeled "Unknown."

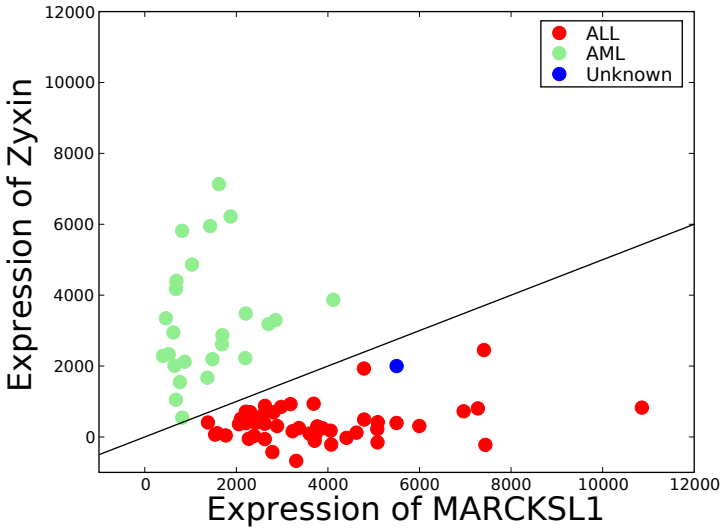


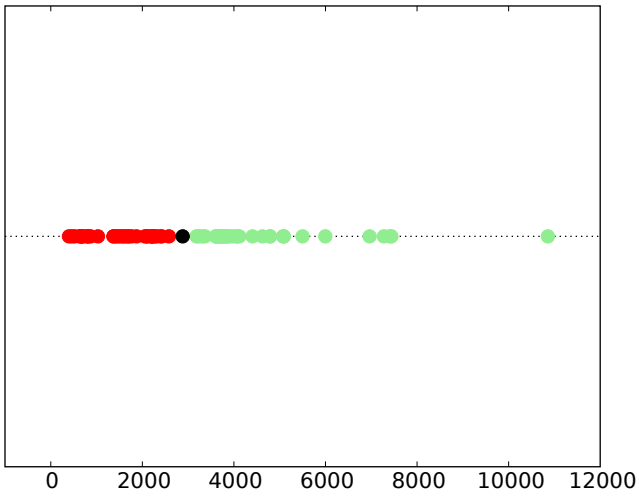
Figure 3: **A separating hyperplane.** Based upon this hyperplane, the inferred labeled of the "Unknown" expression profile is "ALL."

results from a previous study [Guyon et al., 2002], I have selected the genes Zyxin and MARCKSL1. Zyxin encodes an adhesion plaque protein that includes three zinc-binding LIM domains. The Zyxin protein is localized at focal contacts in adherent erythroleukemia cells [Macalma et al., 1996]. MARCKS gene transcription is stimulated by tumor necrosis factor-alpha proteins in human promyelocytic leukemia cells [Harlan et al., 1991]. In the figure, values are proportional to the intensity of the fluorescence on the microarray, so on either axis, a large value indicates that the gene is highly expressed and vice versa. In mathematical terms, I have simplified the SVM's task from classifying 6817-dimensional vectors to classifying two-dimensional vectors. In a two-dimensional plot, each dot represents a two-dimensional vector. Thus, in Figure 2, each expression profile is indicated by a red or green dot, depending upon whether the sample is from a patient with ALL or AML. The SVM must learn to tell the difference between the two groups and, given an unlabeled expression vector such as the one labeled "Unknown" in the figure, predict whether it corresponds to a patient with ALL or AML.

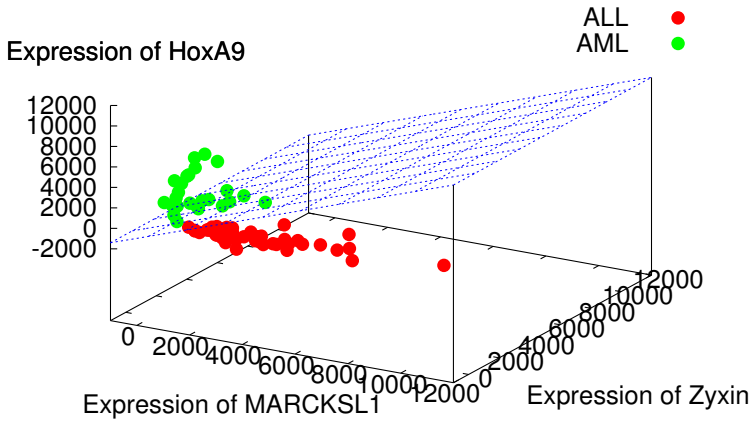
Concept 1: Separating hyperplane

The human eye is very good at pattern recognition. Even a quick glance at Figure 2 shows that the AML profiles form a cluster in the upper left region of the plot, and the ALL profiles cluster in the lower right. A simple rule might state that a patient has AML if the expression level of MARCKSL1 is twice as high as the expression level of Zyxin, and vice versa for ALL. Geometrically, this rule corresponds to drawing a line between the two clusters, as shown in Figure 3. Subsequently, predicting the label of an unknown expression profile is easy: we simply ask whether the new profile falls on the ALL or the AML side of this separating line.

Now, to define the notion of separating hyperplane, consider a situation in which the



(A)



(B)

Figure 4: **Hyperplanes in one and in three dimensions.** In panel (A), the hyperplane is shown as a single black point. In panel (B), the hyperplane is a blue plane.

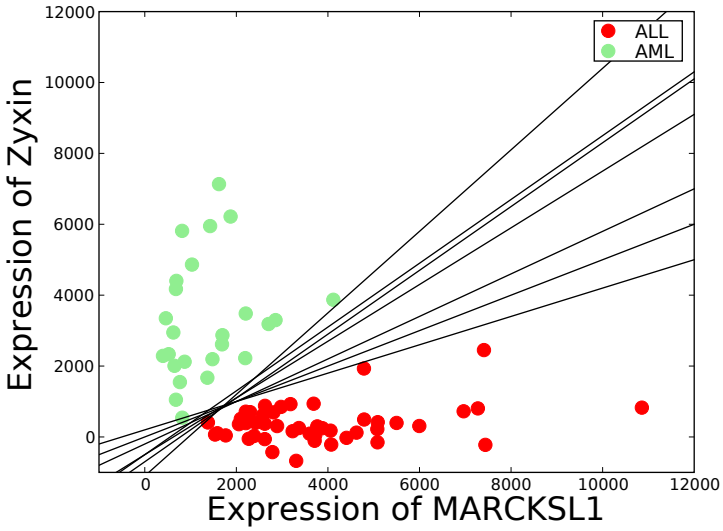


Figure 5: **Many possible separating hyperplanes**

microarray does not contain just two genes. For example, if the microarray contains a single gene, then the "space" in which the corresponding one-dimensional expression profiles reside is a one-dimensional line. We can divide this line in half by using a single point (see Figure 4A). In two dimensions, as shown in Figure 3, a straight line divides the space in half, and in three dimensions, we need a plane to divide the space (Figure 4B). What happens when we move to more than three dimensions? Even though a four-dimensional space is difficult to conceptualize, we can still characterize that space mathematically. For example, we can refer to points in this space by using four-dimensional vectors of expression log ratios, and we can define the separating boundary between ALL and AML profiles in that space. If we define a straight boundary, then that boundary is analagous to the point in one dimension, the line in two dimensions and the plane in three dimensions. The general term for a straight line in a high-dimensional space is a *hyperplane*, and so the separating hyperplane is just, essentially, the line that separates the ALL and AML samples.

Now you can see why I chose to simplify the problem to two dimensions. It is extremely difficult to imagine points in a 6817-dimensional space, such as the expression profiles produced by the Golub *et al.* microarrays. Hopefully, the separating line in Figure 3 is intuitive, and you will trust me when I say that we can find a similar type of separator in the 6817-dimensional gene expression space.

This idea—of treating the objects to be classified as points in a high-dimensional space and finding a line that separates them—is not unique to the SVM. The SVM is distinguished from other hyperplane-based classifiers by the particular hyperplane that it selects. This is the topic of the next section.

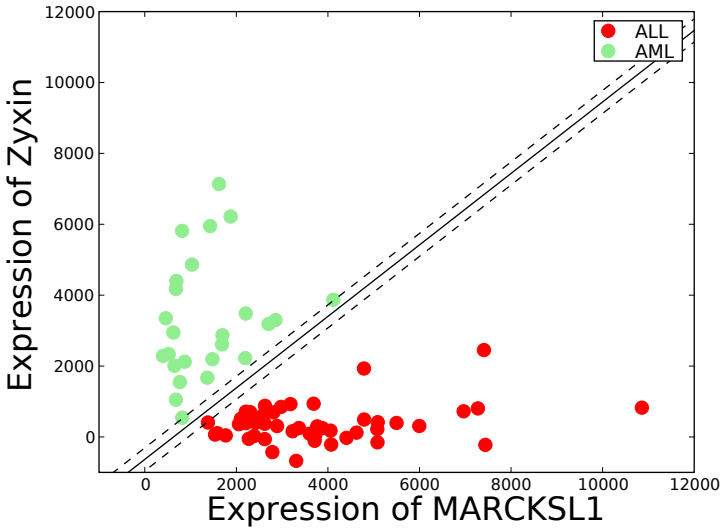


Figure 6: **The maximum margin hyperplane**

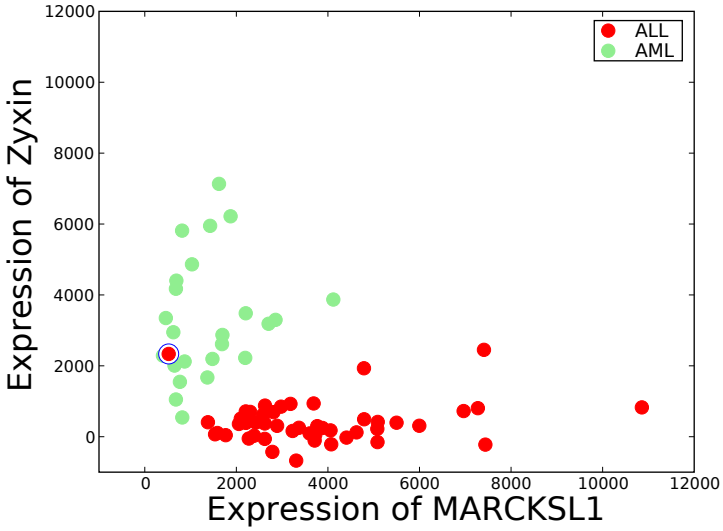
Concept 2: Maximum margin hyperplane

Consider again the classification problem portrayed in Figure 2. We have now established that the goal of the SVM is to identify a line that separates the ALL from the AML expression profiles in this two-dimensional space. However, as shown in Figure 5, many such lines exist. Which one should we choose?

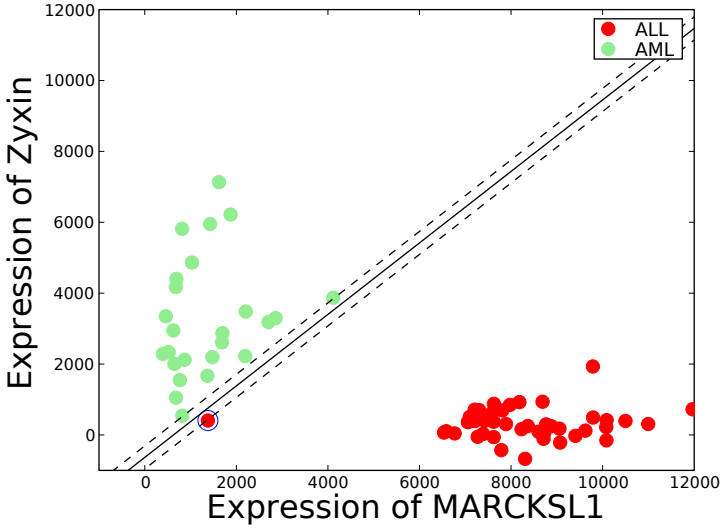
With some thought, and if pressed, you might come up with the simple idea of selecting the line that is, more or less, in the middle. In other words, you could imagine selecting the line that separates the two classes but is maximally far away from any of the given expression profiles. This line is shown in Figure 6.

It turns out that a theorem from the field of statistical learning theory supports exactly this choice [Vapnik and Lerner, 1963, Vapnik, 1998]. If we define the distance from the separating hyperplane to the nearest expression vector as the *margin* of the hyperplane, then the SVM selects the *maximum margin separating hyperplane*. Selecting this particular hyperplane maximizes the SVM's ability to predict the correct classification of previously unseen examples.

This theorem is, in many ways, the key to the SVM's success. Let's take a minute, therefore, to consider some caveats that come with it. First, the theorem assumes that the data on which the SVM is trained are drawn from the same distribution as the data on which it is tested. This is reasonable, since we cannot expect, e.g, an SVM trained on microarray data to be able to classify mass spectometry data. More relevantly, we cannot expect the SVM to perform well if the bone marrow samples for the training data set were prepared using a different protocol than the samples for the test data set. On the other hand, the theorem does not assume that the two data sets were drawn from a particular class of distributions. In particular, the SVM does not assume, e.g., that the training data values are normally distributed.



(A)



(B)

Figure 7: **Two data sets with errors.** In each panel, the circled point corresponds to a gene expression profile that is either incorrectly measured or incorrectly labeled.

Concept 3: Soft margin

So far, I have assumed that the data can be separated using a straight line. Of course, many real data sets are not separable; instead, they look like the one in Figure 7A. In order to handle cases like this, we need to modify the SVM algorithm by adding a *soft margin*. Given the definition of "margin," and given the inseparability problem, you can probably imagine what a soft margin is. But to motivate the issue a bit further, consider the two panels in Figure 7. In panel A, as already mentioned, there is no separating hyperplane. In panel B, we can define the maximum margin hyperplane, but it is somewhat unsatisfactory, because it is so close to the AML examples. By eye, it seems likely that the two gene expression profiles that are circled in panels A and B are mislabeled. It looks like either the microarray measurement is incorrect or somebody misdiagnosed a patient or mislabeled a bone marrow sample. Intuitively, we would like the SVM to be able to allow for this type of error in the data by allowing a few anomalous expression profiles to fall on the wrong side of the separating hyperplane.

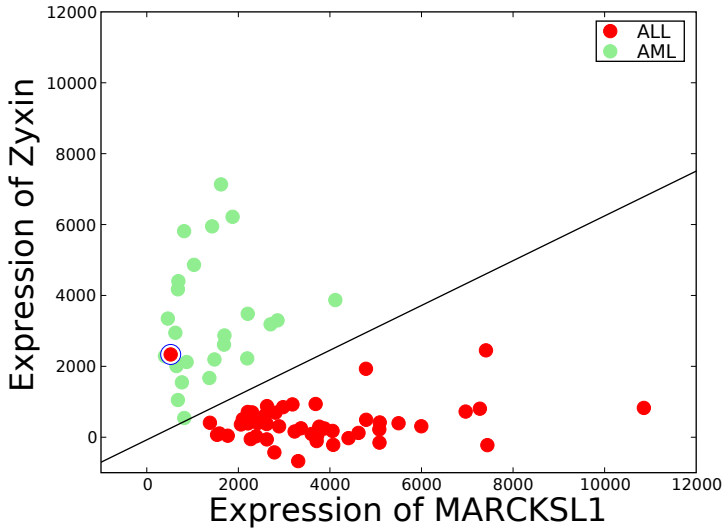
The soft margin allows this to happen. The soft margin is "soft" in the sense that some data points can push their way through it. Figure 8 shows soft margin solutions to the two problems in Figure 7. In both cases, the one outlier example now resides on the same side of the line with members of the opposite class.

Of course, we don't want the SVM to allow too many misclassifications. Hence, introducing the soft margin necessitates introducing a user-specified parameter that controls, roughly, how many examples are allowed to violate the separating hyperplane and how far across the line they are allowed to go. Setting this parameter is complicated by the fact that, as in Figure 8A, we still want to try to achieve a large margin with respect to the correctly classified examples. Hence, the soft margin parameter specifies a trade-off between hyperplane violations and the size of the margin.

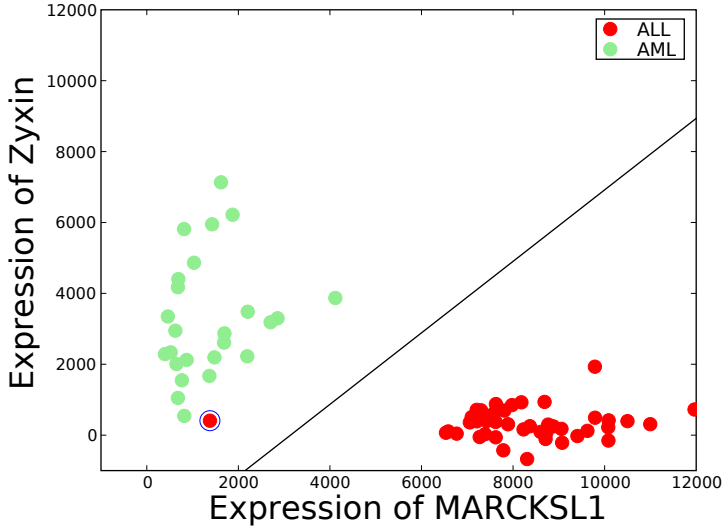
With the machinery that I have described thus far, it is possible to achieve state-of-the-art classification performance in many real application domains. The remaining concept—the *kernel function*—is not always necessary and is considerably more abstract than the first three concepts. The kernel function's primary benefit is to allow the SVM to find a non-linear separating boundary between two classes. In addition, the kernel function expands the SVM's ability to incorporate prior knowledge and to handle non-numeric and heterogeneous data sets.

Concept 4: The kernel function

To explain the kernel function, I am going to simplify my example even further. Rather than a microarray containing two genes, let's assume that we now have only a single gene expression measurement, as shown in Figure 9A. In this case, the maximum margin separating "hyperplane" is a single point, at position 375 on the line, halfway between the lowest AML value and the highest ALL value. Figure 9B shows an analogous, but non-separable example. Here, the AML values are grouped near zero, and the ALL examples have large absolute values. The problem is that no single point can separate the two classes, and introducing a soft margin does not help.

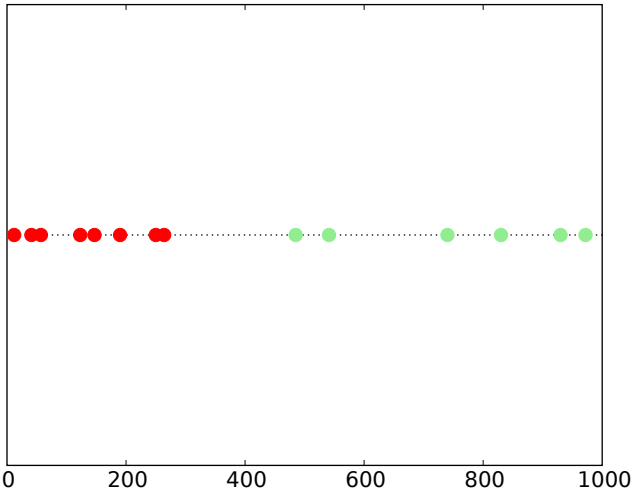


(A)

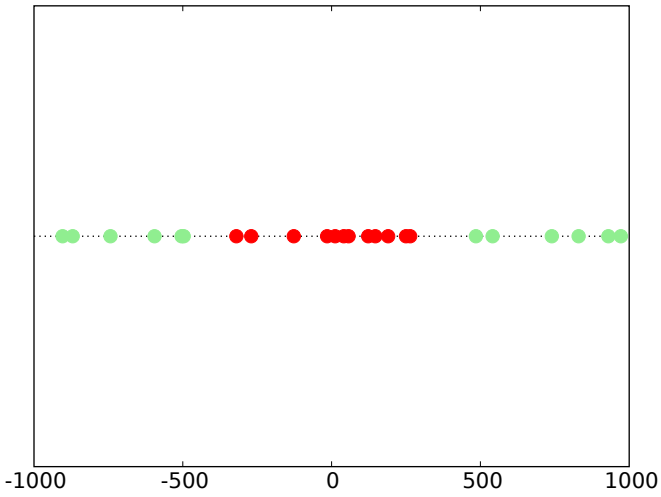


(B)

Figure 8: **A separating hyperplane with a soft margin**



(A)



(B)

Figure 9: **Two one-dimensional data sets.** The data in (A) is separable; the data in (B) is not.

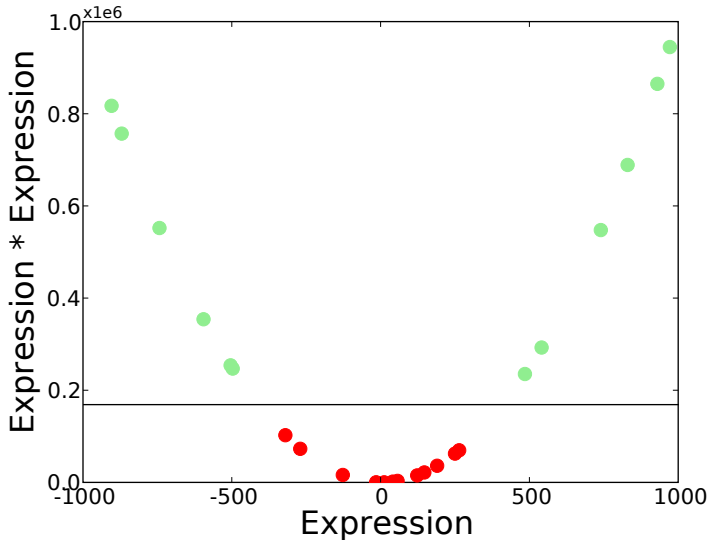


Figure 10: **Separating previously non-separable data.**

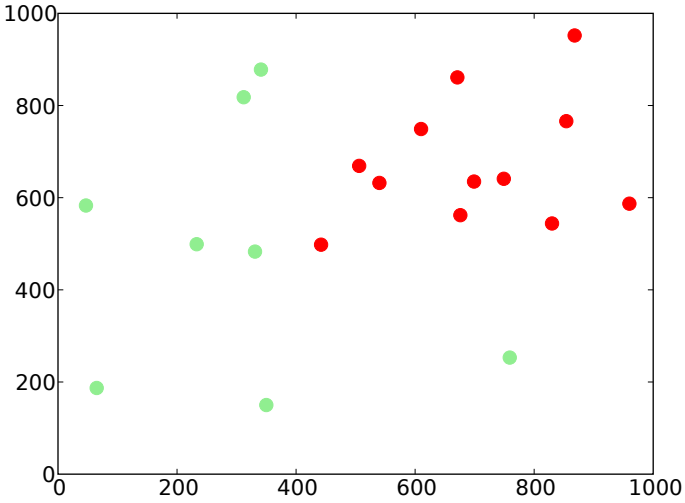
The kernel function provides a solution to this problem, as shown in Figure 10. The figure shows the same data, but with an additional dimension. To get the new dimension, we simply square the original expression values. For example, in this simulated data set, one AML patient had an expression level of 123 for the selected gene. The corresponding two-dimensional vector is (123, 15129), because $123 * 123 = 15129$. Fortuitously, as shown in the figure, we can separate the ALL and AML examples with a straight line in the two-dimensional space, even though the two groups were not separable in the one-dimensional space.

The particular mapping from Figure 9B to Figure 10 is one example of the type of flexibility afforded by the use of a kernel function. In essence, the kernel function is a mathematical trick that allows the SVM to perform classification in the two-dimensional space even when the data is one-dimensional. In general, we say that the kernel function *projects* the data from a low-dimensional space to a space of higher dimension. If we are lucky (or smart) and we choose a good kernel function, then the data will be separable in the resulting higher dimensional space, even if it wasn't separable in the lower dimensional space.

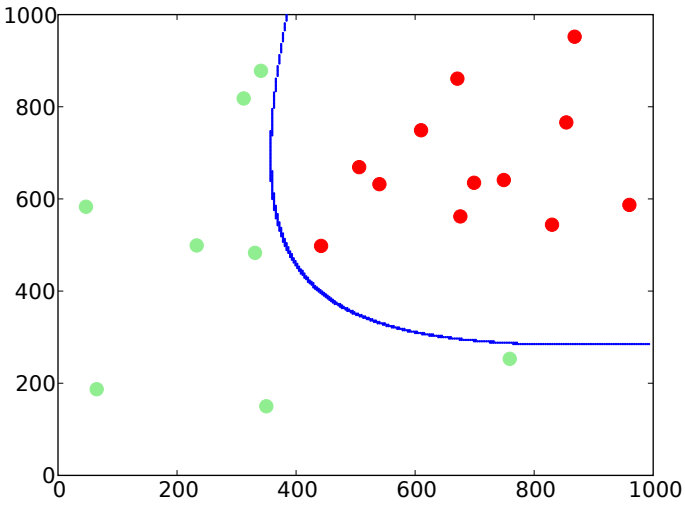
To understand kernels a bit better, now consider the two-dimensional data set shown in Figure 11A. This data cannot be separated using a straight line; however, it turns out that a relatively simple kernel function that projects from two dimensions up to four dimensions will allow the data to be linearly separated. I cannot draw the data in a four-dimensional space, but I can project the SVM hyperplane in that space back down to the original two-dimensional space. The result is shown as a curved line in Figure 11B.

It is possible to prove that, for any given data set with consistent labels (where *consistent* simply means that the data set does not contain two identical objects with opposite labels) there exists a kernel function that will allow the data to be linearly separated.

This observation begs the question, why not always project into a very high-dimensional



(A)



(B)

Figure 11: **A linearly non-separable two-dimensional data set, which is linearly separable in four dimensions.**

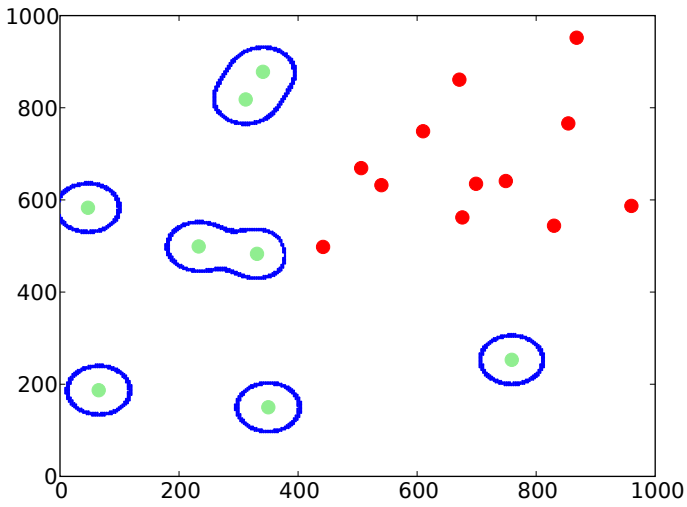


Figure 12: **An SVM that has overfit a two-dimensional data set.**

space, in order to be sure of finding a separating hyperplane? If we did that, then it might seem that we would not need the soft margin, and the original theorem, mentioned above, would still apply. This is a reasonable suggestion, and in fact, the first description of the SVM algorithm did not use the soft margin formulation at all [Boser et al., 1992].

However, projecting into very high-dimensional spaces can be problematic, due to the so-called curse of dimensionality [Bellman, 1961]. The curse is, essentially, that as you increase the number of variables under consideration, you generate an exponentially larger number of possible solutions. Consequently, it becomes harder for any algorithm to select the correct solution from this large set. The SVM is remarkably good at combating the curse of dimensionality. For example, the algorithm can handle classification problems involving relatively few gene expression profiles, each of which contains many, many genes. However, although the curse of dimensionality can be reduced, it can never be fully eliminated. If we take a 6817-dimensional vector and use the same kernel that I used in Figure 11B, then we project our data into a 46 million-dimensional space. Many of the dimensions in this space are irrelevant, corresponding to pairs of genes whose expression bears no relation to the ALL/AML distinction. Any learning algorithm, including the SVM, is unlikely to be able to operate well in such a high-dimensional space.

Figure 12 shows what happens when we project into a space with too many dimensions. The figure contains the same data as Figure 11, but the projected hyperplane comes from an SVM that uses a very high-dimensional kernel function. The result is that the boundary between the classes is very specific to the examples in the training data set. In this case, the SVM is said to *overfit* the data. Clearly, this SVM will not generalize well when presented with new gene expression profiles.

This observation brings us to the largest practical difficulty in applying an SVM classifier to a new data set. We would like to use a kernel function that is likely to allow our data to be separated but that does not introduce too many irrelevant dimensions. How do we

choose this function? Unfortunately, in most cases, the only realistic answer is trial and error. In some cases, the choice of kernel function is obvious. For the Golub *et al.* data, for example, using a kernel function at all is probably not a good idea, because we only have 38 examples, and we already have 6817 gene expression values per example. In this situation, we are more likely to want to reduce the number of dimensions by eliminating some genes from consideration.[1] In a more typical setting, where the number of dimensions is smaller than the number of training set examples, investigators typically begin with a simple SVM, and then experiment with a variety of "standard" kernel functions. An optimal kernel function can be selected from a fixed set of kernels in a statistically rigorous fashion by using a technique known as cross-validation [Hastie et al., 2001]. However, this approach is time-consuming and cannot guarantee that some kernel function that we did not consider would not perform better.

In addition to allowing SVMs to handle non-linearly separable data sets and to incorporate prior knowledge, the kernel function yields at least two additional benefits. First, kernels can be defined on inputs that are not vectors. Gene expression data is a convenient type of data for the SVM, because each expression profile is a vector; i.e., each profile contains the same number of real-valued entries. It is less clear, for example, how to classify protein sequences, which are variable-length and are not even numeric. Conveniently, it turns out that we can define a variety of kernels that operate on protein sequences [Jaakkola et al., 1999, Liao and Noble, 2002, Kuang et al., 2005]. These kernel functions implicitly map the proteins into a high-dimensional space. The mapping is implicit in the sense that we never actually compute the vector representations. Instead, the SVM algorithm works only with protein sequences, applying the kernel function directly to those sequences. This ability to handle non-vector data is critical in biological applications, allowing the SVM to classify DNA and protein sequences, nodes in metabolic, regulatory and protein-protein interaction networks, microscopy images, etc.

The final benefit of the kernel function is that kernels from different types of data can be combined. Imagine, for example, that we are doing biomarker discovery for the ALL/AML distinction, and we have the Golub *et al.* data plus a corresponding collection of mass spectrometry profiles from the same set of patients. It turns out that we can use simple algebra to combine a kernel on microarray data with a kernel on mass spectrometry data. The resulting joint kernel would allow us to train a single SVM to perform classification on both types of data simultaneously. This type of approach has been used successfully to predict gene function [Pavlidis et al., 2001, Lanckriet et al., 2004] and to predict protein-protien interactions [Ben-Hur and Noble, 2005] from a variety of genome-wide data sets in yeast.

Extensions of the SVM algorithm

The most obvious drawback to the SVM algorithm, as described thus far, is that it apparently only handles binary classification problems. We can discriminate between ALL and AML, but how do we discriminate among a large variety of cancer classes? Generalizing to multiclass classification is straightforward and can be accomplished by using any of a variety

[1] This approach is called *feature selection* and is independent of SVM classification.

of methods. Perhaps the simplest approach is to train multiple, one-versus-all classifiers. Essentially, to recognize three classes, A, B and C, you train three separate SVMs to answer the binary questions, "Is it A?" "Is it B?" and "Is it C?" The predicted class is then the example with the strongest "yes" response (or, in some cases, the weakest "no"). This simple approach actually works quite well for cancer classification [Ramaswamy et al., 2001]. More sophisticated approaches also exist, which generalize the SVM optimization algorithm to account for multiple classes [Lee et al., 2001, Weston and Watkins, 1998, Aiolli and Sperduti, 2005, Crammer and Singer, 2001].

The allusion, in the previous paragraph, to a strong "yes" and a weak "no," raises another important issue. A useful classification algorithm should return, for each example that it receives as input, not only a predicted label but also some estimate of the classifier's confidence in its prediction. In the SVM framework, this confidence can be quantified by the distance from the example to the separating hyperplane. Unfortunately, distances in this space have no units associated with them. Platt [1999] suggests a simple method for mapping the distance to the separating hyperplane onto a probability, based upon an empirical curve fitting procedure. This method works fairly well in practice.

Scaling up to large data sets

For data sets of thousands of examples, solving the SVM optimization problem is quite fast. Empirically, running times of state-of-the-art SVM learning algorithms scale approximately quadratically, which means that when you give the SVM twice as much data, it requires four times as long to run. This is obviously not as good as scaling linearly, in which the running time doubles when the data set size doubles, but quadratic running time is competitive with most similar algorithms. SVMs have been successfully trained on data sets containing approximately one million examples, and fast approximation algorithms exist that scale almost linearly and perform nearly as well as the SVM [Bordes et al., 2005].

Comparison to other classification methods

The SVM algorithm is one member in a very large class of methods known as supervised classification algorithms. It is "supervised" in the sense that the SVM requires, during an initial training phase, a collection of objects with known labels, such as gene expression profiles labeled "ALL" and "AML." Only after training can the SVM predict the labels of subsequent, unlabeled objects. This property makes the SVM distinct from clustering methods, which are inherently unsupervised. Examples of clustering methods include hierarchical clustering, the k-means algorithm, self-organizing maps, spectral clustering, etc. A clustering algorithm attempts to identify previously undiscerned clusters in a given data set. A supervised classification algorithm, by contrast, learns to identify members of a given set of clusters.

Among supervised classification methods, SVMs are quite similar to artificial neural networks [Haykin, 1994]. Both methods project a given data set into a high-dimensional space and find a separating hyperplane there. In a neural network, the role of the kernel function

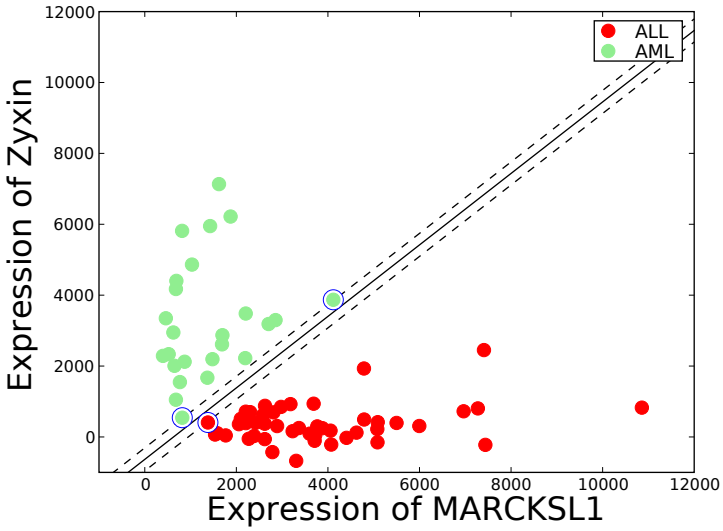


Figure 13: **Support vectors.** The SVM solution assigns weights to each example in the data set. Only those examples that lie near the separating hyperplane receive non-zero weights. These examples are called "support vectors." In the figure, the three support vectors are circled.

is played by the network topology. One drawback to the neural network approach is that it generally does not involve maximizing the margin (although maximum margin formulations now exist). Another drawback is that the backpropagation algorithm for training neural networks only finds a local maximum. As such, the results of the training vary from run to run, depending upon a random initialization of the model parameters. The SVM learning algorithm, by contrast, solves a convex optimization problem, which means that it is guaranteed always to converge to a unique solution.

The primary principle that guided the development of the SVM algorithm is known as Occam's Razor, which states, roughly, that between two hypotheses of equal explanatory power, one should select the simpler of the two. A corollary is, "Do not solve a problem that is harder than the one before you." A supervised classification task involves predicting, for each given test example, a single label. The SVM solves precisely this problem, without attempting, for example, to model the complete distribution from which the example is derived.

This minimalist approach contrasts with, for example, Fisher's linear discriminant (FLD) or logistic regression [Duda and Hart, 1973]. In these methods, members of the two given classes are assumed to come from separate, normal distributions. Each method uses a different strategy to find a hyperplane that optimally separates the two distributions.

In most applications, the SVM performs better than a method such as FLD or logistic regression because the SVM focuses only on the examples that lie near the separating hyperplane. These are, arguably, the examples that matter most to the classification task. Indeed, the SVM solution amounts to assigning a real-valued weight to each training example, and examples that are far from the separating hyperplane receive weights of zero. In Figure 13 the examples that receive non-zero weights are circled. These examples are called *support*

vectors because they support the separating hyperplane. This is the source of the SVM's name.

Interpreting the SVM's output

A consequence of the Occam's Razor approach to classification is that the SVM solves only the task at hand. That is, SVMs are very good at predicting the labels of previously unseen examples that are drawn from the same underlying distribution as the training data. Conversely, SVMs are not very good at providing an explanation for these predictions. As we have seen above, it is possible to extract from the SVM a confidence metric in the form of a probability, but even this extraction is fairly *ad hoc*. If you want a more detailed explanation for the prediction, then you will be hard-pressed to get it out of the SVM, especially if you are using a kernel function that maps the data into an implicit, high-dimensional space.

This inability of the SVM to provide explanations, though frustrating, is profoundly important. As the comparison with FLD and logistic regression suggests, the SVM's power derives in part from its ability to focus only on the portion of the data that is most relevant. In effect, the SVM does not waste any effort attempting to construct a complete picture of the distribution from which the data was drawn. There is an intrinsic trade-off here, between getting the best possible predictions and getting predictions that you can explain. In some settings, prediction accuracy may be paramount, in which case the SVM is a good choice; in other settings, the Occam's Razor approach may be inappropriate.

Further reading and software

The authoritative source for the theory behind SVMs are the books of Vladimir Vapnik [Vapnik, 1995, 1998]. Readers of this tutorial, however, will probably not want to jump directly into those works. Instead, I recommend starting with the introductory chapter in [Schoelkopf et al., 2004]. Alternatively, the book, "Introduction to Support Vector Machines" [Cristianini and Shawe-Taylor, 2000] is reasonably accessible, and is quite comprehensive.

The internet is awash in freely available SVM implementations of varying quality. A good listing is available at `www.kernel-machines.org/software.html`. A nice place to start is a simple JAVA applet at AT&T that allows you to interactively place data points in a two-dimensional plane and find hyperplanes using various kernels (`svm.dcs.rhbnc.ac.uk`). For small classification tasks, my research group has produced a simple web interface (`svm.sdsc.edu`) that will train an SVM on tab-delimited data that you provide.

For real SVM experimentation, the two most commonly used packages are SVMLight (`svmlight.joachims.org`) and LIBSVM (`www.csie.ntu.edu.tw/~cjlin/libsvm`). Some commonly used machine learning toolkits that include SVMs are PyML (`pyml.sourceforge.net`) and Spider (`www.kyb.tuebingen.mpg.de/bs/people/spider`).

Conclusion

The SVM is a pattern recognition algorithm that learns by example to distinguish among various classes of objects. In order to apply the SVM, members of each class to be identified must be available for training. For a given pair of classes, the SVM treats the data as points in a high-dimensional space and attempts to find a separating hyperplane there. The particular hyperplane that it selects is motivated by considerations from statistical learning theory, and makes a trade-off between the desire to find a good separation between the classes and the desire to allow for some noise in the data or the class labels. Using a kernel function gives the SVM additional flexibility to find a good separator, represent heterogeneous types of data, and incorporate prior knowledge. Subsequently, the SVM can predict the class of an unlabeled example by asking which side of the learned hyperplane it lies on.

Using all 6817 gene expression measurements, an SVM can achieve near-perfect classification accuracy on the ALL/AML data set [Furey et al., 2001]. Furthermore, in subsequent work, Ramaswamy et al. [2001] used a much larger data set to demonstrate that SVMs perform better than a variety of competing methods for cancer classification from microarray expression profiles. SVM-related methods have also been used with the Golub *et al.* data set to identify genes related to the ALL/AML distinction [Guyon et al., 2002].

Although this tutorial has focused on cancer classification from gene expression profiles, SVM analysis can be applied to a wide variety of biological data. As we have seen, the SVM boasts a strong theoretical underpinning, coupled with impressive empirical results across a growing spectrum of applications. Thus, SVMs will likely continue to yield valuable insights into the growing quantity and variety of molecular biology data.

Acknowledgments Thanks to Celeste Berg, Martial Hue, Gert Lanckriet, Sheila Reynolds and Jason Weston for comments on the manuscript. This work was supported by award IIS-0093302 from the National Science Foundation and award R33 HG003070 from the National Institutes of Health.

References

F. Aiolli and A. Sperduti. Multiclass classification with multi-prototype support vector machines. *Journal of Machine Learning Research*, 6:817–850, 2005.

D. C. Anderson, W. Li, D. G. Payan, and W. S. Noble. A new algorithm for the evaluation of shotgun peptide sequencing in proteomics: support vector machine classification of peptide MS/MS spectra and SEQUEST scores. *Journal of Proteome Research*, 2(2):137–146, 2003.

R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton UP, 1961.

A. Ben-Hur and W. S. Noble. Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21 suppl 1:i38–i46, 2005.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, Jr., and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences of the United States of America*, 97(1):262–267, 2000.

E. Byvatov and G. Schneider. Support vector machine applications in bioinformatics. *Applied Bioinformatics*, 2(2):67–77, 2003.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge UP, Cambridge, UK, 2000.

S. Degroeve, B. De Baets, Y. Van de Peer, and P. Rouz. Feature subset selection for splice site prediction. *Bioinformatics*, 18:S75–S83, 2002.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2001.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

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

D. M. Harlan, J. M. Graff, D. J. Stumpo, R. L. Eddy, Jr., T. B. Shows, J. M. Boyle, and P. J. Blackshear. The human myristoylated alanine-rich C kinase substrate (MARCKS) gene (macs). analysis of its gene product, promoter, and chromosomal location. *Journal of Biological Chemistry*, 266(22):14399–14405, 1991.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: Data mining, inference and prediction*. Springer, New York, NY, 2001.

S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

S. Hua and Z. Sun. A novel method of protein secondary structure prediction with high segment overlap measure: Support vector machine approach. *Journal of Molecular Biology*, 208(2):397–407, 2001.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, Menlo Park, CA, 1999. AAAI Press.

R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of Bioinformatics and Computational Biology*, 3(3):527–550, 2005.

G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004.

Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. Technical Report TR 1043, University of Wisconsin, Madison, September 2001.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the Sixth Annual International Conference on Computational Molecular Biology*, pages 225–232, Washington, DC, April 18–21 2002.

T. Macalma, J. Otte, M. E. Hensler, S. M. Bockholt, H. A. Louis, M. Kalff-Suske, K. H. Grzeschik, D. von der Ahe, and M. C. Beckerle. Molecular characterization of human zyxin. *Journal of Biological Chemistry*, 271(49):31470–31478, 1996.

W. S. Noble. Support vector machine applications in computational biology. In B. Schoelkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel methods in computational biology*, pages 71–92. MIT Press, Cambridge, MA, 2004.

P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*, pages 242–248, 2001.

J. C. Platt. Probabilities for support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Sciences of the United States of America*, 98(26): 15149–54, 2001.

B. Schoelkopf, K. Tsuda, and J.-P. Vert, editors. *Kernel methods in computational biology*. MIT Press, Cambridge, MA, 2004.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

V. N. Vapnik. *Statistical Learning Theory.* Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.

J. Weston and C. Watkins. Multi-class support vector machines. *Royal Holloway Technical Report CSD-TR-98-04*, 1998.