# KERNELS, SVM

CS5011- MACHINE LEARNING

Murphy 14.1, 14.2.1-14.2.6, 14.3, 14.4, 14.5

+ Wiki + Online Tut notes.

Introduction

- How do we represent a text document or protein sequence, which can be of variable length?

- One approach is to define a generative model for the data, and use the inferred latent representation and/or the parameters of the model as features, and then to plug these features in to standard methods

- Another approach is to assume that we have a way of measuring the similarity between objects, that doesn't require preprocessing them into feature vector format

- For example, when comparing strings, we can compute the edit distance between them
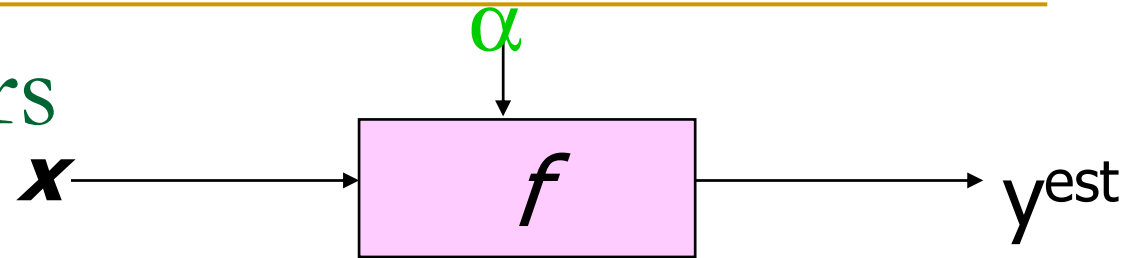
Linear Classifiers
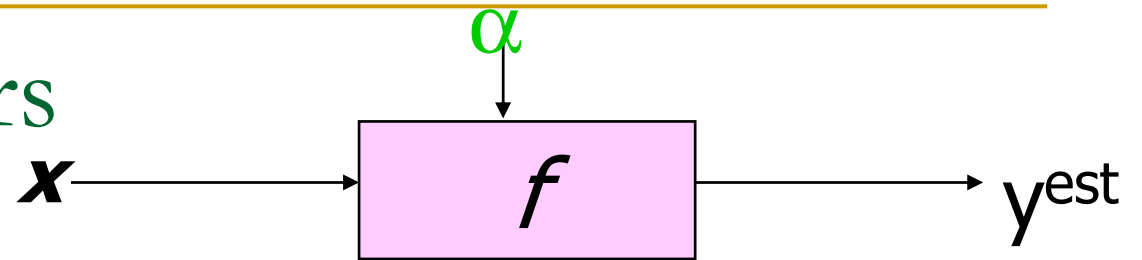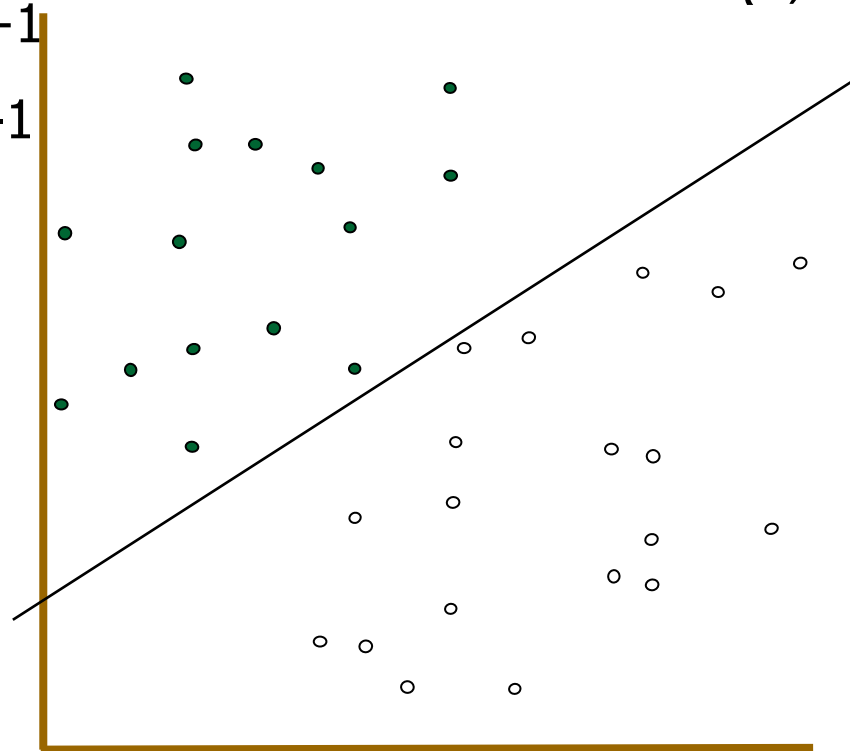
α

$$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$$\boldsymbol{f(x,w,b)} = sign(\boldsymbol{w}\ \boldsymbol{x} + b)$$

• denotes +1

○ denotes -1

How would you classify this data?

Linear Classifiers

α

$x \longrightarrow \boxed{f} \longrightarrow y^{est}$

$f(x, w, b) = sign(w\ x + b)$

- denotes +1
- denotes -1

How would you classify this data?

Linear Classifiers

α

$\boldsymbol{x}$ → f → y^{est}
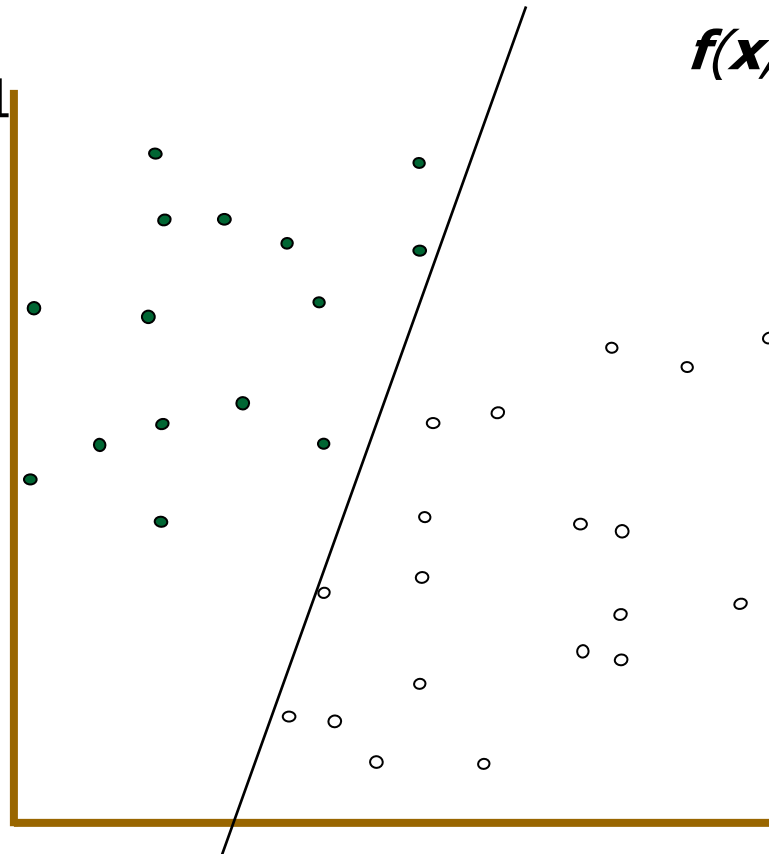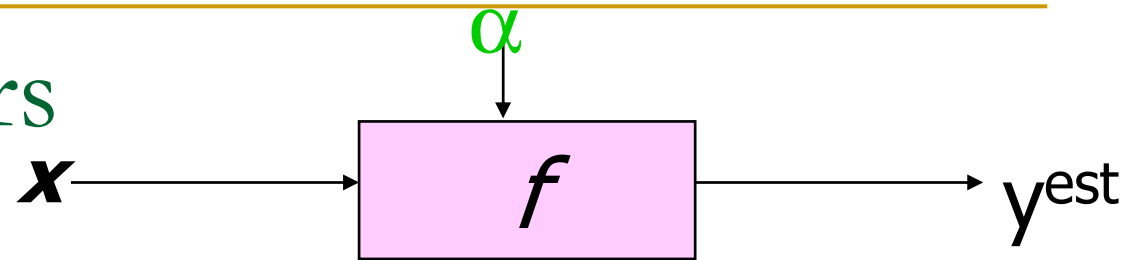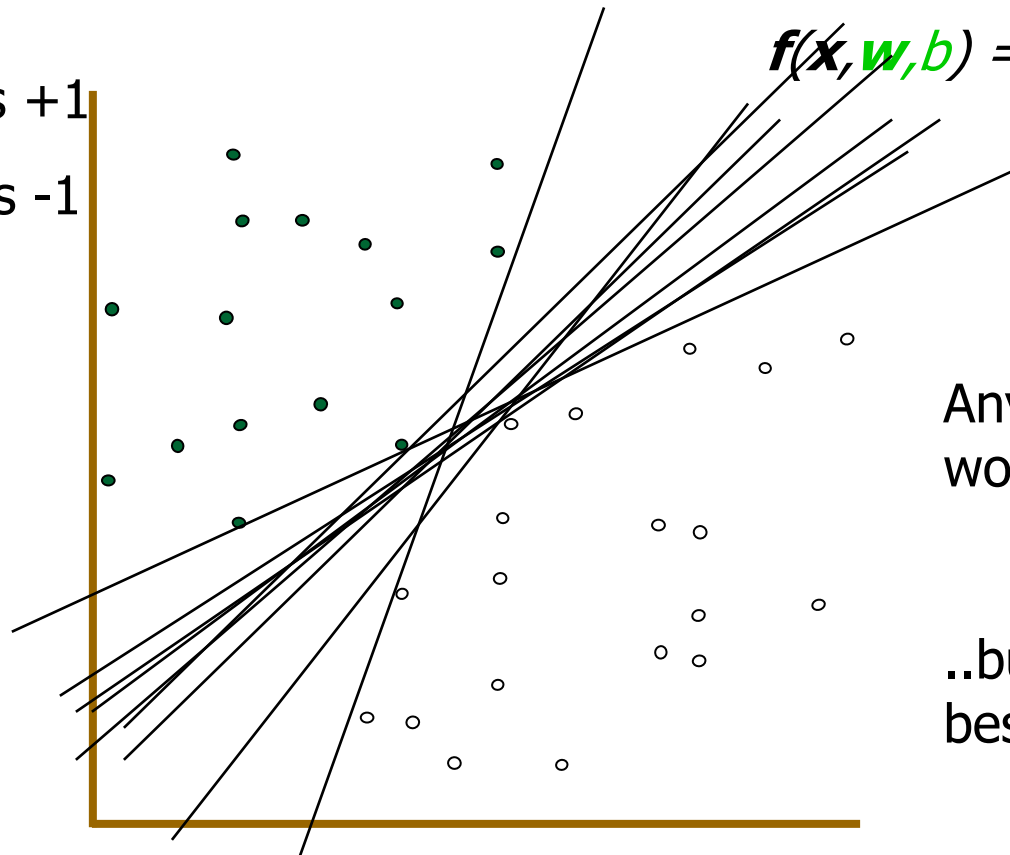
$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\,\boldsymbol{x} + b)$

- denotes +1
- denotes -1

Any of these would be fine..

..but which is best?

Linear Classifiers

$$\alpha$$

$$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow y^{est}$$
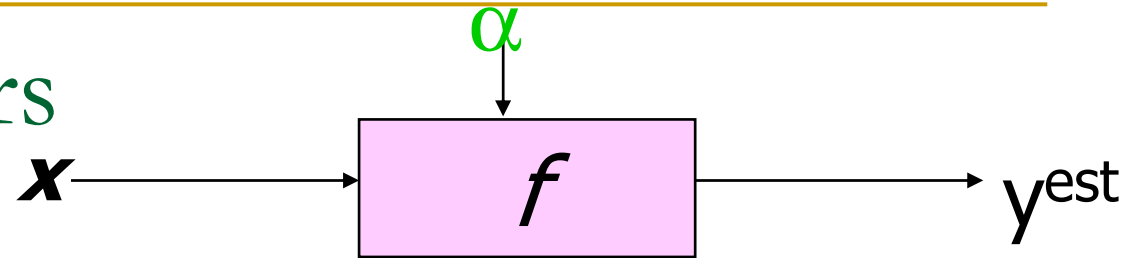
$$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\,\boldsymbol{x} + b)$$

- denotes +1
- denotes -1

How would you classify this data?
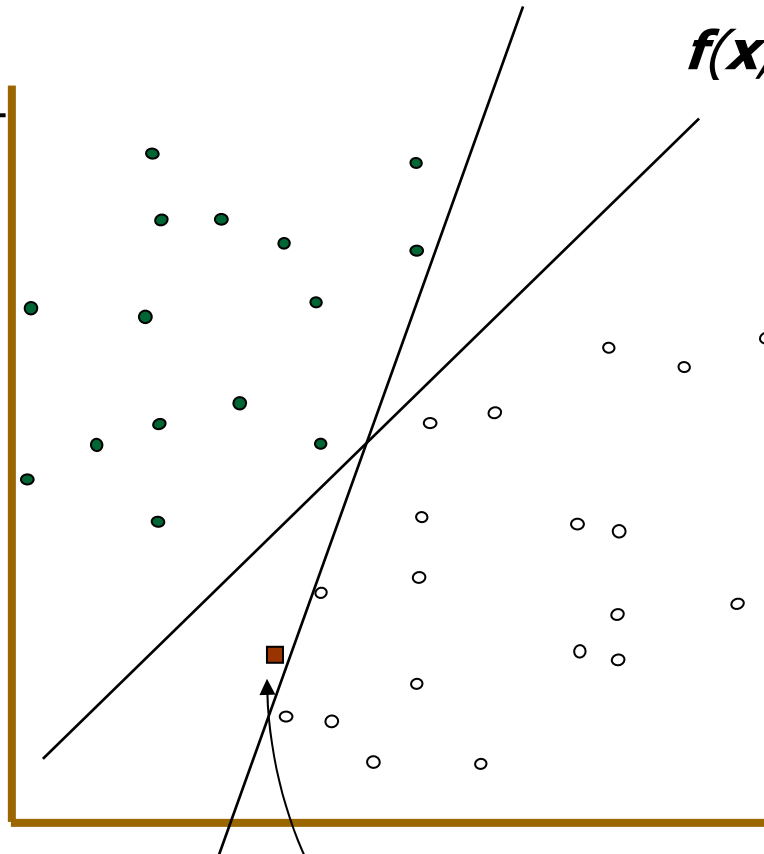
Misclassified to +1 class

Classifier Margin

α

$x \longrightarrow$ [f] $\longrightarrow y^{est}$
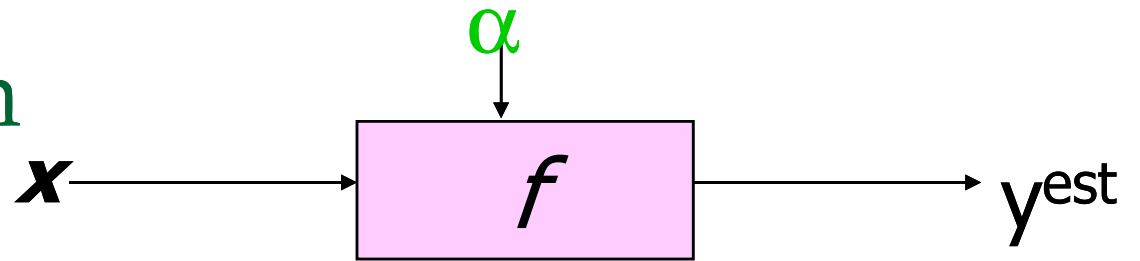
$f(x,w,b) = sign(w\ x + b)$

- denotes +1
- denotes -1
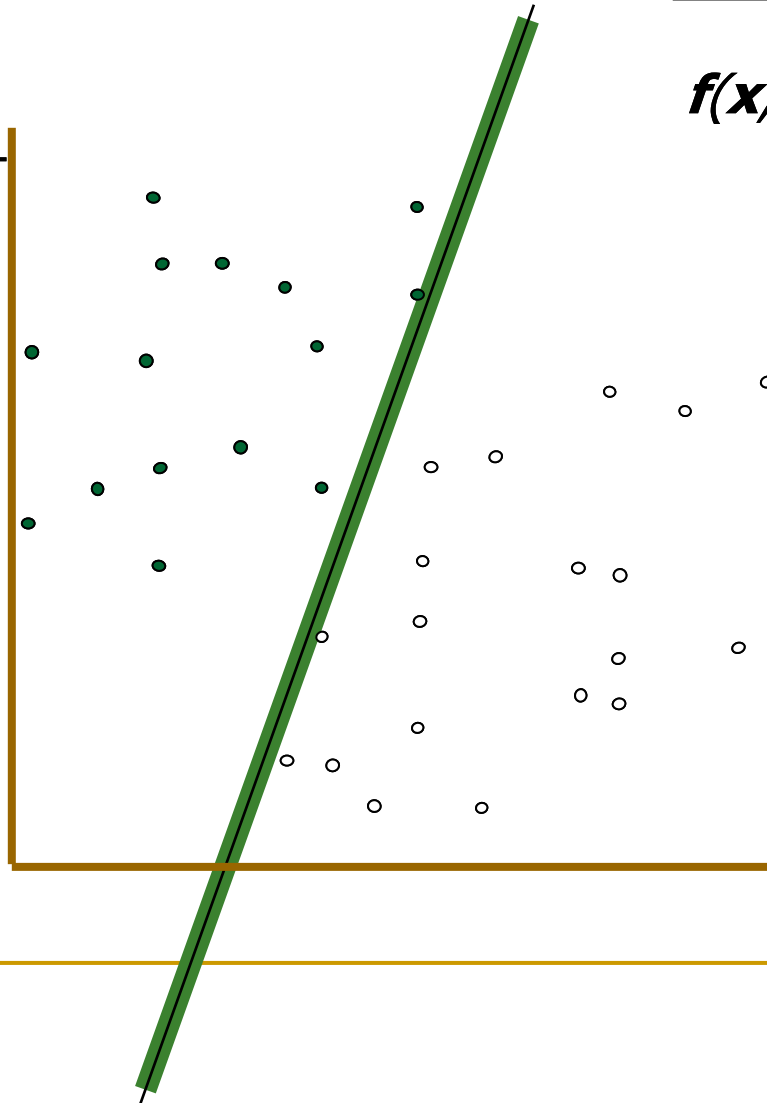
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin

α

$\mathbf{x}$ → | f | → y^{est}

- • denotes +1
- ∘ denotes -1

Support Vectors are those datapoints that the margin pushes up against
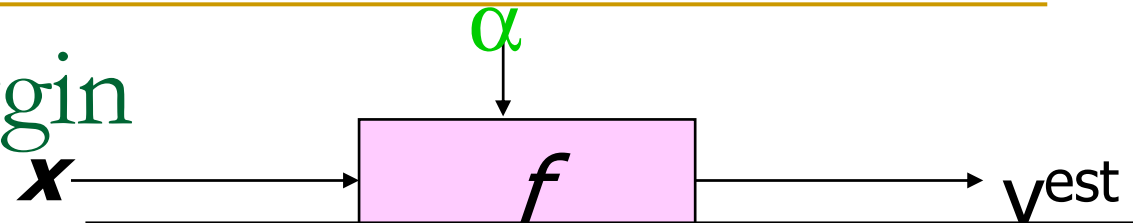
1. Maximizing the margin is good according to intuition

2. Implies that only support vectors are important; other training examples are ignorable.

3. Empirically it works very well.
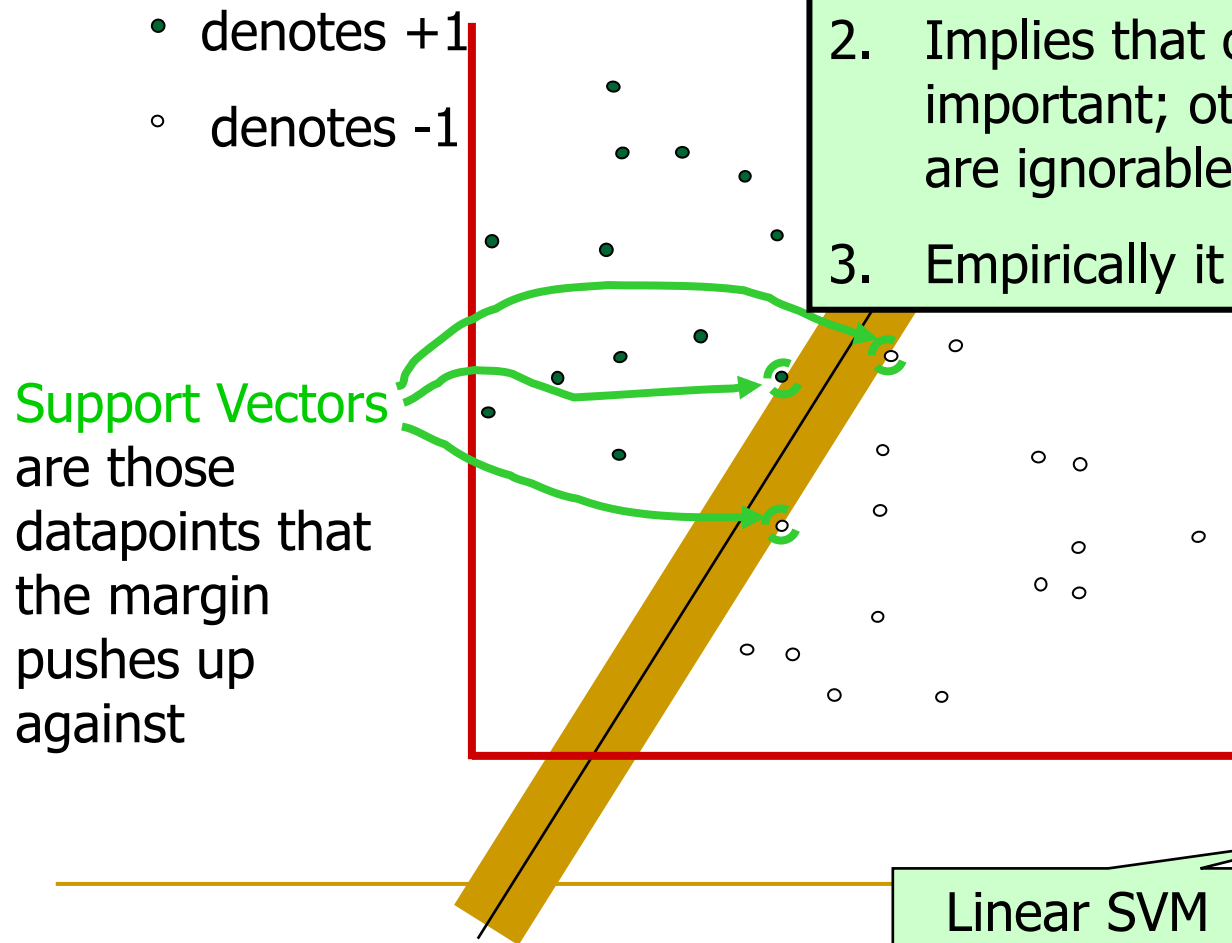
linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Linear SVM Mathematically

"Predict Class = +1" zone

x^+

$\boldsymbol{M}$=Margin Width

wx+b=1
wx+b=0
wx+b=-1

x^-

"Predict Class = -1" zone
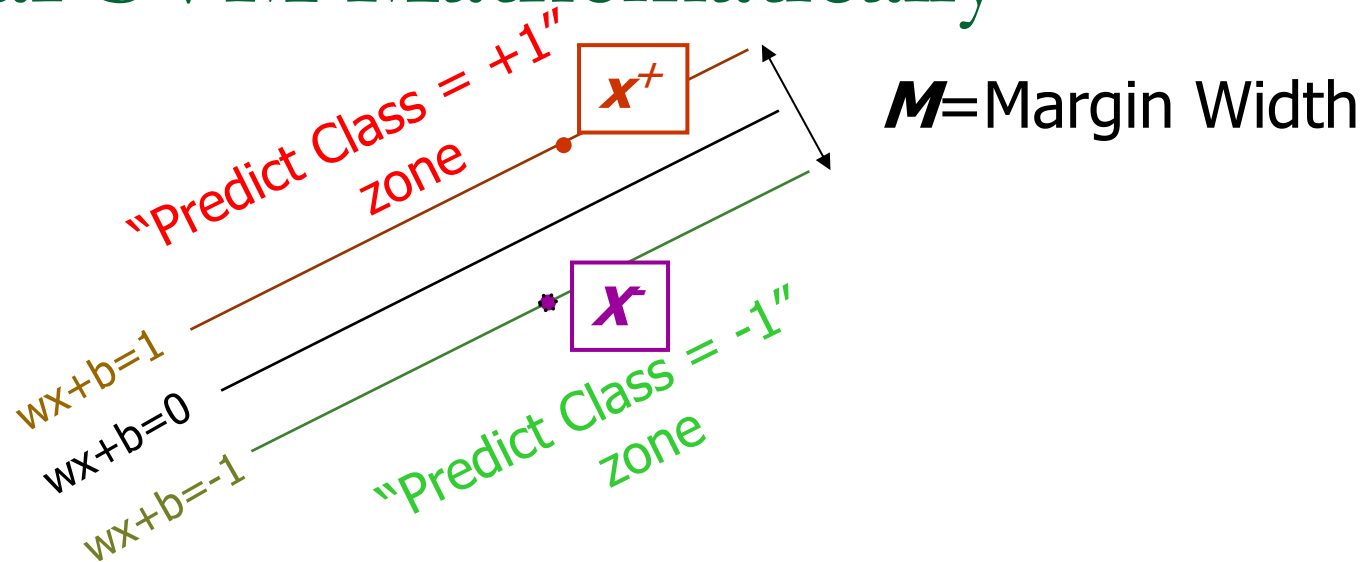
What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x^+} + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x^-} + b = -1$
- $\boldsymbol{w} \cdot (\boldsymbol{x^+ - x^-}) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: **1) Correctly classify all training data**

$$wx_i + b \geq 1 \quad \text{if } y_i = +1$$

$$wx_i + b \leq 1 \quad \text{if } y_i = -1$$

$$y_i(wx_i + b) \geq 1 \quad \text{for all i}$$

2) Maximize the Margin

$$M = \frac{2}{|w|}$$

same as minimize $\quad \frac{1}{2}w^t w$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

- Minimize $\quad \Phi(w) = \frac{1}{2}w^t w$

  subject to $\quad y_i(wx_i + b) \geq 1 \qquad \forall i$

Solving the Optimization Problem

Find **w** and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T\mathbf{w}$ is minimized;
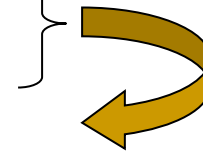
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i (\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **Need to optimize a *quadratic* function subject to *linear* constraints.**

- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**

- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

Find $\alpha_1...\alpha_N$ such that

$Q(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w^T x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.

- Then the classifying function will have the form:

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i^T x} + b$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x_i}$.

- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i^T x_j}$ between all pairs of training points.

Dataset with noise

denotes +1

denotes -1
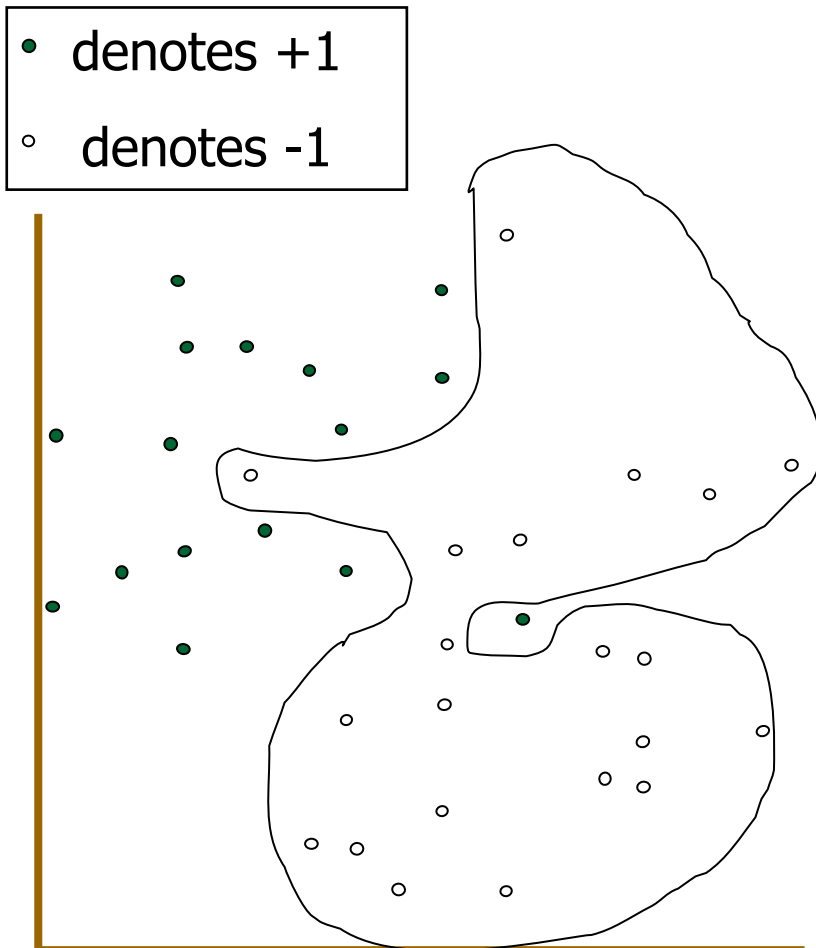
- **Hard Margin:** So far we require all data points be classified correctly

  - No training error

- **What if the training set is noisy?**

  - **Solution 1:** use very powerful kernels

OVERFITTING!

Soft Margin Classification

Slack variables ξi can be added to allow misclassification of difficult or noisy examples.
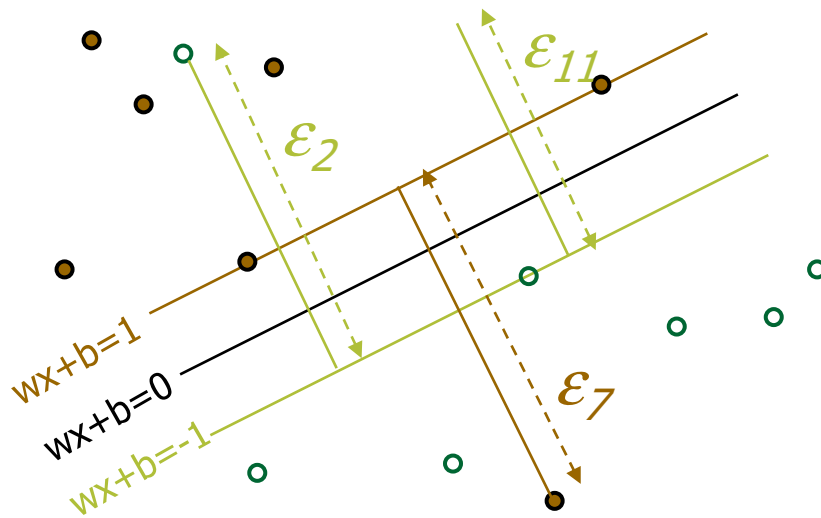
What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Hard Margin v.s. Soft Margin

- **The old formulation:**

  Find $\mathbf{w}$ and b such that

  $\mathbf{\Phi(w)} = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

  $y_i(\mathbf{w^T x_i} + b) \geq 1$

- **The new formulation incorporating slack variables:**

  Find $\mathbf{w}$ and b such that

  $\mathbf{\Phi(w)} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

  $y_i(\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- **Parameter C can be viewed as a way to control overfitting.**

Hard 1-dimensional Dataset

What would SVMs do with this data?

Not a big surprise

$x=0$

Positive "plane"

Negative "plane"
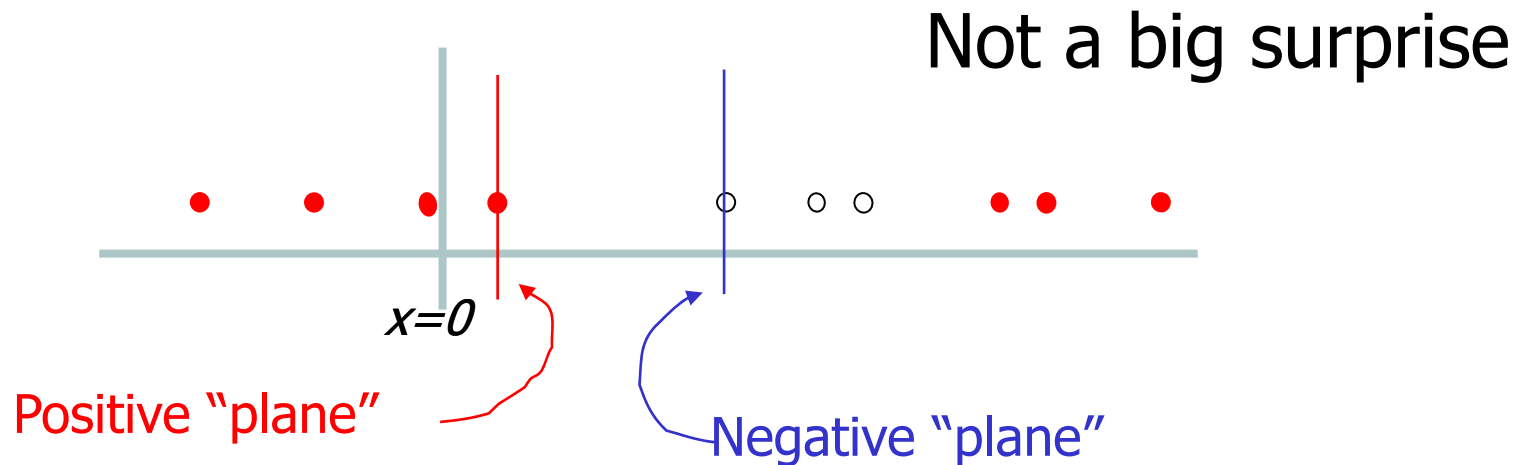
Doesn't look like slack variables will save us this time...

Hard 1-dimensional Dataset

Make up a new feature!

Sort of...

... computed from original feature(s)

$$\mathbf{z}_k = (x_k, x_k^2)$$

Separable! MAGIC!

x=0
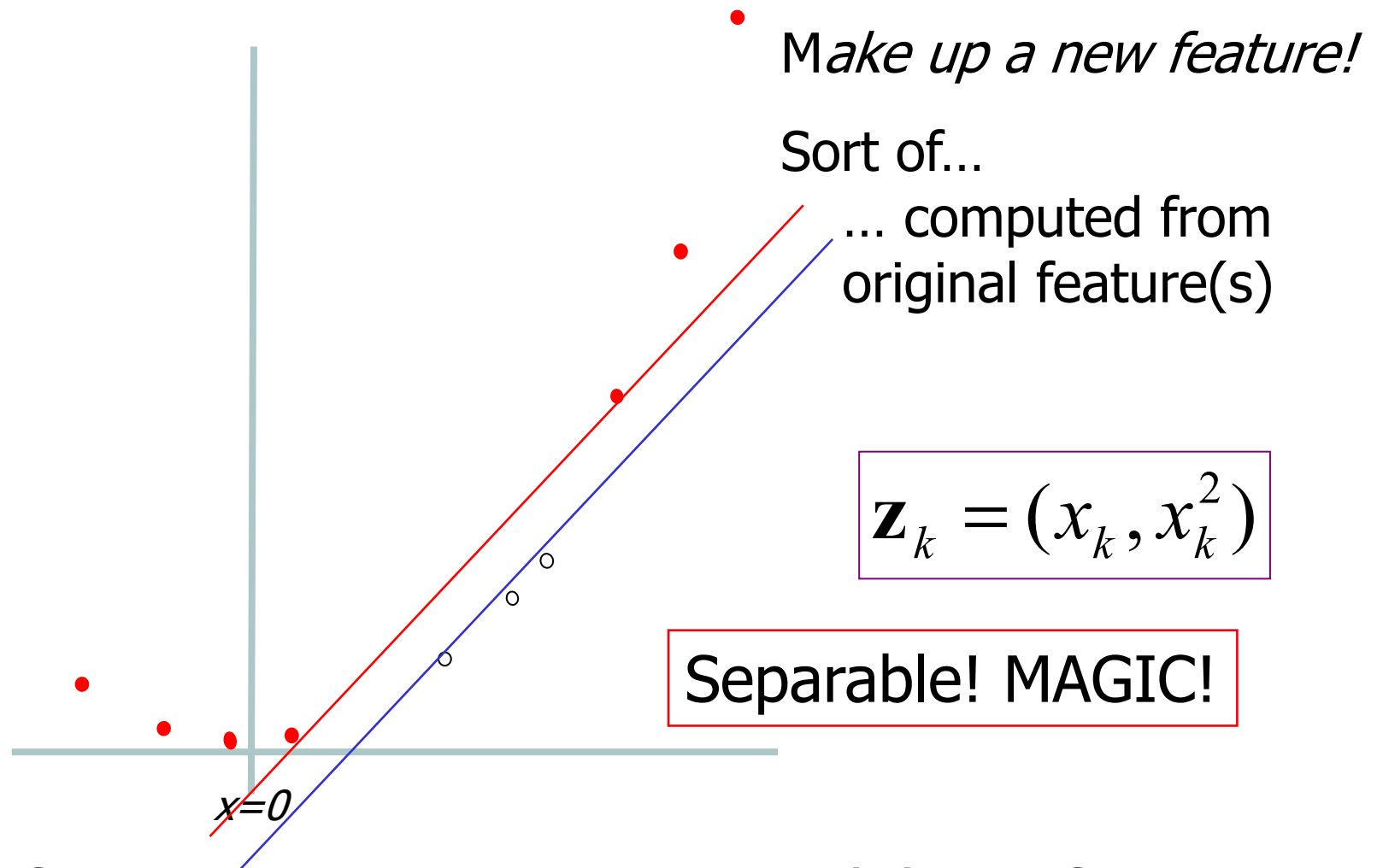
New features are sometimes called *basis functions.*

Now drop this "augmented" data into our linear SVM.

Kernels and Linear Classifiers

Let $\vec{x} = [\vec{x}_1, \vec{x}_2] \in \mathbb{R}^2$ be a vectorial represenation
of object $x \in \mathcal{X}$

Let $\phi : \mathcal{X} \to \mathcal{K} \subset \mathbb{R}^3$ feature map be given by

$$\phi(\vec{x}) \doteq [\vec{x}_1, \vec{x}_2^2, \vec{x}_1 \vec{x}_2]^T \in \mathcal{K} \subset \mathbb{R}^3$$

Def. Feature space: $\mathcal{K}$

We will use linear classifiers in this feature space.

In the original space $\mathbb{R}^2$ for a given $\mathbf{w} \in \mathbb{R}^3$ the decision surface is:

$$\tilde{X}_0(\mathbf{w}) = \{\vec{x} \in \mathbb{R}^2 \mid w_1 \vec{x}_1 + w_2 \vec{x}_2^2 + w_3 \vec{x}_1 \vec{x}_2 = 0\}$$

- This is nonlinear in $\vec{x} \in \mathbb{R}^2$
- This is linear in the feature space $\phi(\vec{x}) \in \mathcal{K} \subset \mathbb{R}^3$

Kernels and Linear Classifiers

$$\phi(\vec{x}) \doteq [\phi_1(\vec{x}), \phi_2(\vec{x}), \phi_3(\vec{x})] \doteq [\vec{x}_1, \vec{x}_2^2, \vec{x}_1\vec{x}_2]^T$$
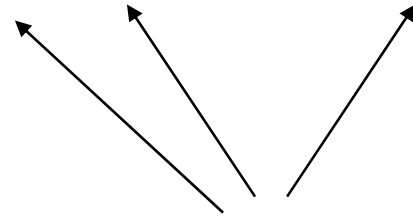
Feature functions

- We seek for a small set of basis vectors $\{\phi_i\}$
  which allows perfect discrimination between
  the classes in $\mathcal{X}$ (**Feature selection**)

- If we have too many features $\Rightarrow$ overfitting can happen.

Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:
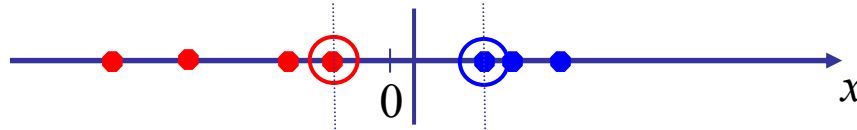
- But what are we going to do if the dataset is just too hard?
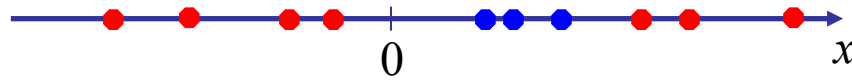
- How about… mapping data to a higher-dimensional space:
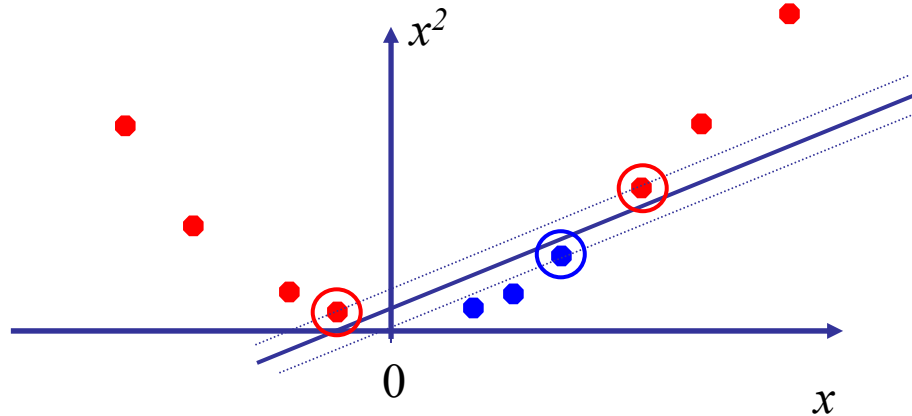
Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:

$$\Phi:\ \mathbf{x} \rightarrow \varphi(\mathbf{x})$$
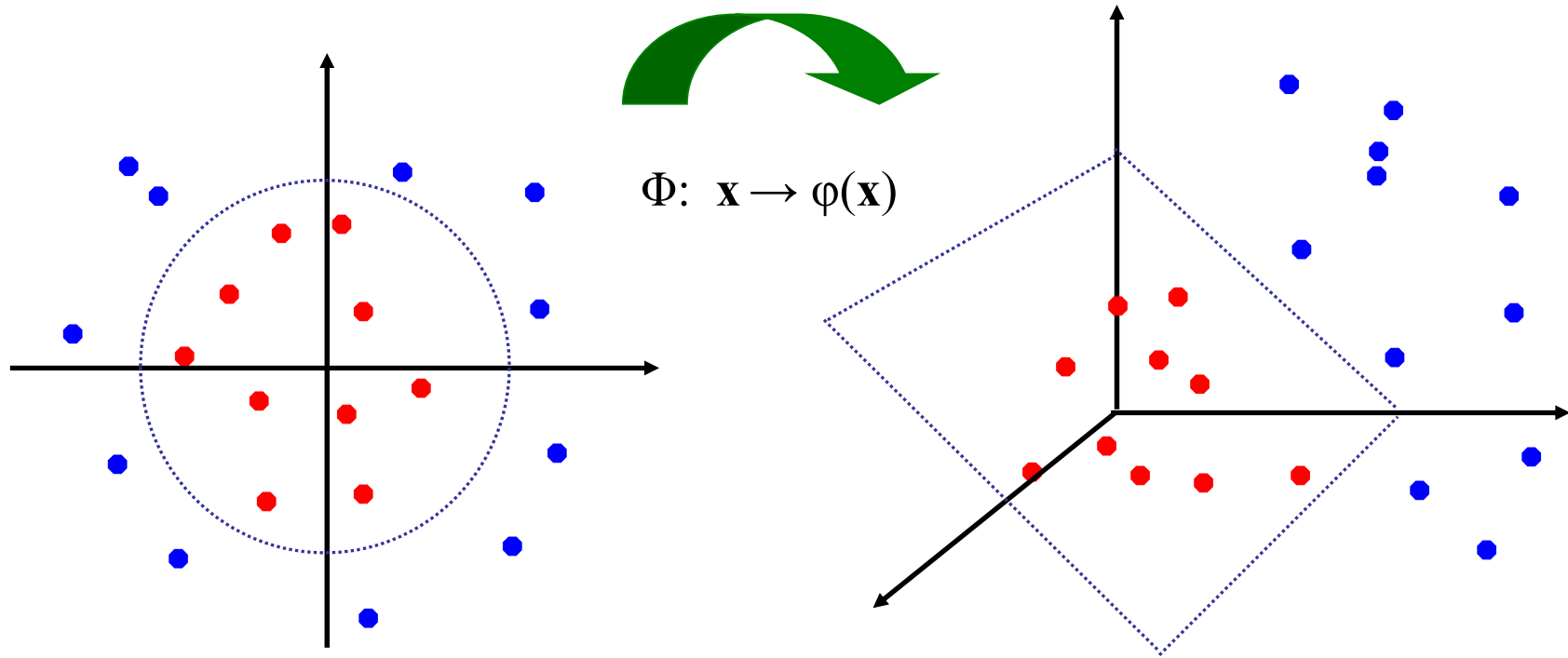
Kernel functions

- We define a kernel function to be a real-valued function of two arguments, $\kappa(\mathbf{x}, \mathbf{x}') \in R$, for $\mathbf{x}, \mathbf{x}' \in X$.

- X is some abstract space

- Typically the function has the following properties:
  - Symmetric
  - Non-negative
  - Can be interpreted as a measure of similarity

- We will discuss several examples of kernel functions

RBF kernels

- **Squared exponential kernel** (SE kernel) or **Gaussian kernel**

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\right)$$

- If $\mathbf{\Sigma}$ is diagonal, this can be written as

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{1}{\sigma_j^2}(x_j - x_j')^2\right)$$

  We can interpret the σ_j as defining the **characteristic length scale** of dimension j

- If $\mathbf{\Sigma}$ is spherical, we get the isotropic kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

  An example of RBF (Radial basis function) kernel (since it is a function of $||x - x'||$) where σ^2 is known as the **bandwidth**

Kernels for comparing documents

- If we use a bag of words representation, where $\mathbf{x}_{ij}$ is the number of times words j occurs in document i, we can use the **cosine similarity**

$$\kappa(\mathbf{x}_i, \mathbf{x}_{i'}) = \frac{\mathbf{x}_i^T \mathbf{x}_{i'}}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_{i'}\|_2}$$

- Unfortunately, this simple method does not work very well
  - Stop words (such as "the" or "and") are not discriminative
  - Similarity is artificially boosted when a discriminative word occurs multiple times
- Replace the word count vector with Term frequency inverse document frequency (**TF-IDF**)

Kernels for comparing documents

- Define the term frequency as:

$$\mathrm{tf}(x_{ij}) \triangleq \log(1 + x_{ij})$$

- This reduces the impact of words that occur many times with a document

- Define inverse document frequency where N is the total number of documents

$$\mathrm{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)}$$

- Our new kernel has the form

$$\kappa(\mathbf{x}_i, \mathbf{x}_{i'}) = \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'})}{||\phi(\mathbf{x}_i)||_2 ||\phi(\mathbf{x}_{i'})||_2}$$

$$\phi(\mathbf{x}) = \text{tf-idf}(\mathbf{x})$$

$$\text{tf-idf}(\mathbf{x}_i) \triangleq [\mathrm{tf}(x_{ij}) \times \mathrm{idf}(j)]_{j=1}^{V}$$

Kernels for comparing documents - Example

$$\text{tf}(x_{ij}) \triangleq \log(1 + x_{ij})$$

$$\text{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)}$$

$$\text{tf-idf}(\mathbf{x}_i) \triangleq [\text{tf}(x_{ij}) \times \text{idf}(j)]_{j=1}^{V}$$

$$\text{df}(j) \triangleq \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)$$

- The number of times each word occurs in a particular document (x_{ij}) which belongs to a collection of 811,400 documents and the number of documents in which each word occurs (document frequency) is given.

x	Doc
car	27
auto	3
insurance	3
best	14

	df	idf
car	18164	1.65
auto	6722	2.08
insurance	19240	1.63
best	25234	1.51

Doc	tf	tf-idf
car	1.45	2.39
auto	0.60	1.25
insurance	0.60	0.98
best	1.18	1.78

$$\phi(\mathbf{x}) = \text{tf-idf}(\mathbf{x})$$

$\phi(x)$ can be used for comparing documents

- idf, tf, tf-idf are calculated

Mercer (positive definite) kernels

- Gram matrix is defined as

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ & \vdots & \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

- If the Gram matrix is positive definite for any set of inputs, the Kernel is a Mercer kernel
- Mercer's theorem: If the Gram matrix is positive definite, we can compute an eigenvector decomposition of it as follows: $\mathbf{K} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$

  where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$
- Now consider an element of $\mathbf{K}$

$$k_{ij} = (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i})^T (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:j})$$

$$k_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \qquad \phi(\mathbf{x}_i) = \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:i}$$

Mercer (positive definite) kernels

- In general, if the kernel is Mercer, then there exists a function ϕ mapping $\mathbf{x} \in X$ to R^D such that

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

- For example, consider the (non-stationary) polynomial kernel $\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$

  If $M = 2$, $\gamma = r = 1$ and $\mathbf{x}, \mathbf{x}' \in R^2$, we have

$$
\begin{aligned}
(1 + \mathbf{x}^T \mathbf{x}')^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\
&= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1)^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2'
\end{aligned}
$$

  This can be written as $\phi(\mathbf{x})^T \phi(\mathbf{x}')$, where

$$\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T$$

Linear kernels

- Deriving the feature vector implied by a kernel is in general quite difficult, and only possible if the kernel is Mercer.

- However, deriving a kernel from a feature vector is easy

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

- If $\varphi(\mathbf{x}) = \mathbf{x}$, we get the linear kernel, defined by κ

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- This is useful if the original data is already high dimensional, and if the original features are individually informative
- Not all high dimensional problems are linearly separable.

Matern kernels

- The Matern kernel, which is commonly used in Gaussian process regression

$$\kappa(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}r}{\ell} \right)$$

  Where $r = ||\mathbf{x} - \mathbf{x}'||$, $\nu > 0$, $\ell > 0$, and K_ν is a modified Bessel function

- As $\nu \to \infty$, this approaches the SE kernel. If $\nu = 1/2$, the kernel simplifies to

$$\kappa(r) = \exp(-r/\ell)$$

String kernels

- The real power of kernels arises when the inputs are structured objects.

- As an example, we now describe one way of comparing two variable length strings using a string kernel

- Consider two strings $\mathbf{x}$ and $\mathbf{x}'$ of lengths D, D', each defined over the alphabet A

- $A = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$

- Let $\mathbf{x}$ be the following sequence of length 110

```
IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDYLQEFLGVMNTEWI
```

- and let $\mathbf{x}$ be the following sequence of length 153

```
PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAERLQENLQAYRTFHVLLA
RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKVLQELSQWTVRSIHDLRFISSHQTGIP
```

String kernels

- These strings have the substring LQE in common. We can define the similarity of two strings to be the number of substrings they have in common.

- Now let $\varphi_s(x)$ denote the number of times that substring s appears in string x

- More formally and more generally, let us say that s is a substring of x if we can write $x = usv$ for some (possibly empty) strings u, s and v.

$$\kappa(x, x') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(x) \phi_s(x')$$

   where $w_s \geq 0$ and A^* is the set of all strings (of any length) from the alphabet A

Using kernels inside GLMs

- We define a kernel machine to be a GLM (generalized linear model) where the input feature vector has the form

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \boldsymbol{\mu}_1), \ldots, \kappa(\mathbf{x}, \boldsymbol{\mu}_K)]$$

 where $\mu_k \in X$ are a set of K centroids

- If κ is an RBF kernel, this is called an RBF network

- We will discuss ways to choose the μ_k parameters

- Note that in this approach, the kernel need not be a Mercer kernel.

- We can use the kernelized feature vector for logistic regression by defining

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Ber}(\mathbf{w}^T \phi(\mathbf{x})).$$

Using kernels inside GLMs

- This provides a simple way to define a non-linear decision boundary

- As an example, consider the data coming from the *exclusive or* or *xor* function.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

(a)　　　　　(b)　　　　　(c)
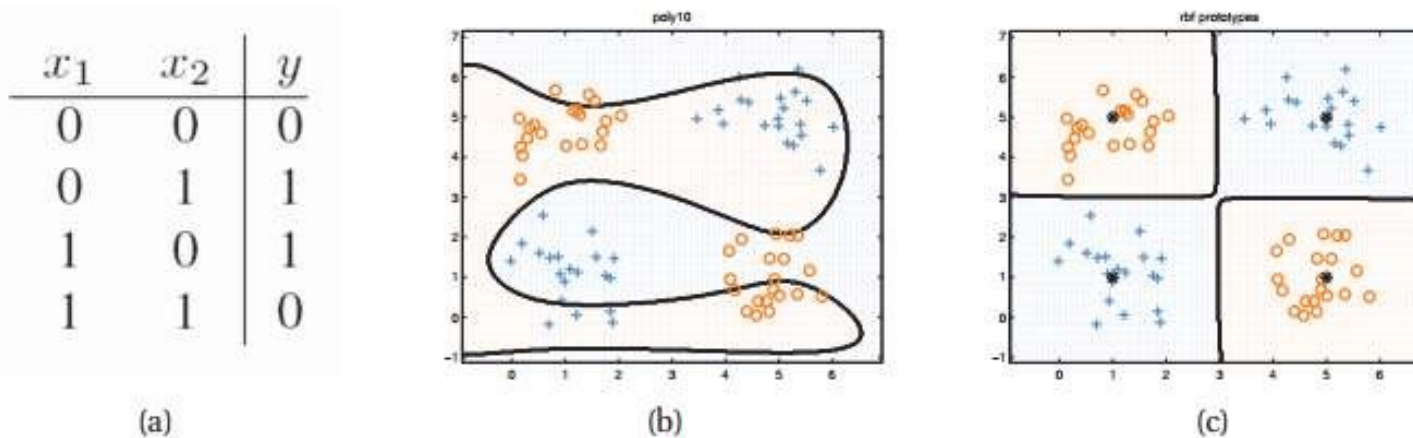
Figure 14.2　(a) xor truth table. (b) Fitting a linear logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses. Figure generated by `logregXorDemo`.

Design Matrix

Consider a simple toy example of classification

(a)
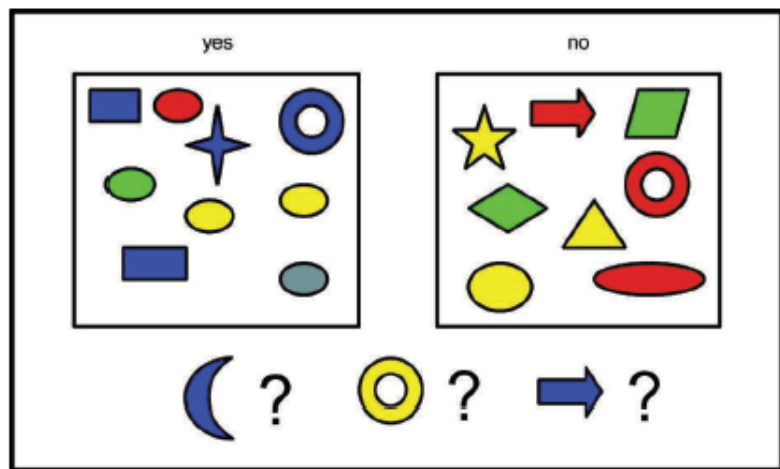
Color	Shape	Size (cm)
Blue	Square	10
Red	Ellipse	2.4
Red	Ellipse	20.7

D features (attributes)

N cases

Label
1
1
0

(b)

Two classes of object which correspond to labels 0 and 1
The inputs are colored shapes as shown in (a). These have
been described by a set of D features or attributes, which
are stored in an $N \times D$ design matrix X, shown in (b).

Design Matrix when a kernelized feature vector is used

RBF Kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

Feature vector:

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \boldsymbol{\mu}_1), \ldots, \kappa(\mathbf{x}, \boldsymbol{\mu}_K)]$$
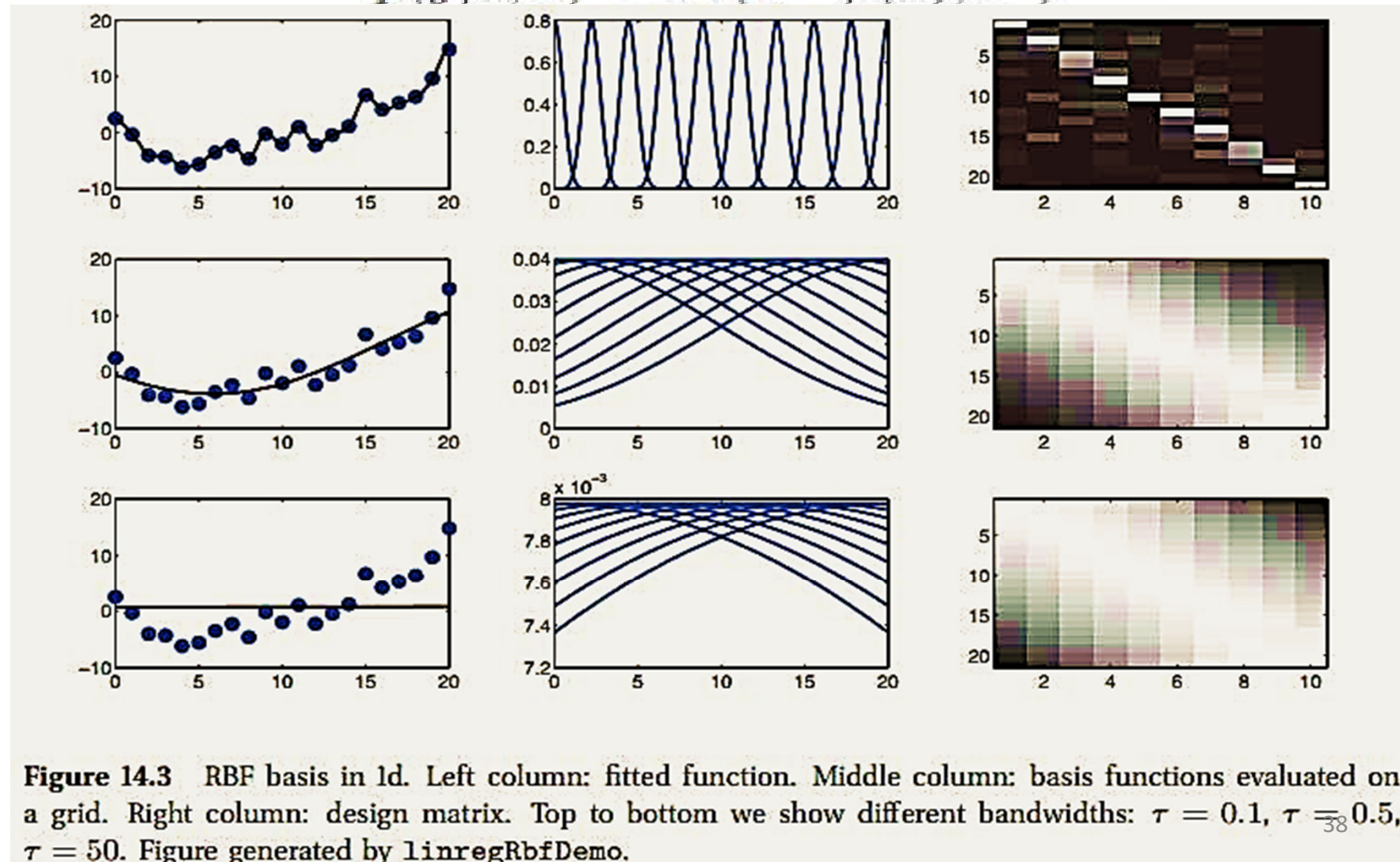
A function is fitted with N data points with K uniformly spaced RBF prototypes ($\mu_1 \ldots \mu_K$). The design matrix is a $N \times K$ matrix given by:

$$\begin{bmatrix} \kappa(\mathbf{x_1}, \boldsymbol{\mu}_1) & \kappa(\mathbf{x_1}, \boldsymbol{\mu}_2) & \ldots & \kappa(\mathbf{x_1}, \boldsymbol{\mu}_K) \\ \kappa(\mathbf{x_2}, \boldsymbol{\mu}_1) & \kappa(\mathbf{x_2}, \boldsymbol{\mu}_2) & \ldots & \kappa(\mathbf{x_2}, \boldsymbol{\mu}_K) \\ \ldots & \ldots & \ldots & \ldots \\ \kappa(\mathbf{x_N}, \boldsymbol{\mu}_1) & \kappa(\mathbf{x_N}, \boldsymbol{\mu}_2) & \ldots & \kappa(\mathbf{x_N}, \boldsymbol{\mu}_K) \end{bmatrix}$$

Using kernels inside GLMs

- Use kernelized feature vector inside a linear regression

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}), \sigma^2).$$

Figure 14.3 RBF basis in 1d. Left column: fitted function. Middle column: basis functions evaluated on a grid. Right column: design matrix. Top to bottom we show different bandwidths: $\tau = 0.1$, $\tau = 0.5$, $\tau = 50$. Figure generated by linregRbfDemo.

L1VMs, RVMs, and other sparse vector machines (L1-Regularized Vector and Relevance vector machine)

- The main issue with kernel machines is: how do we choose the centroids μ_k?

- If the input is low-dimensional Euclidean space, we can uniformly tile the space occupied by the data with prototypes

- However, this approach breaks down in higher numbers of dimensions because of the curse of dimensionality

- A simpler approach is to make each example $\mathbf{x}_i$ be a prototype, so we get

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$$

L1VMs, RVMs, and other sparse vector machines

- Now $D = N$, we have as many parameters as data points
- However, we can use any of the sparsity- promoting priors for w to efficiently select a subset of the training exemplars. We call this a **sparse vector machine**
- Most natural choice is to use L1 regularization resulting in **L1VM** or "L1 regularised vector machine"
- By analogy, we define the use of an L2 regularizer to be a **L2VM** or "L2-regularized vector machine"

L1VMs, RVMs, and other sparse vector machines

- Greater sparsity can be achieved by using Automatic relevance determination (ARD)/ sparse Bayesian learning (SBL) resulting in relevance vector machine or RVM

- Another very popular approach to creating a sparse kernel machine is to use a **support vector machine** or **SVM**

- Rather than using a sparsity-promoting prior, it essentially modifies the likelihood term. Nevertheless, the effect is similar, as we will see
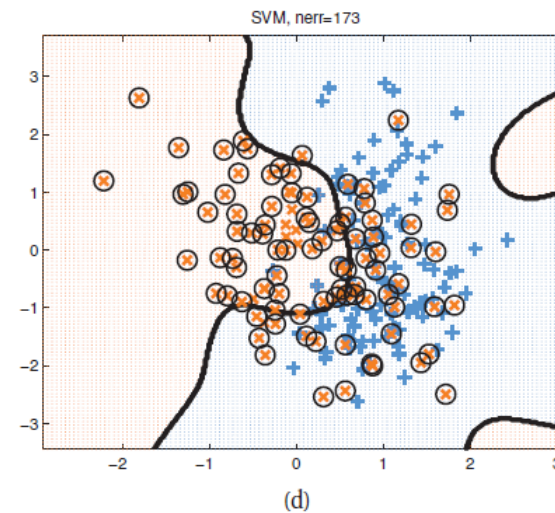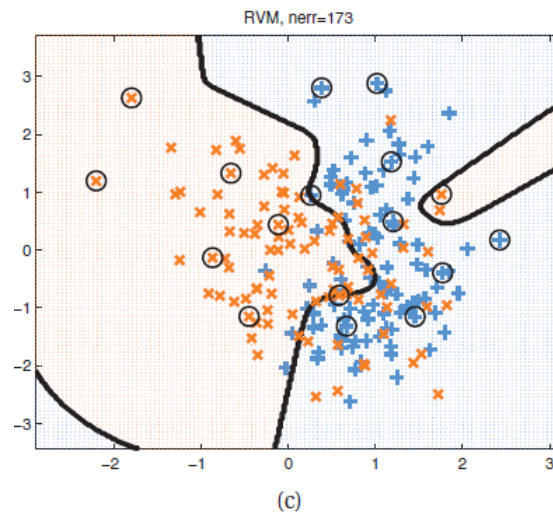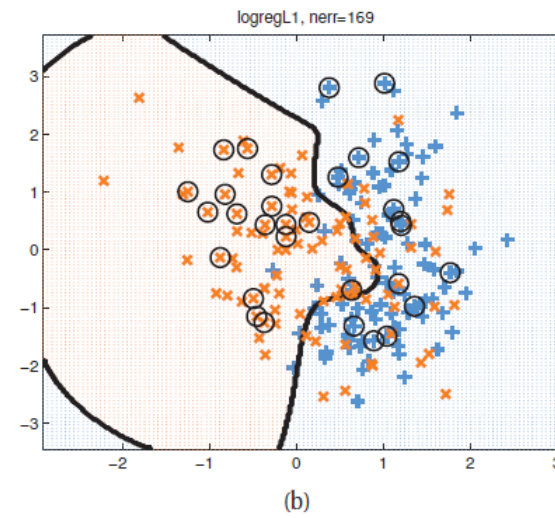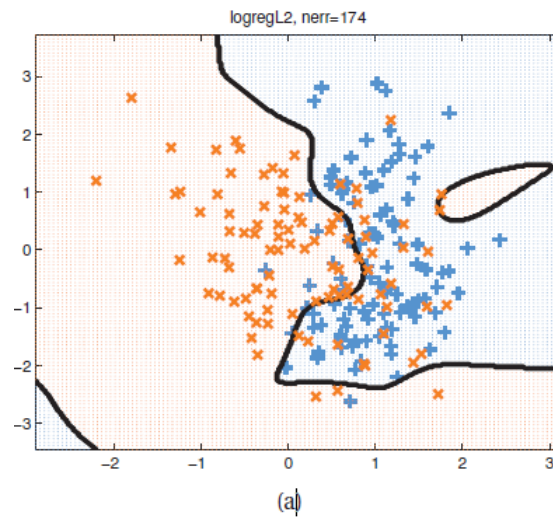
Example of non-linear binary classification using an RBF kernel with bandwidth σ = 0.3. (a) L2VM with λ = 5. (b) L1VM with λ = 1. (c) RVM. (d) SVM with $C = 1/\lambda$ chosen by cross validation. Black circles denote the support vectors
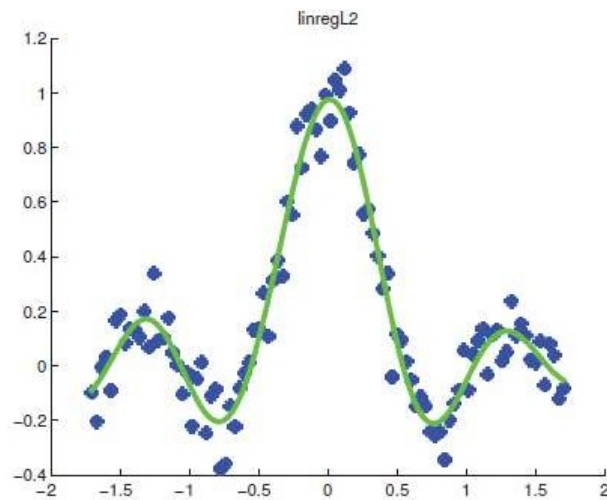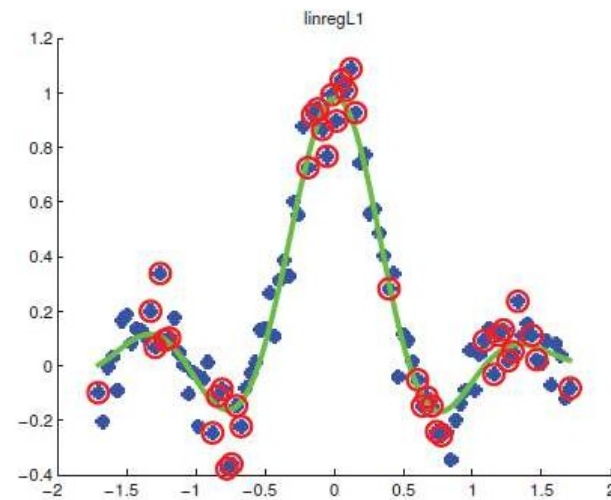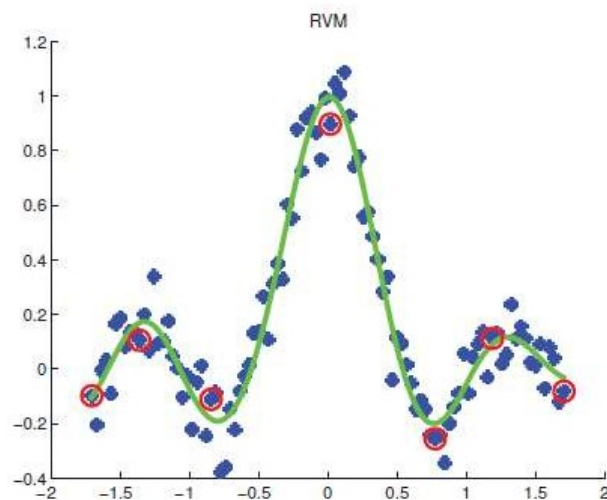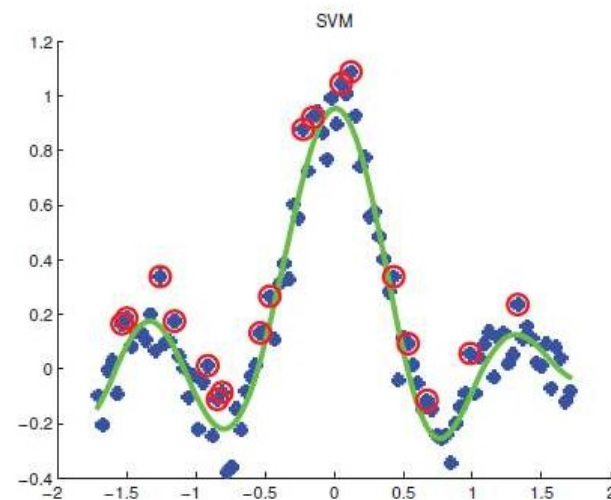
Example of kernel based regression on the noisy sinc function using an RBF kernel with bandwidth σ = 0.3. (a) L2VM with λ = 0.5. (b) L1VM with λ = 0.5. (c) RVM. (d) SVM regression with $C = 1/\lambda$ chosen by cross validation, and ϵ = 0.1. Red circles denote the retained training exemplars.

43

The kernel trick

- Rather than defining our feature vector in terms of kernels, $\boldsymbol{\varphi}(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$, we can work with the original feature vectors $\mathbf{x}$, but modify the algorithm so that it replaces all inner products of the form $< \mathbf{x}, \mathbf{x}' >$ with a call to the kernel function, $\kappa(\boldsymbol{x}, \boldsymbol{x}')$

- This is called the kernel trick.

Kernelized nearest neighbor classification

- Recall that in a 1-NN classifier, we need to compute the Euclidean distance of a test vector to all the training points, find the closest one, and look up its label

- This can be kernelized by observing that

$$||x_i - x_{i'}||_2^2 = \langle x_i, x_i \rangle + \langle x_{i'}, x_{i'} \rangle - 2\langle x_i, x_{i'} \rangle \qquad (14.30)$$

- This allows us to apply the nearest neighbor classifier to structured data objects.

Kernelized K-medoids clustering

- This is similar to K-means, but instead of representing each cluster's centroid by the mean of all data vectors assigned to this cluster, we make each centroid be one of the data vectors themselves

- When we update the centroids, we look at each object (i) that belongs to the cluster (k), and measure the sum of its distances to all the others in the same cluster; we then pick the one which has the smallest such sum

$$m_k = \operatorname*{argmin}_{i:z_i=k} \sum_{i':z_{i'}=k} d(i,i')$$

where z_i is the cluster which i belongs to

$$d(i,i') \triangleq ||\mathbf{x}_i - \mathbf{x}_{i'}||_2^2$$

- This algorithm can be kernelized by using (14.30) to replace the computation of distance "d".

Kernelized ridge regression

- Applying the kernel trick to distance-based methods was straightforward

- It is not so obvious how to apply it to parametric models such as ridge regression

- **The primal problem**
  - Let $\mathbf{x} \in R^D$ be some feature vector, and $\mathbf{X}$ be the corresponding $N \times D$ design matrix
  - Minimize
  $$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda ||\mathbf{w}||^2$$
  - The optimal solution is given by
  $$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y} = (\sum_i \mathbf{x}_i\mathbf{x}_i^T + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y}$$

Kernelized ridge regression

- **The dual problem**
  - Using the matrix inversion lemma

$$\mathbf{w} = \mathbf{X}^T(\mathbf{XX}^T + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$$

  - Takes $O(N^3 + N^2D)$ time to compute. This can be advantageous if D is large

Proof for $X^T \left(XX^T + \lambda I_N\right)^{-1} = \left(X^T X + \lambda I_D\right)^{-1} X^T$

- Start with $\lambda X^T = \lambda X^T$

$$\lambda I_D X^T = \lambda X^T I_N$$

- Add $X^T X X^T$ to both sides

$$X^T X X^T + \lambda I_D X^T = X^T X X^T + \lambda X^T I_N$$

$$\left(X^T X + \lambda I_D\right) X^T = X^T \left(XX^T + \lambda I_N\right)$$

- Left-multiply both sides by $\left(X^T X + \lambda I_D\right)^{-1}$ and right-multiply both sides by $\left(XX^T + \lambda I_N\right)^{-1}$

$$\left(X^T X + \lambda I_D\right)^{-1} \left(X^T X + \lambda I_D\right) X^T \left(XX^T + \lambda I_N\right)^{-1}$$

$$= \left(X^T X + \lambda I_D\right)^{-1} X^T \left(XX^T + \lambda I_N\right) \left(XX^T + \lambda I_N\right)^{-1}$$

- Therefore

$$X^T \left(XX^T + \lambda I_N\right)^{-1} = \left(X^T X + \lambda I_D\right)^{-1} X^T$$

Kernelized ridge regression

- We can partially kernelize this, by replacing $\mathbf{XX}^T$ with the Gram matrix $\mathbf{K}$

- But what about the leading $\mathbf{X}^T$ term?

- Let us define the following **dual variables**:

$$\alpha \triangleq (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$$

- Then we can rewrite the **primal variables** as follows

$$\mathbf{w} = \mathbf{X}^T\alpha = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

- This tells us that the solution vector is just a linear sum of the N training vectors. When we plug this in at test time to compute the predictive mean, we get

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x} = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$$

Kernelized ridge regression

- So we have successfully kernelized ridge regression by changing from primal to dual variables

- This technique can be applied to many other linear models, such as logistic regression

- The cost of computing the dual variables $\boldsymbol{\alpha}$ is $O(N^3)$, whereas the cost of computing the primal variables $\boldsymbol{w}$ is $O(D^3)$

- However, prediction using the dual variables takes $O(ND)$ time, while prediction using the primal variables only takes $O(D)$ time

Support vector machines (SVMs)

- Consider the ℓ_2 regularized empirical risk function

$$J(\mathbf{w}, \lambda) = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \lambda ||\mathbf{w}||^2 \qquad \hat{y}_i = \mathbf{w}^T \mathbf{x}_i + w_0$$

- If L is quadratic loss, this is equivalent to ridge regression

- We can rewrite these equations in a way that only involves inner products of the form $\mathbf{x}^T \mathbf{x}$, which we can replace by calls to a kernel function, $\kappa(\mathbf{x}, \mathbf{x})$

- This is kernelized, but not sparse

- If we replace the quadratic loss with some other loss function, we can ensure that the solution is sparse, so that predictions only depend on a subset of the training data, known as **support vectors**

- This combination of the kernel trick plus a modified loss function is known as a **support vector machine** or **SVM**
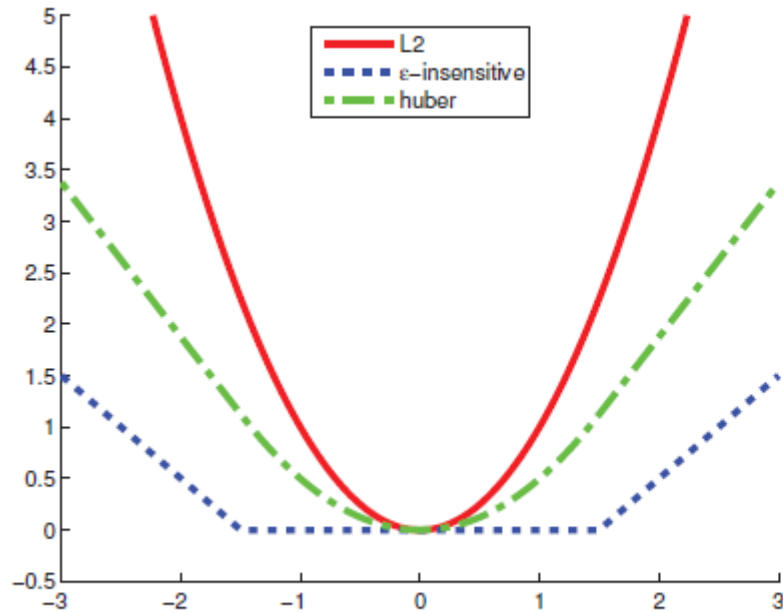
SVMs for regression

- The problem with kernelized ridge regression is that the solution vector w depends on all the training inputs

- We now seek a method to produce a sparse estimate
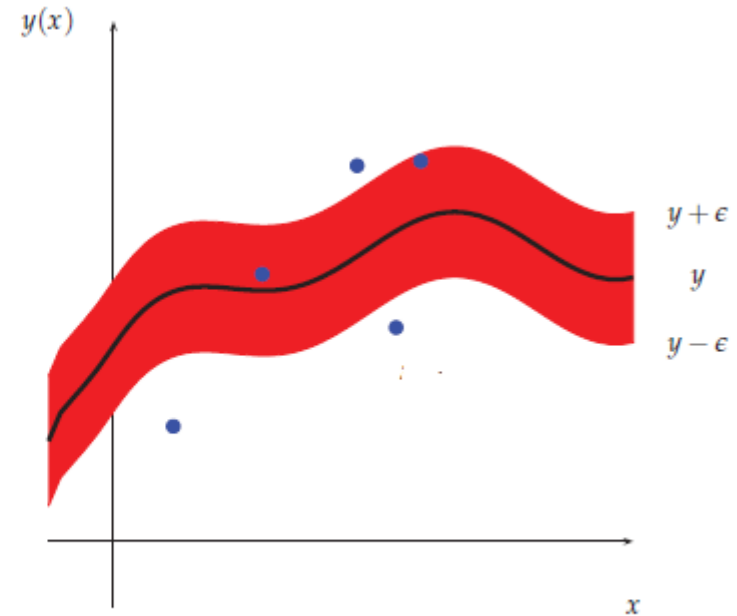
- Consider the epsilon insensitive loss function

$$L_\epsilon(y, \hat{y}) \triangleq \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

- This means that any point lying inside an ϵ-tube around the prediction is not penalized

SVMs for regression

(a)
(b)

(a) Illustration of ℓ_2, Huber and ϵ-insensitive loss functions, where $\epsilon = 1.5$

(b) Illustration of the ϵ-tube used in SVM regression.

Huber Loss: $L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for} |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$

SVMs for regression

- The corresponding objective function

$$J = C \sum_{i=1}^{N} L_\epsilon(y_i, \hat{y}_i) + \frac{1}{2}||\mathbf{w}||^2$$

- where $\hat{y}_i = f(\mathbf{x}_i) = \boldsymbol{w}^T \mathbf{x}_i + w_0$ and $C = 1/\lambda$ is a regularization constant

- This objective is convex and unconstrained, but not differentiable, because of the absolute value function in the loss term

SVMs for regression

- One popular approach is to formulate the problem as a constrained optimization problem

- In particular, we introduce slack variables to represent the degree to which each point lies outside the tube

$$y_i \leq f(\mathbf{x}_i) + \epsilon + \xi_i^+$$
$$y_i \geq f(\mathbf{x}_i) - \epsilon - \xi_i^-$$

- We can rewrite the objective $J = C \sum_{i=1}^{N} L_\epsilon(y_i, \hat{y}_i) + \frac{1}{2}||\mathbf{w}||^2$ as follows:

$$J = C \sum_{i=1}^{N} (\xi_i^+ + \xi_i^-) + \frac{1}{2}||\mathbf{w}||^2$$

- This is a quadratic function of $\mathbf{w}$, and must be minimized subject to the linear constraints as well as the positivity constraints $\xi_i^+ \geq 0$ and $\xi_i^- \geq 0$

SVMs for regression

- This is a standard quadratic program in $2N + D + 1$ variables.
- The optimal solution has the form

$$\hat{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}_i$$

where $\alpha_i \geq 0$

- Furthermore, it turns out that the $\boldsymbol{\alpha}$ vector is sparse, because we don't care about errors which are smaller than ϵ. The $\mathbf{x}_i$ for which $\alpha_i > 0$ are called the **support vectors**. These are points for which the errors lie on or outside the ϵ-tube

SVMs for regression

- Once the model is trained, we can then make predictions using

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \hat{\mathbf{w}}^T \mathbf{x}$$

- Plugging in the definition of $\hat{\mathbf{w}}$ we get

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \sum_i \alpha_i \mathbf{x}_i^T \mathbf{x}$$

- Finally, we can replace $\mathbf{x}_i^T \mathbf{x}$ with $\kappa(\mathbf{x}_i, \mathbf{x})$ to get a kernelized solution

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$
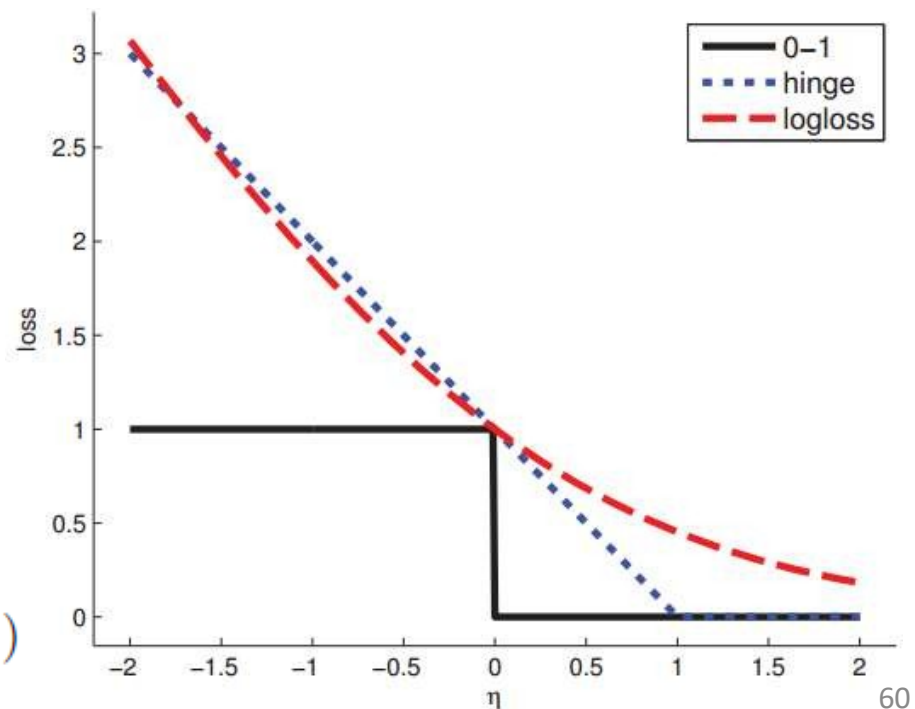
SVMs for classification

- The Hinge loss is defined as
$$L_{\text{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+$$

- We have assumed the labels are $y \in \{1, -1\}, \eta = f(\mathbf{x})$ is our "confidence" in choosing label $y = 1$; however, it need not have any probabilistic semantics

Illustration of various loss functions for binary classification. The horizontal axis is the margin η, the vertical axis is the loss.

Logloss: $L_{\text{nll}}(y, \eta) = \log(1 + e^{-y\eta})$

SVMs for classification

- The overall objective has the form

$$\min_{\mathbf{w}, w_0} \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N}(1 - y_i f(\mathbf{x}_i))_+$$

- Once again, this is non-differentiable, because of the max term. However, by introducing slack variables ξ_i, one can show that this is equivalent to solving

$$\min_{\mathbf{w}, w_0, \xi} \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i \quad \text{s.t.} \quad \xi_i \geq 0, \; y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i, i = 1 : N$$

- This is a quadratic program in $N + D + 1$ variables, subject to $O(N)$ constraints. Standard solvers take $O(N^3)$ time

SVMs for classification

- One can show that the solution has the form

$$\hat{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}_i$$

  where $\boldsymbol{\alpha}$ is sparse (because of the hinge loss)

- The $\boldsymbol{x}_i$ for which $\alpha_i > 0$ are called support vectors; these are points which are either incorrectly classified, or are classified correctly but are on or inside the margin

SVMs for classification

- At test time, prediction is done using

$$\hat{y}(\mathbf{x}) = \mathrm{sgn}(f(\mathbf{x})) = \mathrm{sgn}\left(\hat{w}_0 + \hat{\mathbf{w}}^T\mathbf{x}\right)$$

- Using the kernel trick we have

$$\hat{y}(\mathbf{x}) = \mathrm{sgn}\left(\hat{w}_0 + \sum_{i=1}^{N}\alpha_i \kappa(\mathbf{x}_i, \mathbf{x})\right)$$

This takes $O(sD)$ time to compute, where $s \leq N$ is the number of support vectors. This depends on the sparsity level, and hence on the regularizer C
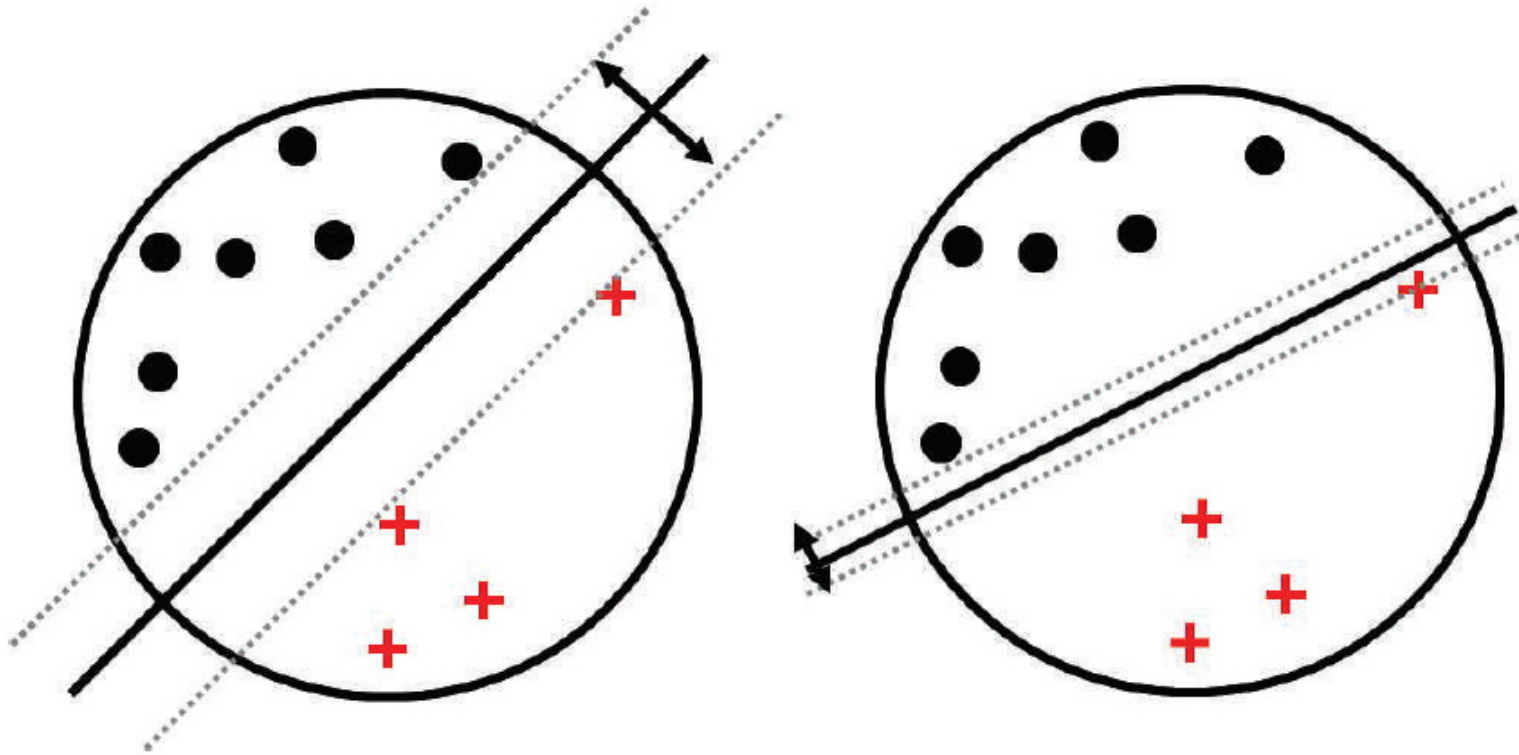
The large margin principle

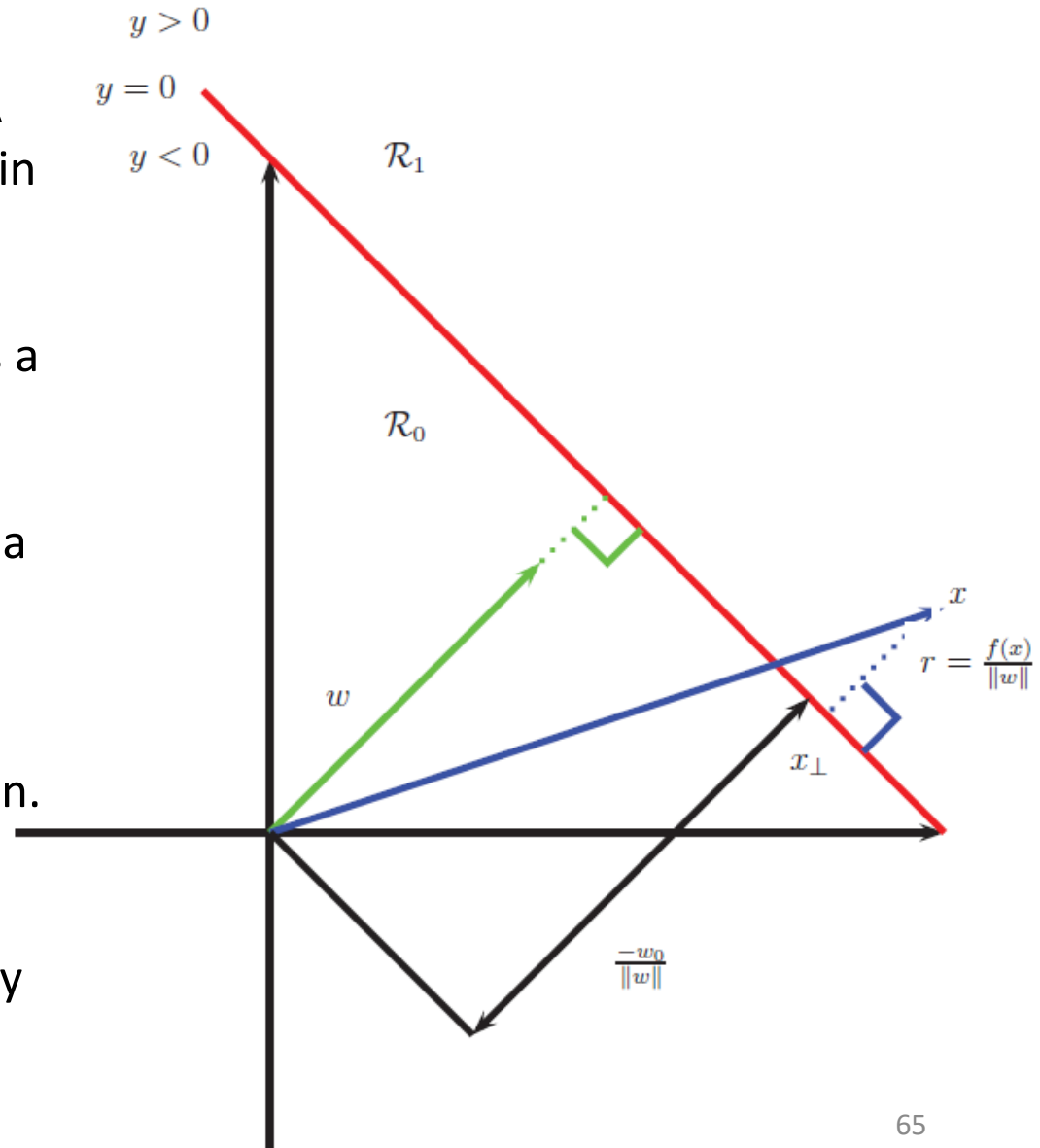Illustration of the large margin principle
Left: a separating hyper-plane with large margin
Right: a separating hyper-plane with small margin

The large margin principle

Illustration of the geometry of a linear decision boundary in 2d. A point **x** is classified as belonging in decision region R_1 if $f(\mathbf{x}) > 0$, otherwise it belongs in decision region R_0; here $f(\mathbf{x})$ is known as a **discriminant function**. The decision boundary is the set of points such that $f(\mathbf{x}) = 0$. $\boldsymbol{w}$ is a vector which is perpendicular to the decision boundary. The term w_0 controls the distance of the decision boundary from the origin. The signed distance of **x** from its orthogonal projection onto the decision boundary, $\boldsymbol{x}_\perp$, is given by $f(\mathbf{x})/||\boldsymbol{w}||$.

$y > 0$

$y = 0$

$y < 0$

$\mathcal{R}_1$

$\mathcal{R}_0$

w

x

$r = \frac{f(x)}{||w||}$

$x_\perp$

$\frac{-w_0}{||w||}$

The large margin principle

- Here, we derive the Equation form a completely different perspective.

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{||\mathbf{w}||}$$

- where r is the distance of $\mathbf{x}$ from the decision boundary whose normal vector is $\mathbf{w}$, and $\mathbf{x}_\perp$ is the orthogonal projection of $\mathbf{x}$ onto this boundary

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = (\mathbf{w}^T\mathbf{x}_\perp + w_0) + r\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||}$$

- Now $f(\mathbf{x}_\perp) = 0$ so $0 = \mathbf{w}^T\mathbf{x}_\perp + w_0$

- Hence

$$f(\mathbf{x}) = r\frac{\mathbf{w}^T\mathbf{w}}{\sqrt{\mathbf{w}^T\mathbf{w}}} \qquad r = \frac{f(\mathbf{x})}{||\mathbf{w}||}$$

The large margin principle

- We would like to make this distance $r = f(\mathbf{x})/||\boldsymbol{w}||$ as large as possible

- Intuitively, the best one to pick is the one that maximizes the margin, i.e., the perpendicular distance to the closest point

- In addition, we want to ensure each point is on the correct side of the boundary, hence we want $f(\mathbf{x}_i)\, y_i > 0$.

- So our objective becomes

$$\max_{\mathbf{w}, w_0} \min_{i=1}^{N} \frac{y_i(\mathbf{w}^T \mathbf{x}_i + w_0)}{||\mathbf{w}||}$$

The large margin principle

- Our objective:

$$\max_{\mathbf{w}, w_0} \min_{i=1}^{N} \frac{y_i(\mathbf{w}^T \mathbf{x}_i + w_0)}{||\mathbf{w}||}$$

- Note that by rescaling the parameters using $\mathbf{w} \rightarrow k\mathbf{w}$ and $w_0 \rightarrow kw_0$, we do not change the distance of any point to the boundary, since the k factor cancels out when we divide by $||\mathbf{w}||$.
- Therefore let us define the scale factor such that $y_i f_i = 1$ for the point that is closest to the decision boundary
- We therefore want to optimize

$$\min_{\mathbf{w}, w_0} \frac{1}{2} ||\mathbf{w}||^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, i = 1 : N$$

- The constraint says that we want all points to be on the correct side of the decision boundary with a margin of at least 1

Soft margin constraints

- If the data is not linearly separable (even after using the kernel trick), there will be no feasible solution in which $y_i f_i \geq 1$ for all i.

- We replace the hard constraints with the **soft margin constraints** that $y_i f_i \geq 1 - \xi_i$.

- Our objective was:

$$\min_{\mathbf{w}, w_0} \frac{1}{2} ||\mathbf{w}||^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, i = 1 : N$$

- The new objective becomes

$$\min_{\mathbf{w}, w_0, \boldsymbol{\xi}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i \quad \text{s.t.} \quad \xi_i \geq 0, \; y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i$$

Soft margin constraints

- We therefore have introduced slack variables $\xi_i \geq 0$ such that $\xi_i = 0$ if the point is on or inside the correct margin boundary, and $\xi_i = |y_i - f_i|$ otherwise
- $0 < \xi_i \leq 1$ the point lies inside the margin, but on the correct side of the decision boundary
- If $\xi_i > 1$, the point lies on the wrong side of the decision boundary
- Points with circles around them are support vectors.

$y = -1$

$\xi > 1$

$y = 0$

$\xi < 1$
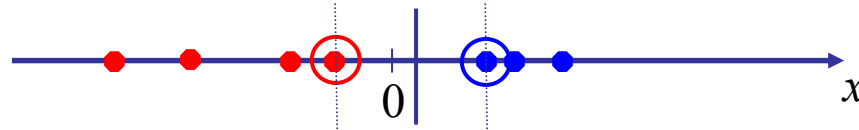
$y = 1$
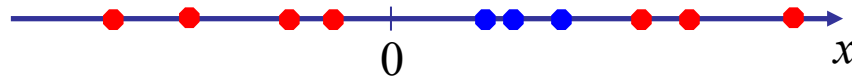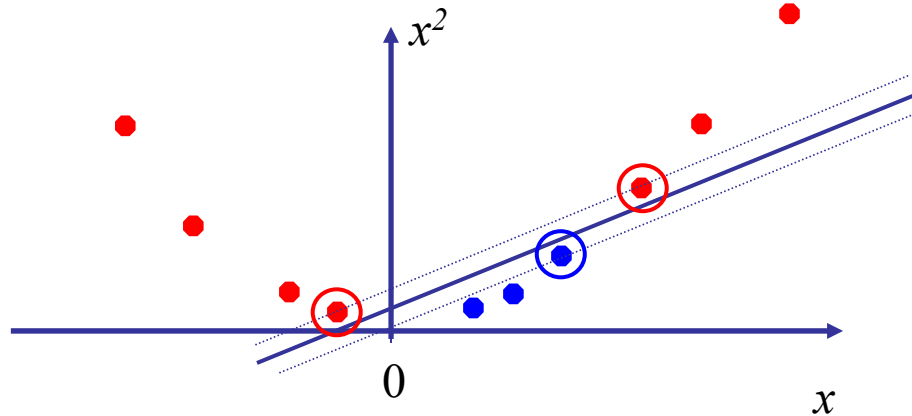
$\xi = 0$

$\xi = 0$
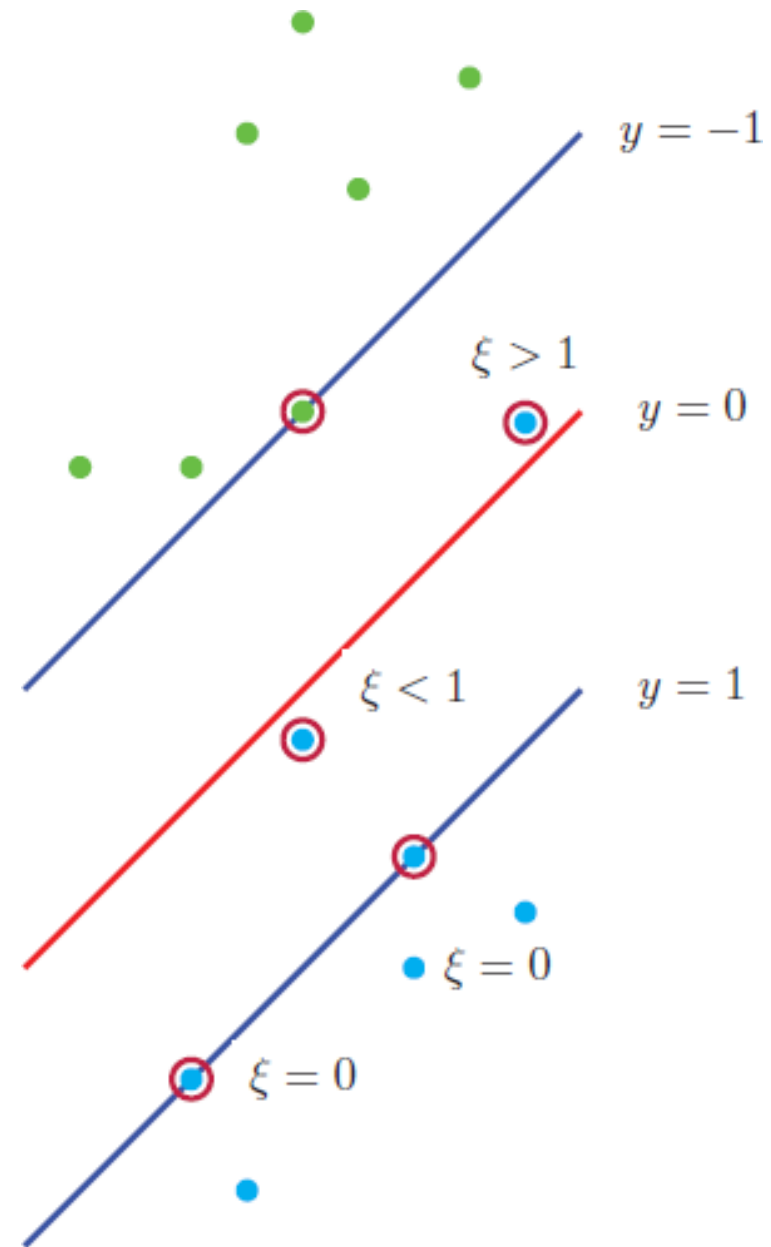
(b)

Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:

- But what are we going to do if the dataset is just too hard?

- How about… mapping data to a higher-dimensional space:

Kernels

Definition: (Gram matrix, kernel matrix)

Gram matrix $G \in \mathbb{R}^{m \times m}$ of kernel k at $\{x_1, \ldots, x_m\}$:

$$\left.\begin{array}{l} \text{Given a kernel } k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \\ \text{and a training set } \{x_1, \ldots, x_m\} \end{array}\right\} \Rightarrow G_{ij} \doteq k(x_i, x_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

Definition: (Feature space, kernel space)

$$\mathcal{K} \doteq span\{\phi(x) \mid x \in \mathcal{X}\} \subset \mathbb{R}^n$$

Kernel technique

Definition:

Matrix $G \in \mathbb{R}^{m \times m}$ is positive semidefinite (PSD)
$$\Leftrightarrow G \text{ is symmetric, and } 0 \le \beta^T G \beta \; \forall \beta \in \mathbb{R}^{m \times m}$$

$$\left. \begin{array}{l} \text{Given a kernel } k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \\ \text{and a training set } \{x_1, \ldots, x_m\} \end{array} \right\} \Rightarrow G_{ij} \doteq k(x_i, x_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathcal{K}}$$

Lemma:

The Gram matrix is symmetric, PSD matrix.

Proof:
$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{n \times m} \Rightarrow G = \mathbf{X}^T \mathbf{X} \in \mathbb{R}^{m \times m}$$
$$0 \le \langle \mathbf{X}\beta, \mathbf{X}\beta \rangle_{\mathcal{K}} = \beta^T G \beta$$

The "Kernel Trick"

- **To produce linear separability in Higher Dimension, the linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^Tx_j$**

- **If every data point is mapped into high-dimensional space via some transformation Φ: $x \to \varphi(x)$, the dot product becomes:**

$$K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$$

- **A *kernel function* is some function that corresponds to an inner product in some expanded feature space.**

- **Example:**

  2-dimensional vectors $x = [x_1 \ x_2]$; let $K(x_i,x_j) = (1 + x_i^Tx_j)^2$,

  Need to show that $K(x_i,x_j) = \varphi(x_i)^T\varphi(x_j)$:

  $K(x_i,x_j) = (1 + x_i^Tx_j)^2$,

  $$= 1+ x_{i1}^2x_{j1}^2 + 2\ x_{i1}x_{j1}\ x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$= [1 \ \ x_{i1}^2 \ \sqrt{2}\ x_{i1}x_{i2} \ \ x_{i2}^2 \ \sqrt{2}x_{i1} \ \sqrt{2}x_{i2}]^T [1 \ \ x_{j1}^2 \ \sqrt{2}\ x_{j1}x_{j2} \ \ x_{j2}^2 \ \sqrt{2}x_{j1} \ \sqrt{2}x_{j2}]$$

$$= \ \varphi(x_i)^T\varphi(x_j), \quad \text{where } \varphi(x) = \ [1 \ \ x_1^2 \ \sqrt{2}\ x_1x_2 \ \ x_2^2 \ \sqrt{2}x_1 \ \sqrt{2}x_2]$$

What Functions are Kernels?

- **For some functions $K(x_i, x_j)$ checking that**

    $$K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j) \text{ can be cumbersome.}$$

- **Mercer's theorem:**

    Every semi-positive definite symmetric function is a kernel

- **Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:**

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_N)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_N)$
…	…	…	…	…
$K(\mathbf{x}_N,\mathbf{x}_1)$	$K(\mathbf{x}_N,\mathbf{x}_2)$	$K(\mathbf{x}_N,\mathbf{x}_3)$	…	$K(\mathbf{x}_N,\mathbf{x}_N)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^\mathsf{T}\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i}, \mathbf{x_j}) = (1 + \mathbf{x_i}^\mathsf{T}\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{\left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2}{2\sigma^2})$$

- Sigmoid: $K(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\beta_0 \mathbf{x_i}^\mathsf{T}\mathbf{x_j} + \beta_1)$

Non-linear SVMs Mathematically

- **Dual problem formulation:**

  Find $\alpha_1...\alpha_N$ such that
  $Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i, x_j)$ is maximized and
  (1) $\Sigma\alpha_i y_i = 0$
  (2) $\alpha_i \geq 0$ for all α_i

- **The solution is:**

  $f(x) = \Sigma\alpha_i y_i K(x_i, x_j) + b$

- **Optimization techniques for finding α_i's remain the same!**

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space

- It does not need to represent the space explicitly, simply by defining a kernel function

- The kernel function plays the role of the dot product in the feature space.

Properties of SVM

- **Flexibility in choosing a similarity function**
- **Sparseness of solution when dealing with large data sets**

  - **only support vectors are used to specify the separating hyperplane**

- **Ability to handle large feature spaces**

  - **complexity does not depend on the dimensionality of the feature space**

- **Overfitting can be controlled by soft margin approach**
- **Nice math property:** a simple convex optimization problem which is guaranteed to converge to a single global solution
- **Feature Selection**

SVM Applications

- **SVM has been used successfully in many real-world problems**

    - text (and hypertext) categorization

    - image classification

    - bioinformatics (Protein classification, Cancer classification)

    - hand-written character recognition

Probabilistic output

- An SVM classifier produces a hard-labeling, $y(\mathbf{x}) = sign(f(\mathbf{x}))$.

- However, we often want a measure of confidence in our prediction

- One heuristic approach is to interpret $f(\mathbf{x})$ as the log-odds ratio, $\log(p(y = 1|\mathbf{x})/p(y = 0|\mathbf{x}))$

$$p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(af(\mathbf{x}) + b)$$

- where *a, b* can be estimated by maximum likelihood on a separate validation set

- However, the resulting probabilities are not particularly well calibrated

Probabilistic output
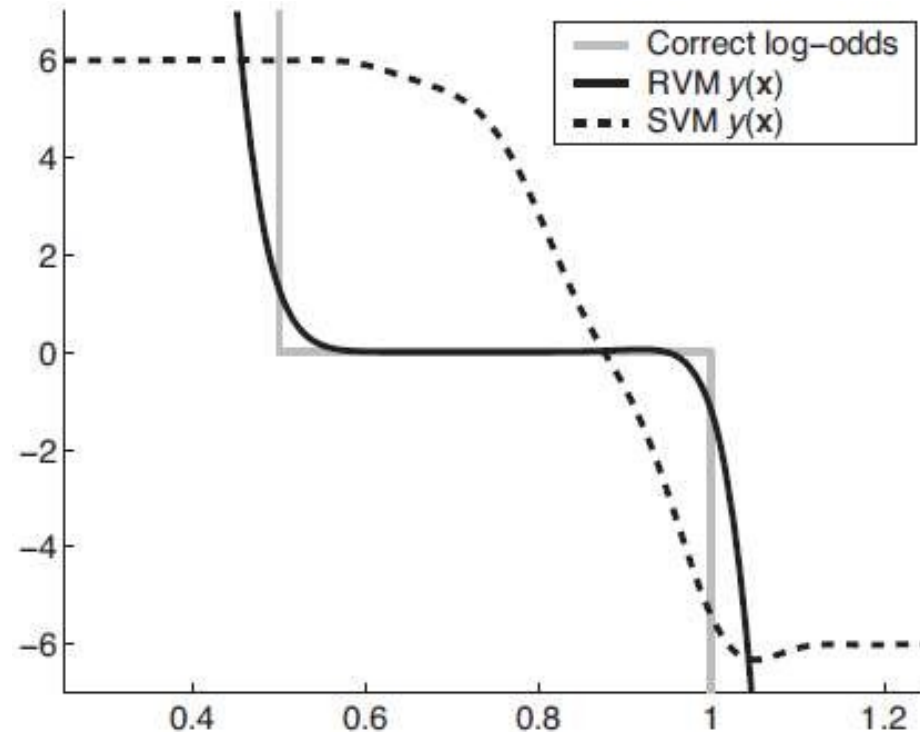
Log-odds vs *x* for 3 different methods

Suppose we have 1d data where $p(x/y = 0)$ =Unif(0, 1) and $p(x/y = 1)$ = Unif(0.5, 1.5). Since the class-conditional distributions overlap in the middle, the log-odds of class 1 over class 0 should be zero in [0.5, 1.0], and infinite outside this region.

SVMs for multi-class classification

- A binary logistic regression model is "upgraded" to the multi-class case, by replacing the sigmoid function with the softmax, and the Bernoulli distribution with the multinomial.

- Upgrading an SVM to the multi-class case is not so easy, since the outputs are not on a calibrated scale and hence are hard to compare to each other

- The obvious approach is to use a **one-versus-the-rest** approach (also called **one-vs-all**), in which we train C binary classifiers, $f_c(\mathbf{x})$, where the data from class c is treated as positive, and the data from all the other classes is treated as negative

SVMs for multi-class classification

- However, this can result in regions of input space which are ambiguously labeled.

- The green region is predicted to be both class 1 and class 2.
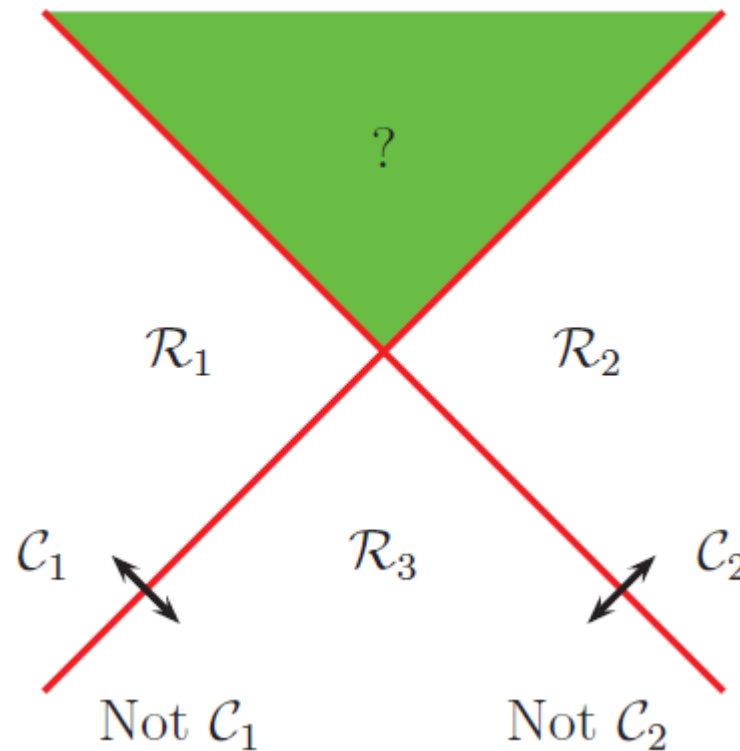
SVMs for multi-class classification

- Another approach is to use the **one-versus-one** or OVO approach, also called **all pairs**, in which we train $C(C-1)/2$ classifiers to discriminate all pairs $f_{C,C'}$

- We then classify a point into the class which has the highest number of votes. However, this can also result in ambiguities
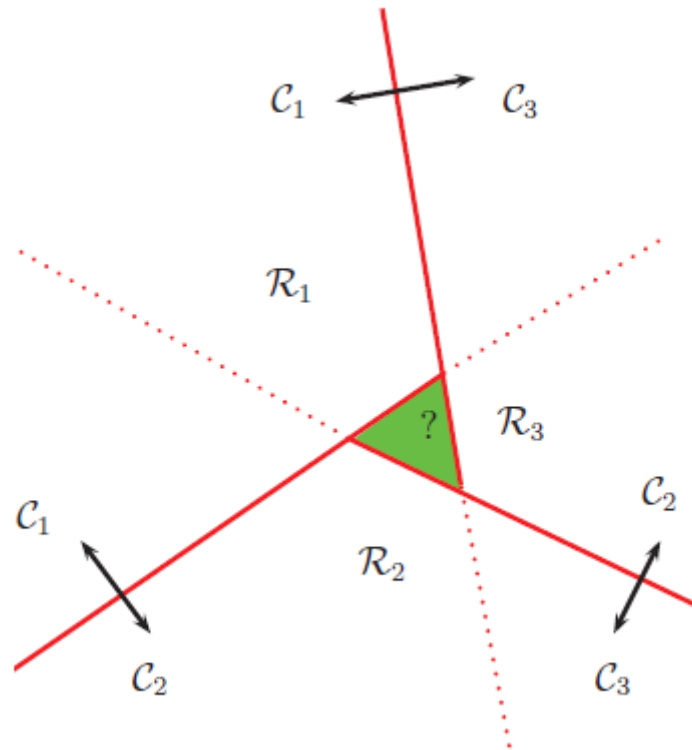
Choosing C

- Typically C is chosen by cross-validation.

- C interacts quite strongly with the kernel parameters.

- To choose C efficiently, one can develop a path following algorithm

- The basic idea is to start with λ large, so that the margin $1/||\boldsymbol{w}(\lambda)||$ is wide, and hence all points are inside of it and have $\alpha_i = 1$

- By slowly decreasing λ, a small set of points will move from inside the margin to outside, and their α_i values will change from 1 to 0, as they cease to be support vectors

SVM vs. Other Methods

	NB	Dec. Trees	kNN	linear SVM $C = 0.5$	linear SVM $C = 1.0$	rbf-SVM $\sigma \approx 7$
earn	96.0	96.1	97.8	98.0	98.2	98.1
acq	90.7	85.3	91.8	95.5	95.6	94.7
money-fx	59.6	69.4	75.4	78.8	78.5	74.3
grain	69.8	89.1	82.6	91.9	93.1	93.4
crude	81.2	75.5	85.8	89.4	89.4	88.7
trade	52.2	59.2	77.9	79.2	79.2	76.6
interest	57.6	49.1	76.7	75.6	74.8	69.1
ship	80.9	80.9	79.8	87.4	86.5	85.8
wheat	63.4	85.5	72.9	86.6	86.8	82.4
corn	45.2	87.7	71.4	87.5	87.8	84.6
microavg.	72.3	79.4	82.6	86.7	87.5	86.4

SVM classifier break-even F1 results are shown for the 10 largest categories and for micro-averaged performance over all 90 categories on the Reuters-21578 data set. (https://nlp.stanford.edu/IR-book/html/htmledition/experimental-results-1.html)

Summary of key points

- Summarizing the above discussion, we recognize that SVM classifiers involve three key ingredients:
    - The kernel trick : prevent underfitting
    - Sparsity, large margin principle : prevent overfitting