Lecture 18: Support Vector Machines

BIOS635

03/26/2020

Some slides taken/modified from other people's lectures online.

Kernel regression/classification algorithm

Convert from a distance d(·,·) to a Gaussian kernel K(·,·)

$$K(\mathbf{x}, \mathbf{x}_i) = \exp\{\frac{-d^2(\mathbf{x},\mathbf{x}_i)}{\sigma^2}\}$$

Given training data $D = \{\mathbf{x}_i, y_i\}$, Kernel function $K(\cdot, \cdot)$ and input $\mathbf{x}$

- (regression) if $y \in \mathbf{R}$, return weighted average:

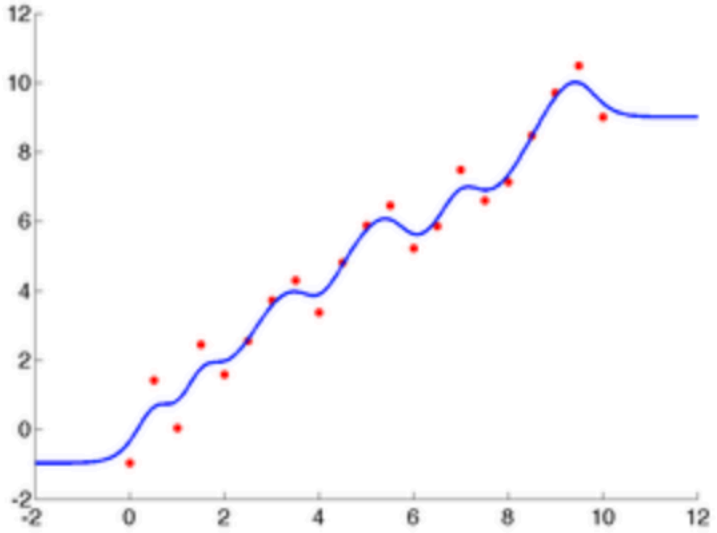
$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i)}$$

- (classification) if $y \in \pm 1$, return weighted majority:

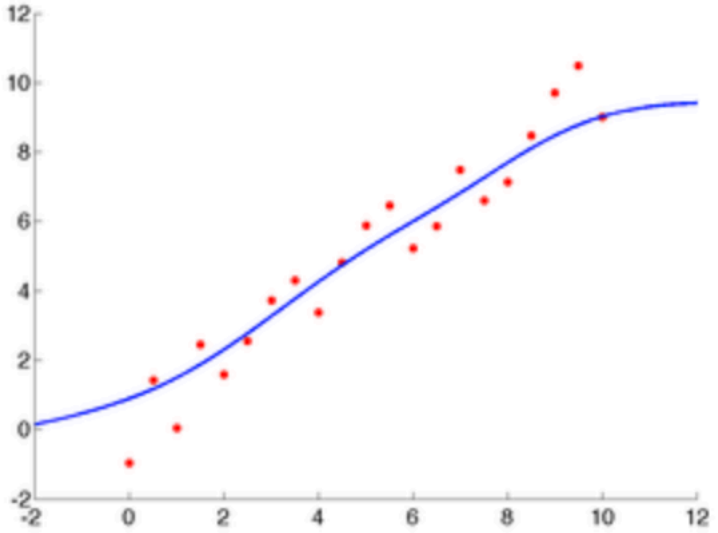
$$\hat{y}(\mathbf{x}) = sign(\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) y_i)$$

Kernel regression: example

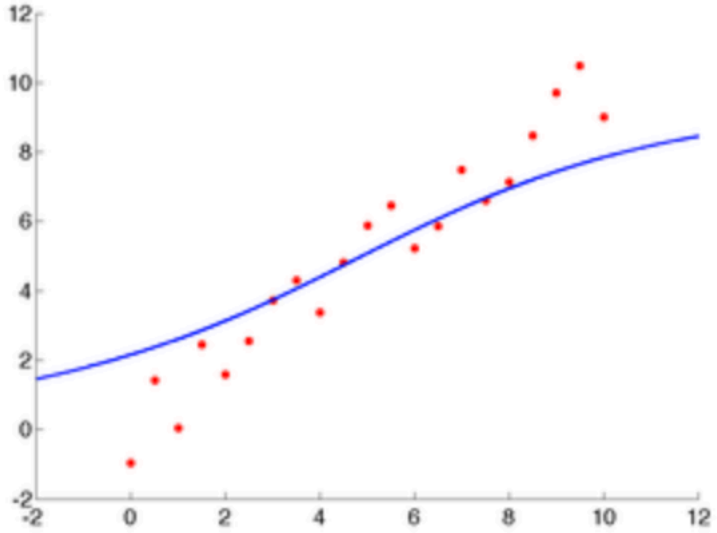
Gaussian kernel, σ = 0.5



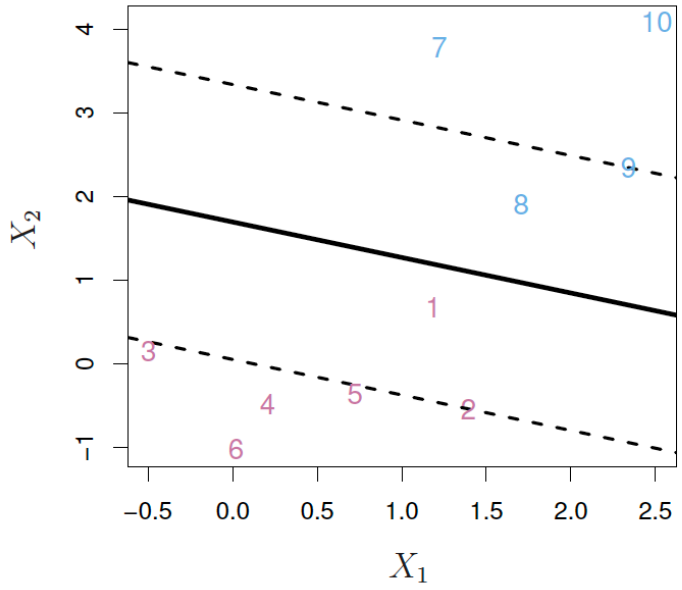
Gaussian kernel, σ = 2

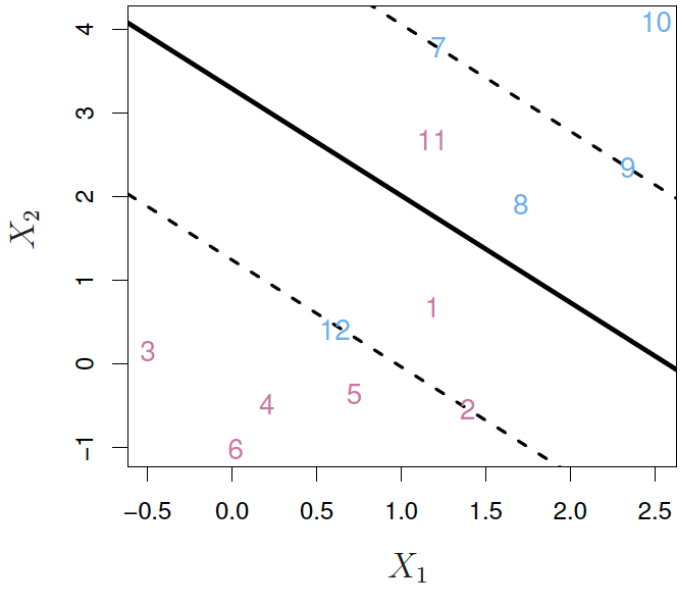


Gaussian kernel, σ = 4



Support vector classifier



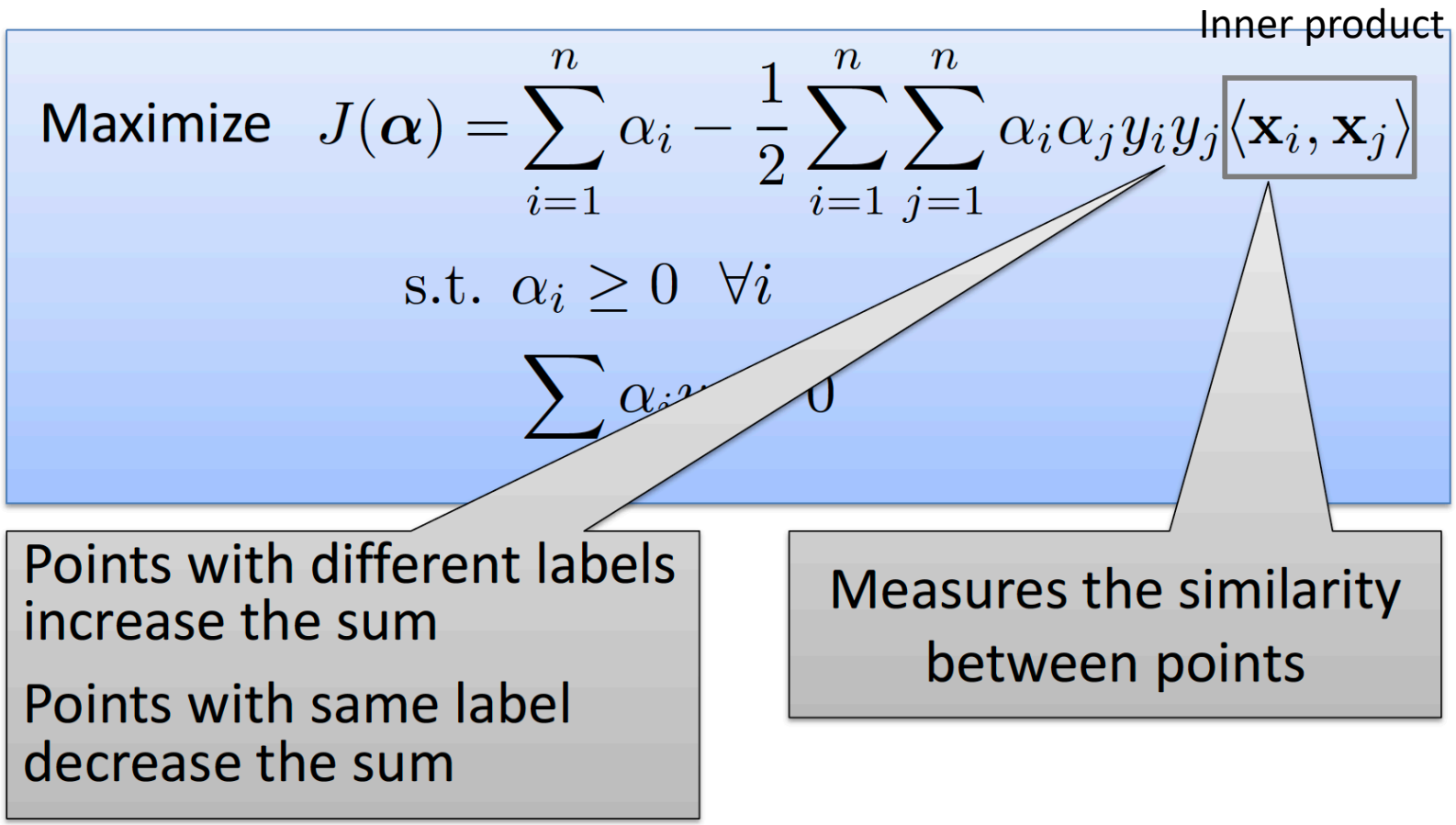


$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \; M \quad \text{subject to} \; \sum_{j=1}^{p} \beta_j^2 = 1,$$

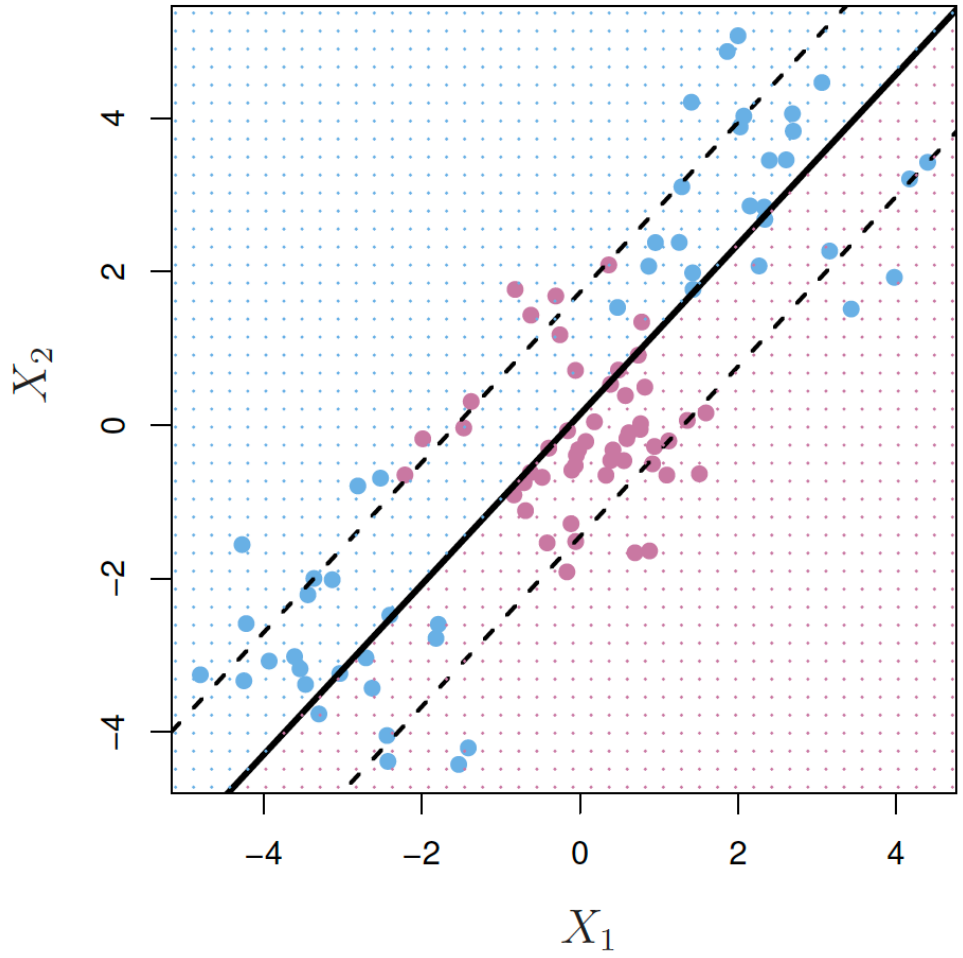
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

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

Dual representation



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.

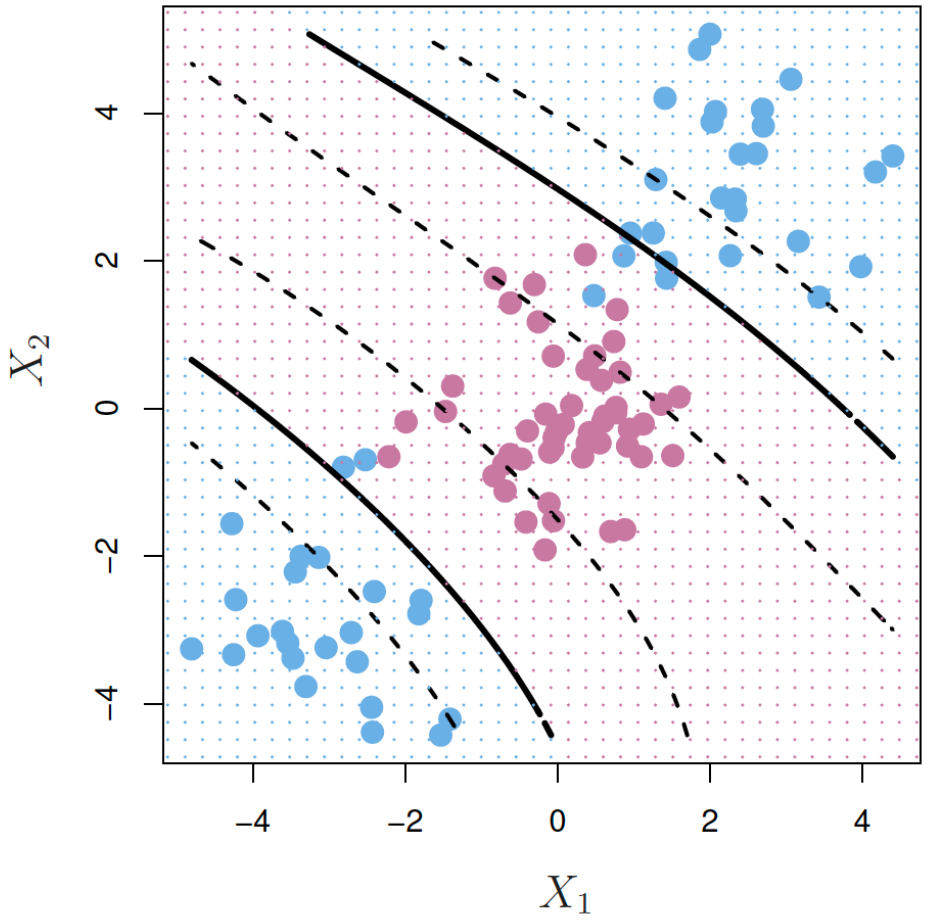
What to do?

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

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of *kernels*.

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

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

$f(x)$ is a linear function of x: correspondence between α_i and β_j.

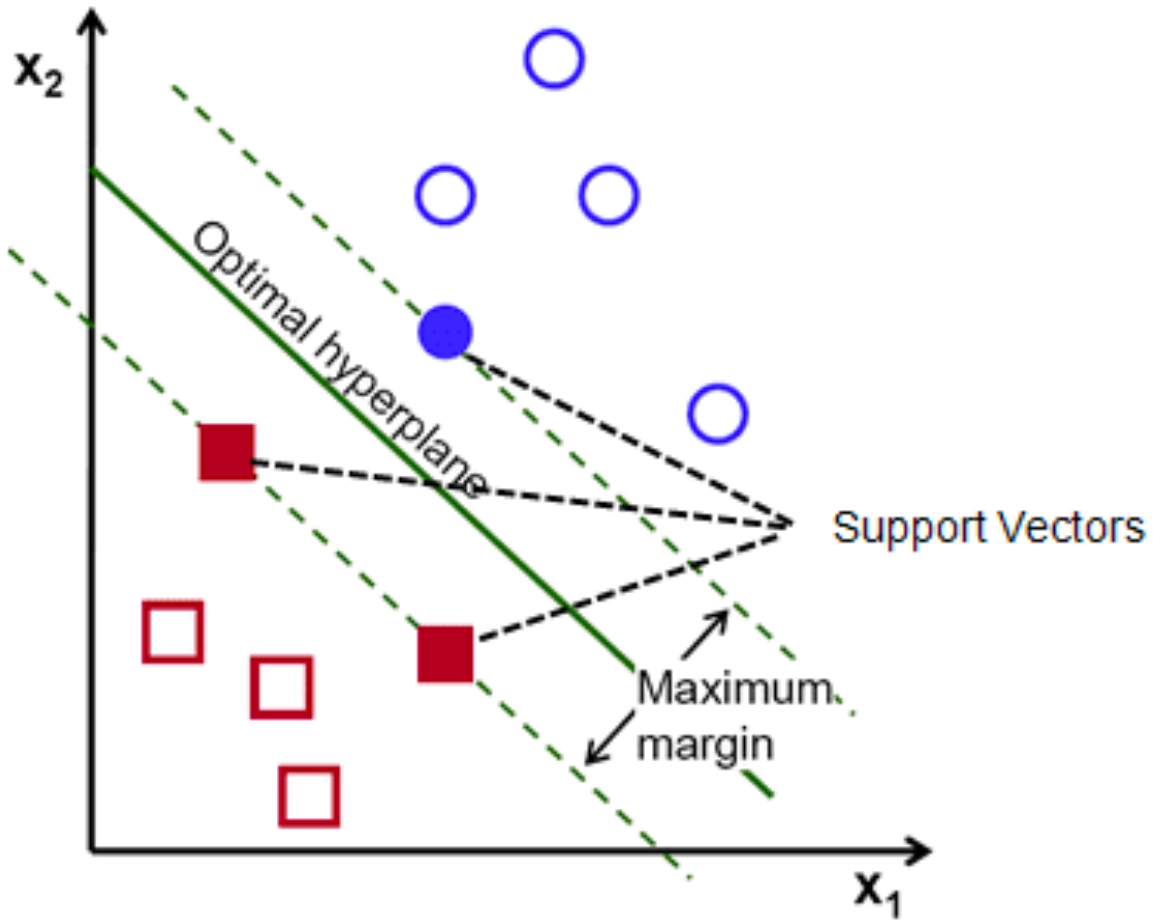
- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

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 $\hat{\alpha}_i > 0$.

Support vectors



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.

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

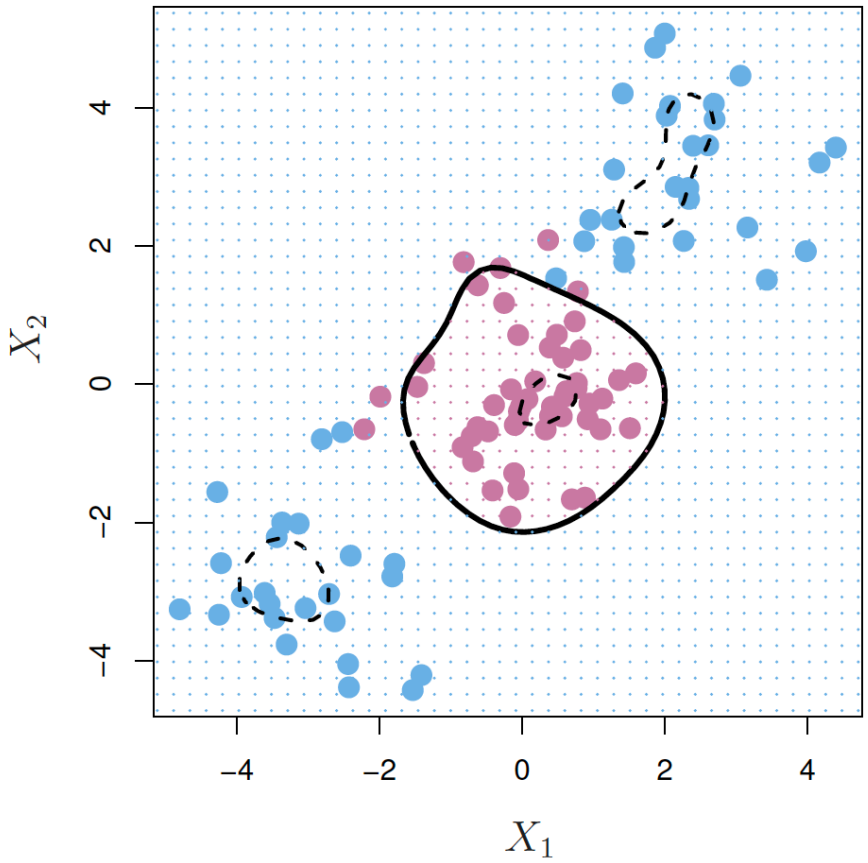
computes the inner-products needed for d dimensional polynomials

- The solution has the form

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

Radial kernel

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

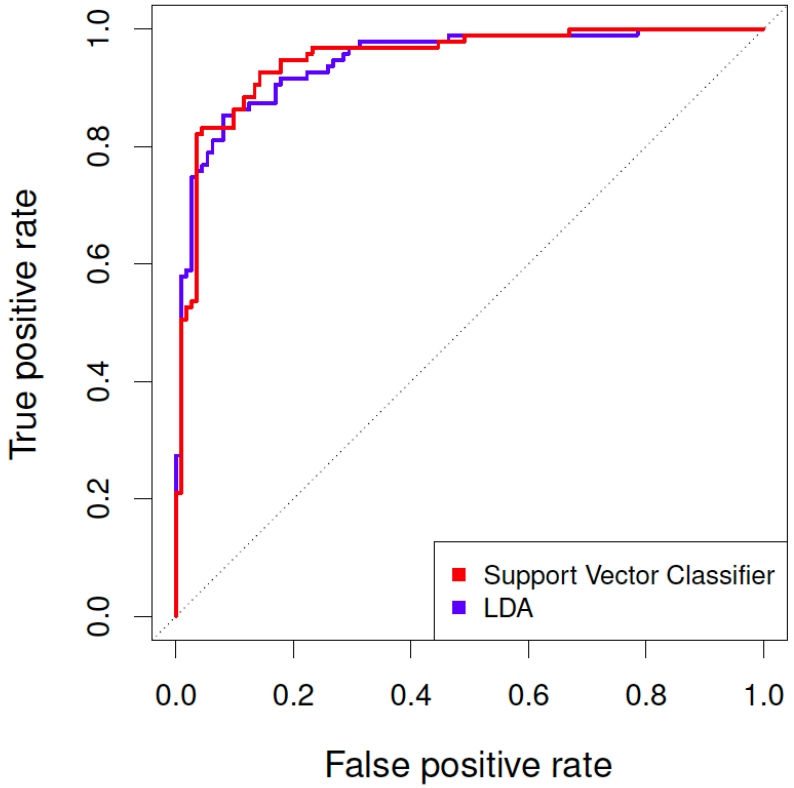


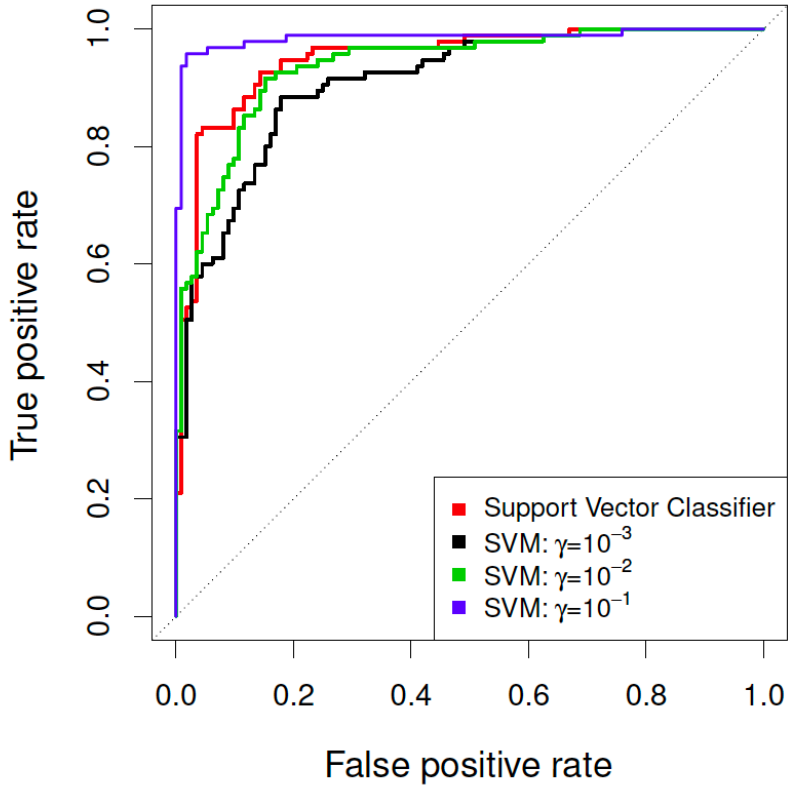
$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

Controls variance by squashing down most dimensions severely

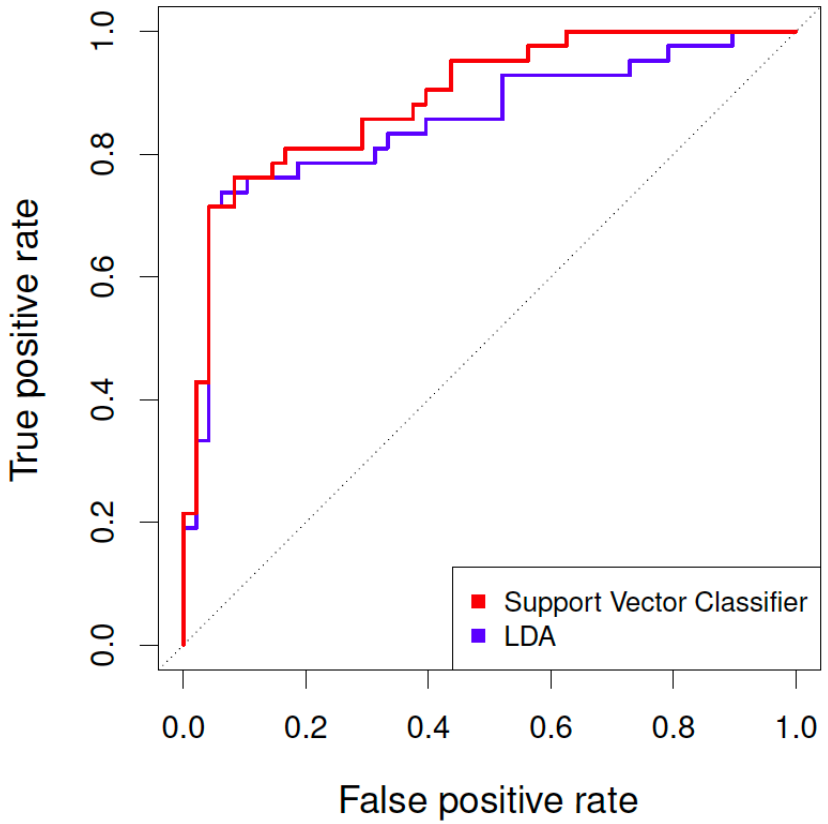
Example: heart training data

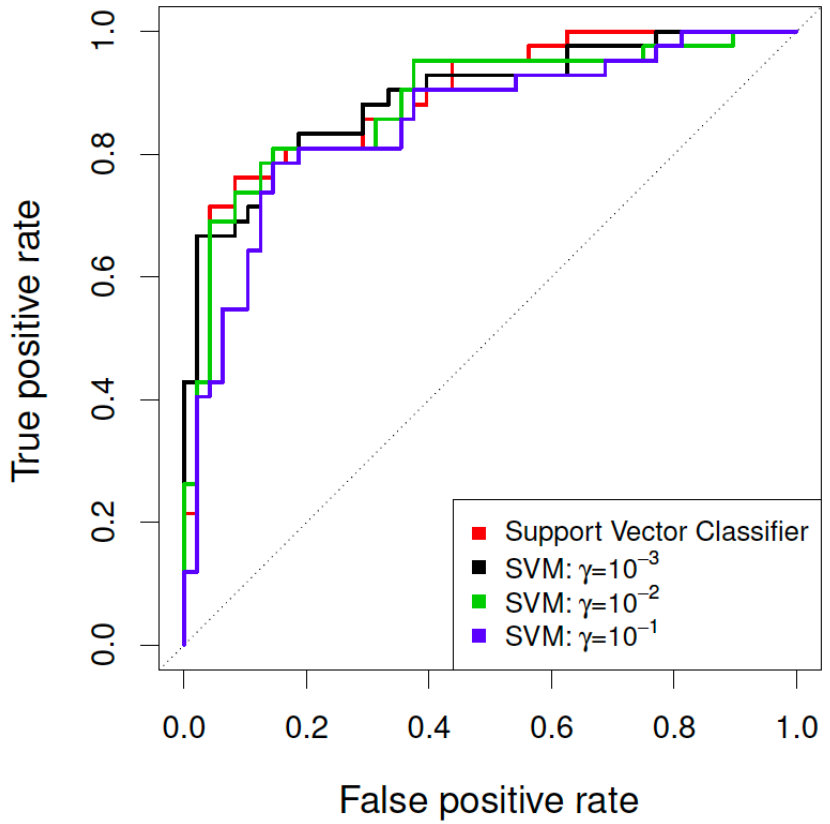




ROC curve is obtained by changing the threshold 0 to threshold t in $\hat{f}(X) > t$, and recording *false positive* and *true positive* rates as t varies. Here we see ROC curves on training data.

Example: heart test data





SVM with more than two 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.

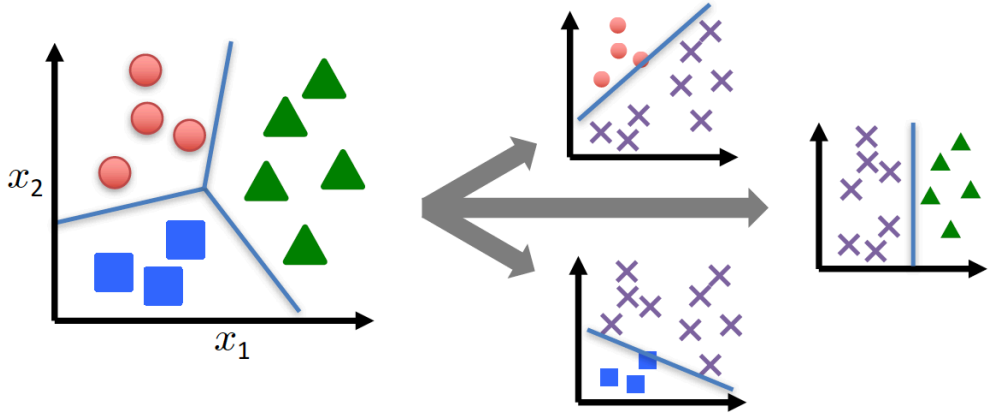
Note: Many SVM packages already have multi-class classification built in.

Logistic regression with more than two classes

So far we have discussed logistic regression with two classes. It is easily generalized to more than two classes. One version (used in the R package `glmnet`) has the symmetric form

$$\Pr(Y = k|X) = \frac{e^{\beta_{0k}+\beta_{1k}X_1+...+\beta_{pk}X_p}}{\sum_{\ell=1}^{K} e^{\beta_{0\ell}+\beta_{1\ell}X_1+...+\beta_{p\ell}X_p}}$$

Here there is a linear function for *each* class.

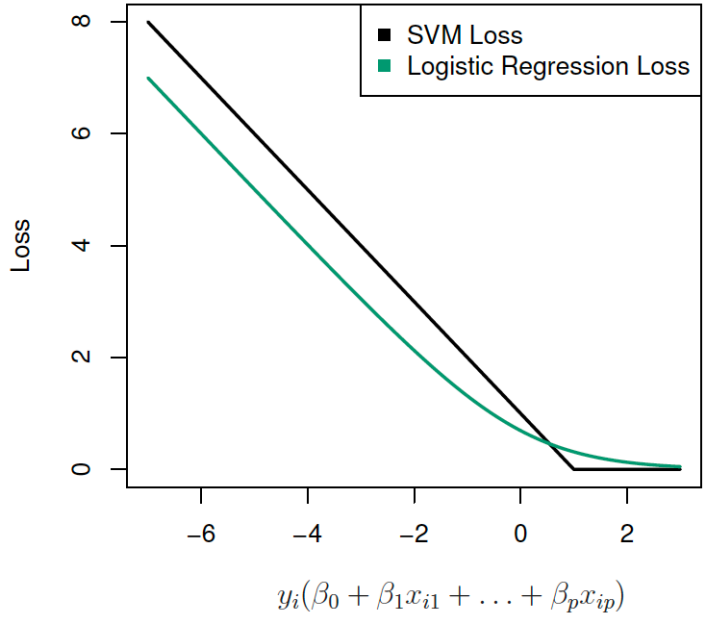


Multiclass logistic regression is also referred to as *multinomial regression*.

SVM versus logistic regression

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

$$\min_{\beta} \left\{ \sum_i \max[0, 1 - y_i \beta^T x_i] + \lambda \|\beta\|^2 \right\}$$



This has the form *loss plus penalty*.
The loss is known as the *hinge loss*.
Very similar to "loss" in logistic regression

$$\min_{\beta} \left\{ \sum_i \log[1 + \exp(1 - y_i \beta^T x_i)] + \lambda \|\beta\|^2 \right\}$$

Which to use: SVM or logistic regression?

- When classes are (nearly) separable, SVM does better than 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.

SVMs vs Logistic Regression (Advice from Andrew Ng)

If p is large (relative to n) (e.g., p > n with p = 10,000, n = 10-1,000)

- Use logistic regression or SVM with a linear kernel

If p is small (up to 1,000), n is intermediate (up to 10,000)

- Use SVM with Gaussian kernel

If p is small (up to 1,000), n is large (50,000+)

- Create/add more features, then use logistic regression or SVM without a kernel

Neural networks likely to work well for most of these settings, but may be slower to train.

Conclusions

- SVMs find optimal linear separator
- The kernel trick makes SVMs learn non-linear decision surfaces
- Strength of SVMs:
  - Good theoretical and empirical performance
  - Supports many types of kernels
- Disadvantages of SVMs:
  - “Slow” to train/predict for huge data sets (but relatively fast!)
  - Need to choose the kernel (and tune its parameters)