Introduction to Statistical Learning and Machine Learning

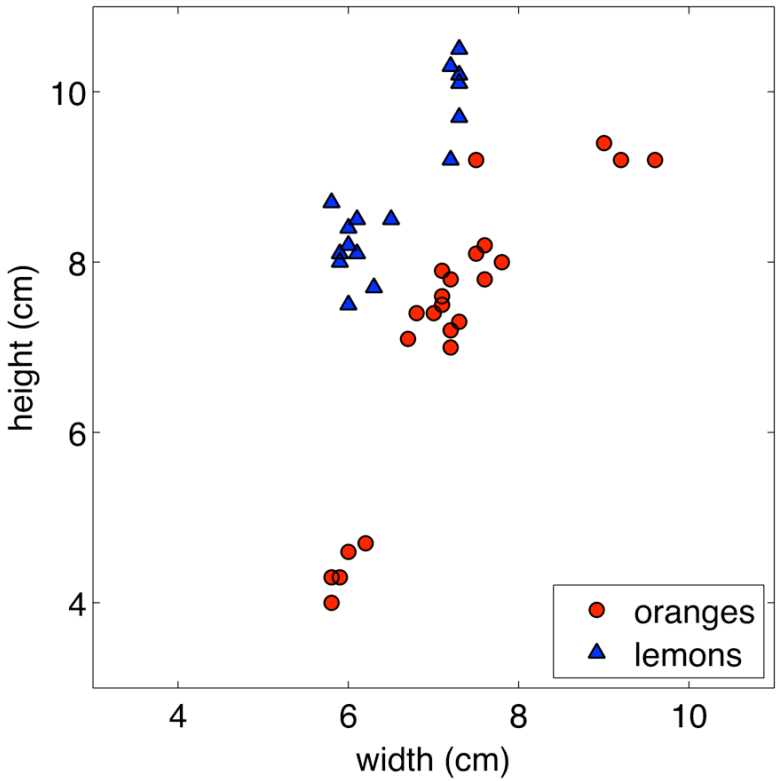
Chap 5 & Chap6 – SVM and Kernel Methods

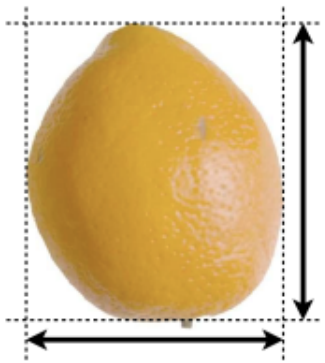
Yanwei Fu

School of Data Science, Fudan University

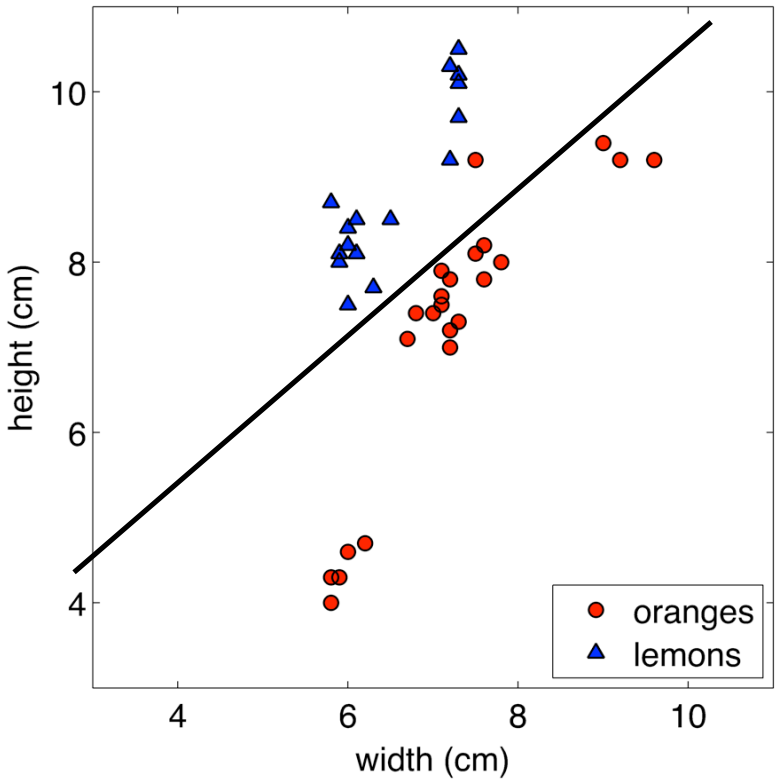
Recap – Nearest Neighbour, Logitic Regression

Classication: Oranges and Lemons



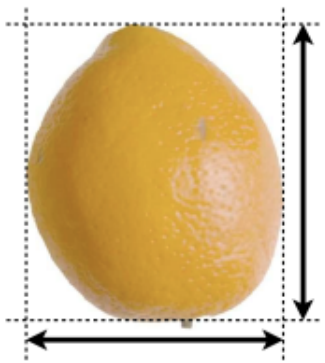


Classication: Oranges and Lemons



Can construct simple linear decision boundary:

$$y = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$$



What is the meaning of "linear" classication

- Classification is intrinsically non-linear
  - It puts non-identical things in the same class, so a difference in the input vector sometimes causes zero change in the answer
- Linear classification means that the part that adapts is linear (just like linear regression)

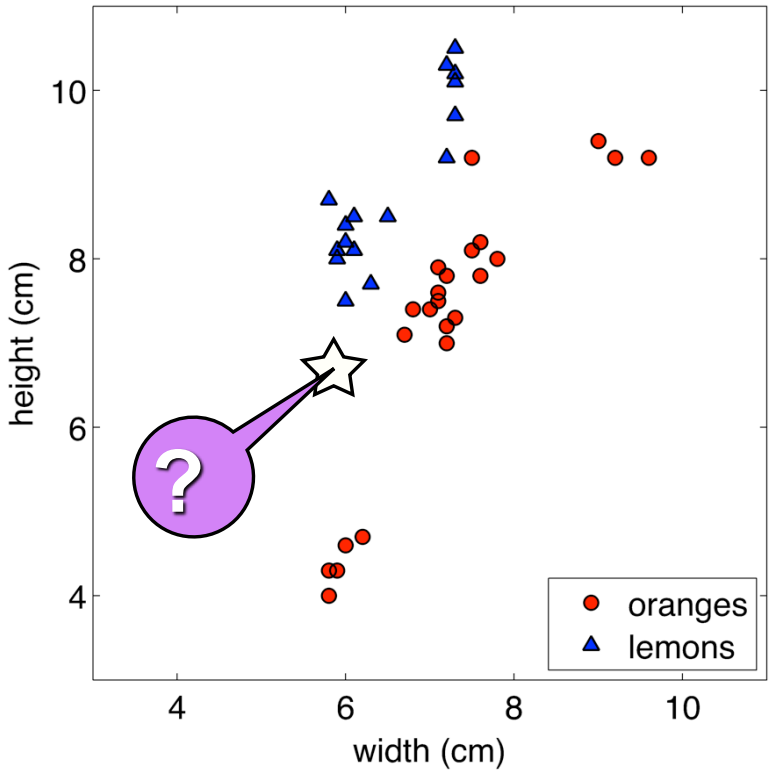
$$z(x) = \mathbf{w}^T\mathbf{x} + w_0$$

  with adaptive $\mathbf{w}, w_0$
- The adaptive part is follow by a non-linearity to make the decision

$$y(\mathbf{x}) = f(z(\mathbf{x}))$$

- What f have we seen so far in class?

Classication as Induction



Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)
- Learning amounts to simply storing training data
- Test instances classified using similar training instances
- Embodies often sensible underlying assumptions:
  - Output varies smoothly with input
  - Data occupies sub-space of high-dimensional input space

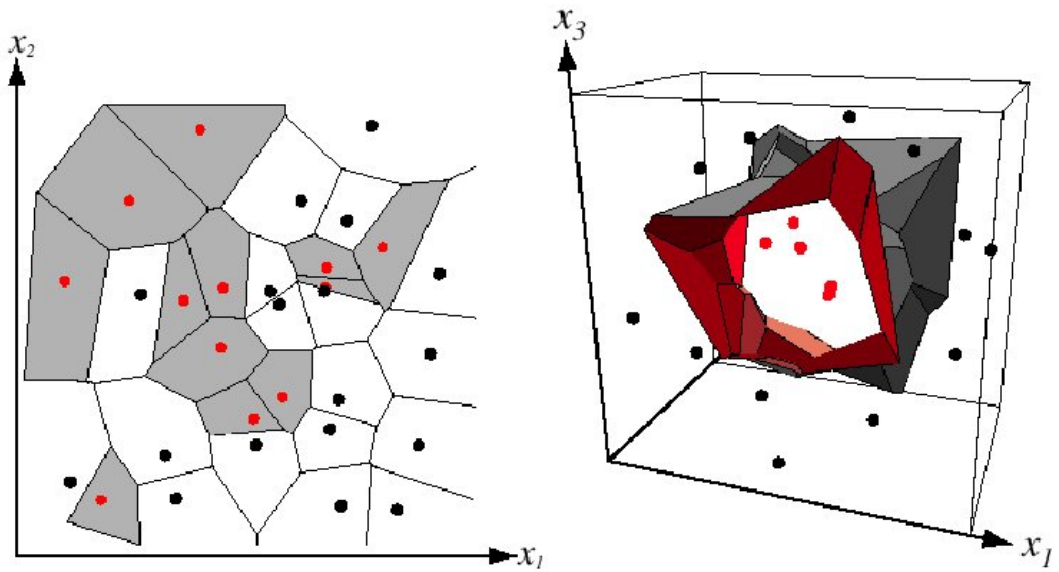
- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)
- Distance typically defined to be Euclidean:

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

- Algorithm
  1. find example $(\mathbf{x}^*, t^*)$ closest to the test instance $\mathbf{x}^{(q)}$
  2. output $y^{(q)} = t^*$
- Note: we don't need to compute the square root. Why?

Nearest Neighbors Decision Boundaries

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred
- Decision boundaries: Voronoi diagram visualization
  - show how input space divided into classes
  - each line segment is equidistant between two points of opposite classes



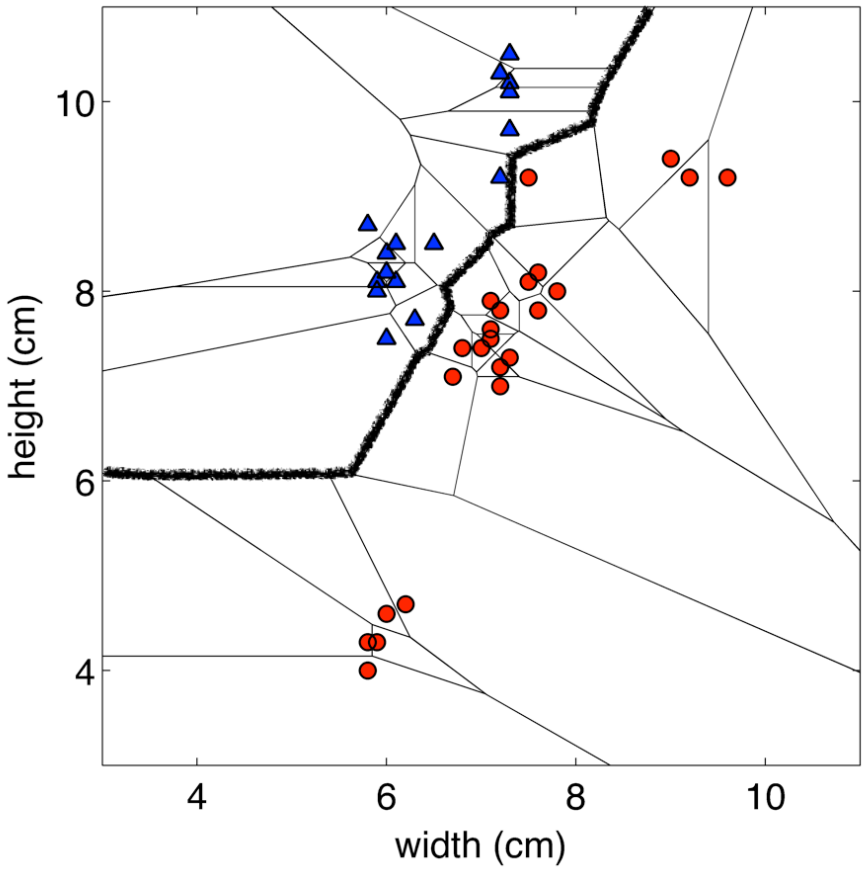
- Nearest neighbors sensitive to mis-labeled data ("class noise") $\rightarrow$ smooth by having k nearest neighbors vote
- Algorithm:
  1. find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
  2. classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$$

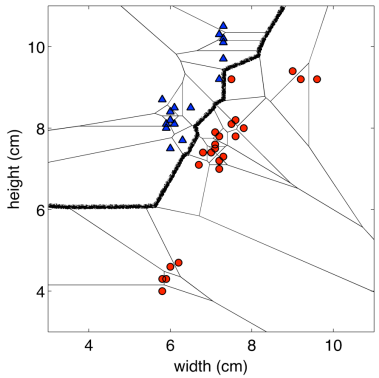
k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important
  - normalize scale
- Irrelevant, correlated attributes add noise to distance measure
  - eliminate some attributes
  - or vary and possibly adapt weight of attributes
- Non-metric attributes (symbols)
  - Hamming distance
- Brute-force approach: calculate Euclidean distance to test point from each stored point, keep closest: $O(dn^2)$. We need to reduce computational burden:
  1. Use subset of dimensions
  2. Use subset of examples
     - Remove examples that lie within Voronoi region
     - Form efficient search tree (kd-tree), use Hashing (LSH), etc

Decision Boundary K-NN



K-NN Summary



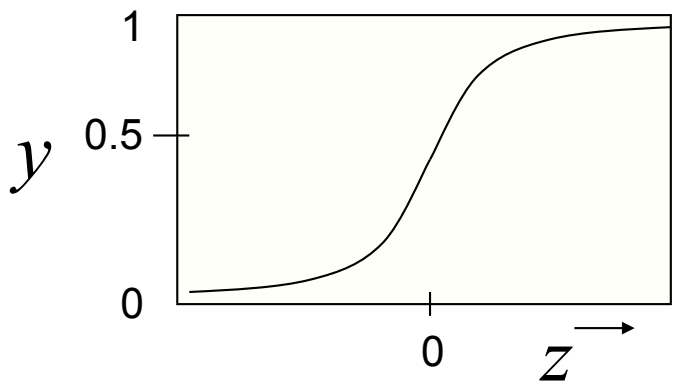
- Single parameter (k) $\rightarrow$ how do we set it?
- Naturally forms complex decision boundaries; adapts to data density
- Problems:
  - Sensitive to class noise.
  - Sensitive to dimensional scales.
  - Distances are less meaningful in high dimensions
  - Scales with number of examples
- Inductive Bias: What kind of decision boundaries do we expect to find?

- An alternative: replace the $sign(\cdot)$ with the sigmoid or logistic function
- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



- The output is a smooth function of the inputs and the weights

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

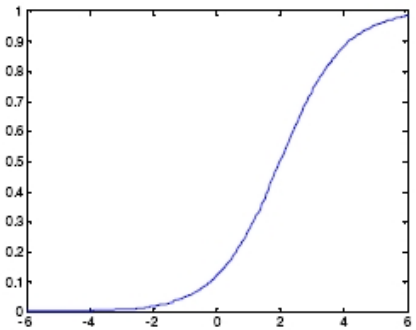
  - One parameter per data dimension (feature)
  - Features can be discrete or continuous
  - Output of the model: value $y \in [0, 1]$
  - This allows for gradient-based learning of the parameters: smoothed version of the $sign(\cdot)$

Shape of Logistic Function

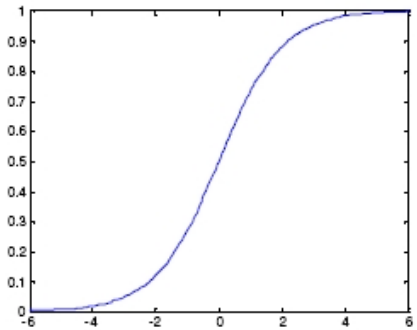
- Let's look at how modifying **w** changes the function shape
- 1D example:

$$y = \sigma(w_1 x + w_0)$$

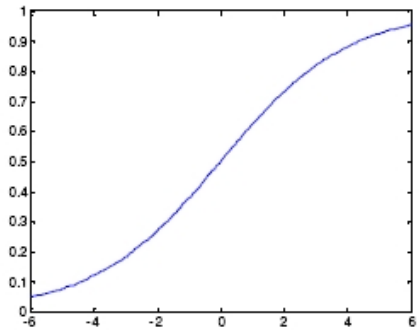
$w_0 = -2, w_1 = 1$



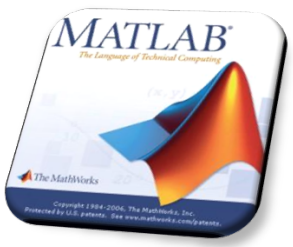
$w_0 = 0, w_1 = 1$



$w_0 = 0, w_1 = 0.5$



- Demo



Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C = 0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Substituting we have

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- Supposed we have two classes, how can I compute $p(C = 1|\mathbf{x})$?
- Use the marginalization property of probability

$$p(C = 1|\mathbf{x}) + p(C = 0|\mathbf{x}) = 1$$

- Thus (show matlab)

$$p(C = 1|\mathbf{x}) = 1 - \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

Conditional likelihood(MLE)

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$\begin{aligned} p(t^{(i)} | \mathbf{x}^{(i)}) &= p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \\ &= \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \end{aligned}$$

- We might want to learn the model, by maximizing the conditional likelihood

$$\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- Convert this into a minimization so that we can write the loss function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

- Why is this equivalent to maximize the conditional likelihood?
- Is there a closed form solution?
- It's a convex function of $\mathbf{w}$. Can we get the global optimum?

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

  with $\{t\} = (t^{(1)}, \cdots, t^{(N)})$, and $\{\mathbf{x}\} = (\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)})$
- We can define priors on parameters $\mathbf{w}$
- This is a form of regularization
- Helps avoid large weights and overfitting

$$\max_{\mathbf{w}} \log \left[p(\mathbf{w}) \prod_i p(t^{(i)}|\mathbf{x}^{(i)},\mathbf{w}) \right]$$

- What's $p(\mathbf{w})$?

Regularized Logistic Regression

- For example, define prior: normal distribution, zero mean and identity covariance $p(\mathbf{w}) = \mathcal{N}(0, \alpha\mathbf{I})$
- This prior pushes parameters towards zero
- Including this prior the new gradient is

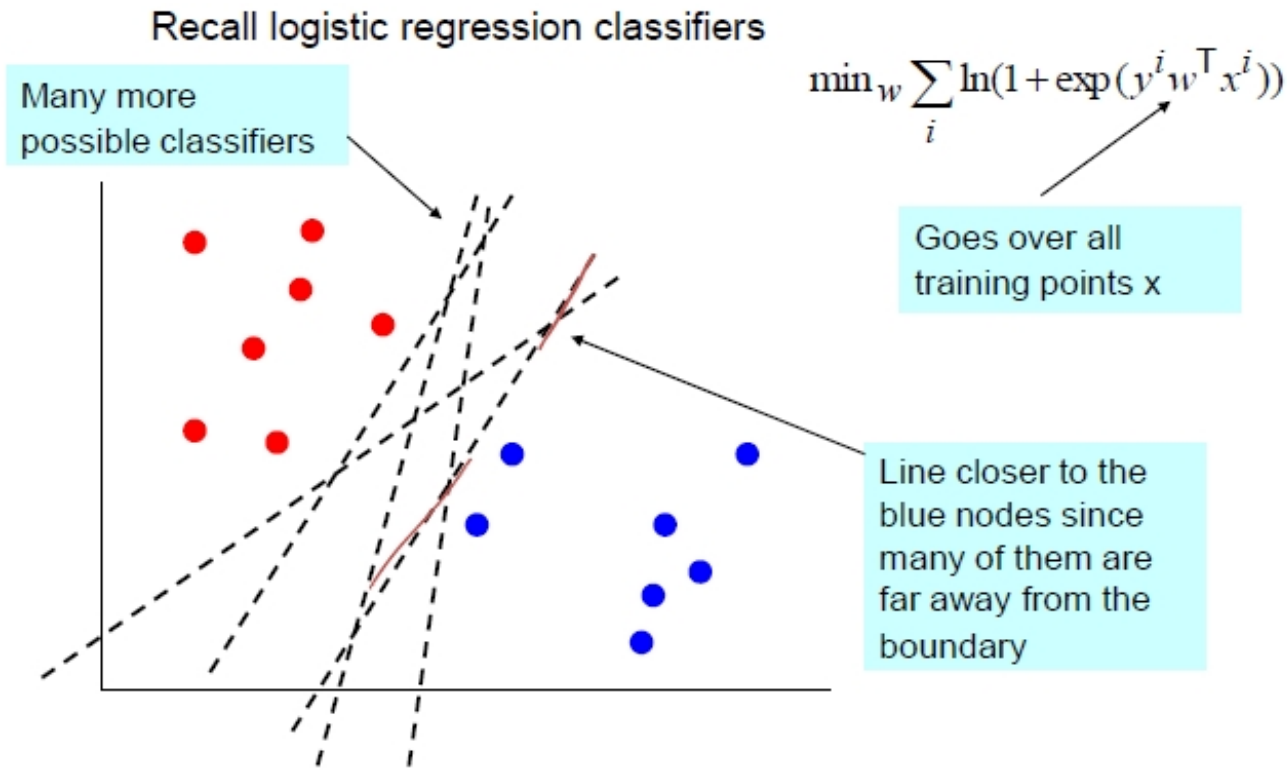
$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \lambda \frac{\partial \ell(\mathbf{w})}{\partial w_j} - \lambda \alpha w_j^{(t)}$$

  where t here refers to iteration of the gradient descent
- How do we decide the best value of α?

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation
- Use the first set (i.e., training) to estimate the weights **w** for different values of α
- Use the second set (i.e., validation) to estimate the best α, by evaluating how well the classifier does in this second set
- This test how well you generalized to unseen data
- The parameter α is the importance of the regularization, and it's a hyper-parameter

Support Vector Machines



$$y = \begin{cases} 1 & \text{if } (\mathbf{w}^T\mathbf{x} + b) \geq 0 \\ -1 & \text{if } (\mathbf{w}^T\mathbf{x} + b) < 0 \end{cases}$$

Here we approach the two-class classification problem 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.

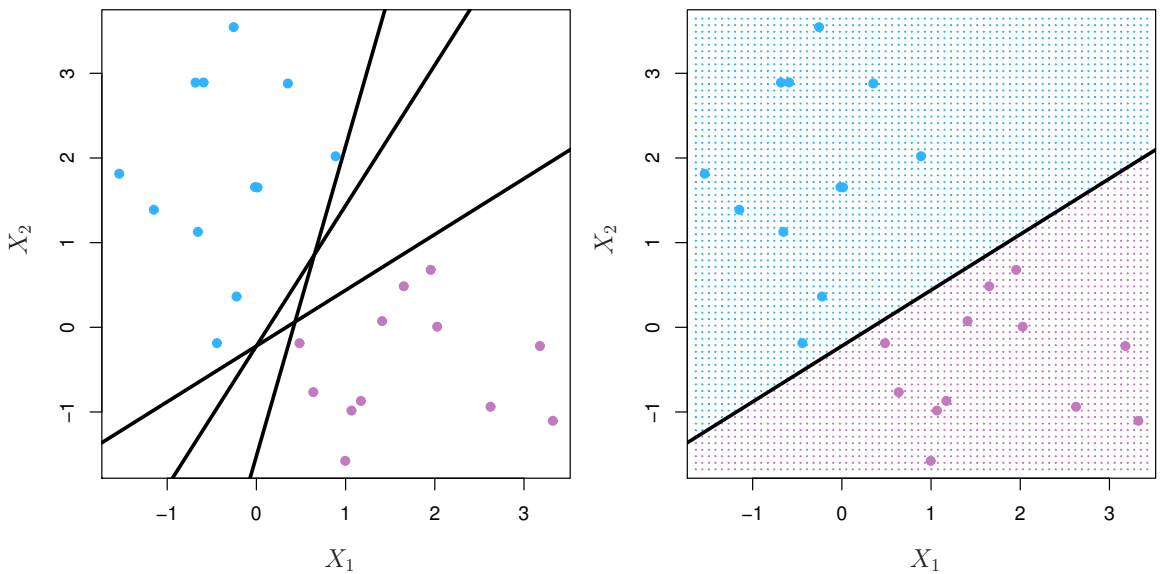
What is a Hyperplane?

- 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.

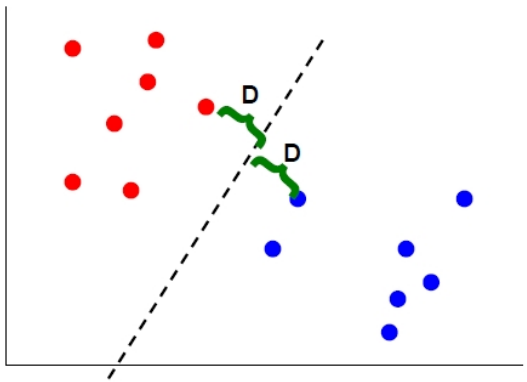
Separating Hyperplanes



- 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 Classification

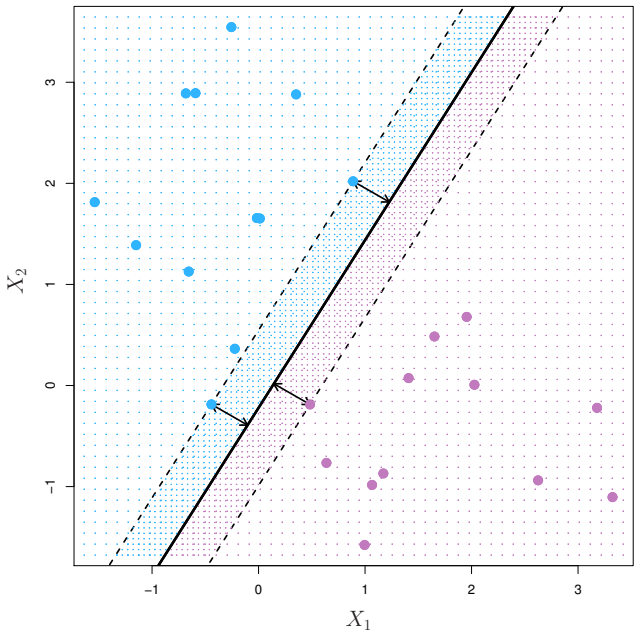
- Instead of fitting all the points, focus on boundary points
- Aim: learn a boundary that leads to the largest margin (buffer) from points on both sides



- Why: intuition; theoretical support; and works well in practice
- Subset of vectors that support (determine boundary) are called the support vectors

Maximal Margin Classier

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



Constrained optimization problem

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

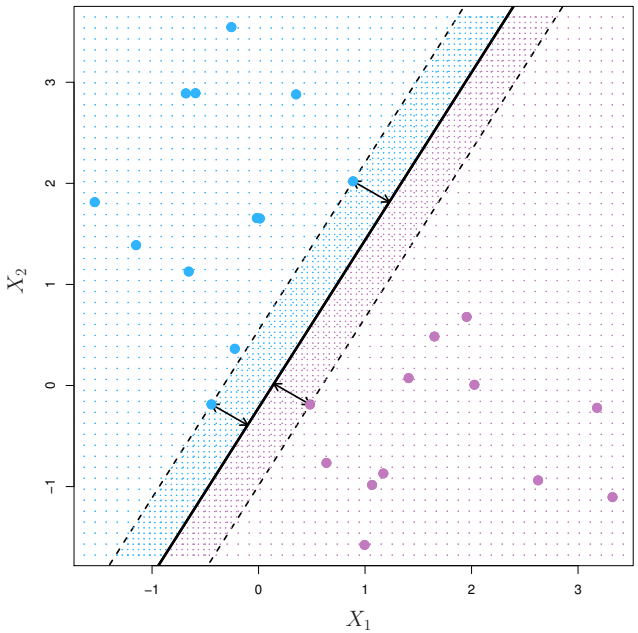
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

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

$$\text{for all} \quad i = 1, \ldots, N.$$

Maximal Margin Classier

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



Constrained optimization problem

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

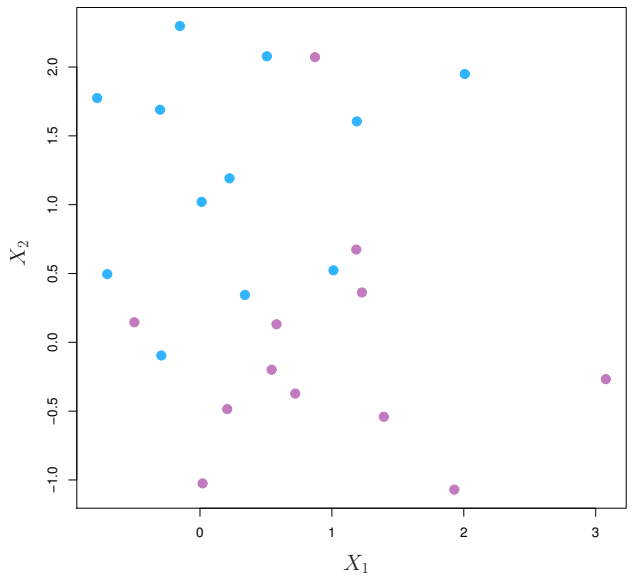
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

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

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

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

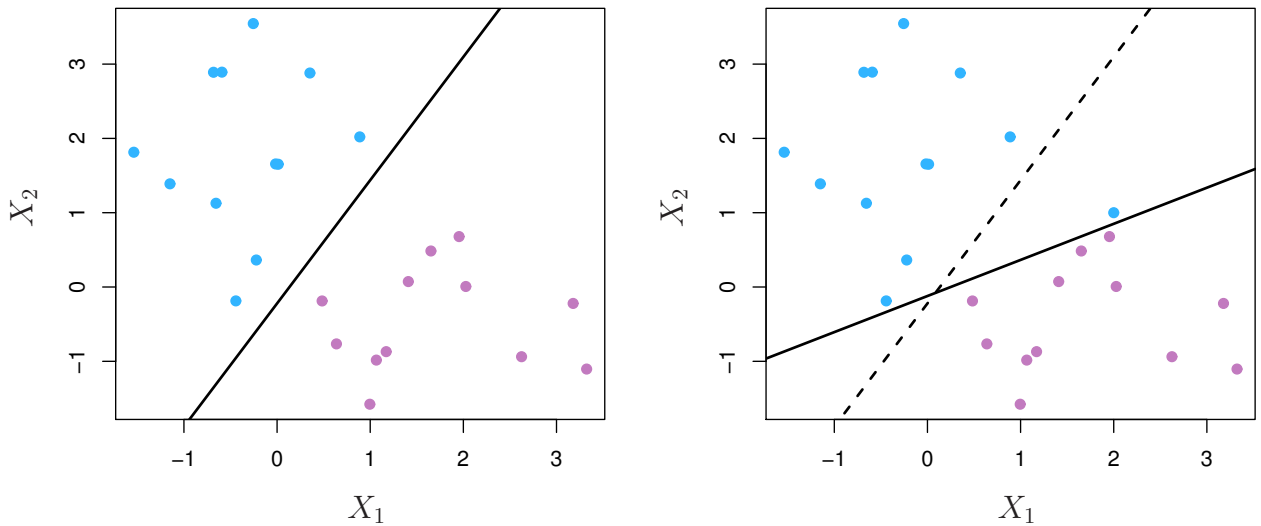
Non-separable Data



The data on the left are not separable by a linear boundary.

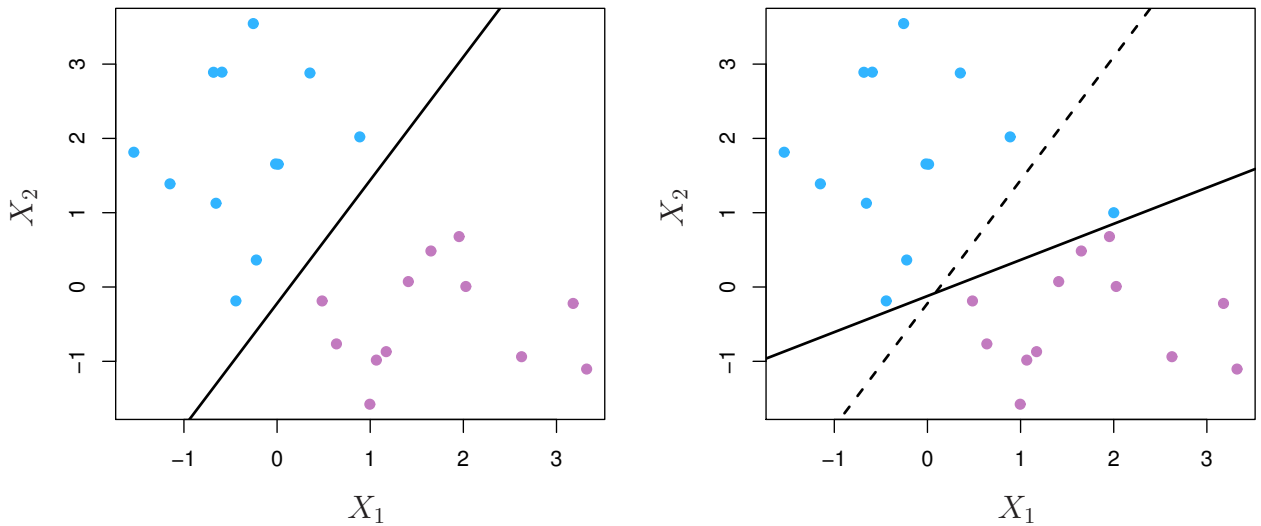
This is often the case, unless $N < p$.

Noisy Data



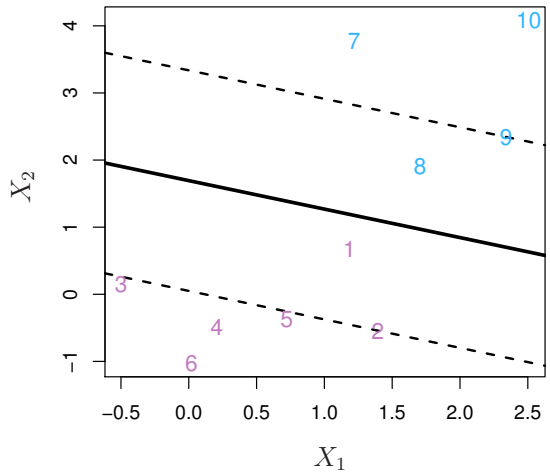
Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

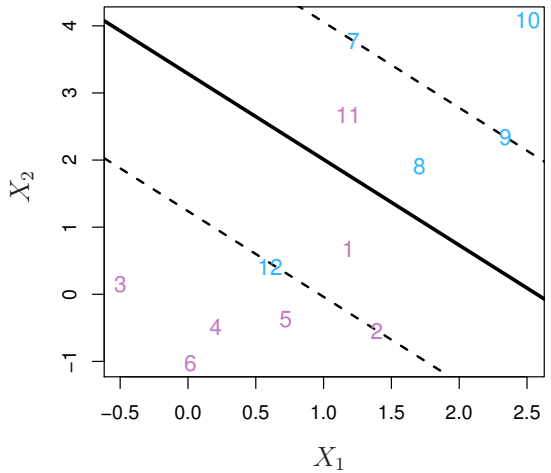
Noisy Data



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.



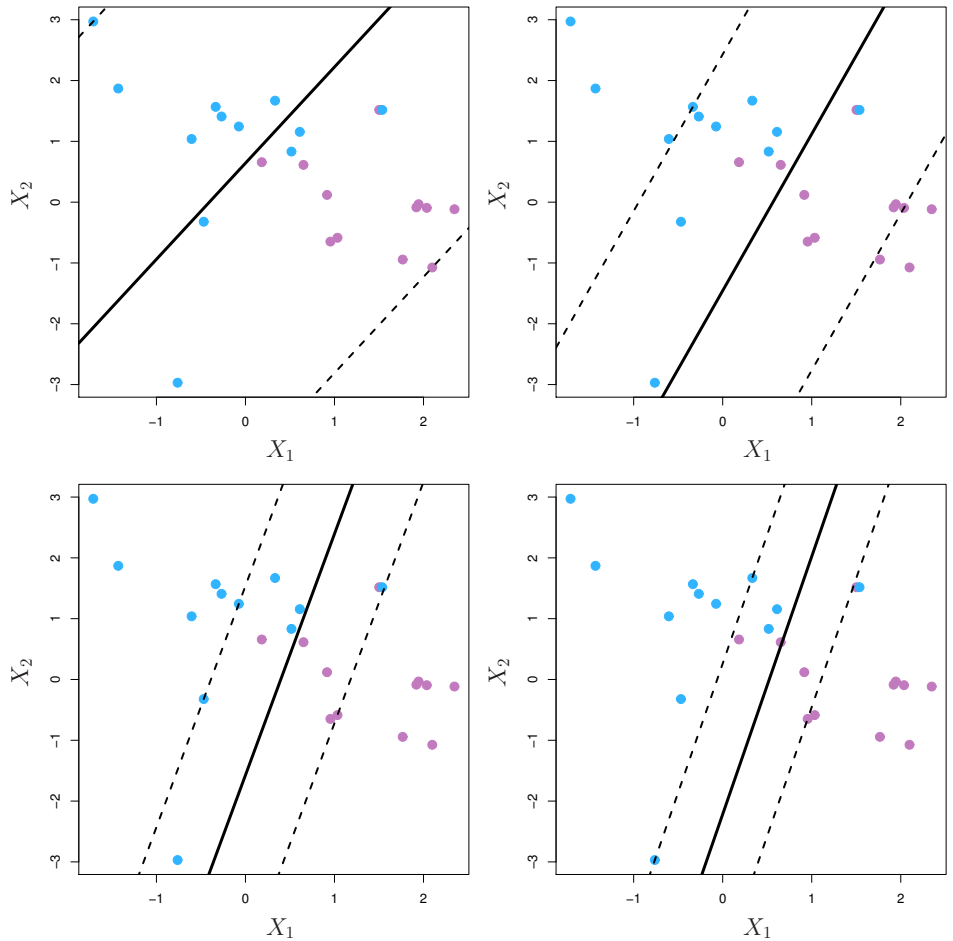


$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

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

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

C is a regularization parameter



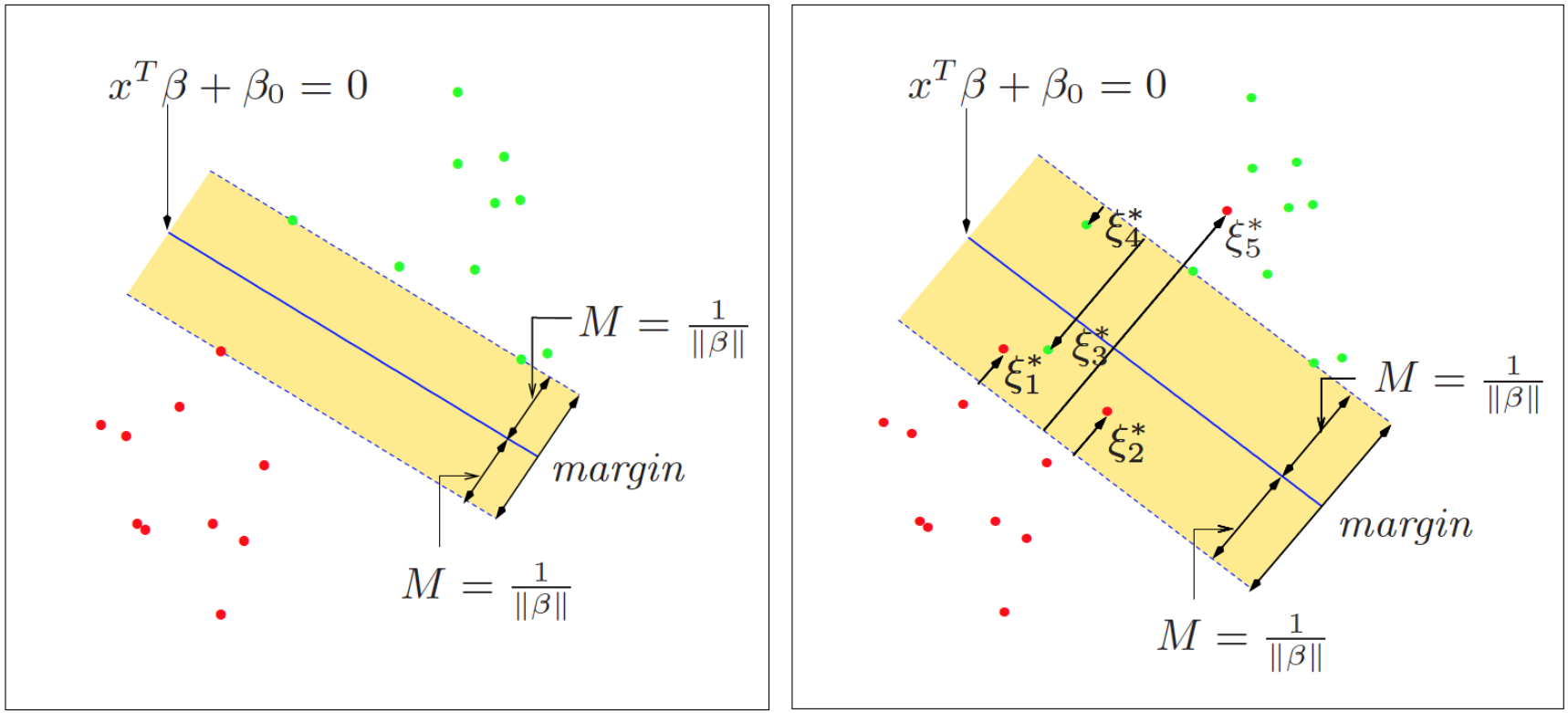
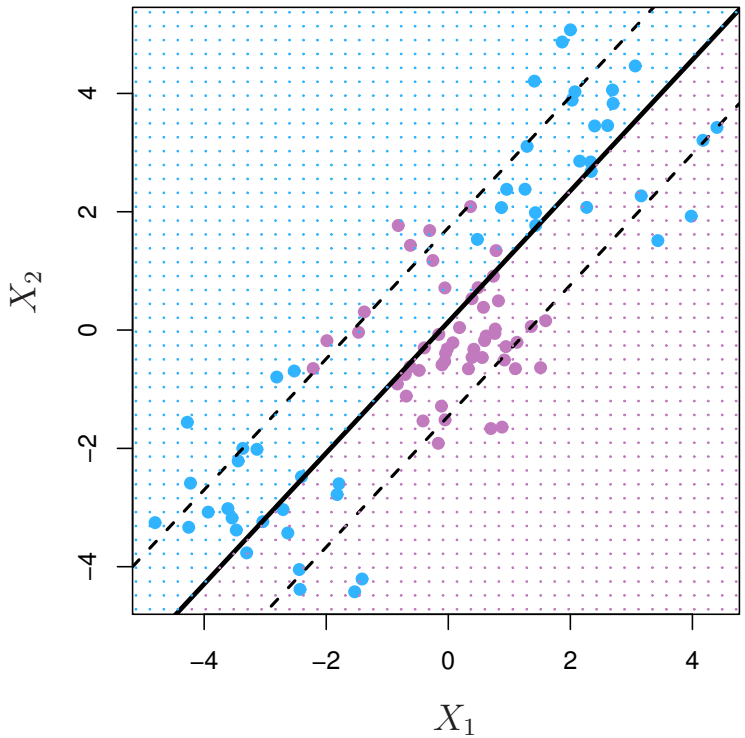


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$. *The right panel shows the nonseparable (overlap) case. The points labeled* ξ_j^* *are on the wrong side of their margin by an amount* $\xi_j^* = M\xi_j$; *points on the correct side have* $\xi_j^* = 0$. *The margin is maximized subject to a total budget* $\sum \xi_i \leq$ *constant. Hence* $\sum \xi_j^*$ *is the total distance of points on the wrong side of their margin.*

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.

- 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

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

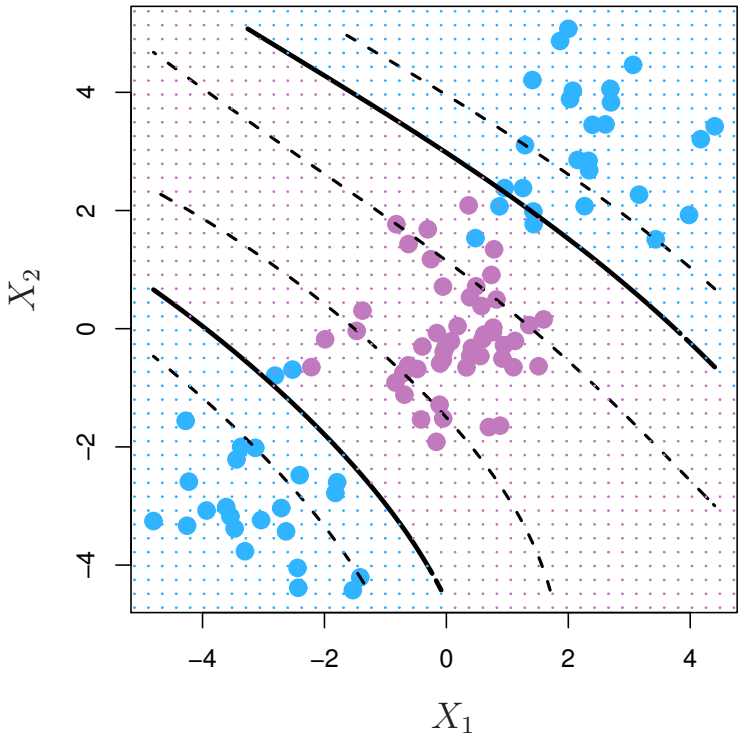
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

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

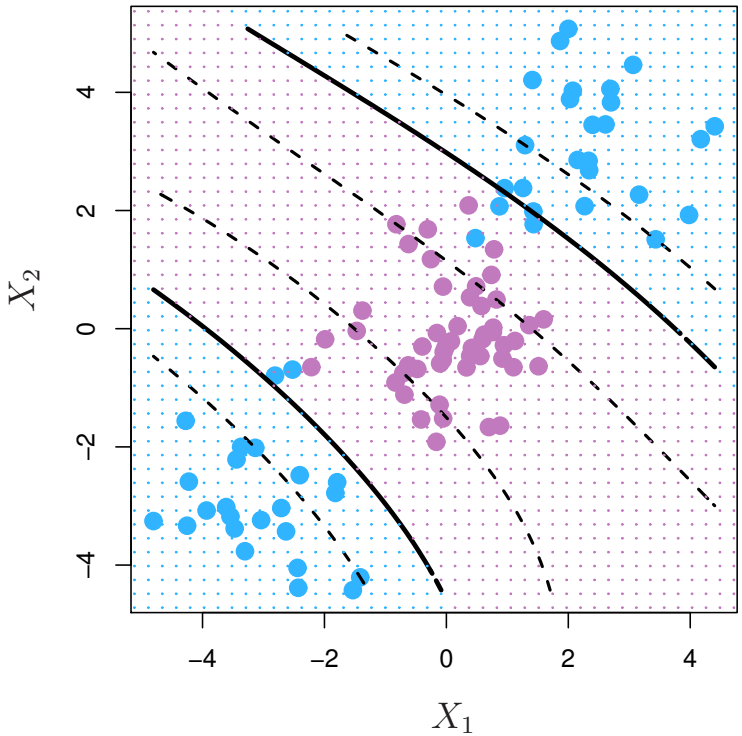


Cubic Polynomials

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



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

- 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*.
- 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

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

- The linear support vector classifier can be represented as

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

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

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

- 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

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

- The linear support vector classifier can be represented as

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

- 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]

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

- 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!

- 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 $d = 2$.

- 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.

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

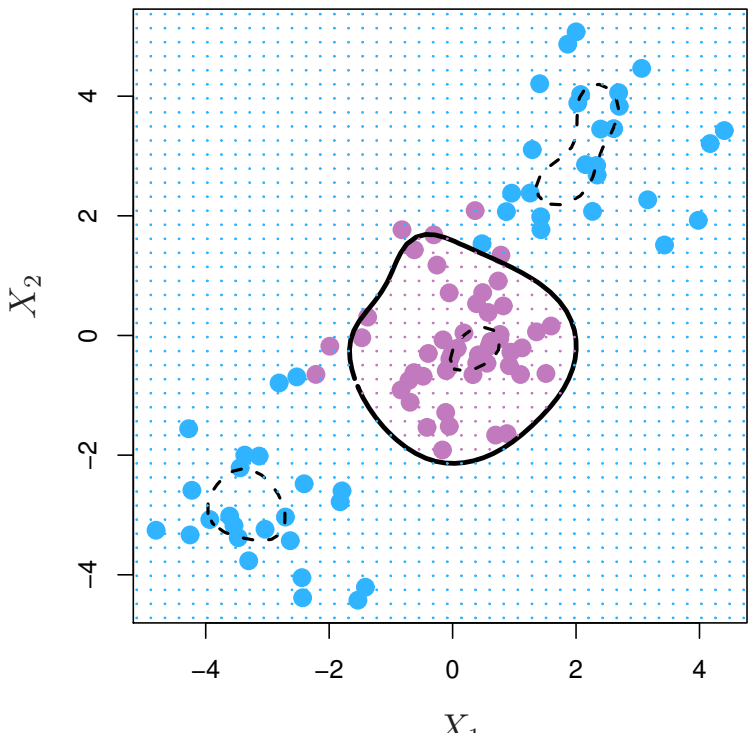
computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

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

- The solution has the form

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

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

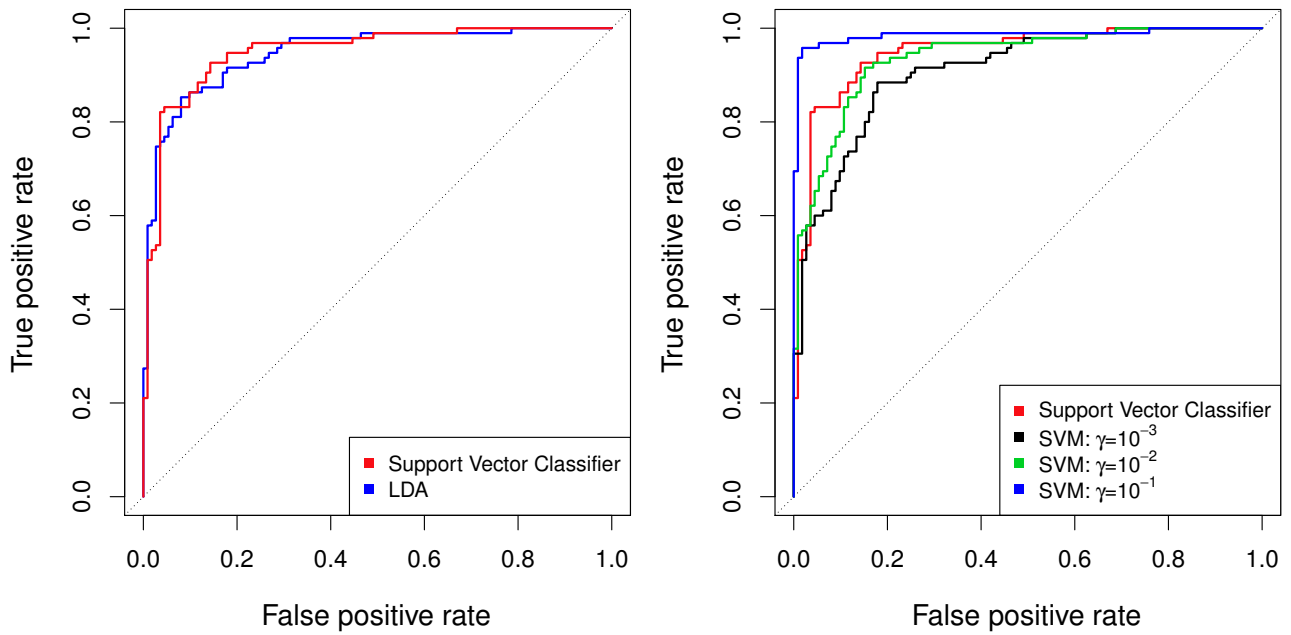


$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

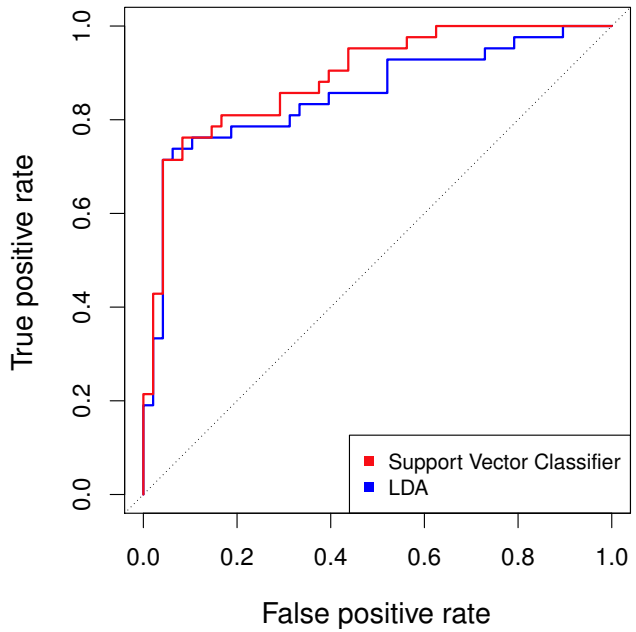
Controls variance by squashing down most dimensions severely

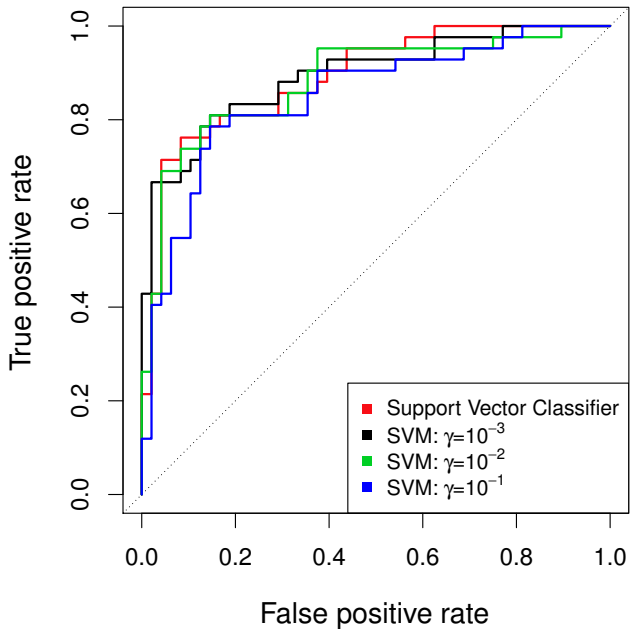
Example: Heart Data



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 continued: Heart Test Data





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?

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.

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.

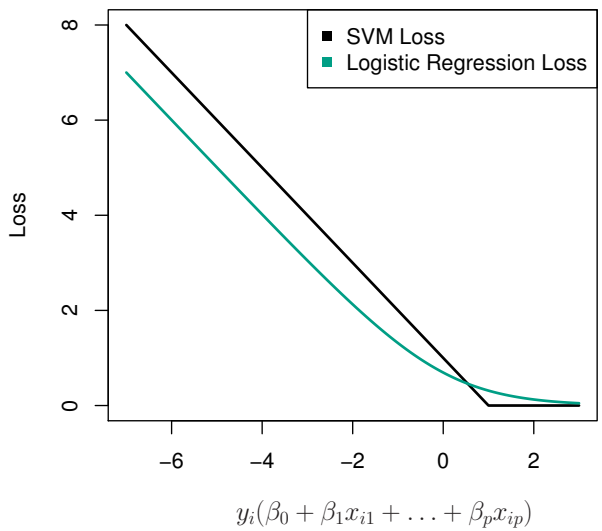
OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

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

- 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.

Advanced issues of Kernels and SVM

Detailed duality, please refer to Page 215 – 229, (Chap 5), Stephen Byod et al. “Convex Optimization” 2004, Cambridge University Press

Checking if a given function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel can be hard.

- $k(x, \bar{x}) = \tanh(1 + \langle x, \bar{x} \rangle)$?
- $k(x, \bar{x}) = \exp(\ -$ edit distance between two strings x and $\bar{x}\ )$?
- $k(x, \bar{x}) = 1 - \|x - \bar{x}\|^2$?

Easier: construct functions that are garanteed to be kernels:

Construct explicitly:

- any $\phi : \mathcal{X} \to \mathbb{R}^m$ induces a kernel $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle$.
  in particular any $f : \mathcal{X} \to \mathbb{R}$, $\quad k(x, \bar{x}) = f(x) f(\bar{x})$

Construction from other kernels:

- If k is a kernel and $\alpha \in \mathbb{R}^+$, then $k + \alpha$ and αk are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.
- if k is a kernel, then $\exp(k)$ is a kernel.

Optimizing the SVM Dual (kernelized)

How to solve the QP

$$\max_{\alpha^1,\ldots,\alpha^n \in \mathbb{R}} \quad -\frac{1}{2}\sum_{i,j=1}^{n} \alpha^i \alpha^j y^i y^j k(x^i, x^j) + \sum_{i=1}^{n} \alpha^i$$

subject to $\sum_i \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$, for $i = 1, \ldots, n$.

Observations:

- Kernel matrix K (with entries $k_{ij} = k(x^i, x^j)$) might be too big to fit into memory.
- In the optimum, many of the α_i are 0 and do not contribute. If we knew which ones, we would save a lot of work

Optimizing the SVM Dual (kernelized)

Working set training [Osuna 1997]

1: $S = \emptyset$
2: **repeat**
3: $\alpha \leftarrow$ solve QP with variables α_i for $i \in S$ and $\alpha_i = 0$ for $i \notin S$
4: **for** $i = 1 \ldots, n$ **do**
5: **if** if $i \in S$ and $\alpha_i = 0$ **then** remove i from S
6: **if** if $i \notin S$ and α_i not optimal **then** add i to S
7: **end for**
8: **until** convergence

Advantages:

- objective value increases monotonously
- converges to global optimum

Disadvantages:

- each step is computationally costly, since S can become large

Sequential Minimal Optimization (SMO) [Platt 1998]

1: $\alpha \leftarrow 0$
2: **repeat**
3: pick index i such that α_i is not optimal
4: pick index $j \neq i$ arbitrarily (usually based on some heuristic)
5: $\alpha_i, \alpha_j \leftarrow$ solve QP for α_i, α_j and all other α_k fixed
6: **until** convergence

Advantages:

- convergences monotonously to global optimum
- each step optimizes a subproblem of smallest possible size:
  2 unknowns (1 doesn't work because of constraint $\sum_i \alpha_i y_i = 0$)
- subproblems have a closed-form solution

Disadvantages:

- many iterations are required
- many kernel values $k(x^i, x^j)$ are computed more than once
  (unless K is stored as matrix)

SVMs Without Bias Term– Optimization

For optimization, the *bias term* is an annoyance

- In primal optimization, it often requires a different stepsize.
- In dual optimization, it is not straight-forward to recover.
- It couples the dual variables by an equality constraint: $\sum_i \alpha_i y_i = 0$.

We can get rid of the bias by the **augmentation trick**.

Original:

- $f(x) = \langle w, x \rangle_{\mathbb{R}^d} + b, \quad \text{with } w \in \mathbb{R}^d, b \in \mathbb{R}.$

New augmented:

- linear: $f(x) = \langle \tilde{w}, \tilde{x} \rangle_{\mathbb{R}^{d+1}}, \quad \text{with } \tilde{w} = (w, b),\ \tilde{x} = (x, 1).$
- generalized: $f(x) = \langle \tilde{w}, \tilde{\phi}(x) \rangle_{\tilde{\mathcal{H}}}$ with $\tilde{w} = (w, b),\ \tilde{\phi}(x) = (\phi(x), 1).$
- kernelize: $\tilde{k}(x, \bar{x}) = \langle \tilde{\phi}(x), \tilde{\phi}(\bar{x}) \rangle_{\tilde{\mathcal{H}}} = k(x, \bar{x}) + 1.$

SVMs Without Bias Term– Optimization

SVM without bias term – primal optimization problem

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi^i$$

subject to, for $i = 1, \ldots, n$,

$$y^i\langle w, x^i\rangle \geq 1 - \xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

Difference: no b variable to optimize over

SVMs Without Bias Term– Optimization

SVM without bias term – primal optimization problem

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi^i$$

subject to, for $i = 1,\ldots,n$,

$$y^i\langle w, x^i\rangle \geq 1-\xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

Difference: no b variable to optimize over

SVM without bias term – dual optimization problem

$$\max_{\alpha} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i,x^j) + \sum_i \alpha_i$$

subject to, $\quad 0 \leq \alpha_i \leq C, \quad$ for $i = 1,\ldots,n.$

Difference: no constraint $\sum_i y_i\alpha_i = 0$.

Linear SVM Optimization in the Dual

Stochastic Coordinate Dual Ascent

$\alpha \leftarrow \mathbf{0}$.
for $t = 1, \ldots, T$ **do**
  $i \leftarrow$ random index (uniformly random or in epochs)
  solve QP w.r.t. α_i with all α_j for $j \neq i$ fixed.
end for
return α

Properties:

- converges monotonically to global optimum
- each subproblem has smallest possible size

Open Problem:

- how to make each step efficient?

SVM Optimization in the Dual

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha \in [0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i, x^j) + \sum_i \alpha_i$

SVM Optimization in the Dual

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha\in[0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i,x^j)+\sum_i \alpha_i$

When all α_j except α_i are fixed: $\mathbf{max}_{\alpha_i\in[0,C]}\, F(\alpha_i)$, with

$$F(\alpha_i) = -\frac{1}{2}\alpha_i^2 k(x^i,x^i) + \alpha_i\Big(1 - y^i\sum_{j\neq i}\alpha_j y^j\, k(x^i,x^j)\Big) + \text{const.}$$

$$\frac{\partial}{\partial\alpha_i}F(\alpha_i) = -\alpha_i k(x^i,x^i) + \Big(1 - y^i\sum_{j\neq i}\alpha_j y^j\, k(x^i,x^j)\Big) + \text{const.}$$

$$\alpha_i^{\text{opt}} = \alpha_i + \frac{1 - y^i\sum_{j=1}^n \alpha_j y^j\, k(x^i,x^j)}{k(x^i,x^i)}, \qquad \alpha_i = \begin{cases} 0 & \text{if } \alpha_i^{\text{opt}} < 0, \\ C & \text{if } \alpha_i^{\text{opt}} > C, \\ \alpha_i^{\text{opt}} & \text{otherwise.} \end{cases}$$

(except if $k(x^i,x^i)=0$, but then $k(x^i,x^j)=0$, so α_i has no influence)

Observation: each update has complexity $O(n)$.

(Generalized) Linear SVM Optimization in the Dual

Let $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle_{\mathbb{R}^d}$ for explicitly known $\phi : \mathcal{X} \to \mathbb{R}^d$.

$$\alpha_i^{\text{opt}} = \alpha_i + \frac{1 - y^i \sum_j \alpha_j y^j \, k(x^i, x^j)}{k(x^i, x^i)},$$

remember $w = \sum_j \alpha_j y_j \phi(x^j)$

$$= \alpha_i + \frac{1 - y^i \langle w, \phi(x^i) \rangle}{\|\phi(x^i)\|^2},$$

- each update takes $O(d)$, independent of n
  - $\langle w, \phi(x^i) \rangle$ takes at most $O(d)$ for explicit $w \in \mathbb{R}^d, \phi(x^i) \in \mathbb{R}^d$
  - we must also take care that w remains up to date (also at most $O(d)$)

(Generalized) Linear SVM Optimization in the Dual

SCDA for (Generalized) Linear SVMs [Hsieh, 2008]

initialize $\alpha \leftarrow \mathbf{0}$, $w \leftarrow \mathbf{0}$

for $t = 1, \ldots, T$ **do**

$i \leftarrow$ random index (uniformly random or in epochs)

$\delta \leftarrow \frac{1 - y^i \langle w, \phi(x^i) \rangle}{\|\phi(x^i)\|^2}$

$$\alpha_i \leftarrow \begin{cases} 0, & \text{if } \alpha_i + \delta < 0, \\ C, & \text{if } \alpha_i + \delta > C, \\ \alpha_i + \delta, & \text{otherwise.} \end{cases}$$

$w \leftarrow w + \delta y^i \phi(x^i)$

end for

return α, w

Properties:

- converges monotonically to global optimum
- complexity of each step is independent of n
- resembles stochastic gradient method, but **automatic step size**

Appendix–Practical Issues in Machine Learning Experiments

You've trained a new predictor, $g : \mathcal{X} \to \mathcal{Y}$, and you want to tell the world how good it is. How to measure this?

Reminder:

- The average loss on the training set, $\frac{1}{|\mathcal{D}_{trn}|} \sum_{(x,y)\in\mathcal{D}_{trn}} \ell(y, g(x))$ tells us (almost) nothing about the future loss.
  Reporting it would be misleading as best.
- The relevant quantity is the expected risk,

$$\mathcal{R}(g) = \mathbb{E}_{(x,y)\sim p(x,y)}\ \ell(y, g(x))$$

  which unfornately we cannot compute, since $p(x, y)$ is unknown.
- If we have data $\mathcal{D}_{tst} \overset{i.i.d.}{\sim} p(x, y)$, we have,

$$\frac{1}{|\mathcal{D}_{tst}|} \sum_{(x,y)\in\mathcal{D}_{tst}} \ell(y, g(x)) \quad \overset{|\mathcal{D}_{tst}|\to\infty}{\longrightarrow} \quad \mathbb{E}_{(x,y)\sim p(x,y)}\ \ell(y, g(x))$$

- Problem: samples $\ell(y, g(x))$ must me independent, otherwise law of large numbers doesn't hold.
- Make sure that g is independent of $\mathcal{D}_{tst}$.

Classifier Training (idealized)

input training data $\mathcal{D}_{trn}$
input learning procedure A
$g \leftarrow A[\mathcal{D}]$ (apply A with $\mathcal{D}$ as training set)
output resulting classifier $g : \mathcal{X} \rightarrow \mathcal{Y}$

Classifier Evaluation

input trained classifier $g : \mathcal{X} \rightarrow \mathcal{Y}$
input test data $\mathcal{D}_{tst}$
apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Classifier Training (idealized)

input training data $\mathcal{D}_{trn}$
input learning procedure A
$g \leftarrow A[\mathcal{D}]$ (apply A with $\mathcal{D}$ as training set)
output resulting classifier $g : \mathcal{X} \to \mathcal{Y}$

Classifier Evaluation

input trained classifier $g : \mathcal{X} \to \mathcal{Y}$
input test data $\mathcal{D}_{tst}$
apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Remark: In commercial applications, this is realistic:

- given some training set one builds a single system,
- one deploys it to the customers,
- the customers use it on their own data, and complain if disappointed

In research, one typically has no customer, but only a fixed amount of data to work with, so one *simulates* the above protocol.

Classifier Training and Evaluation

input data $\mathcal{D}$
input learning method A
  split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it
  $g \leftarrow A[\mathcal{D}_{trn}]$ // learn a predictor from $\mathcal{D}_{trn}$
  apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Classifier Training and Evaluation

input data $\mathcal{D}$
input learning method A
  split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it
  $g \leftarrow A[\mathcal{D}_{trn}]$ // learn a predictor from $\mathcal{D}_{trn}$
  apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Remark. $\mathcal{D}_{tst}$ should be as small as possible, to keep $\mathcal{D}_{trn}$ as big as possible, but large enough to be convincing.

- sometimes: 50%/50% for small datasets
- more often: 80% training data, 20% test data
- for large datasets: 90% training, 10% test data.

Remark: The split because $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! Even if the learning algorithm doesn't see it, you looking at it can and will influence your model design or parameter selection (human overfitting).
- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.

Remark: The split because $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! Even if the learning algorithm doesn't see it, you looking at it can and will influence your model design or parameter selection (human overfitting).
- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.

In practice we often want more: not just evaluate one classifier, but

- select the best algorithm or parameters amongst multiple ones

We simulate the classifier evaluation step during the training procedure. This needs (at least) one additional data split:

Training and Selecting between Multiple Models

input data $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
  split $\mathcal{D} = \mathcal{D}_{trnval} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place (do not look at it)

  split $\mathcal{D}_{trnval} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{val}$ disjointly
  for all models $A_i \in \mathcal{A}$ **do**
    $g_i \leftarrow A_i[\mathcal{D}_{trn}]$
    apply g_i to $\mathcal{D}_{val}$ and measure performance $E_{val}(A_i)$
  end for
  pick best performing A_i

  (optional) $g_i \leftarrow A_i[\mathcal{D}_{trnval}]$ // retrain on larger dataset
  apply g_i to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

How to split? For example $1/3$–$1/3$–$1/3$ or 70%–10%–20%.

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Problems.

- small $\mathcal{D}_{val}$ is bad: E_{val} could be bad estimate of g_A's true performance, and we might pick a suboptimal method.
- large $\mathcal{D}_{val}$ is bad: $\mathcal{D}_{trn}$ is much smaller than $\mathcal{D}_{trnval}$, so the classifier learned on $\mathcal{D}_{trn}$ might be much worse than necessary.
- retraining the best model on $\mathcal{D}_{trnval}$ might overcome that, but it comes at a risk: just because a model worked well when trained on $\mathcal{D}_{trn}$, this does not mean it'll also work well when trained on $\mathcal{D}_{trnval}$.

Leave-one-out Evaluation (for a single model/algorithm)

input algorithm A
input loss function ℓ
input data $\mathcal{D}$ (trnval part only: test part set aside earlier)
  for all $(x^i, y^i) \in \mathcal{D}$ **do**
    $g^{\neg i} \leftarrow A[\ \mathcal{D} \setminus \{(x^i, y^i)\}\ ]$ // $\mathcal{D}_{trn}$ is $\mathcal{D}$ with i-th example removed
    $r^i \leftarrow \ell(y^i, g^{\neg i}(x^i))$ // $\mathcal{D}_{val} = \{(x^i, y^i)\}$, disjoint to $\mathcal{D}_{trn}$
  end for
output $R_{loo} = \frac{1}{n} \sum_{i=1}^{n} r^i$ (average leave-one-out risk)

Properties.

- Each r^i is a unbiased (but noisy) estimate of the risk $\mathcal{R}(g^{\neg i})$
- $\mathcal{D} \setminus \{(x^i, y^i)\}$ is almost the same as $\mathcal{D}$, so we can hope that each $g^{\neg i}$ is almost the same as $g = A[\mathcal{D}]$.
- Therefore, R_{loo} can be expected a good estimate of $\mathcal{R}(g)$

Problem: slow, trains n times on $n-1$ examples instead of once on n

Compromise: use fixed number of small $\mathcal{D}_{val}$

K-fold Cross Validation (CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into K equal sized disjoint parts
  for $k = 1, \ldots, K$ **do**
    $g^{\neg k} \leftarrow A[\,\mathcal{D} \setminus \mathcal{D}_k\,]$
    $r^k \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} \ell(y^i, g^{\neg k}(x))$
  end for
output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{n} r^k$ (K-fold cross-validation risk)

Observation.

- for $K = |\mathcal{D}|$ same as leave-one-out error.
- approximately k times increase in runtime.
- most common: $k = 10$ or $k = 5$.

Problem: training sets overlap, so the error estimates are correlated.
Exception: $K = 2$

5×2 Cross Validation (5×2-CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  for $k = 1, \ldots, 5$ **do**
    Split $\mathcal{D} = \mathcal{D}_1 \dot{\cup} \mathcal{D}_2$
    $g_1 \leftarrow A[\mathcal{D}_1]$,
    $r_1^k \leftarrow$ evaluate g_1 on $\mathcal{D}_2$
    $g_2 \leftarrow A[\mathcal{D}_2]$,
    $r_2^k \leftarrow$ evaluate g_2 on $\mathcal{D}_1$
    $r^k \leftarrow \frac{1}{2}(r_k^1 + r_k^2)$
  end for
output $E_{5\times 2} = \frac{1}{5} \sum_{k=1}^{5} r^k$

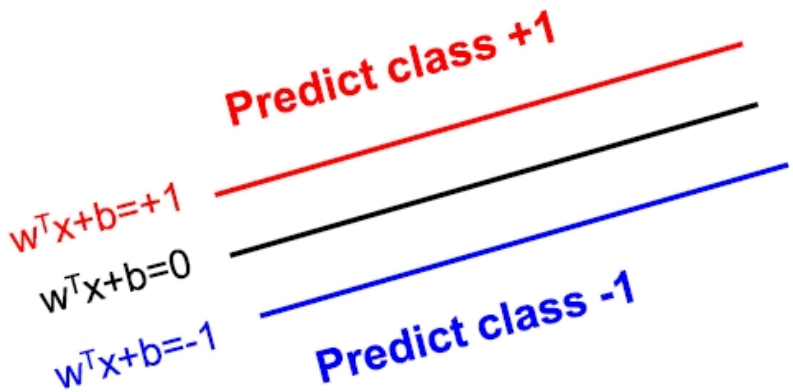
Observation.

- 5×2-CV is really the average of 5 runs of 2-fold CV
- very easy to implement: shuffle the data and split into halfs
- within each run the training sets are disjoint and the classifiers g_1 and g_2 are independent

Problem: training sets are smaller than in 5- or 10-fold CV.

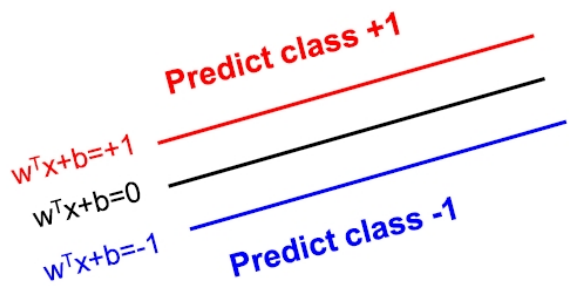
Appendix–Geometry of the Linear SVM

- Max margin classifier: inputs in margin are of unknown class



$$y = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 1 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} + b \leq -1 \\ \text{Undefined} & \text{if } -1 \leq \mathbf{w}^T\mathbf{x} + b \leq 1 \end{cases}$$

Summary: Geometry of the Linear SVM



- The vector $\mathbf{w}$ is orthogonal to the $+1$ plane.
  If $\mathbf{u}$ and $\mathbf{v}$ are two points on that plane, then

$$\mathbf{w}^T(\mathbf{u}-\mathbf{v})=0$$

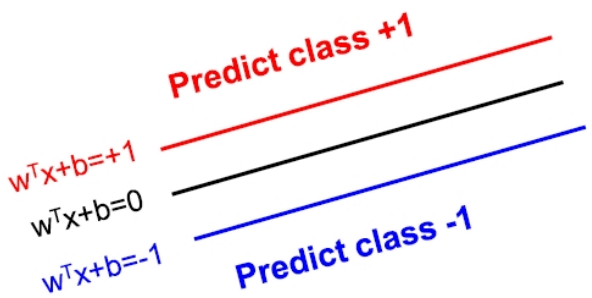
- Same is true for -1 plane
- Also: for point $\mathbf{x}_+$ on $+1$ plane and $\mathbf{x}_-$ nearest point on -1 plane:

$$\mathbf{x}_+ = \lambda\mathbf{w}+\mathbf{x}_-$$

Summary: Computing the Margin

- Also: for point $\mathbf{x}_+$ on $+1$ plane and $\mathbf{x}_-$ nearest point on -1 plane:

$$\mathbf{x}_+ = \lambda \mathbf{w} + \mathbf{x}_-$$



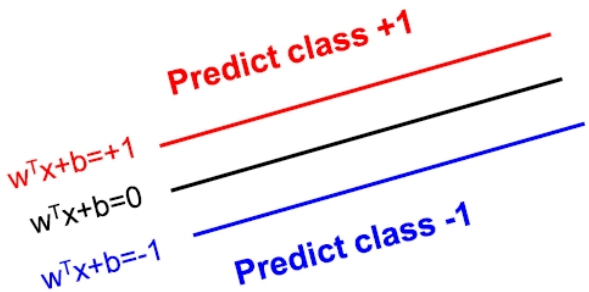
$$\begin{aligned} \mathbf{w}^T \mathbf{x}_+ + b &= 1 \\ \mathbf{w}^T (\lambda \mathbf{w} + \mathbf{x}_-) + b &= 1 \\ \mathbf{w}^T \mathbf{x}_- + b + \lambda \mathbf{w}^T \mathbf{w} &= 1 \\ -1 + \lambda \mathbf{w}^T \mathbf{w} &= 1 \end{aligned}$$

Therefore

$$\lambda = \frac{2}{\mathbf{w}^T \mathbf{w}}$$

Summary: Computing the Margin

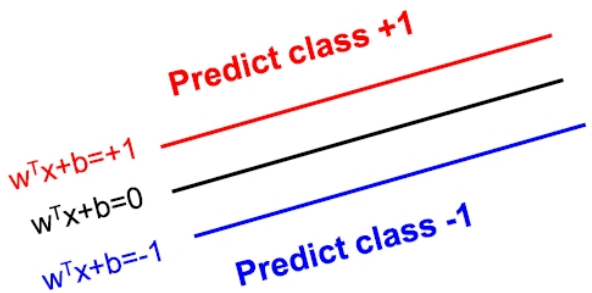
- Define the margin M to be the distance between the $+1$ and -1 planes
- We can now express this in terms of $\mathbf{w}$ to maximize the margin we minimize the length of $\mathbf{w}$



$$
\begin{aligned}
M &= ||\mathbf{x}_+ - \mathbf{x}_-|| \\
&= ||\lambda \mathbf{w}|| = \lambda \sqrt{\mathbf{w}^T \mathbf{w}} \\
&= 2\frac{\sqrt{\mathbf{w}^T \mathbf{w}}}{\mathbf{w}^T \mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}} = \frac{2}{||\mathbf{w}||}
\end{aligned}
$$

Summary: Learning a Margin-Based Classier

- We can search for the optimal parameters ($\mathbf{w}$ and b) by finding a solution that:
  1. Correctly classifies the training examples: $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$
  2. Maximizes the margin (same as minimizing $\mathbf{w}^T\mathbf{w}$)



$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2$$

$$s.t. \forall i \quad (\mathbf{w}^T\mathbf{x}^{(i)} + b)t^{(i)} \geq 1,$$

- This is call the primal formulation of Support Vector Machine (SVM)
- Can optimize via projective gradient descent, etc.
- Apply Lagrange multipliers: formulate equivalent problem

Appendix–Gradient Descendent of Logitic Regression

Loss Function of Logitic Regression

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

- Why is this equivalent to maximize the conditional likelihood?
- Is there a closed form solution?
- It's a convex function of $\mathbf{w}$. Can we get the global optimum?

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size λ

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \lambda \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} \qquad p(C = 1|\mathbf{x}) = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- You can write this in vector form

$$\bigtriangledown \ell(\mathbf{w}) = \left[\frac{\partial \ell(\mathbf{w})}{\partial w_0}, \cdots, \frac{\partial \ell(\mathbf{w})}{\partial w_k} \right]^T, \qquad \text{and} \qquad \triangle(\mathbf{w}) = -\lambda \bigtriangledown \ell(\mathbf{w})$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)}, \mathbf{w}) - \sum_{i=1}^{N} (1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w})$$

  where the probabilities are

$$p(C=0|\mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x}, \mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

  and $z = \mathbf{w}^T\mathbf{x} + w_0$

- We can simplify

$$\begin{aligned} \ell(\mathbf{w}) &= \sum_i t^{(i)} \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)} + \sum_i (1-t^{(i)}) \log(1+\exp(-z^{(i)})) \\ &= \sum_i \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)} \end{aligned}$$

- Now it's easy to take derivatives

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

$$\frac{\partial \ell}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

- What's $x_j^{(i)}$?
- And simplifying

$$\frac{\partial \ell}{\partial w_j} = \sum_i x_j^{(i)} \left(t^{(i)} - p(C = 1 | \mathbf{x}^{(i)}) \right)$$

- Don't get confused with indexes: j for the weight that we are updating and i for the training example
- Logistic regression has linear decision boundary

Acknowledgement

Some slides are in courtesy of
(1)Chap 9 of James *et. al.* "An Introduction to Statistical Learning with applications in R", 2011;
(2) Lecture 05,15, CSC 411 by Raquel Urtasun & Rich Zemel, University of Toronto