STA 414/2104

Statistical Methods for Machine Learning and Data Mining

Radford M. Neal, University of Toronto, 2014

Week 11

Large Margin Classifiers

The Large Margin Hard Classifier

Some classification methods produce *only* a predicted class for a test case, with no probability distribution for that class.

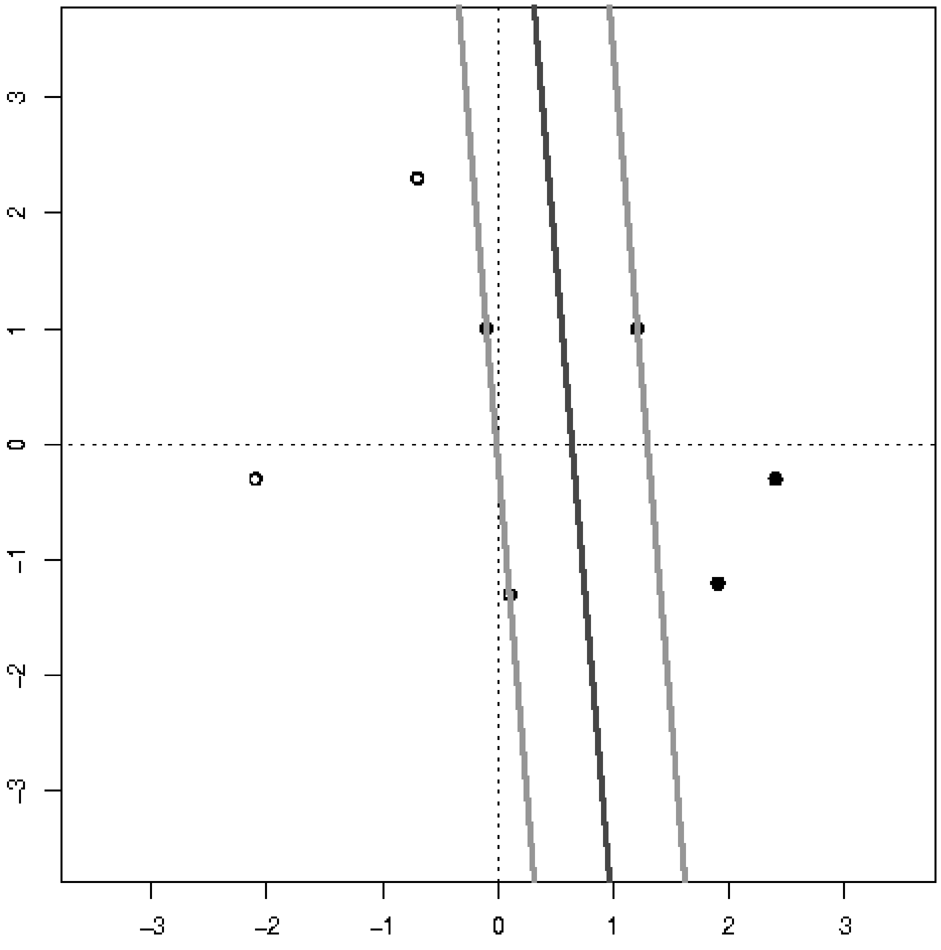
Large margin classifiers are in this category. They are the basis for the popular *Support Vector Machine (SVM)* classifiers.

In their simplest form, large margin classifiers apply only to perfectly separable binary classification problems, in which there is a hyperplane that separates the training cases in one class from the training cases in the other class.

There will usually be many hyperplanes separating the classes. The idea is to pick the separating hyperplane that has the largest *margin* — the minimum distance of a training case from the line.

An Illustration with Two Inputs

Here is a large margin classifier in two dimensions (where a hyperplane is a straight line). There are four training cases in Class -1 (white) on the left, and three in Class $+1$ (black) on the right. The dark line is the separating hyperplane, used to predict the class of a test case. The lighter lines show the margin.



Finding the Separating Hyperplane with Largest Margin

We can define a hyperplane by the equation $w^T x + b = 0$. We can use w and b to classify test cases to the class $\text{sign}(w^T x + b)$. (We'll use -1 and $+1$ as class labels.)

Note that negating both w and b will swap which side of the hyperplane has which class, and multiplying both w and b by any positive constant won't change classifications.

When finding w and b from the training cases, we will impose the constraint that all training cases are classified correctly — that is,

$$y_i(w^T x_i + b) \; > \; 0, \qquad \text{for } i = 1, \ldots, n$$

But we want to also maximize the margin, which is

$$\min_{i=1,\ldots,n} \; y_i(w^T x_i + b) \, / \, ||w||$$

This is equivalent to the following optimization problem:

$$\text{minimize } ||w||^2, \quad \text{subject to } y_i(w^T x_i + b) \; \geq \; 1 \text{ for } i = 1, \ldots, n$$

(The minimizaton will shrink w to where at least one inequality above is an equality, at which point the margin will be $1/||w||$, so maximizing the margin is the same as minimizing $||w||^2$.)

Characteristics of the Maximum Margin Separating Hyperplane

The previous slide characterizes the maximum margin hyperplane as minimizing a quadratic function, subject to linear inequality constraints.

This is a convex optimization problem. It has a unique solution, which can be found reasonably efficiently by standard methods (or more efficiently using specialized methods).

The solution is locally sensitive to a subset of the training cases, called the *support vectors* — typically, but not always, less (often much less) than the full set of training cases.

Of course, all training cases have to be looked at before the support vectors can be identified. But when there are only a few support vectors, the computations do go faster.

Why Might a Large Margin Classifier be Good?

The maximum margin separating hyperplane seems intutively like it should be better than some other separating hyperplanes, such as one that goes very close to a training point.

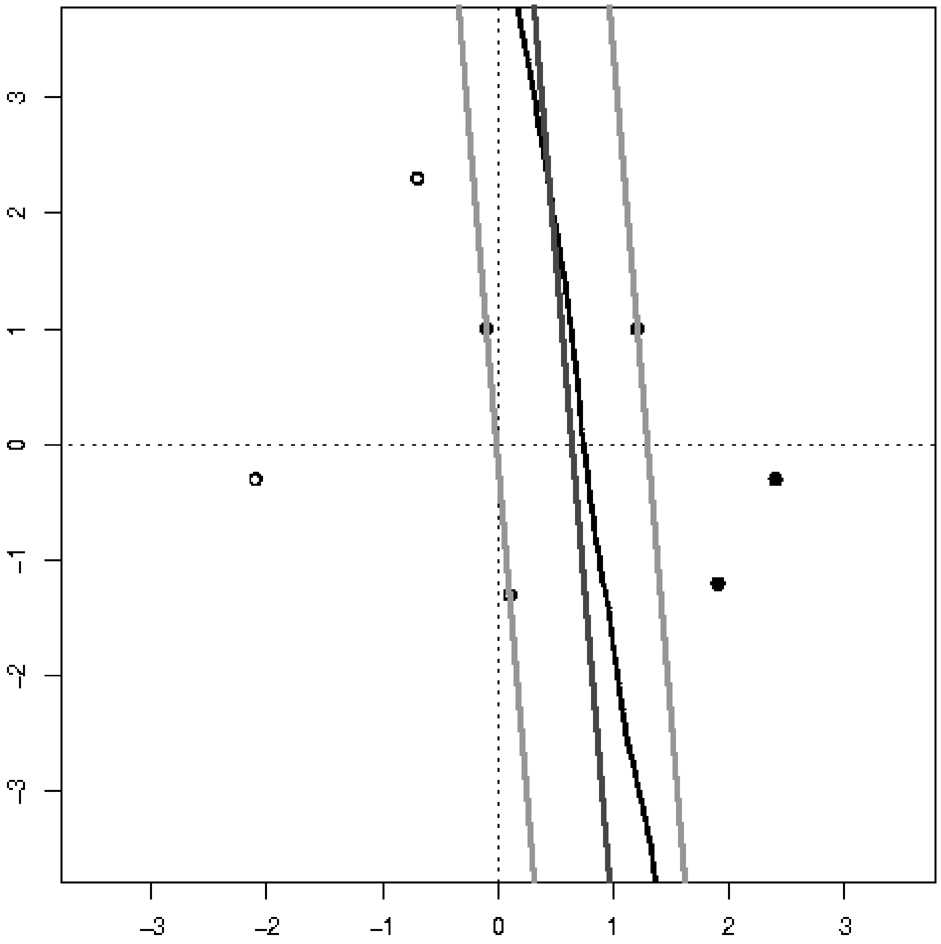
It's also the same as you get from logistic regression with coefficients that maximize the log likelihood minus an infinitesimal quadratic penalty.

Some theorists have attempted to justify large margin classifiers using "VC-dimension" arguments — that relate to how much potential there is for overfitting — but it's not clear these arguments actually succeed.

Perhaps one can see the large margin classifier as approximating Bayesian predictions (based on a vague prior distribution)...

Comparison with a Bayesian Hard Linear Classifier

The data set in the earler slide illustrating a large margin classifier is the same as the one I used in the introduction to Bayesian inference. Here from that demonstration is the curve where the classes have equal predictive probability, together with the maximum margin classifie:



Support Vector Machines

Another Way to Find the Hyperplane with Largest Margin

Recall that we if define a hyperplane by the equation $w^T x + b = 0$, we can find the maximum margin hyperplane by solving the following optimization problem:

$$\text{minimize } ||w||^2, \quad \text{subject to } y_i(w^T x_i + b) \geq 1 \text{ for } i = 1, \ldots, n$$

We can always write

$$w \;=\; \sum_{i=1}^{n} a_i x_i \;+\; \delta$$

where $\delta^T x_i = 0$ for all $i = 1, \ldots, n$, for some (not necessarily unique) set of a_i. With this representation of w,

$$||w||^2 \;=\; \Big(\sum_{i=1}^{n} a_i x_i + \delta\Big)^T \Big(\sum_{i'=1}^{n} a_{i'} x_{i'} + \delta\Big) \;=\; \sum_{i=1}^{n} \sum_{i'=1}^{n} a_i a_{i'} (x_i^T x_{i'}) \;+\; ||\delta||^2$$

and

$$y_i(w^T x_i + b) \;=\; y_i \Big(\sum_{i'=1}^{n} a_{i'} (x_i^T x_{i'}) \;+\; b \Big)$$

Since the constraints don't depend on δ, the minimization will set $\delta = 0$, so we can assume that $w = \sum_{i=1}^{n} a_i x_i$.

Another Way to Find the Hyperplane... (Continued)

So we see that we can find $w = \sum_{i=1}^{n} a_i x_i$ and b as follows:

$$\text{minimize} \sum_{i=1}^{n} \sum_{i'=1}^{n} a_i a_{i'} (x_i^T x_{i'}),$$

$$\text{subject to } y_i \Big(\sum_{i'=1}^{n} a_{i'} (x_i^T x_{i'}) + b \Big) \geq 1 \text{ for } i = 1, \ldots, n$$

This is also a quadratic programming problem — minimize a quadratic function of the a_i subject to linear constraints on the a_i and b — which could be solved by standard (and fairly efficient) methods.

However, the solution may not be unique (though the resulting w is). If the problem is formulated a bit differently, the result can be made unique, and often many of the a_i will be zero (with non-zero a_i only for the support vectors).

The formulation above does show one crucial property — the minimization depends only on inner products of input vectors (ie, on $x_i^T x_{i'}$). Predictions for test cases also depend only on such inner products, since we will classify x_* according to the sign of $w^T x_* + b = \sum_{i=1}^{n} a_i (x_*^T x_i) + b$.

Large Margin Classifiers Using Basis Functions

Rather than find a large margin classifier based on the original input vector, x, we can use a vector of basis function values, $\phi(x) = [\phi_1(x)\ \phi_2(x)\ \cdots\ \phi_m(x)]^T$.

The classes may be separable by a hyperplane in this space even if they aren't in the original space.

Finding $a_1, \ldots, a_n$ and b can be done as before, using inner products, $\phi(x_i)^T\phi(x_{i'})$.

A test case with input vector x_* is classified by the sign of $\sum_{i=1}^{n} a_i \left(\phi(x_*)^T\phi(x_i)\right) + b$.

Since all that matters are these inner products, we can define

$$K(x, x') \;=\; \phi(x)^T\phi(x') \;=\; \sum_{j=1}^{m} \phi_j(x)\phi_j(x')$$

and then look at $K(x_i, x_{i'})$ for training cases i and i', and $K(x_*, x_i)$ for a test case.

So once we have a formula for $K(x, x')$, we can forget about the ϕ functions.

Classification (and regression) methods based on this "kernel trick" are known as *Support Vector Machines* (abbreviated to "SVM").

Letting the Number of Basis Functions Go to Infinity

Since all we need is a formula for the "kernel function",

$$K(x,x') \;=\; \sum_{j=1}^{m} \phi_j(x)\phi_j(x')$$

we can consider letting the number of basis functions, m, go to infinity, as long the resulting infinite sum has a finite limit, and can be computed efficiently.

This is essentially identical to what we did earlier for Gaussian process models. The noise-free covariance function corresponding to a Bayesian linear basis function model with independent zero-mean normal priors for coefficients, with the variance of the coefficient for ϕ_j being ω_j^2, was found to be

$$K(x,x') \;=\; \sum_{j=0}^{m-1} \omega_j^2 \phi_j(x)\phi_j(x')$$

This becomes the same as above if we absorb a factor ω_j into the definition of ϕ_j (and replace 0 to $m-1$ with 1 to m).

Possible Kernel Functions

The possible kernel functions for a support vector machine are the same as the possible covariance functions for a Gaussian process model — all those that produce positive semi-definite matrices at any set of points.

Mercer's Theorem says that all such positive definite kernels can be represented in the form $K(x, x') = \sum \phi_j(x)\phi_j(x')$, though sometimes all but a finite number of the ϕ_j will be identically zero.

So the class of models defined using linear basis functions is the same as the class of models defined using a kernel/covariance function.

Commonly used kernel functions include $K(x, x') = (1 + x^T x')^d$, corresponding to polynomial basis functions to degree d, and $K(x, x') = \exp(-\rho^2 ||x - x'||^2)$.

Note that for an SVM (unlike for a Gaussian process), multiplying the kernel function by a positive constant does not change things.

More Elaborations on Support Vector Machines

- Which kernel function is best is usually not clear. Cross validation can be used to choose one.

- Finding a separating hyperplane (even if always possible in an infinite dimensional space) may not be a good idea, when class labels are actually "noisy". Introducing "slack variables" allows for some mis-classified points.

- Classification problems with more than two classes can be handled in various ways — eg, combining results from pairwise binary classifiers.

- Regression problems can be handled by using a "loss" function that is "ϵ-insensitive" — where small errors cost zero.

Support Vector Machines vs. Gaussian Process Models

SVM and GP models have a strong common element — the positive semi-definite kernel/covariance function. How do they compare otherwise?

Advantages of support vector machines:

- The number of support vectors is often much less than the total size of the training set, reducing computation time for training and prediction.
- Binary classification can be done directly, with a relatively fast optimization procedure, whereas Gaussian process classification requires handling a distribution over “latent variables”.

Advantages of Gaussian process models:

- The covariance function has a probabilistic interpretation — one can sample from the prior over functions that it defines — which can guide the choice of a suitable covariance function.
- Finding good parameters of the covariance function can be done by maximum likelihood (or by Bayesian methods), without the need for cross validation.
- Classification problems with more than two classes can be handled naturally.
- Regression can be done using a conventional Gaussian model for residuals.

Kernel PCA

PCA on Basis Function Values

Rather than do PCA on the original vector of p values, x, we can do it on the values of m basis functions, $\phi(x) = [\phi_1(x), \ldots, \phi_m(x)]$.

If $m > p$, the basis function values will lie in a p-dimensional space embedded in an m-dimensional space. This would make modeling the data as a multivariate Gaussian be drastically incorrect, but PCA makes no distributional assumptions — it just finds the directions of maximum sample variance.

If m is big, we should of course use the trick that was presented earlier for when p is big — find the eigenvectors of the $n \times n$ matrix XX^T rather than the eigenvectors of the $p \times p$ matrix X^TX.

Note that even if the x values have been centred to have sample mean of zero, the $\phi(x)$ values will probably not be centred. So we'll have to subtract the sample mean for each basis function before finding eigenvectors.

Details of PCA on Basis Function Values

Let Φ be the $n \times m$ matrix of basis function values for the n observed items, so $\Phi_{ik} = \phi_k(x_i)$.

If we let $\mathbf{1}_n$ be a vector of n ones, we can get a row vector of sample means of the basis functions as $(1/n)\mathbf{1}_n^T\Phi$. The matrix of centred basis function values can then be written as

$$\widetilde{\Phi} \quad = \quad \Phi \; - \; \mathbf{1}_n[(1/n)\mathbf{1}_n^T\Phi] \quad = \quad [I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]\,\Phi$$

where $I_{n\times n}$ is the $n \times n$ identity matrix and $\mathbf{1}_{n\times n}$ is the $n \times n$ matrix of all ones.

We now find the eigenvectors of

$$\widetilde{\Phi}\widetilde{\Phi}^T \quad = \quad [I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]\,\Phi\Phi^T\,[I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]$$

If v is such an eigenvector, of length one, with eigenvalue λ, then $\widetilde{\Phi}^T v/\sqrt{\lambda}$ is an eigenvector of $\widetilde{\Phi}^T\widetilde{\Phi}$, also of length one, and with eigenvalue λ.

Projections of Basis Function Vectors for Test Points

What we're usually interested in are the projections of the basis function vectors for test points on the principal components.

Let $\widetilde{\Phi}^T v/\sqrt{\lambda}$ be a principal component direction, where v is an eigenvalue of $\widetilde{\Phi}\widetilde{\Phi}^T$ with eigenvalue λ, and recall that $\widetilde{\Phi} \;=\; [I_{n\times n} \;-\; \mathbf{1}_{n\times n}/n]\,\Phi$.

The projection in this direction of the centred basis function values of a point x_* is

$$\begin{aligned}
[\phi(x_*)^T - (1/n)\mathbf{1}_n^T\Phi]\,\widetilde{\Phi}^T v/\sqrt{\lambda} &= [\phi(x_*)^T - (1/n)\mathbf{1}_n^T\Phi]\,\Phi^T[I_{n\times n} \;-\; \mathbf{1}_{n\times n}/n]\,v/\sqrt{\lambda} \\
&= [\phi(x_*)^T\Phi^T - \mathbf{1}_n^T\Phi\Phi^T/n]\,[I_{n\times n} \;-\; \mathbf{1}_{n\times n}/n]\,v/\sqrt{\lambda}
\end{aligned}$$

Applying the Kernel Trick

All these operations involve $\phi(x)$ only via inner products. We can define

$$K(x, x') \quad = \quad \phi(x)^T \phi(x')$$

and then define the $n \times n$ matrix K by $K_{ij} = K(x_i, x_j)$. We then can compute

$$\widetilde{K} \quad = \quad \widetilde{\mathbf{\Phi}}\widetilde{\mathbf{\Phi}}^T \quad = \quad [I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]\, K \,[I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]$$

If the $n \times n$ matrix $\widetilde{K}$ has unit length eigenvectors $v_1, v_2, \ldots, v_n$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, then the projection of a data point x_* on the m'th principal component is

$$[k \; - \; \mathbf{1}_n^T K/n]\,[I_{n\times n} \; - \; \mathbf{1}_{n\times n}/n]\, v_m/\sqrt{\lambda_m}$$

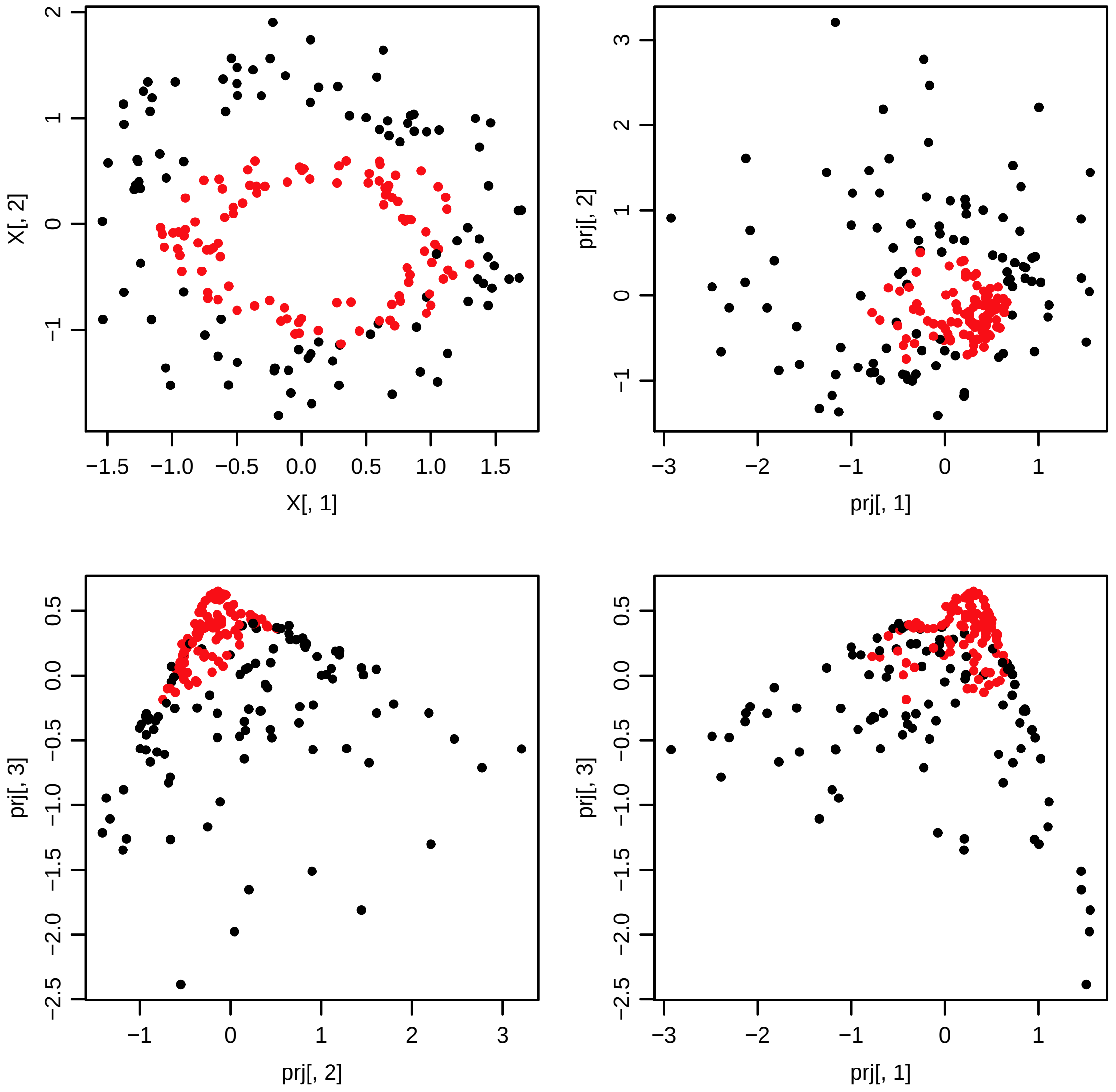
where k is the vector of dimension n with $k_i = K(x_*, x_i)$.

Since ϕ no longer appears explicitly in these formulas, we can let the number of basis functions go to infinity, as long as we know how to compute $K(x, x')$.

Example of Kernel PCA

Kernel PCA for 2D data from two classes, using $K(x, x') = \exp(-||x - x'||^2)$.

Original data and pairs of projections on PC1, PC2, and PC3:


X[, 1]
X[, 2]
prj[, 1]
prj[, 2]
prj[, 3]