Support Vector Machines

a popular classification method in the computer science community:
- one of the best black box classifier

Here we approach the two-c[illegible]blem in a direct way:

> *We try and find a plane that separates the classes in feature space.*

If we cannot, we get creative in two ways:

- We soften what we mean by "separates", and
- We enrich and enlarge the feature space so that separation is possible.

What is a Hyperplane?

- A hyperplane in p dimensions is a flat affine subspace of dimension $p - 1$.
- In general the equation for a hyperplane has the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

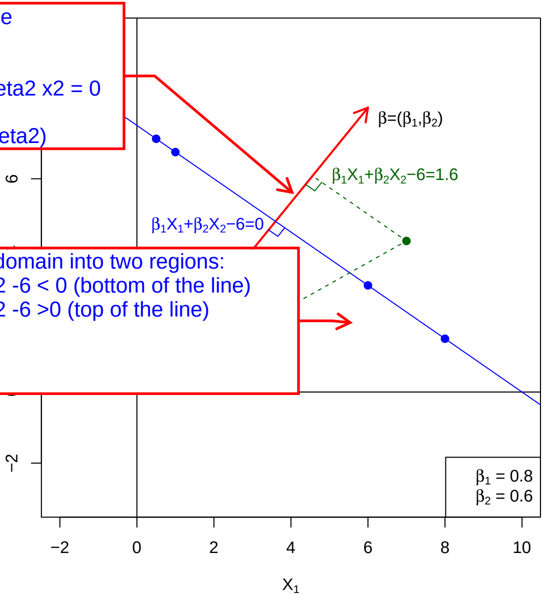
- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

Hyperplane in 2 Dimensions

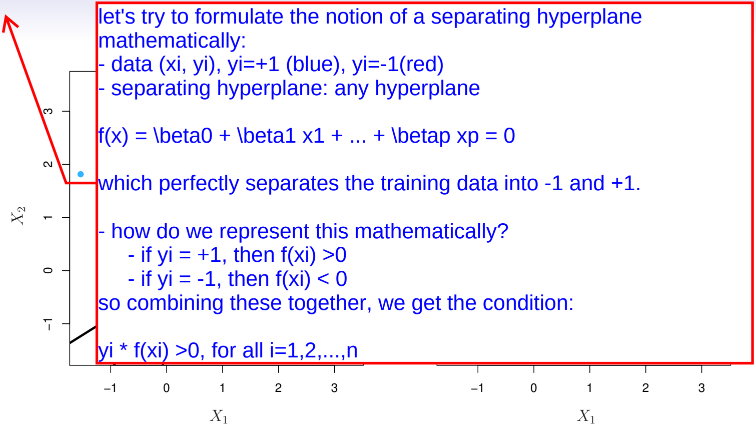
the normal vector for the hyperplane

\beta0 + \beta1 x1 + \beta2 x2 = 0

is the vector (\beta1, \beta2)



we can then split the domain into two regions:
- \beta1 x1 + \beta2 x2 -6 < 0 (bottom of the line)
- \beta1 x1 + \beta2 x2 -6 >0 (top of the line)



let's try to formulate the notion of a separating hyperplane mathematically:

- data (xi, yi), yi=+1 (blue), yi=-1(red)
- separating hyperplane: any hyperplane

f(x) = \beta0 + \beta1 x1 + ... + \betap xp = 0

which perfectly separates the training data into -1 and +1.

- how do we represent this mathematically?
    - if yi = +1, then f(xi) >0
    - if yi = -1, then f(xi) < 0

so combining these together, we get the condition:

yi * f(xi) >0, for all i=1,2,...,n

- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Maximal Margin Classifier

Intuition behind Maximal Margin Classifier (MMC):
- suppoer I give you a separating hyperplane f(x) = 0
- claim: for any point x, |f(x)| measures the closest distance from x to the hyperplane (under the constraint of \sum \betaj^2 = 1)
    - can show using linear algebra

- the condition yi*f(xi) >=M:
    - provides a margin for the separation condition
    - ensures each point is at least distance M from the separating hyperplane f(x) = 0
    - margin M provides a measure of "confidence" that an observation is correctly classified (larger M => greater confidence)

- we wish to find a hyperplane (by finding \beta0,...\betap) which maximizes this margin M.

- support vectors:
    - data points which lie right on the margin of th

nd the one that makes the
o classes.

rained optimization problem

$$\text{aximize}\, M \quad {}_{0,\beta_1,\ldots,\beta_p}$$

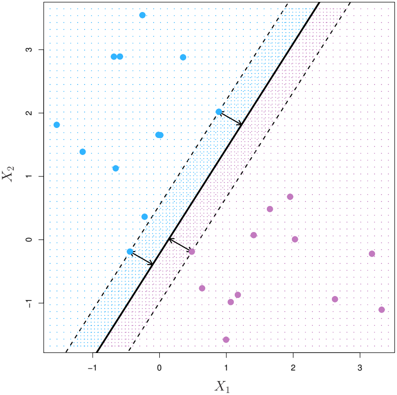
$$\text{ubject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$${}_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{or all} \;\; i = 1, \ldots, N.$$

Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

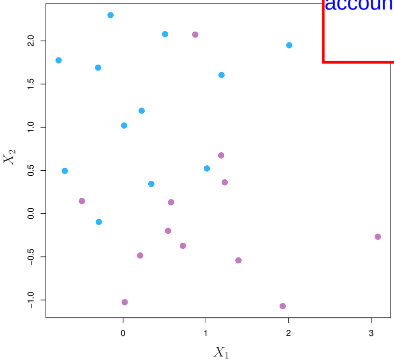
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$
$$\text{for all } \; i = 1, \ldots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently

Non-separable Data

What if the data does not have a separating hyperplane ?
- MMC optimization has no solution for M>0
- this is often the case for lower-dimensional data (N>p)
- we need to extend the notion of "separation" to account for this setting

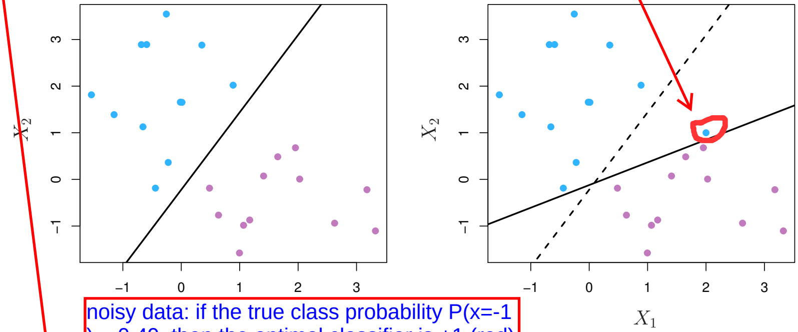


not separable by a linear boundary.

This is often the case, unless $N < p$.

Noisy D

sensitivity: if a new observation is added to the training point, MMC can change dramatically
- high variance, overfitting



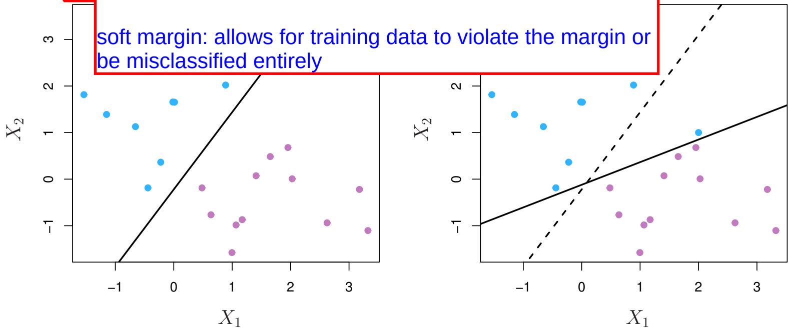
noisy data: if the true class probability P(x=-1) = 0.49, then the optimal classifier is +1 (red) , but we can easily get a label of -1 (blue) in the training data
 - by perfectly classifying the data point, MMC would give an incorrect classification
 - overfitting, high variance of MMC

Some
noisy. This can lead to a
poor
lassifier.

Noisy Data

hard margin: all training data are classified correctly and satisfies margin (distance of M away from the hyperplane) => MMC

soft margin: allows for training data to violate the margin or be misclassified entirely



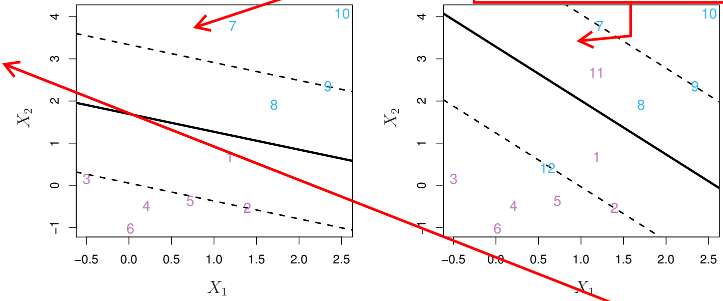
Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

The *support vector classifier* maximizes a *soft* margin.

Support Vector Classifier

observations 1 & 8: violate margin but correctly classified

observations 11 & 12: violate margin and misclassified



Comparing to MMC, what's new here?
- \epsilon_i: slack (relaxation) variables which relax the hard margin constraint
  - \epsilon_i = 0: constraint becomes a hard margin (i-th point must be classified

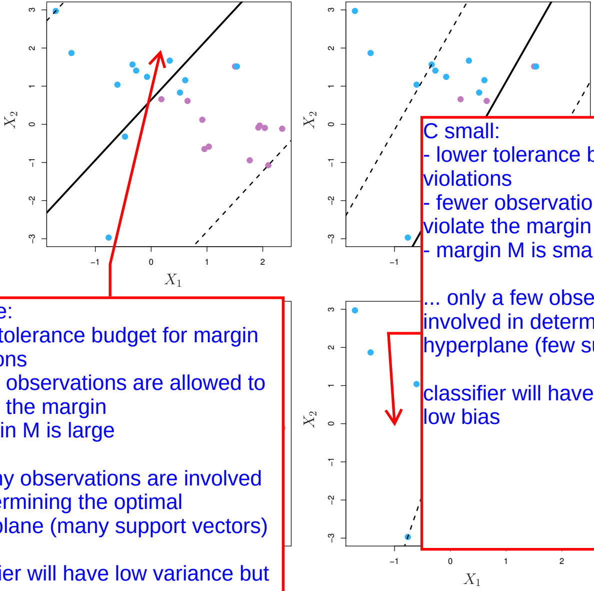
$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \ \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C,$$

C is a reg meter

controls the bias-variance trade-off of the classifier



C large:
- high tolerance budget for margin violations
- more observations are allowed to violate the margin
- margin M is large

... many observations are involved in determining the optimal hyperplane (many support vectors)

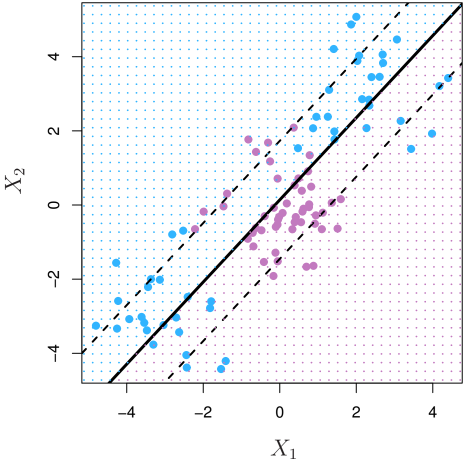
classifier will have low variance but high bias

C small:
- lower tolerance budget for margin violations
- fewer observations are allowed to violate the margin
- margin M is small

... only a few observations are involved in determining the optimal hyperplane (few support vectors)

classifier will have high variance but low bias

Linear boundary can fail



Sometime a linear boundary simply won't work, no matter what value of C.

The example on the left is such a case.

What to do?

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,. . .. Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,. . .. Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

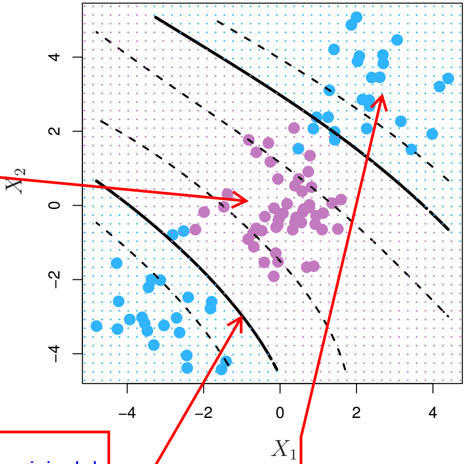
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Cubic Polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support vector classifier in the enlarged space solves the problem in the lower-dimensional space



f(x) < 0

f(x) = 0
- nonlinear in the original d=2 dimensional space
- linear in the expanded 9-dimensional space

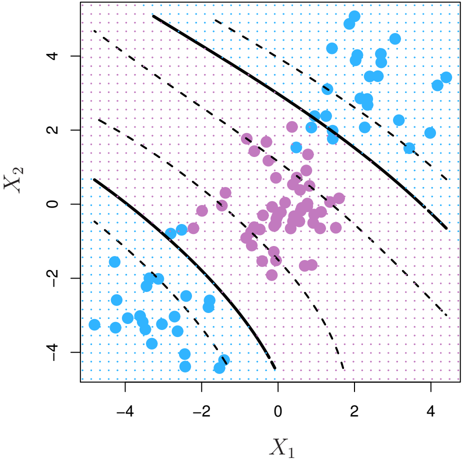
f(x) > 0

Cubic Polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space



$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2=0$$

Nonlinearities and Kernels

one computational bottleneck with previous strategy:
- p variables, d-th order polynomials for the classifier
- O(d^p) total features to optimize in the SVC
- thus, if you want to fix highly complex nonlinear classifiers, SVC optimization can be very expensive

kernels provide a nice computational trick to bypass this issue

- Polynomials (especiall rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of *kernels*.
- Before we discuss these, we must understand the role of *inner products* in support-vector classifiers.

Inner products and support vectors

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \text{— \textit{inner product between vectors}}$$

Inner products and support vectors

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \textit{— inner product between vectors}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \quad \textit{— } n \textit{ parameters}$$

Inner products and support vectors

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \text{— \textit{inner product between vectors}}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \quad \text{— \textit{n parameters}}$$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

Inner products and support vectors

note:
- if \alpha_i = 0: point i does not determine or affect the SVC (point i is not a support vector)
- if \alpha i is nonzero: point i is a support vector
- we know there are only a few support vectors => only a few of the \alpha's would be non-zero

CLAIM: the optimal SVC solution f(x) can be written in this form, for some choice of \beta_0 and \alpha_1, ... \alpha_n

— *inner* *ors*

- vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

— *n parameters*

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

$\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$. [see slide 8]

Kernels and Support Vector Machines

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!

Kernels and Support Vector Machines

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!
- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}x_{i'j}\right)^d$$

computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

Kernels and Support Vector Machines

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!
- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}x_{i'j}\right)^d$$

computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

Try it for $p = 2$ and $d = 2$.

and Support Vector Machines

can view a kernel conceptually as a way of measuring similarities (inner product) over expanded feature space

pute inner-products between observations, we

lassifier. Can be

us solve the nonlinear classifier quicker?
- earlier SVC optimization with expanded features required O(d^p) variables for optimization - many variables!
- in this kernel representation, we reduce the number of optimization variables to n +1 variables - much less!

how did we get this computational advantage? the kernel encodes all of the nonlinear features that we want, into a single similarity function.

- Some special *kernel functions* c

$$K(x_i, x_{i'}) = \Bigg(1 -$$

Solving this particular kernel representation of the SVC (with a given kernel) - support vector machines (with kernel K)

inner-products n

polynomials — $\binom{p+d}{d}$ basis fun

Try it for $p = 2$ and $d = 2$.

- The solution has the form

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

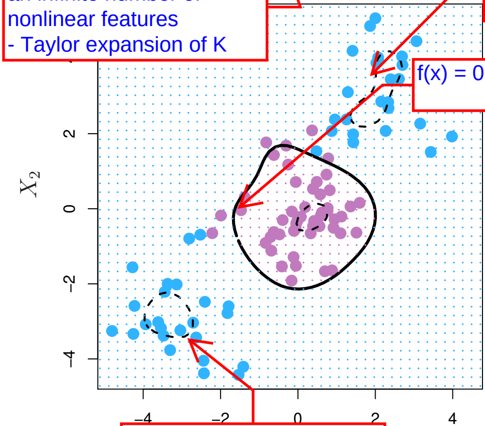
nel

view this as a way of measuring similarities between two features

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

this kernel corresponds to an infinite number of nonlinear features
- Taylor expansion of K

f(x) = +M



f(x) = 0

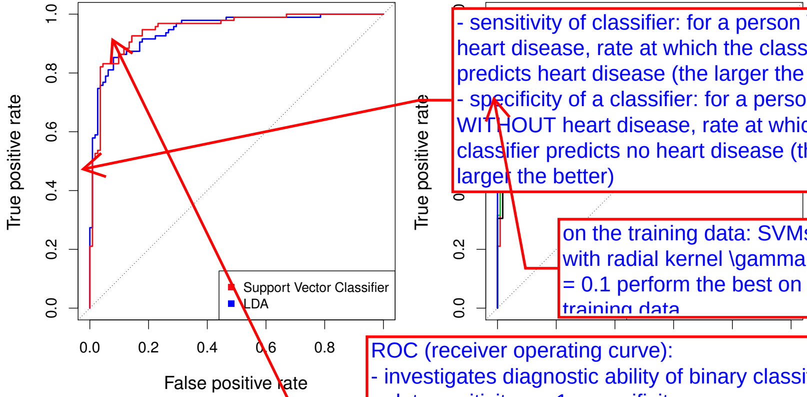
$$\sum_{i\in\mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

Controls variance by squashing down most dimensions severely

f(x) = -M

Example: Heart Data



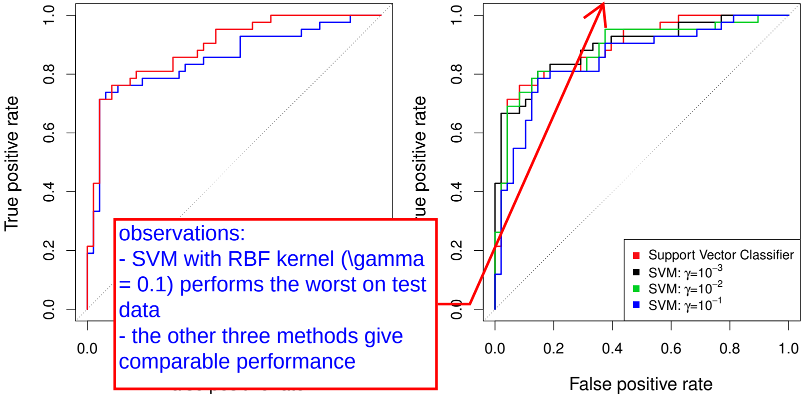
- sensitivity of classifier: for a person WITH heart disease, rate at which the classifier predicts heart disease (the larger the better)
- specificity of a classifier: for a person WITHOUT heart disease, rate at which the classifier predicts no heart disease (the larger the better)

on the training data: SVMs with radial kernel \gamma = 0.1 perform the best on training data

ROC curve is obtained by ch
t in $\hat{f}(X) > t$, and recording
rates as t varies. Here we se

ROC (receiver operating curve):
- investigates diagnostic ability of binary classifier
- plot sensitivity vs. 1 - specificity
- ROC curve is obtained by varying the discrimination threshold T in the classifier f(x) = T
    - T = 0: original classifier (from SVM)
    - T very large: all points are classified as 0 (perfect sensitivity)
    - T very negative: all points classified as 1 (perfect specificity)

Example continued: Heart Test Data



observations:
- SVM with RBF kernel (\gamma = 0.1) performs the worst on test data
- the other three methods give comparable performance

SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x),\ k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

SVMs: more than 2 classes?

- for each category k, we fit SVM classifiers with category k coded as +1, all others coded as -1
- recall that |\hat{f}_k(x)| quantifies the distance from the point x to the classification hyperplane
- a larger value of this then suggests greater "confidence" that x belongs in the k-th class

The SVM as defined w … do if we have $K > 2$ classe…

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

- for each pair of categories (k,l), fit SVM classifiers with class k coded as +1, class l coded as -1
- pick the most frequently assigned class

SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

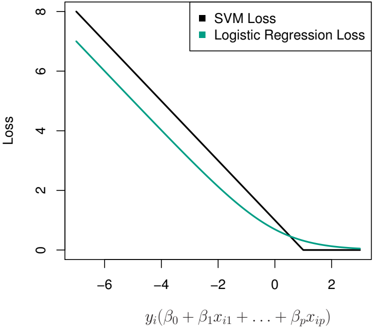
OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

Support Vector versus Logistic Regression?

With $f(X) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$ can rephrase support-vector classifier optimization as

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max\left[0, 1 - y_i f(x_i)\right] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$



This has the form *loss plus penalty*.

The loss is known as the *hinge loss*.

Very similar to "loss" in logistic regression (negative log-likelihood).

Which to use: SVM or Logistic Regression

In general, SVMs vs LR:
- SVMs marginally better for prediction accuracy
- LR: yields a probabilistic classifier which gives a measure of predictive (classification) uncertainty
    - should be considered when such uncertainty is important in the application

- When classes are (LR. So does LDA.
- When not, LR (with ridge penalty) and SVM very similar.
- If you wish to estimate probabilities, LR is the choice.
- For nonlinear boundaries, kernel SVMs are popular. Can use kernels with LR and LDA as well, but computations are more expensive.