Advanced Computational Econometrics: Machine Learning

Chapter 7: Support Vector Machines

Thierry Denœux

July-August 2019

Support Vector Machines

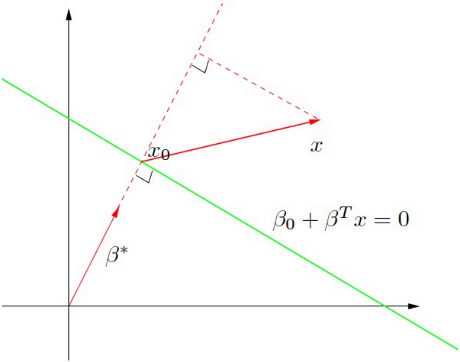
- In this chapter we describe new methods for linear and nonlinear classification.
- Optimal separating hyperplanes are first introduced for the case when two classes are linearly separable. Then we cover extensions to the nonseparable case, where the classes overlap.
- These techniques are then generalized to what is known as the support vector machine (SVM), which produces nonlinear boundaries by constructing a linear boundary in a large, transformed version of the predictor space.

Overview

Overview

Hyperplane

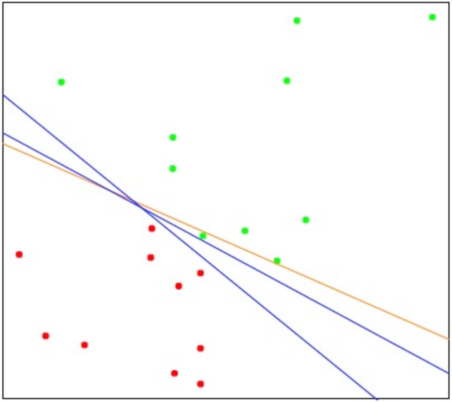
- In $\mathbb{R}^p$, a hyperplane H is defined by the equation $g(x) = 0$ with $g(x) = \beta_0 + \beta^T x$. We have $g(x) > 0$ on one side of H and $g(x) < 0$ on the other side.



- For any two points x_1 and x_2 lying in H, $\beta^T(x_1 - x_2) = 0$, hence $\beta^* = \beta / \|\beta\|$ is the vector normal to the surface of H.
- Let $x_0 \in H$. The signed distance of any point x to H is

$$d_s(x, H) = \beta^{*T}(x - x_0) = \frac{\beta^T x + \beta_0}{\|\beta\|} = \frac{g(x)}{\|\beta\|}$$

Linearly separable data



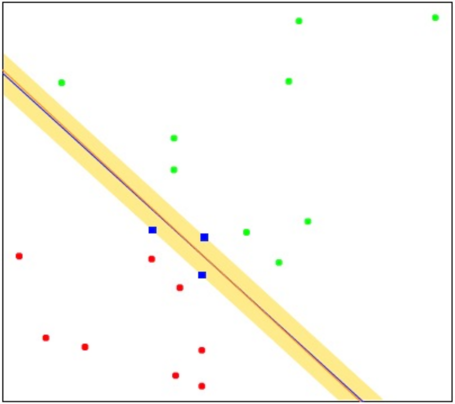
- Consider a two-class data set $\{(x_i, y_i)\}_{i=1}^n$ with $y_i \in \{-1, 1\}$.
- It is said to be linearly separable if there exists a hyperplane $H : g(x) = 0$ that separates the two classes, i.e., such that

$$g(x_i)y_i > 0, \quad \forall i.$$

Optimal separating hyperplane

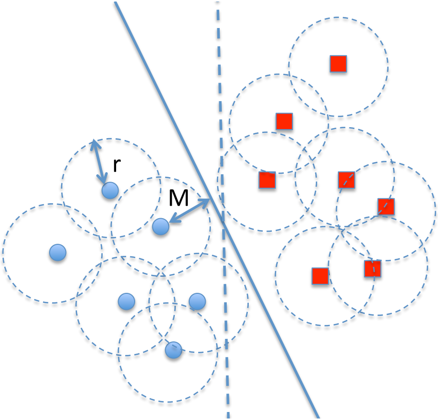
- Let H be a separating hyperplane. The distance between H and a learning vector x_i is
$$d(x_i, H) = \frac{g(x_i)y_i}{\|\beta\|}$$
- The margin of H is the smallest distance between H and a learning vector x_i:
$$M = \min_i d(x_i, H).$$
- The optimal separating hyperplane (OSH) is the hyperplane with the largest margin.
- The learning vectors x_i such that $d(x_i, H) = M$ are called the support vectors of H.

Example



The shaded region delineates the maximum margin separating the two classes. There are 3 support vectors, and the OSH is the blue line. The boundary found using logistic regression is the red line. In this case, it is very close to the OSH.

The OSH is more likely to separate future data



- Future data can be assumed to be "close" to past data.
- Assume they will lie with a distance r of a past data point.
- If $M > r$, the hyperplane will classify future data perfectly.

Overview

How to find the OSH?

- The OSH can be found by solving the following optimization problem:

$$\max_{\beta,\beta_0} M$$

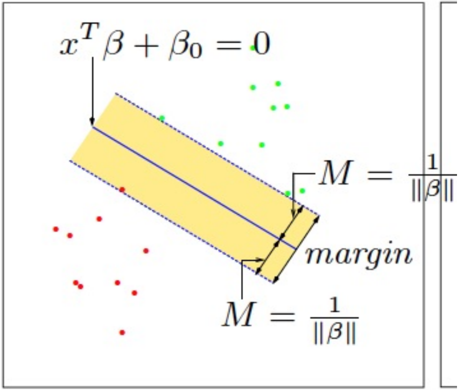
$$\text{subject to} \quad \frac{y_i(\beta^T x_i + \beta_0)}{\|\beta\|} \geq M, \quad i = 1, \ldots, n.$$

- If (β, β_0) is a solution, so is $(\lambda\beta, \lambda\beta_0)$ for any λ. Hence, we can fix $\|\beta\| = 1/M$ and reformulate the problem as

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$

$$\text{subject to} \quad y_i(\beta^T x_i + \beta_0) \geq 1, \quad i = 1, \ldots, n.$$

Interpretation



- The constraints define an empty band or margin around the linear decision boundary of thickness $1/\|\beta\|$.
- The vectors x_i such that $y_i(\beta^T x_i + \beta_0) = 1$ are the support vectors.

Reminder on constrained optimization

Lagrange function

- Consider the following minimization problem:

$$\min_{\beta} f(\beta) \tag{1}$$

subject to the constraints $c_i(\beta) \geq 0$, $i = 1, \ldots, n$, where f and the c_i's are differentiable functions.

- The Lagrange function is defined by

$$L(\beta, \alpha) = f(\beta) - \sum_{i=1}^{n} \alpha_i c_i(\beta),$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is the vector of Lagrange multipliers.

Reminder on constrained optimization

Theorem (Karush-Kuhn-Tucker conditions)

If function f has a minimum for some value β^* in the feasibility region, the following Karush-Kuhn-Tucker (KKT) conditions are verified for some numbers α_i^*, $i = 1, \ldots, n$:

$$\begin{aligned}
\frac{\partial L}{\partial \beta}(\beta^*, \alpha^*) &= 0 && \text{(2a)}\\
c_i(\beta^*) &\geq 0, \quad i = 1, \ldots, n && \text{(2b)}\\
\alpha_i^* c_i(\beta^*) &= 0 \quad i = 1, \ldots, n && \text{(2c)}\\
\alpha_i^* &\geq 0 \quad i = 1, \ldots, n. && \text{(2d)}
\end{aligned}$$

Reminder on constrained optimization (continued)

Theorem (Wolfe dual)

Problem (1) is equivalent to the following problem (Wolfe dual):

$$\max_{\beta,\alpha} L(\beta, \alpha) \tag{3}$$

subject to

$$\frac{\partial L}{\partial \beta} = 0 \tag{4}$$

$$\alpha_i \geq 0 \quad i = 1, \ldots, n. \tag{5}$$

The active constraints $c_i(\beta^*) = 0$ *correspond to indices* i *such that* $\alpha_i^* > 0$.

Lagrange function

- Let us come back to the problem $\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$ subject to $y_i(\beta^T x_i + \beta_0) \geq 1$, $i = 1, \ldots, n$.
- This is a convex optimization problem (quadratic criterion with linear inequality constraints), so the solution exists and it is unique.
- The Lagrange (primal) function is

$$L_P(\beta, \beta_0, \alpha) = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\beta^T x_i + \beta_0) - 1], \tag{6}$$

- Setting the derivatives to zero, we obtain:

$$\frac{\partial L_P}{\partial \beta} = \beta - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{7}$$

$$\text{and} \quad \frac{\partial L_P}{\partial \beta_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{8}$$

Dual problem

- Substituting (7) and (8) in (6) we obtain the Wolfe dual

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

- The solution is obtained by maximizing L_D subject to the constraints

$$\alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{9}$$

- This can be done using standard quadratic programming software. We will discuss a specialized optimization algorithm later.

Interpreting the solution

Support vectors

- The solution α^* must satisfy the KKT conditions, which include (7), (8), (9) and

$$\alpha_i^*[y_i({\beta^*}^T x_i + \beta_0^*) - 1] = 0, \quad i = 1, \ldots, n. \tag{10}$$

- From these we can see that
  - if $\alpha_i^* > 0$, then $y_i({\beta^*}^T x_i + \beta_0^*) = 1$, i.e., x_i is a support vector
  - $y_i({\beta^*}^T x_i + \beta_0^*) > 1$, x_i is not a support vector, and $\alpha_i^* = 0$

Interpreting the solution

Computing β^* and β_0^*

- From (7) we see that the solution vector β^* is defined in terms of a linear combination of the support vectors

$$\beta^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i = \sum_{i \in S} \alpha_i^* y_i x_i \tag{11}$$

with $S = \{i | \alpha_i^* > 0\}$ and β_0^* can be found from (10).

- For any support vector x_i, we have

$$y_i({\beta^*}^T x_i + \beta_0^*) = 1,$$

from which we can get β_0^*.

Resulting classifier

- The equation of the OSH is

$$g^*(x) = {\beta^*}^T x + \beta_0^* = \sum_{i \in S} \alpha_i^* y_i x_i^T x + \beta_0^* = 0$$

- The corresponding classifier is

$$D(x) = \text{sign } g^*(x).$$

- The classifier is based only on support vectors, which are close to the boundary between classes.

Overview

Extension to non-separable data

- Until now, we have assumed that the data are linearly separable.
- This will generally not be the case with real data, so the technique derived so far is not really useful in practice.
- We need to propose an alternative formulation for the non-separable case.

Weakening the constraints

- Suppose that the classes overlap in predictor space.
- One way to deal with the overlap is to still maximize the margin M, but allow for some points to be on the wrong side of the margin.
- Define the slack variables $\xi = (\xi_1, \xi_2, \ldots, \xi_n)$ with $\xi_i \geq 0$. The constraints can be modified as

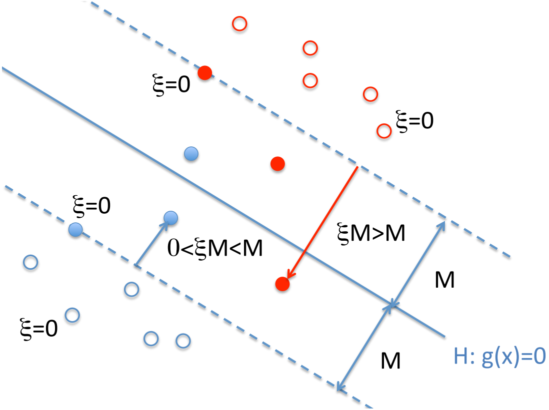
$$\frac{y_i(\beta^T x_i + \beta_0)}{\|\beta\|} \geq M(1 - \xi_i), \quad i = 1, \ldots, n.$$

- As before, fixing $\|\beta\| = 1/M$, this is equivalent to

$$y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i, \quad i = 1, \ldots, n.$$

- The value ξ_i is the proportional amount by which vector x_i is on the wrong side of its margin.

Interpretation



- The filled points are on the wrong side of their margin by an amount $M\xi_i$.
- Points on the correct side have $\xi = 0$.
- Misclassified points have $\xi_i > 1$.

Optimization problem

- The optimization problem now becomes:

$$\min_{\beta,\beta_0,\{\xi_i\}} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i,$$

subject to

$$\xi_i \geq 0, \quad i = 1,\ldots,n$$

$$y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i, \quad i = 1,\ldots,n$$

where C is a hyperparameter.

Lagrange function

- The Lagrange (primal) function is

$$L_P(\beta,\beta_0,\xi,\alpha,\mu) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i[y_i(\beta^T x_i + \beta_0) - (1-\xi_i)] - \sum_{i=1}^n \mu_i\xi_i.$$

- Setting the derivatives w.r.t. β, β_0 and ξ to zero, we get

$$\beta = \sum_{i=1}^n \alpha_i y_i x_i, \quad \sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i = C - \mu_i \tag{12}$$

Dual formulation

- By substituting (12), we obtain the Lagrangian dual objective function

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j, \tag{13}$$

which has exactly the same form as in the previous problem.

- We maximize L_D subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.
- The SMO (sequential minimal optimization) algorithm gives an efficient way of solving this problem.

SMO algorithm

- The SMO is a grouped coordinate ascent algorithm.
- Maximizing $L_D(\alpha)$ one α_i at a time does not work, because due to the constraint

$$\sum_{i=1}^{n} \alpha_i y_i = 0,$$

variable α_i is uniquely determined from the other α_j's through the equation

$$\alpha_i = -y_i \sum_{j \neq i} \alpha_j y_j.$$

- Instead, the SMO algorithm maximizes $L_D(\alpha)$ w.r.t. to each pair of variables (α_i, α_j) sequentially.

SMO algorithm (continued)

Repeat until convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $L_D(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

To test for convergence of this algorithm, we can check whether the KKT conditions are satisfied to within some tolerance (see next slide).

Interpretation of the solution

- The solution verifies the KKT conditions (12) and

$$\alpha_i^*[y_i({\beta^*}^T x_i + \beta_0^*) - (1 - \xi_i^*)] = 0, \quad i = 1, \ldots, n \tag{14}$$

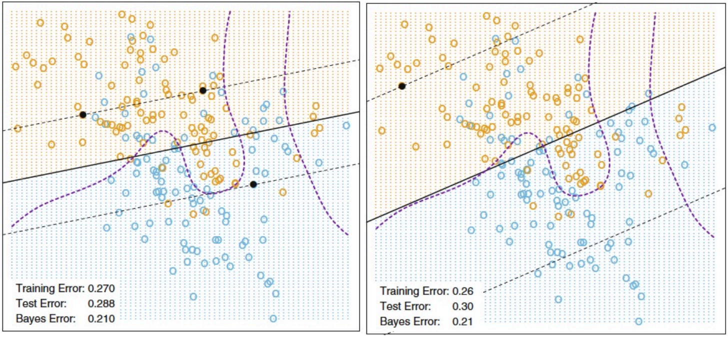
$$\mu_i^* \xi_i^* = 0, \quad i = 1, \ldots, n \tag{15}$$

- As before, the support vectors are the points such that $\alpha_i^* > 0$.
- From (12) and (15), the supports vectors such that $\alpha_i^* < C$ verify $\xi_i^* = 0$: they lie on the edge of the margin. The remainder ($\xi_i^* > 0$) have $\alpha_i^* = C$.
- The support vectors such that $\xi_i^* > 1$ are misclassified.
- From (14) we can see that any of the margin points ($\alpha_i^* > 0$, $\xi_i^* = 0$) can be used to solve for β_0^*, and we typically use an average of all the solutions for numerical stability.

Tuning C

- The tuning parameter of this procedure is the cost parameter C.
- The optimal value for C can be estimated by cross-validation.
- The margin is smaller for larger C. Hence larger values of C focus attention more on points near the decision boundary, while smaller values involve data further away.
- The leave-one-out cross-validation error can be bounded above by the proportion of support vectors in the data. The reason is that leaving out an observation that is not a support vector will not change the solution. Hence these observations, being classified correctly by the original boundary, will be classified correctly in the cross-validation process. However this bound tends to be too high, and not generally useful for choosing C.

Example



$C = 10000$ $\quad\quad$ $C = 0.01$

The support vectors ($\alpha_i^* > 0$) are all the points on the wrong side of their margin. The black solid dots are those support vectors falling exactly on the margin ($\xi_i^* = 0$, $\alpha_i^* > 0$). In the left (resp., right) panel 62% (resp., 85%) of the observations are support vectors.

Application in R

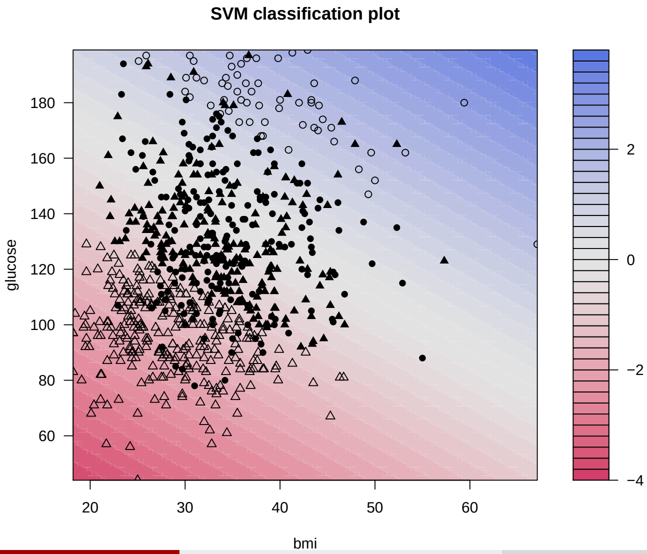
```
library("kernlab")

ii<-which((pima$glucose>0) & (pima$bmi>0))

svmfit<-ksvm(as.factor(class)~ glucose+bmi,data=pima[ii,],
             type="C-svc",kernel="vanilladot",C=10)

plot(svmfit,data=pima[ii,],grid=100)
```

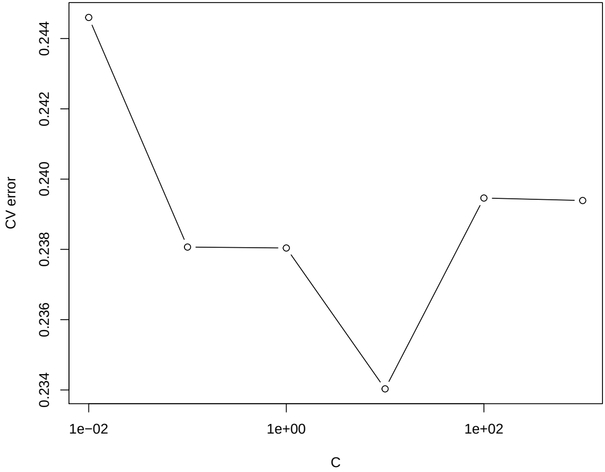
Result


SVM classification plot
glucose
180
160
140
120
100
80
60
20
30
40
50
60
bmi
2
0
−2
−4

Selection of C by cross-validation

```
CC<-c(0.01,0.1,1,10,100,1000)
N<-length(CC)
err<-rep(0,N)
for(i in 1:N){
err[i]<-cross(ksvm(as.factor(class)~glucose+bmi,data=pima[ii,],
            type="C-svc",kernel="vanilladot",C=CC[i],cross=5))
}
plot(CC,err,type="b",log="x",xlab="C",ylab="CV error")
```

Cross-validation result



Overview

Overview

Extension to non-linear classification

- The support vector classifier described so far finds linear boundaries in the predictor space.
- As with other linear methods, we can make the procedure more flexible by enlarging the predictor space using basis expansions such as polynomials or splines.
- Linear boundaries in the enlarged space generally achieve better training-class separation, and translate to nonlinear boundaries in the original space.

Extension to non-linear classification (continued)

- Once the basis functions $\Phi_m(x)$, $m = 1, \ldots, M$ are selected, the procedure is the same as before:
  - We fit the SV classifier using predictors $\Phi(x_i) = (\Phi_1(x_i), \Phi_2(x_i), \ldots, \Phi_M(x_i))$, $i = 1, \ldots, n$, and produce the (nonlinear) function $g^*(x) = \Phi(x)^T\beta^* + \beta_0$.
  - The classifier is $D^*(x) = \text{sign}(g^*(x))$ as before.
- In SVM, the mapping $x \rightarrow \Phi(x)$ will be defined implicitly, and M will be potentially very large (even infinite!).

The OSH depends only on dot-products

A key feature of the OSH is that it depends only on the dot products between input vectors:

- The solution is found by maximizing

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j, \tag{16}$$

  subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.
- The optimal discriminant function is

$$g^*(x) = \sum_{i \in S} \alpha_i^* y_i x_i^T x + \beta_0^* = 0,$$

  where β_0^* also depends only on the dot products $x_i^T x_j$.

Dot-products in the transformed input space

- Assume that the input vector x is replaced by $\Phi(x)$ for some transformation $\Phi : \mathbb{R}^p \to \mathcal{H}$.
- The objective function will become

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \Phi(x_i), \Phi(x_j) \rangle \tag{17}$$

and the optimal discriminant function will be

$$g^*(x) = \sum_{i \in S} \alpha_i^* y_i \langle \Phi(x_i), \Phi(x) \rangle + \beta_0^* = 0,$$

where $\langle \cdot, \cdot \rangle$ denotes the dot-product in $\mathcal{H}$.
- All we need is a method to compute dot-products in $\mathcal{H}$.

The "kernel trick"

- If there exists a kernel function $\mathcal{K} : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$ such that

$$\mathcal{K}(x, x') = \langle \Phi(x), \Phi(x') \rangle,$$

  then the transformation Φ will be defined implicitly.
- This is the "kernel trick".

Example

- Assume $p = 2$ and $\mathcal{K}(x, x') = (x^T x')^2$.
- We have
$$\mathcal{K}(x, x') = \Phi(x)^T \Phi(x')$$
with
$$\Phi : x \longrightarrow \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$
- Function Φ is defined implicitly by the kernel function $\mathcal{K}$.

Overview

Mercer condition

Theorem

A kernel function $\mathcal{K}$ corresponds to a dot-product in some space $\mathcal{H}$ iff it verifies the following Mercer condition:

$$\forall f : \mathbb{R}^p \rightarrow \mathbb{R} \text{ s.t. } \int f(x)^2 dx < \infty, \quad \int \mathcal{K}(x, x') f(x) f(x') dx dx' \geq 0.$$

If the Mercer condition is not verified, the Wolf dual problem may not have a solution. In practice, the method may still work most of the time with a kernel function that does not meet this condition.

Popular kernel functions

- Three popular choices for $\mathcal{K}$ in the SVM literature are

$$\begin{aligned}
\mathcal{K}(x,x') &= (x^T x' + 1)^d, \quad d > 0 && \text{(polynomial kernel)}\\
\mathcal{K}(x,x') &= \exp\left[-\gamma \|x - x'\|^2\right], \quad \gamma > 0 && \text{(RBF or Gaussian kernel)}\\
\mathcal{K}(x,x') &= \tanh(\kappa_1 x^T x' + \kappa_2) && \text{(MLP kernel).}
\end{aligned}$$

- The polynomial and Gaussian verify the Mercer condition, but the MLP kernel does not.
- With the MLP kernel, the discriminant function is

$$g(x) = \sum_{i \in S} \alpha_i^* y_i \tanh(\kappa_1 x_i^T x + \kappa_2) + \beta_0^*.$$

It is the transfer function of a neural network with $n_S = \text{card}(S)$ hidden units (see chapter on neural networks).

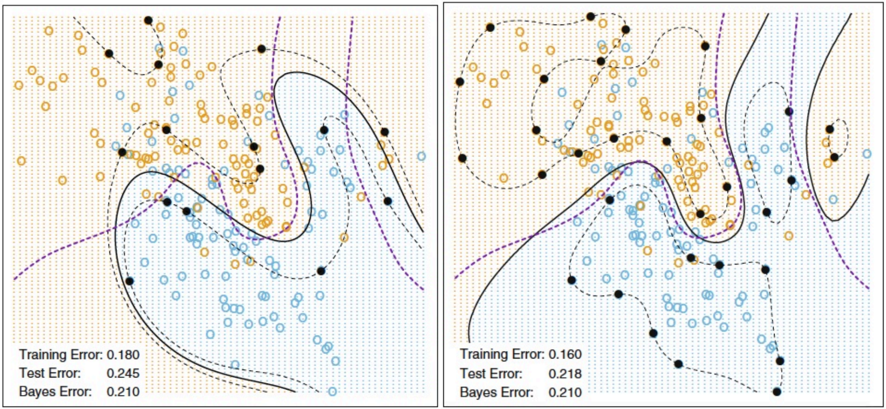
Influence of C

- The role of parameter C is clearer in an enlarged predictor space, since perfect separation is often achievable there. (The dimension of $\mathcal{H}$ may be very large and even infinite.)
- A large value of C will discourage any positive ξ_i, and lead to an overfit wiggly boundary in the original predictor space; a small value of C will encourage a small value of $\|\beta\|$, which in turn causes $g(x)$ and hence the boundary to be smoother.
- Both C and the kernel parameters (d, γ, etc.) are usually tuned by cross-validation.

Example

SVM - Degree-4 Polynomial in Feature Space

SVM - Radial Kernel in Feature Space

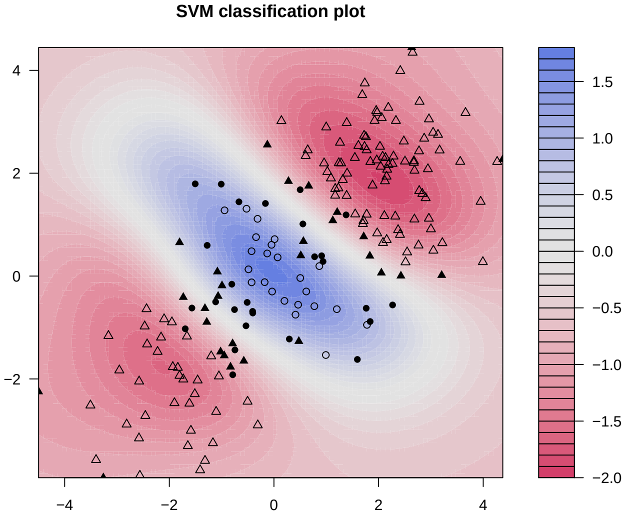


Application in R

```
x<-matrix(rnorm(200*2),ncol=2)
y<-as.factor(c(rep(-1,150),rep(1,50)))
x[1:100,]<- x[1:100,]+2
x[101:150,]<- x[101:150,]-2

svmfit<-ksvm(x,y,type="C-svc",kernel="rbfdot",
           kpar=list(sigma=1),C=1)
plot(svmfit,data=x,grid=100)
```

Result



SVM as a penalization method

- Let $g(x_i) = \beta^T \Phi(x_i) + \beta_0$. The problem

$$\min_{\beta,\beta_0,\{\xi_i\}} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \xi_i,$$

subject to $\xi_i \geq 0$ and $y_i g(x_i) \geq 1 - \xi_i$, $i = 1, \ldots, n$ is equivalent to the unconstrained optimization problem:

$$\min_{\beta,\beta_0} \sum_{i=1}^{n} [1 - y_i g(x_i)]_+ + \frac{\lambda}{2}\|\beta\|^2,$$

where $[\cdot]_+$ denotes the positive part, with $\lambda = 1/C$.

- This has the form loss + penalty, with the "hinge" loss function $J(y, g(x)) = [1 - yg(x)]_+$.

Estimation of posterior probabilities

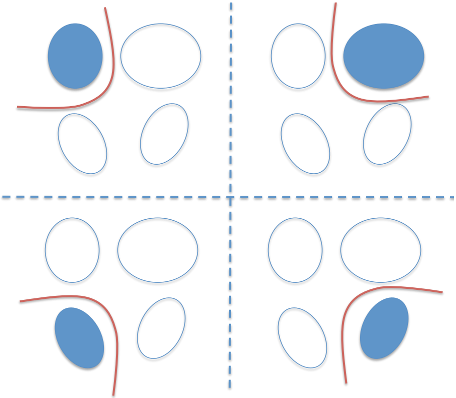
- The SVM classifier gives us a decision function, but it does not provide estimates of conditional class probabilities $P(x) = \mathbb{P}(Y = +1 \mid X = x)$.
- One approach to estimate these probabilities is to use logistic regression, with the output $g(x)$ of the SVM classifier as the predictor. We then have
$$\widehat{P}(x) = \frac{1}{1 + \exp[-(a + b \cdot g(x))]}$$
- To avoid overfitting, it is preferable to estimate the additional parameters a and b from a validation dataset or using cross-validation.

Overview

From binary to multi-class classification

- So far, the discussion has been restricted to binary classification.
- To apply the SVM technique to multi-class classification, we usually decompose the multi-class problem into several binary problems.
- How?

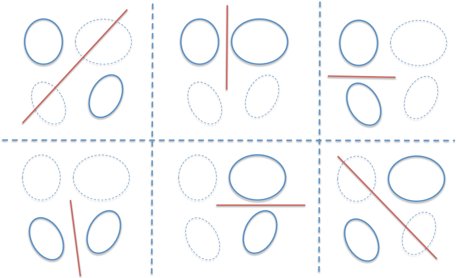
One-against-all approach



- c binary classifiers are trained to discriminate between each class and the $c-1$ others.
- Let g_k^* be the discriminant function that classifies class k vs. all other classes. We have

$$D(x) = \arg\max_k g_k^*(x).$$

One-against-one approach



- $c(c-1)/2$ binary classifiers are trained to discriminate between each pair of classes.
- Each classifier votes for a class.
- The majority class is selected.

One-against-all or one-against-one?

- The one-against-one approach implies solving a larger number of binary classification problems, but each one of them is simpler and involves fewer data points.
- The one-against-one approach often outperforms one-against-all and is preferred in most cases, except when the number of classes is very large.
- Function `ksvm` in R package `kernlab` uses the one-against-one approach for multi-class classification.

Example

```
letter <- read.table("letter-recognition.data",header=FALSE)
n<-nrow(letter)
napp=10000
train<-sample(1:n,napp)
letter.test<-letter[-train,]
letter.train<-letter[train,]

fit<- ksvm(as.factor(V1)~.,kernel="rbfdot",C=1,data=letter.train)

# Number of votes for each class
pred<-predict(fit,newdata=letter.test,type = "votes")

# Error rate calculation
pred<-predict(fit,newdata=letter.test,type = "response")
mean(letter.test$V1 != pred)

0.0807
```