Lecture 5: Kernel machines

Helen Yannakoudakis & Oana Cocarascu

Department of Informatics
King's College London

(Version 1.5)

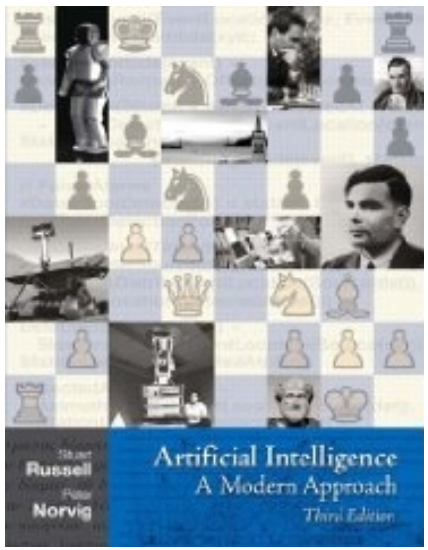
Today

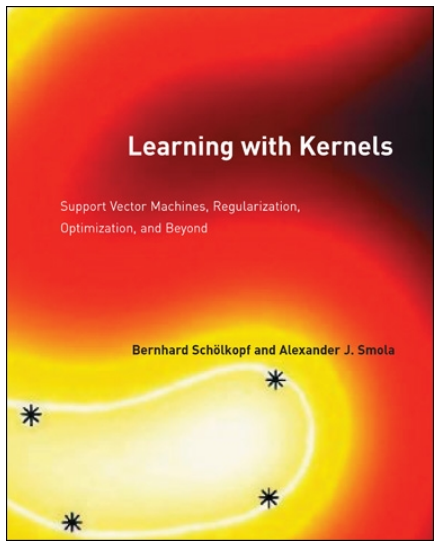
- Introduction
- Inductive Learning
- Probabilistic models 1
- Probabilistic models 2
- Kernel Methods
- Neural Networks
- Evolutionary Algorithms
- Reinforcement Learning 1
- Reinforcement Learning 2
- Learning from Demonstration

Today

- Support vector machines
- Support vector regression
- Gaussian process models
- Non-parametric models

- Today's lecture is from:





Kernel machines

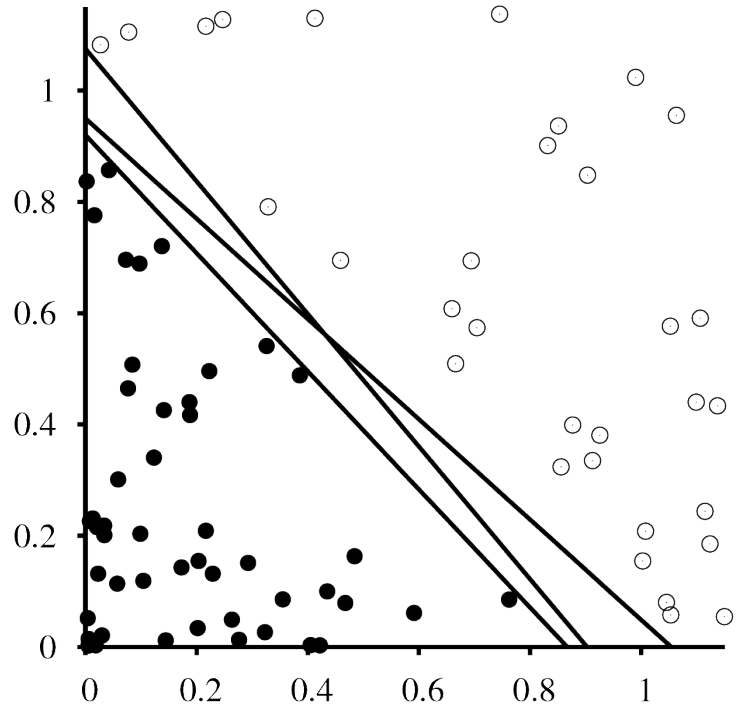
- Start by thinking about linear classifiers.
- Easy to learn and understand
  - Gradient descent.
- But they can only learn linear decision boundaries.
- Could combine many linear classifiers to learn complex boundaries
  - Neural networks.
- But, difficult to train.
- Many local minima/maxima in the high dimensional weight space.

Kernel machines

- Efficient training
- Complex non-linear functions
- Support Vector Machines (SVMs)

(shutterstock.com, food-iepics)

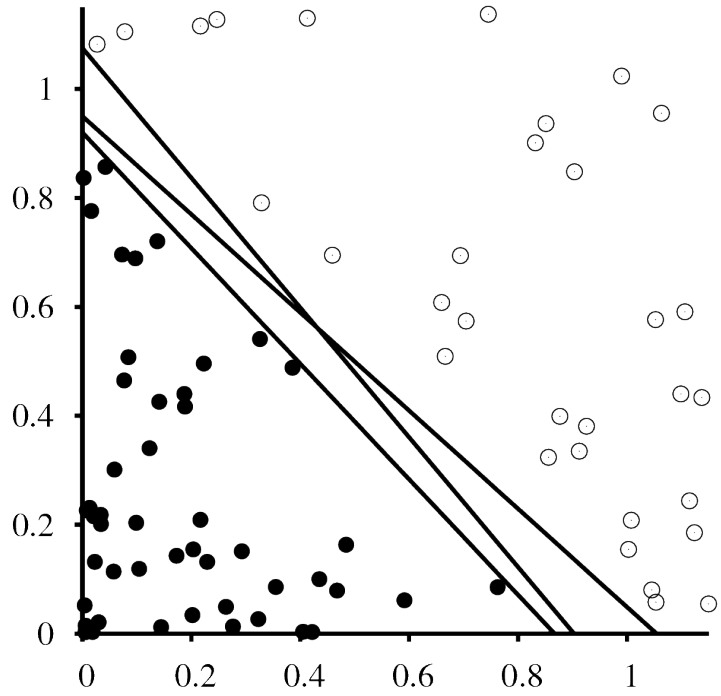
Simple linear classifier



(Russell & Norvig)

- Binary classifier, three possible solutions.

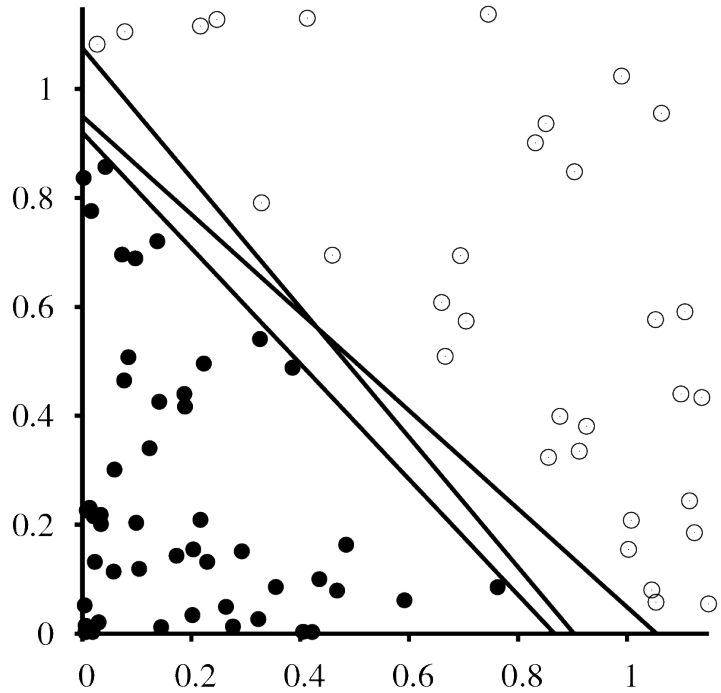
Simple linear classifier



(Russell & Norvig)

- All are consistent with the examples.
- Same accuracy etc.
- BUT not equal — not all examples are equal.

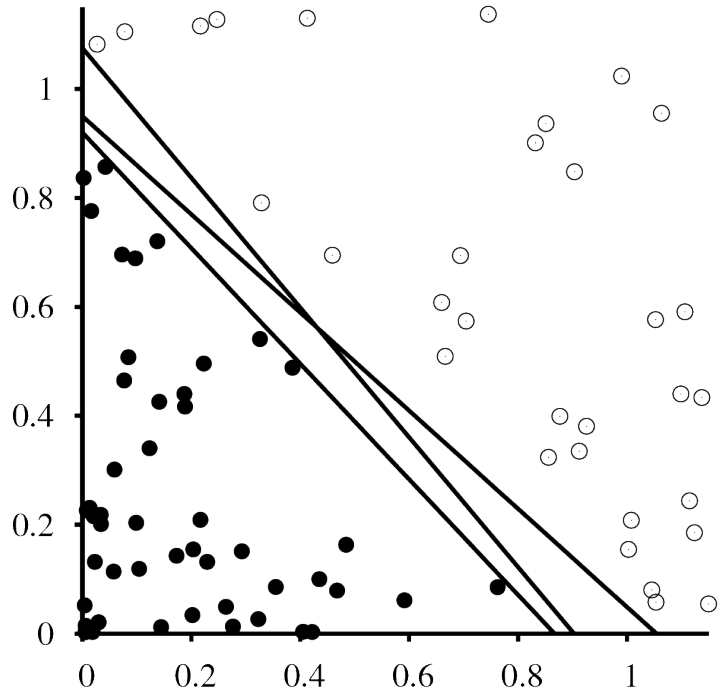
Simple linear classifier



(Russell & Norvig)

- Consider lowest line.
- Very close to the black circles.
- Maybe other examples will cross the line.

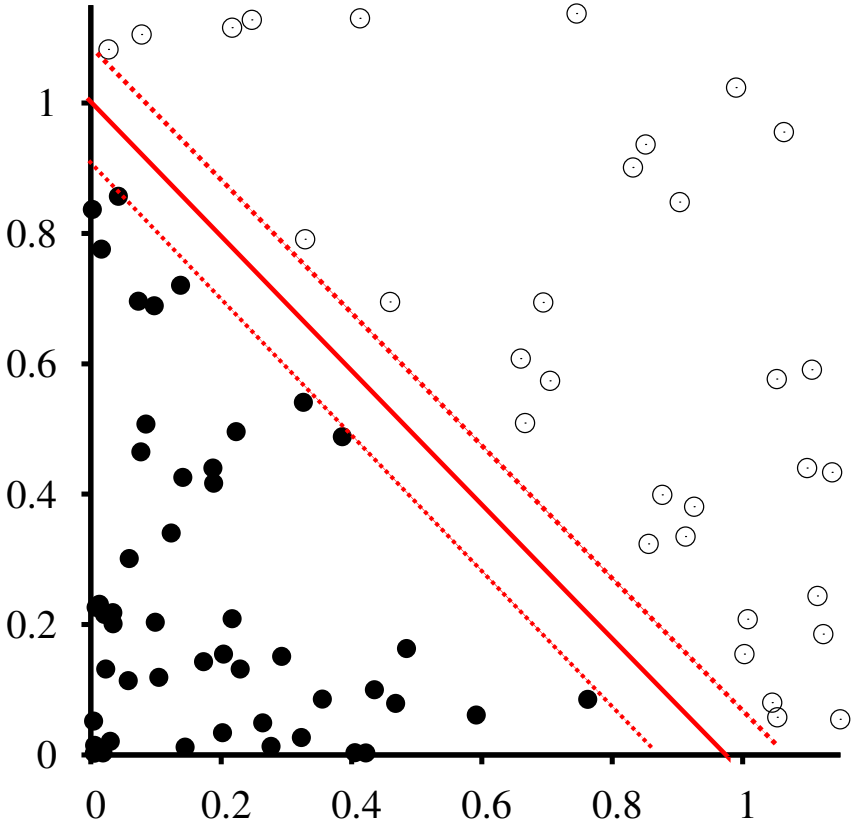
Simple linear classifier



(Russell & Norvig)

- Typical classifers minimize expected empirical loss on training data.
- Kernel machines attempt to minimize generalization loss.

Simple linear classifier

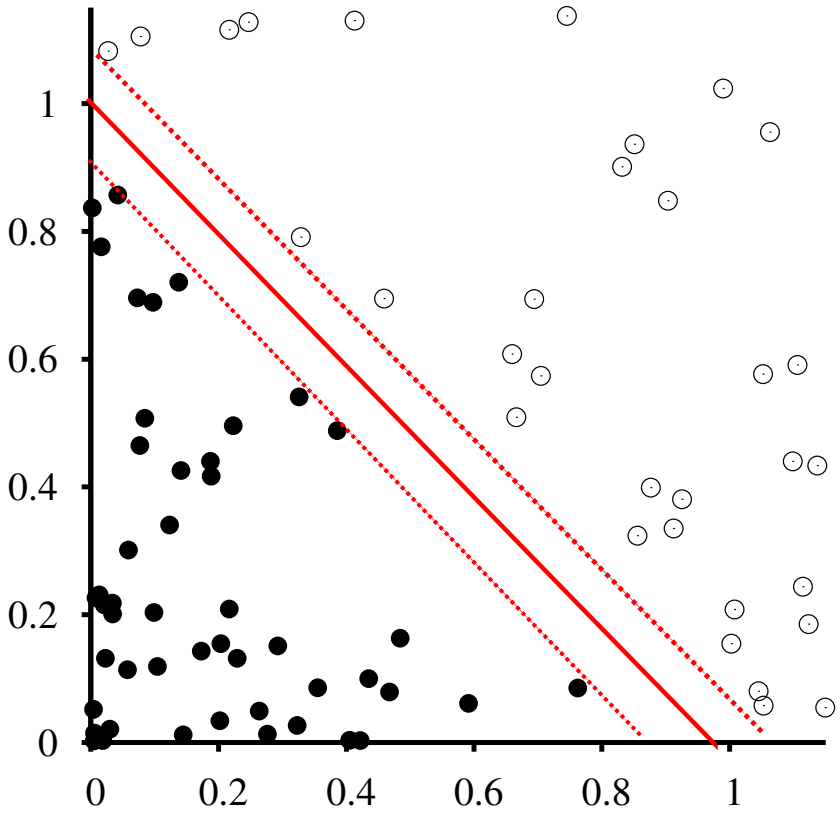


(Russell & Norvig)

- Computational Learning Theory tells us that we do this by maximising the margin on the examples we have seen so far → max margin separator.

- Margin: width of the area bounded by dashed lines (ie, twice the distance from the separator to the nearest datapoint).

What is the maximum margin?



(Russell & Norvig)

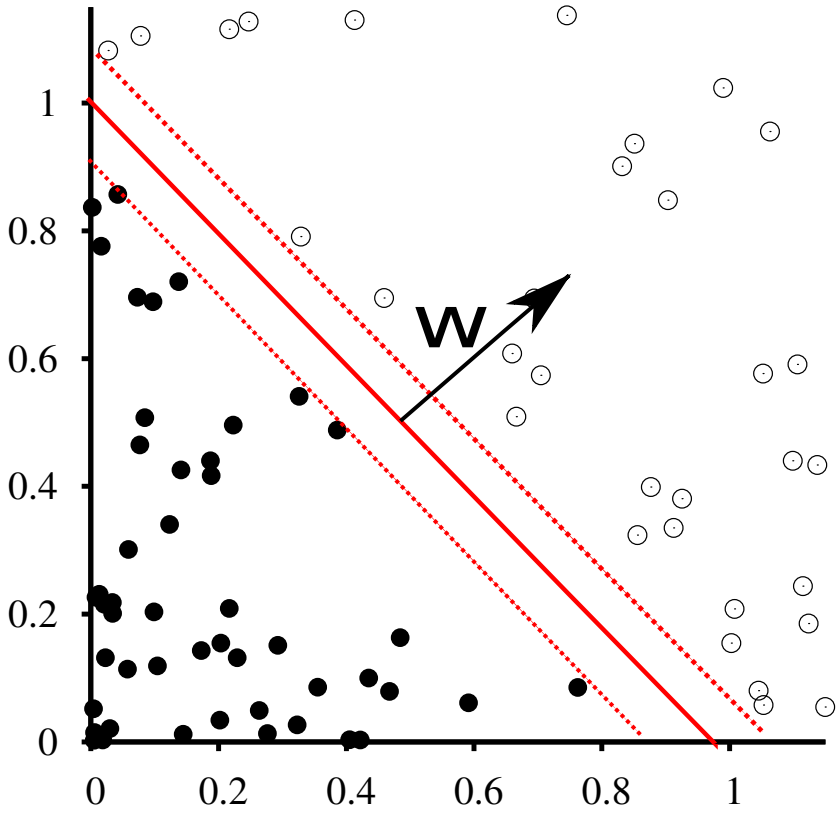
- Linear classifier has the form $y = \mathbf{w}^T\mathbf{x} + b$

Reminder:

$y = w_0 x_0 + w_1 x_1$

w_0 is the intercept (now b)

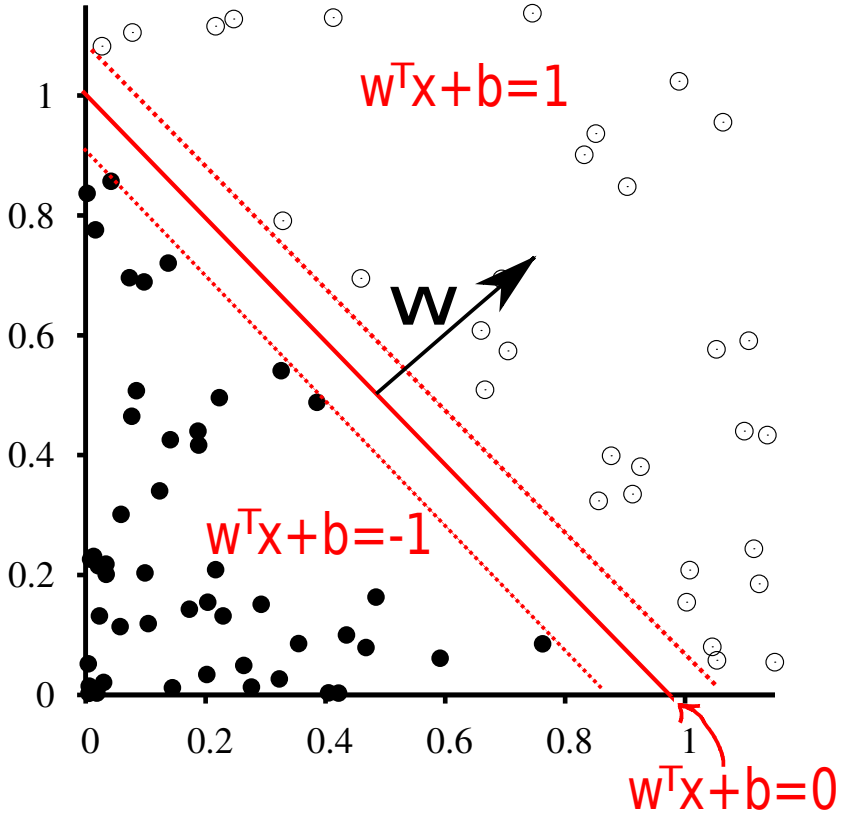
What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- Linear classifier has the form $y = \mathbf{w}^T\mathbf{x} + b$
- Changing **w** will change the decision boundary (separator).
- Separator that has the max distance to the nearest datapoints (in the training set) in either class.

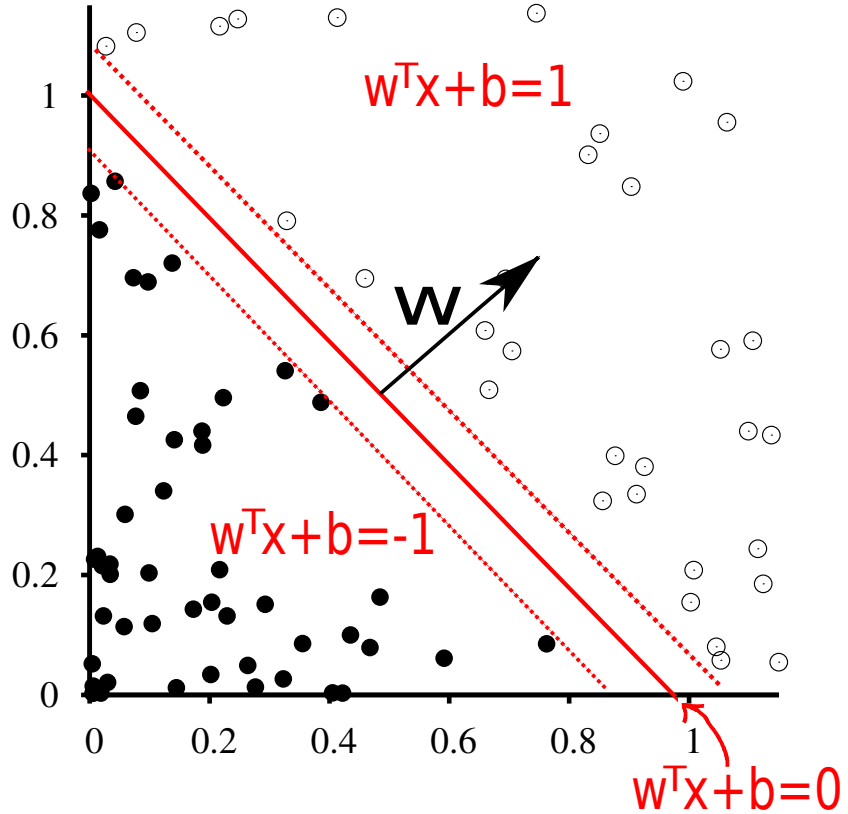
What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- Finding the optimal decision boundary (separator) = Finding the biggest margin.
- Margin: the width of the area bounded by dashed lines.

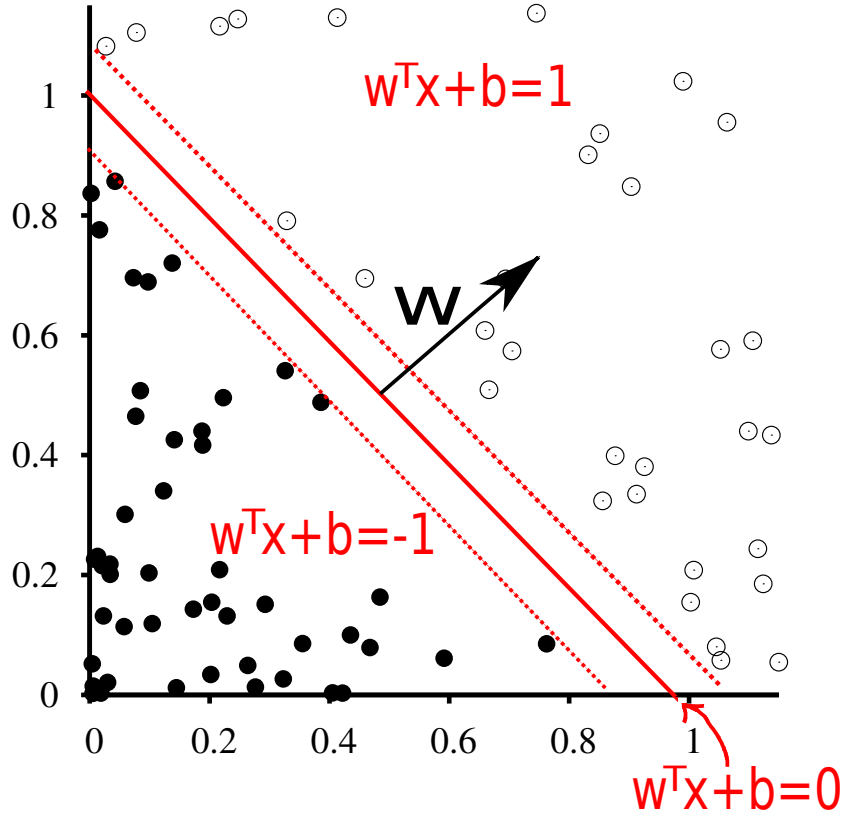
What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- Finding the optimal decision boundary (separator) = Finding the biggest margin.
- Margin: the width of the area bounded by dashed lines.
- Decision function fully specified by a (small) subset of training samples, the support vectors.
- The points closest to the separator are the support vectors (they "hold up" the separator).

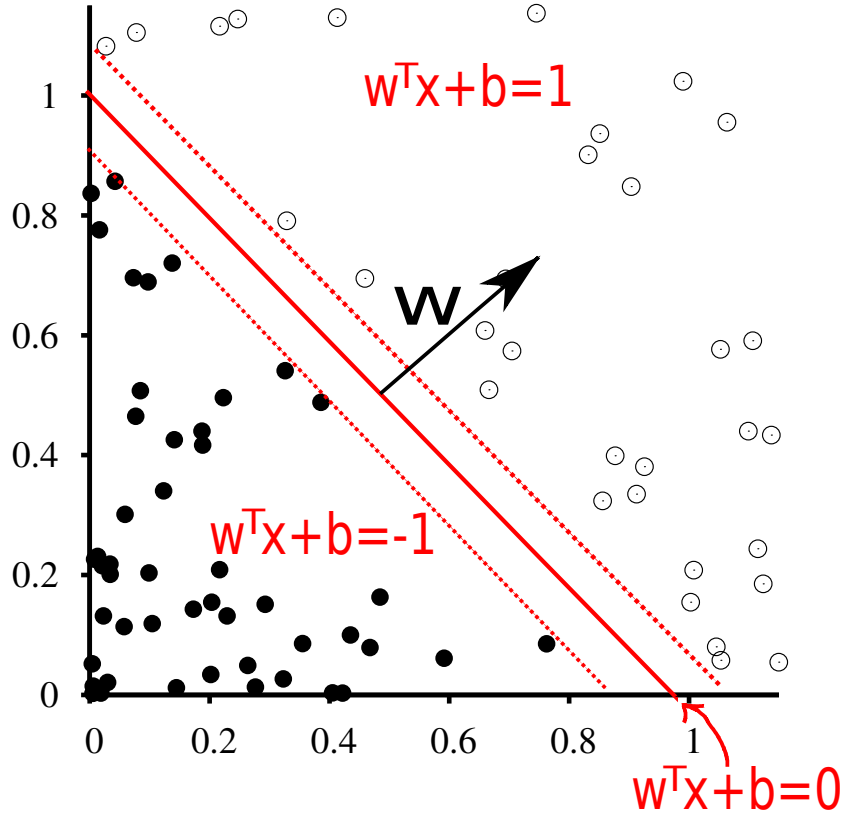
What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- What we want is:
  $\mathbf{w}^T\mathbf{x}_i + b \geqslant 0$ when $y_i = +1$
  $\mathbf{w}^T\mathbf{x}_i + b < 0$ when $y_i = -1$

What is the maximum margin?



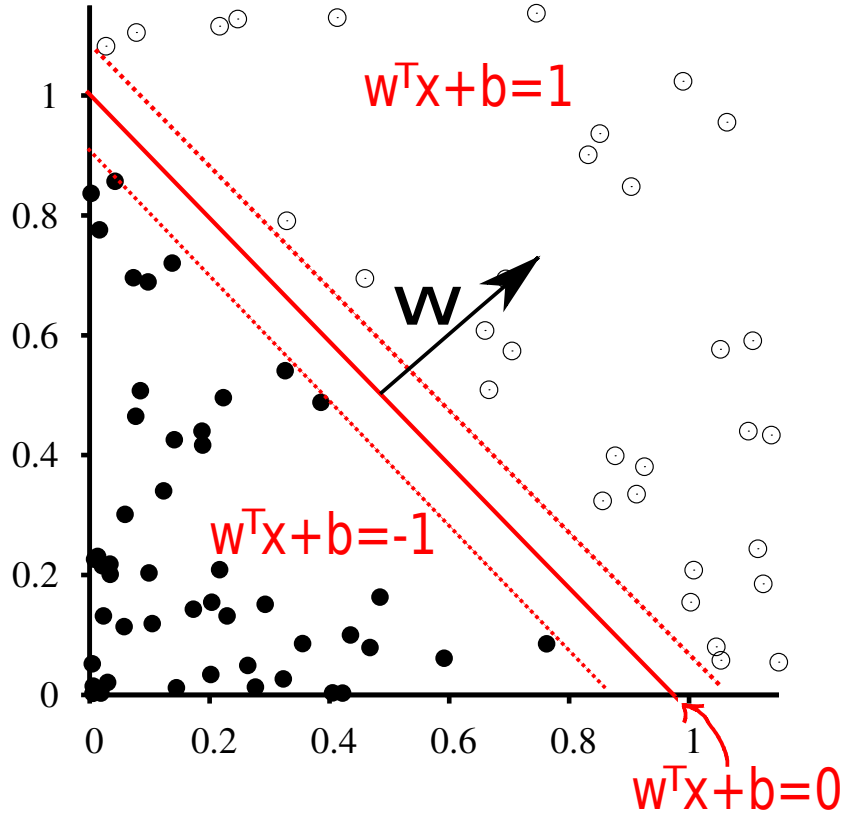
(Russell & Norvig, Brendan Michael)

- What SVMs do:

  $\mathbf{w}^T\mathbf{x}_i + b \geqslant +1$ when $y_i = +1$

  $\mathbf{w}^T\mathbf{x}_i + b \leqslant -1$ when $y_i = -1$

What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- What SVMs do:

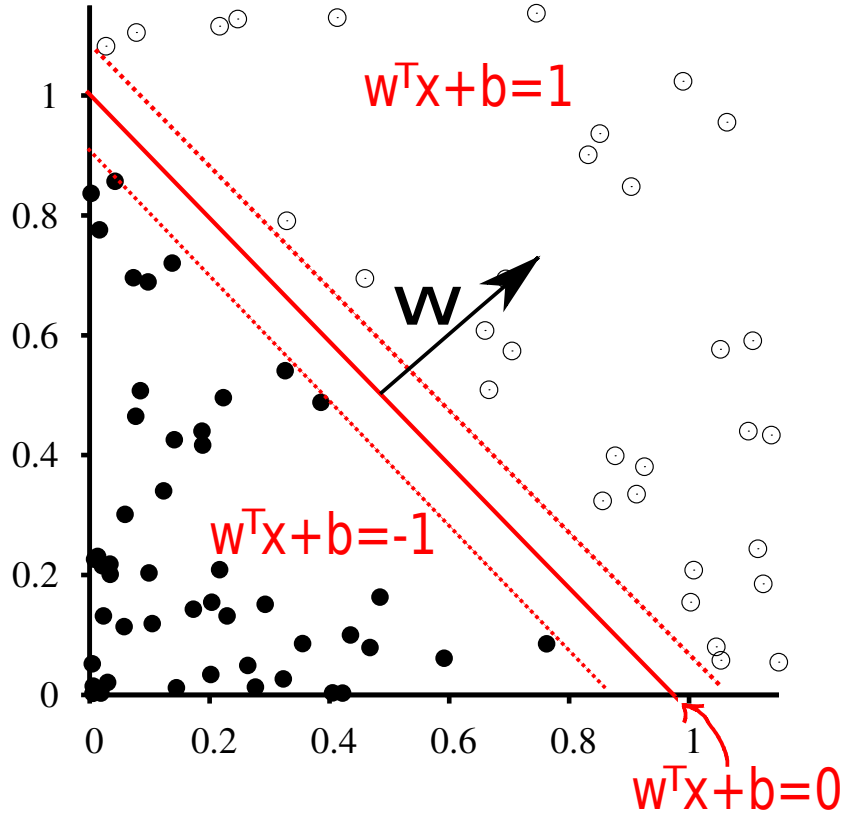
  $\mathbf{w}^T\mathbf{x}_i + b \geqslant +1$ when $y_i = +1$

  $\mathbf{w}^T\mathbf{x}_i + b \leqslant -1$ when $y_i = -1$
- Our margin is then defined by:

  $h1 : \mathbf{w}^T\mathbf{x} + b = +1$

  $h2 : \mathbf{w}^T\mathbf{x} + b = -1$

What is the maximum margin?



(Russell & Norvig, Brendan Michael)

- The distance between h_1 or h_1 and our decision boundary is $\frac{1}{||\mathbf{w}||}$
  - $||\mathbf{w}||$ is norm of vector $\mathbf{w}$ (euclidean length): $\sqrt{w_1^2 + w_2^2}$
- Total distance between h_1 and h_2 is then $\frac{2}{||\mathbf{w}||}$

How do we find **w**?

- Gradient descent to search the space of **w**, look for the maximum margin separator that classifies examples correctly.

$$\max_{\mathbf{w},b} \frac{2}{||\mathbf{w}||}$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1, i = 1, \ldots, m$$

The constraint simply means that:

$\mathbf{w}^T\mathbf{x}_i + b \geqslant +1$ when $y_i = +1$

$\mathbf{w}^T\mathbf{x}_i + b \leqslant -1$ when $y_i = -1$

- ie, learnt parameter gives $+1$ for class $+1$, and -1 for class -1.

How do we find **w**?

- However, this maximisation problem is unwieldy.
- Easier to convert into a quadratic optimization problem with linear constraints

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

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1, i = 1, \ldots, m$$

- Objective function + inequality constraints $\rightarrow$ constrained optimisation problem

How do we find it?

- Such problems are "dealt with" using Lagrange multipliers $\alpha_i \geqslant 0$ and a Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1)$$

- The minimum is found by taking its partial derivatives and setting them equal to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i$$

- Substituting these (and simplifying) we get a dual representation:

$$\arg\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

How do we find it?

- So, in the dual representation, the optimal separator in the input space is found by solving (we like this because of the dot product between feature vectors):

$$\arg\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

where m is the number of training examples, and:

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

Note

For the dual:

$$\arg\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

- The expression:

  $$\mathbf{x}_i \cdot \mathbf{x}_j$$

  is important, so be sure you understand what it means.
- $\mathbf{x}_i$ and $\mathbf{x}_j$ are vectors of features:

  $$\mathbf{x}_i = \{x_{i1}, \ldots x_{in}\}$$
  $$\mathbf{x}_j = \{x_{j1}, \ldots x_{jn}\}$$

  and:

  $$\mathbf{x}_i \cdot \mathbf{x}_j = \sum_{k=1}^{n} x_{ik} x_{jk}$$

- e.g. $[1, 2, 3] \cdot [4, 2, 3] = 17$

How do we find it?

- This is a quadratic programming problem.
  - Good software packages to solve this.
- Can convert back to **w** using:

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

- There is a single, global maximum (convex).
- The data is only involved as a dot product.
- The weights α_i associated with each data point are all zero, except for the points closest to the decision boundary (support vectors).

  (only the support vectors determine the position of the decision boundary)

  (since there are many fewer support vectors than datapoints, we aren't in danger of overfitting)

Decision function

- Remember that for a specific $\mathbf{x}$ we want to know whether it is $+1$ or a -1.
- We can stay in the dual representation.
- If we substitute, the decision function can be written as:

$$h(\mathbf{x}) = sign\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right)$$

- Obviously we only need to do this for the $\mathbf{x}_i$ that are support vectors.

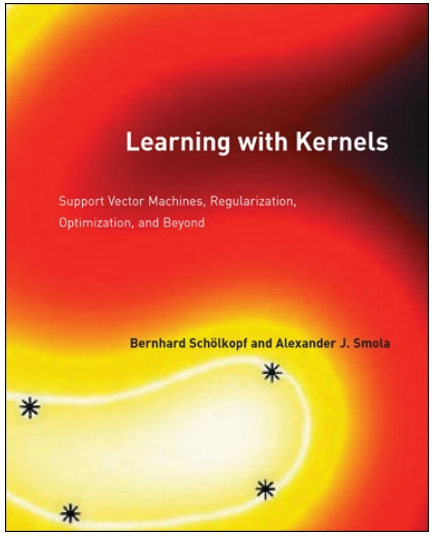
Note

- Establishing the dual representation involves rewriting

$$\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$$

using Lagrange multipliers.

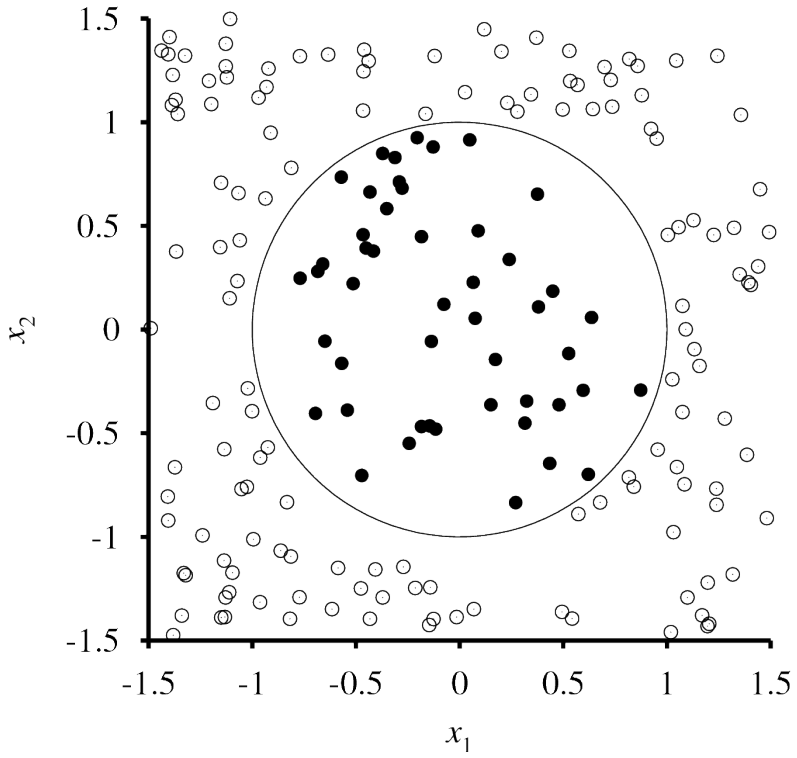
  - For more details on this, see Scholkopf and Smola, pages 11–14.



But. . .

- You promised non-linearity

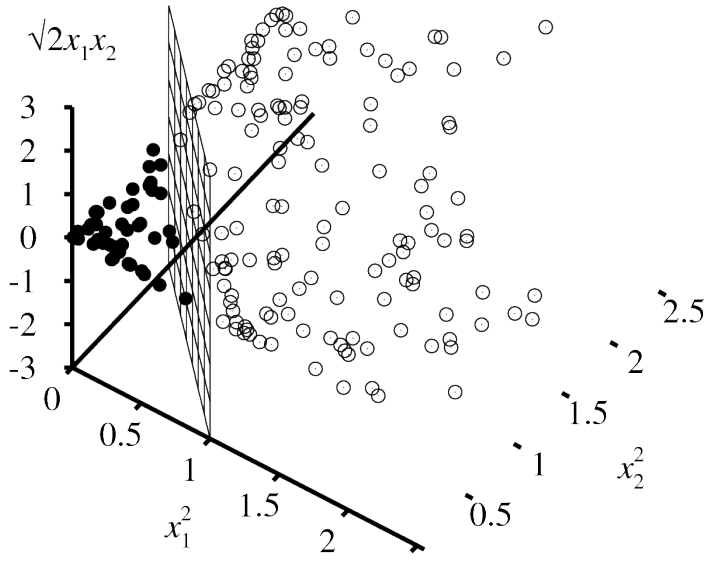
A simple example



(Russell & Norvig)

- Can't separate black circles from white circles with a linear boundary.

A simple example



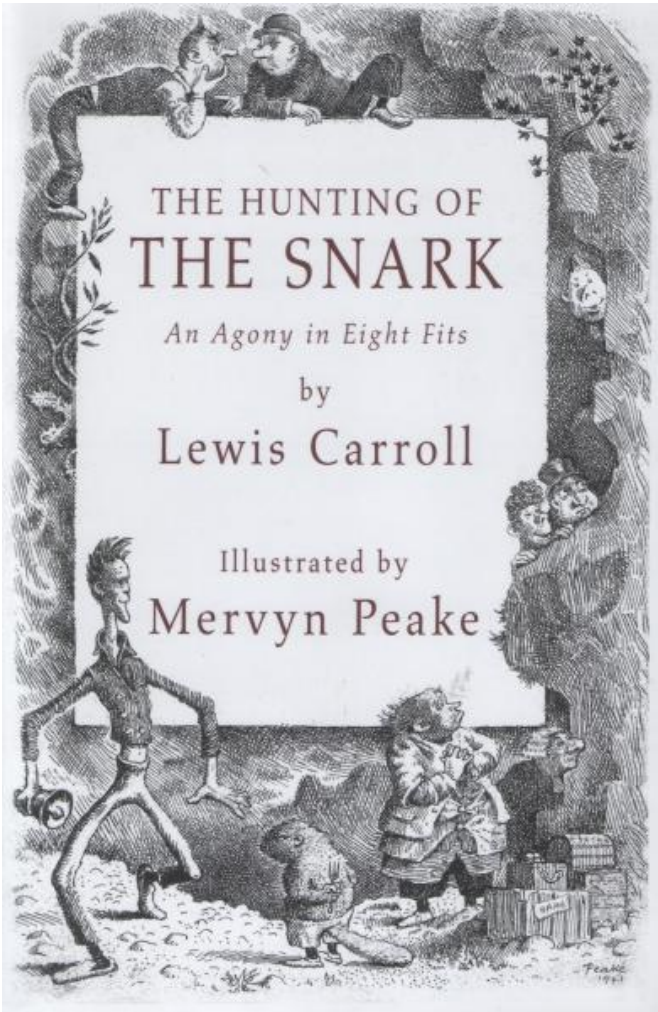
(Russell & Norvig)

- Can separate black circles from white circles with a linear boundary.

A simple example

- In this three dimensional space, the separator (decision boundary) is a plane.
- In a higher dimensional space, the separator will be a hyperplane.

The Kernel Trick, Fit the First



(Mervyn Peake's illustrations to "The Hunting of the Snark")

The Kernel Trick, Fit the First

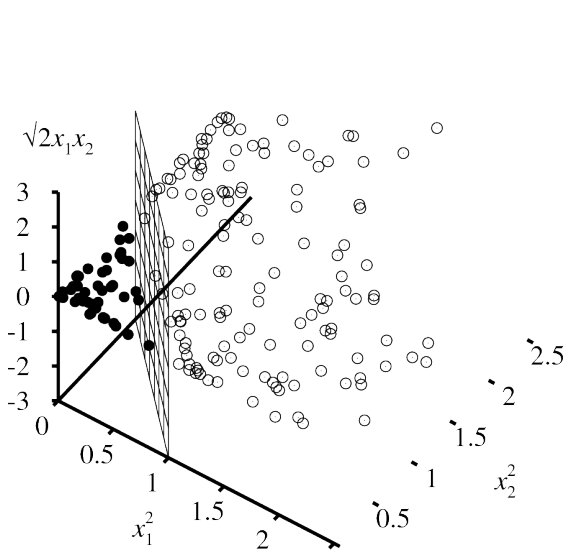
- Initial space:

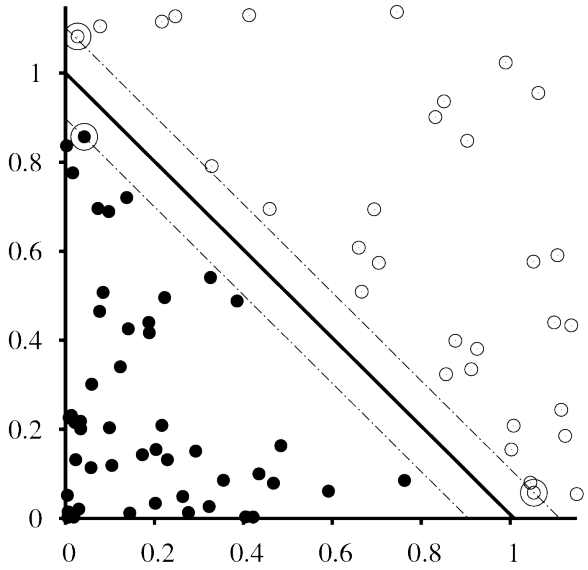
$$\mathbf{x} = (x_1, x_2)$$

- Now, map $\mathbf{x}$ to $F(\mathbf{x})$ such that:

$$f_1 = x_1^2$$
$$f_2 = x_2^2$$
$$f_3 = \sqrt{2} x_1 x_2$$

The Kernel Trick, Fit the First





(Russell & Norvig)

- Projecting into x_1^2, x_2^2 space gives the linear classifier we saw before.

The Kernel Trick, Fit the First

- Data will always be linearly separable if we map them into a space with enough dimensions.
- In general, N data points, will always be linearly separable in $N - 1$ dimensions.

Kernel trick, Fit the Second

- What exactly is a kernel?
- It is the mapping from the input space $\mathbf{x}$ to the higher dimension space.
- Instead of:

$$\arg\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

we write:

$$\arg\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (F(\mathbf{x}_i) \cdot F(\mathbf{x}_j))$$

Kernel Trick, Fit the Second

- We could replace $\mathbf{x}$ by $F(\mathbf{x})$ in any ML algorithm.
- Neat thing is that the dot product has some nice properties.
- Chief among these is that:

$$F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$$

  can be computed without first computing $F()$ for each data point.
- In our simple example:

$$\begin{aligned} F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) &= F(\mathbf{x}_i \cdot \mathbf{x}_j) \\ &= (\mathbf{x}_i \cdot \mathbf{x}_j)^2 \end{aligned}$$

- Mapping into high dimensional space and then taking dot product = dot product squared.

Kernel Trick, Fit the Second

Dot-product of polynomials:

- Degree=1

$$F(\mathbf{u}) \cdot F(\mathbf{v}) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = \mathbf{u} \cdot \mathbf{v}$$

- Degree=2

$$F(\mathbf{u}) \cdot F(\mathbf{v}) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2 =$$
$$= (u_1 v_1 + u_2 v_2)^2 = (\mathbf{u} \cdot \mathbf{v})^2$$

- Can be proven that for any degree d: $F(\mathbf{u}) \cdot F(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$
  ie, dot product to the power of d = mapping into high dimensional space and then taking dot product.

Kernel Trick, Fit the Second

Simple example

- Given input data $\mathbf{x} \in \mathbb{R}^2$
- We can project that to 3D space with $F : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$F(\mathbf{x}) = [\mathbf{x}_1^2, \mathbf{x}_2^2, \sqrt{(2)}\mathbf{x}_1\mathbf{x}_2]^T$$

- Then if we want to compute the inner product between two inputs

$$F(\mathbf{x}) \cdot F(\mathbf{z}) = [\mathbf{x}_1^2, \mathbf{x}_2^2, \sqrt{(2)}\mathbf{x}_1\mathbf{x}_2] \cdot [\mathbf{z}_1^2, \mathbf{z}_2^2, \sqrt{(2)}\mathbf{z}_1\mathbf{z}_2]$$

$$= \mathbf{x}_1^2\mathbf{z}_1^2 + \mathbf{x}_2^2\mathbf{z}_2^2 + 2\mathbf{x}_1\mathbf{x}_2\mathbf{z}_1\mathbf{z}_1$$

- But this is a computation in the $\mathbb{R}^3$ space (imagine if $F : \mathbb{R}^2 \rightarrow \mathbb{R}^{3000}$!

Kernel Trick, Fit the Second

Advantage of using kernel functions: can compute what the dot product would be if we had actually projected the data.

Simple example:

- So instead, we can use a kernel function $K(\mathbf{x}, \mathbf{z})$
- $K(\mathbf{x}, \mathbf{z}) = F(\mathbf{x}) \cdot F(\mathbf{z})$
- e.g. for quadratic, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$
- Since $\mathbf{x} \cdot \mathbf{z}$ is a scalar, once we have computed the dot product, just have to do the kernel computation on this one number.
- We've computed the inner product in the feature space $\mathbb{R}^3$ without ever having to transform data into $\mathbb{R}^3$!

Kernel Trick, Fit the Second

- This expression:

$$(\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

  is the kernel function.
- Usually written as:

$$K(\mathbf{x}_i, \mathbf{x}_j)$$

- For any $K(\mathbf{x}_i, \mathbf{x}_j)$, solving

$$\arg\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

  will find a linear separator in the higher dimension feature space associated with $K(\mathbf{x}_i, \mathbf{x}_j)$.
- Kernel trick: plugging these kernels, optimal linear separators can be found efficiently in feature spaces with billions of dimensions.
- The resulting linear separators, when mapped back to the original input space, can correspond to arbitrarily wiggly, non-linear decision boundaries.

Kernel Trick, Fit the Second

- Nothing special about:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

- Other kernel functions correspond to other feature spaces:
  - Polynomial kernels:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$$

    or

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$$

    corresponds to a feature space that is exponential in d.
  - Gaussian kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp\Big(- \frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\sigma^2} \Big)$$

Other kernel functions

- The sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\kappa(\mathbf{x}_i \cdot \mathbf{x}_j) + \Theta)$$

where d is an integer and $\sigma, \kappa, \Theta \in \Re$

Empirically, all of the above kernel functions have been found to give SV classifiers with similar accuracies and similar set of support vectors.

Other kernel functions

- Also notable is the radial basis function kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2s^2})$$

- This defines a spherical kernel with centre $\mathbf{x}_i$ and radius s.
- Comparing this to the Gaussian kernel, we can see this is basically the same thing (writing s for σ).

Kernel Trick, Fit the Second

- Indeed, Mercer's theorem (1909) shows that any "reasonable" kernel corresponds to some feature space.

(st-andrews.ac.uk)

Reasonable?

- A kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is reasonable if it is positive definite.
- That is, for some feature space $\mathbf{X}$:

$$\sum_{i,j=1}^{n} c_i, c_j K(\mathbf{x}_i, \mathbf{x}_j) \geqslant 0$$

for any n, and any scalars c_i, c_j and any $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$

Kernel Trick, Fit the Second

Note:

- Feature spaces can be big.
- Hence the concern about the feature space relative to the number of training examples.
- The polynomial kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$$

corresponds to a feature space that is exponential in d

Other uses of kernels

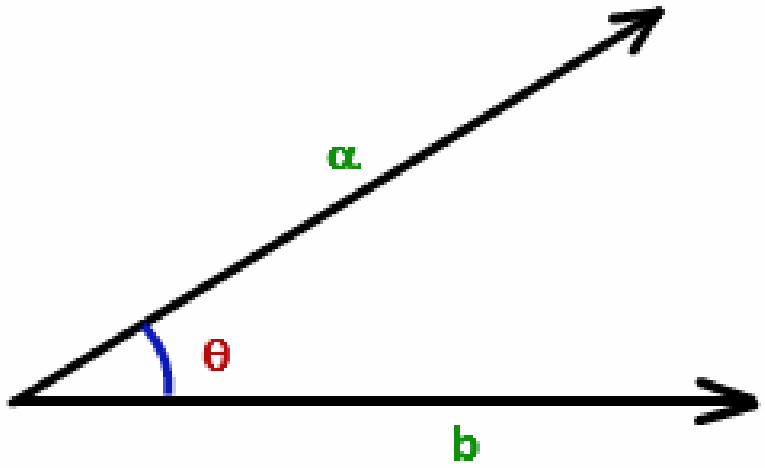
- Remember that there are two parts to the learning here.
  1. A method for finding optimal linear separators.
  2. The kernel trick
- Don't have to use them together.
- Can use the optimal linear classifier piece to learn linear classifiers.
  - Linear kernel
- Can kernelize any learning algorithm where we use dot products of pairs of data points.
  - *k*-nearest neighbour
  - Perceptron learning

Dot product = similarity

(Studio Canal)

What is the dot product?

- Measure of similarity
- When $\theta = 0$, $a \cdot b$ is maximum.
- When $\theta = 0$, a and b are the same.
- When $\theta = \frac{\pi}{2}$, $a \cdot b$ is zero.
- When $\theta = \frac{\pi}{2}$, a and b are orthogonal.

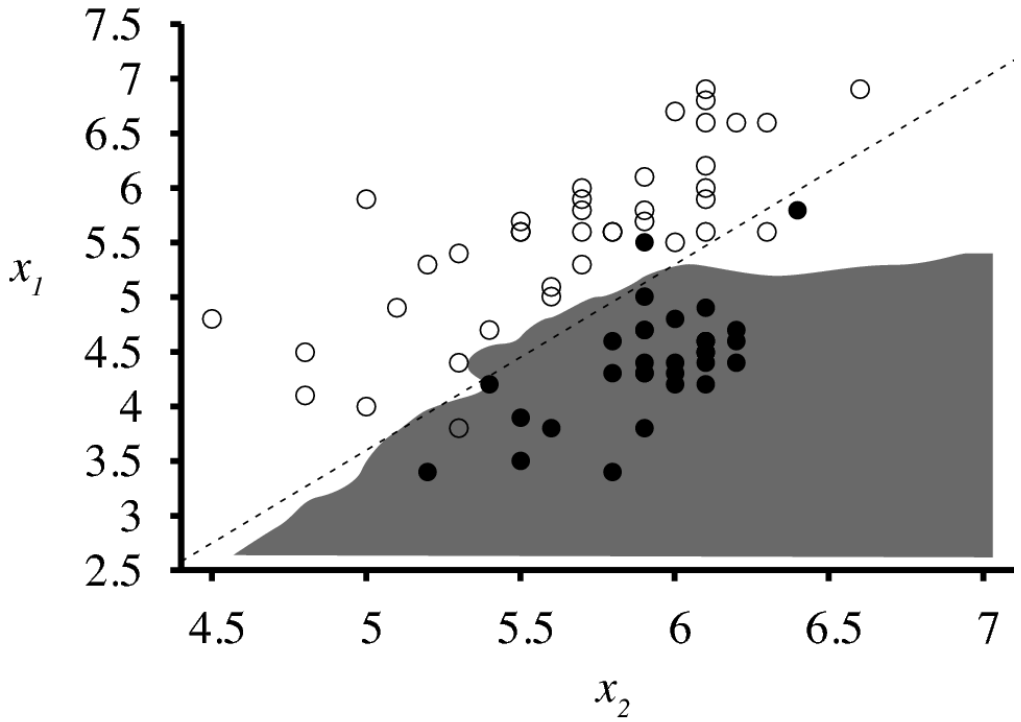


$$a.b = |a|\,|b|\,\cos\theta$$

(courses.cs.washington.edu/courses/cse490j

- So, in k-nearest neighbour, the dot product of one feature with another is a measure of "nearness" between them.

k-nearest neighbour



(Russell & Norvig)

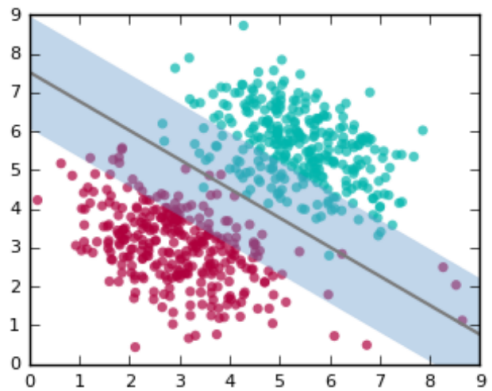
- Classify by finding the *k* nearest examples and going with the majority classification.
- Instead of distance, compute similarity.

Everything is Connected

(BBC America)

Soft margin classifiers

- In practice, a separating hyperplane may not exist.
- For some problems, even in the high dimension space of the kernel, there is no hyperplane that cleanly separates the training examples.
- A high level of noise can cause this to happen.
- What then?
- We relax the constraints under which we look for the hyperplane.
- Soft margin classifier.



(blog.statsbot.co)

Soft margin classifiers

- We do this by allowing slack variables:

$$\xi_i \geqslant 0$$

for all $i = 1 \ldots m$

- Then, instead of looking for

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geqslant 1$$

for all i in the original weight/feature space, we look for:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geqslant 1 - \xi_i$$

for all i.

Soft margin classifiers

- A classifier that generalizes well is then found by controlling both the "capacity" of the classifier, measured by:

$$||\mathbf{w}||$$

and the amount of slack, measured by:

$$\sum_{i=1}^{m} \xi_i$$

This latter provides an upper bound on the number of training errors.

Soft margin classifiers

- One way to achieve this, is to minimise:

$$\tau(w, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i$$

subject to:

$$\xi_i \geqslant 0$$

for all $i = 1, \ldots, m$, and:

$$y_i(\mathbf{w} \cdot x + b) \geqslant 1 - \xi_i$$

for all i.

Soft margin classifiers

$$\tau(w, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i$$

- The constant $C > 0$ determines the amount of slack that is allowed.
- This controls the trade-off between margin maximization
  - which needs more slack

  and minimizing training errors
  - which needs less slack
- Typically have to search to find the optimal C for a given problem.

Soft margin classifiers

We then do the usual:

1. Incorporate a kernel
2. Rewrite in terms of Lagrange multipliers

Soft margin classifiers

- We end up solving:

$$\arg\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the additional constraints:

$$0 \leqslant \alpha_i \leqslant C$$

for all $i = 1, \ldots, m$ and:

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

Soft margin classifiers

- The only difference with the separable case is the upper bound C on the α_i.
- This limits the influence of individual data points.
- The intuition is that this prevents outliers from having a big effect on the separator.

Soft margin classifiers

- We end up with a hyperplane separator such that the support vectors
  - The $\mathbf{x}_i$ which lie on the margins.

  are the ones with the slack variables $\xi_i = 0$.

Support Vector Regression

- Regression, as we already know, is a generalization of classification.
- Rather than, as above, looking for output:

$$y \in \{1, -1\}$$

we are interested in output:

$$y \in \Re$$

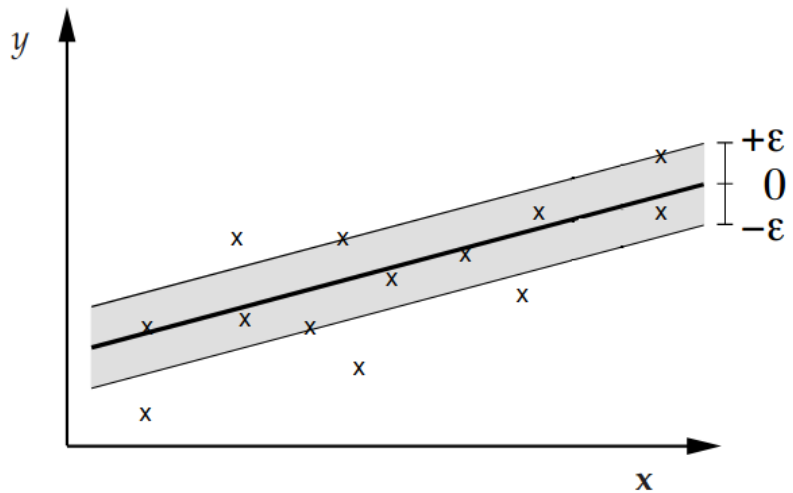
we do this by learning a function $f(\mathbf{x})$ which approximates y.

Support Vector Regression

- We quantify the loss incurred by predicting $f(\mathbf{x})$ rather than y as:

$$\begin{aligned} c(\mathbf{x}, y, f(\mathbf{x})) &= |y - f(\mathbf{x})|_\epsilon \\ &= \max\{0, |y - f(\mathbf{x})| - \epsilon\} \end{aligned}$$

- This is Vapnik's ϵ-insensitive loss function
- In effect we fit a tube with radius ϵ to the data:



Support Vector Regression

- To estimate a linear regression:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

we minimise:

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

Support Vector Regression

- Can transform this into a constrained optimization problem by introducing slack variables, akin to the soft margin case.
- This time we need two kinds:
- We have ξ for the case where:

$$f(\mathbf{x}_i) - y_i > \epsilon$$

and ξ^* for the case where:

$$y_i - f(\mathbf{x}_i) > \epsilon$$

- (We write both ξ together as $\xi^{(*)}$)

Support Vector Regression

- Then we minimise:

$$\tau(w, \xi^{(*)}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*)$$

subject to:

- $f(\mathbf{x}_i) - y_i \leqslant \epsilon + \xi_i$
- $y_i - f(\mathbf{x}_i) \leqslant \epsilon + \xi_i^*$
- $\xi_i, \xi_i^* \geqslant 0$ for all i

The conditions on the ξ_i and ξ_i^* mean that errors smaller than ϵ do not require a non-zero ξ_i and ξ_i^*, and so don't affect the minimization.

Support Vector Regression

- Again we do the whole Lagrange multiplier thing, and end up with a regression estimate of:

$$f(\mathbf{x}) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) K(\mathbf{x}_i, \mathbf{x}) + b$$

- SVM idea, applied to regression.

Gaussian process models

- Combination of SVM and probabilistic methods.
- Or:
  What to do if you think SVMs aren't complicated enough.

Gaussian process models

- Take the basic likelihood calculation:

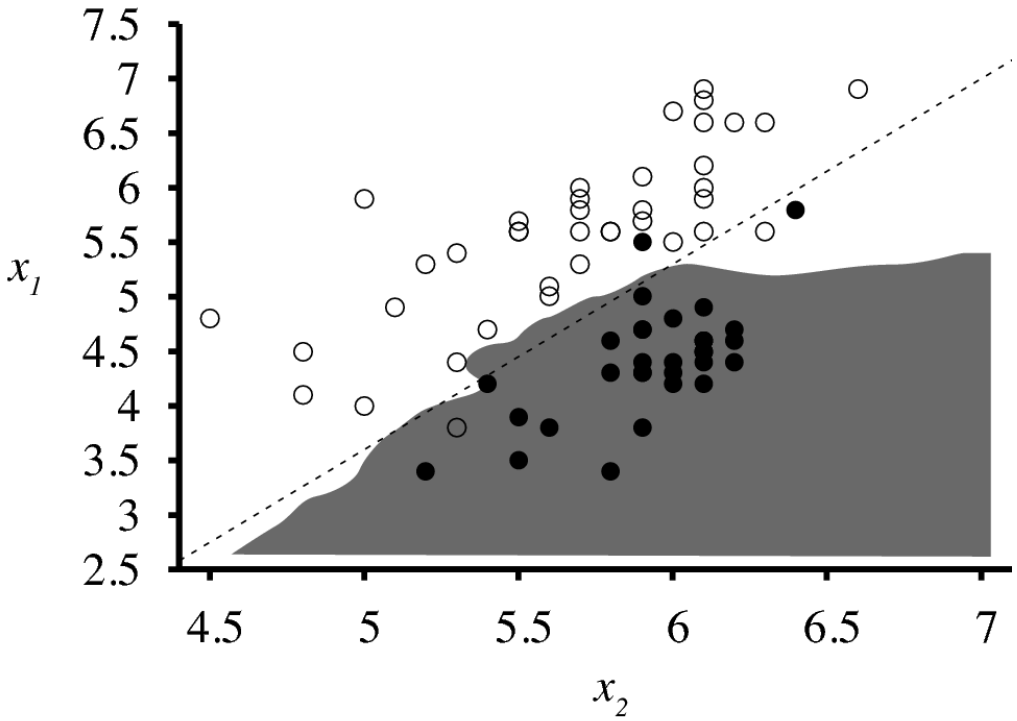
$$p(h|\mathcal{D}) = \frac{p(\mathcal{D}|h)p(h)}{p(\mathcal{D})}$$

- Model $p(\mathcal{D}|h)$ as a Gaussian.
- This is the linear model.
- Now kernelize it.

Non-parametric methods

- Consider methods to be non-parametric if they cannot be characterised by a bounded set of parameters.
- Instance-based methods like k-nearest neighbour are non-parametric because they require us to remember all the examples.
- Grows without bound.

Non-parametric methods



(Russell & Norvig)

- Classify by finding the *k* nearest examples and going with the majority classification.

Non-parametric methods

- Requires a notion of distance.
- Typically the Minkowski distance:

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = \left(\sum_i |\mathbf{x}_{ji} - \mathbf{x}_{qi}|^p \right)^{\frac{1}{p}}$$

  which generalises Euclidian and Manhattan distances to p dimensions.
- Also use the Mahalanobis distance which takes into account the covariance between dimensions.

Non-parametric methods

- We also now know that we can use the dot product:

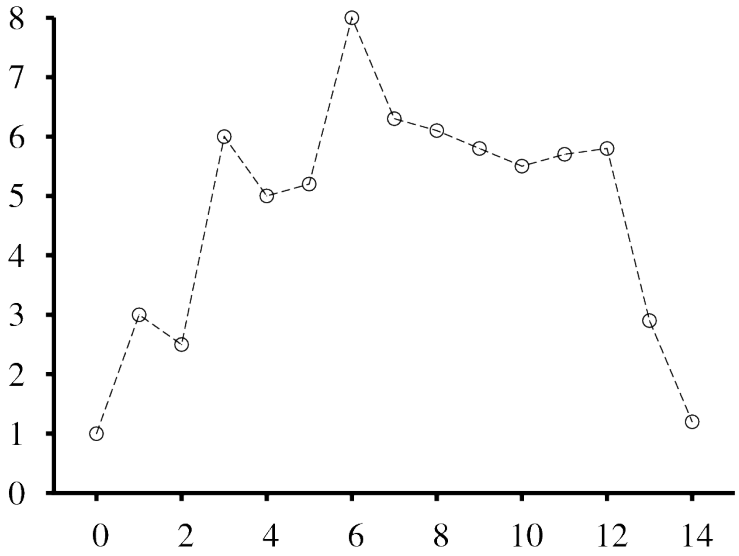
$$\mathbf{x}_j \cdot \mathbf{x}_q$$

to find the nearest, where the metric is similarity between vectors.

Non-parametric methods

- Now let's look at non-parametric regression.

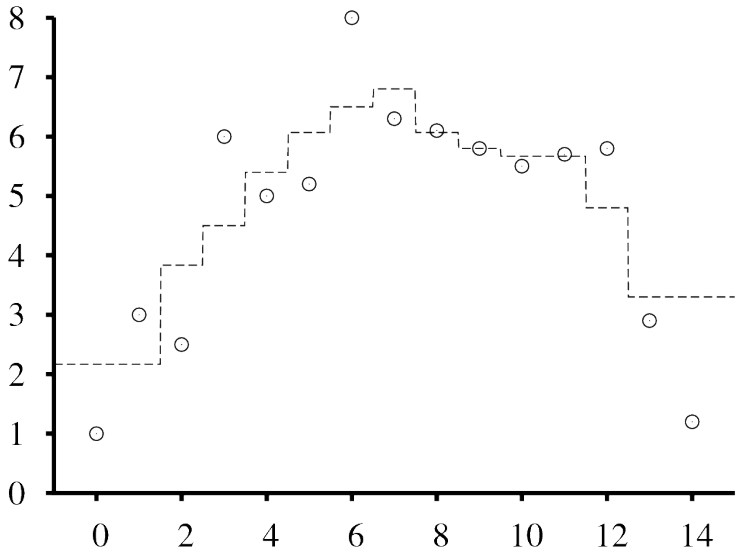
Non-parametric methods



(Russell & Norvig)

- Piecewise non-parametric linear regression model.
- When given a test x_q, uses training examples immediately to the left and right of x_q.
- Good for non-noisy data, poor for noisy data.

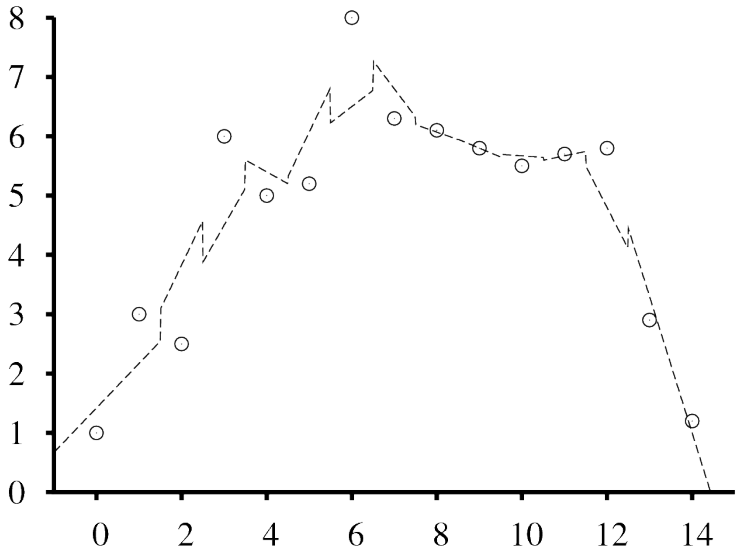
Non-parametric methods



(Russell & Norvig)

- k-nearest neighbours average.
- $k = 3$.
- Discontinuous, poor at the edges.

Non-parametric methods



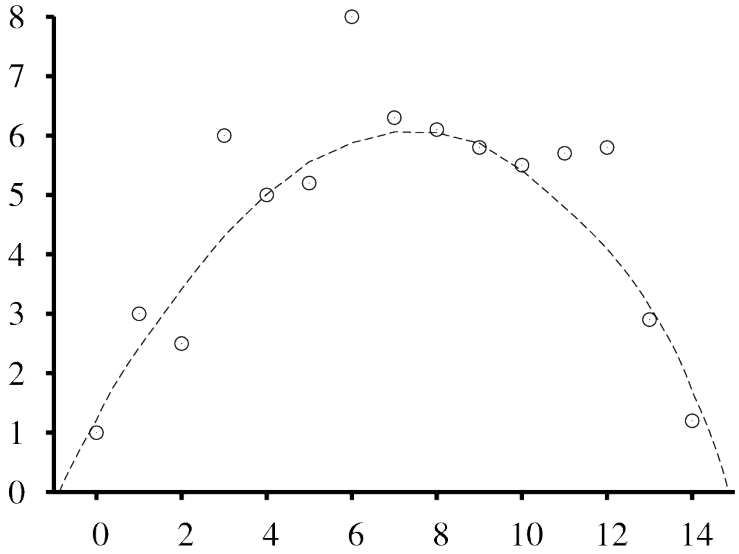
(Russell & Norvig)

- *k*-nearest neighbour linear regression, finds best line through *k* examples.
- Still discontinuous.

Non-parametric methods

- Choose k through cross-validation.

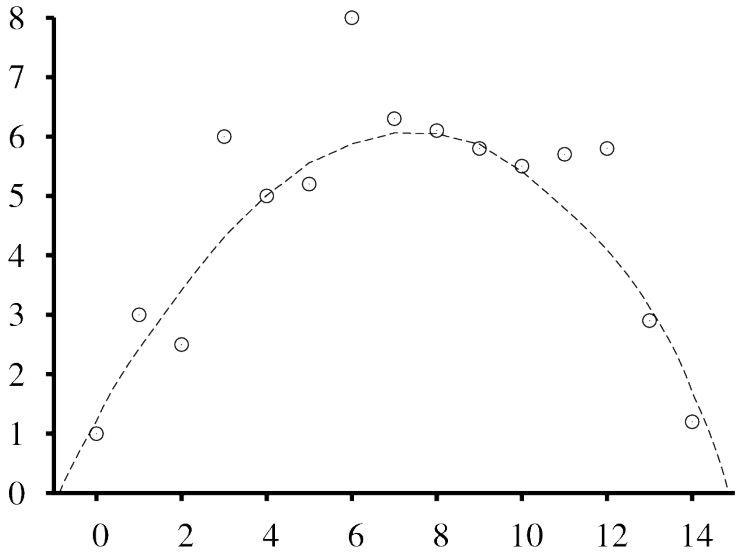
Non-parametric methods



(Russell & Norvig)

- Locally weighted regression.
- Prevents discontinuity by discounting further points more and more.
- No sudden drop off = no discontinuity.

Non-parametric methods



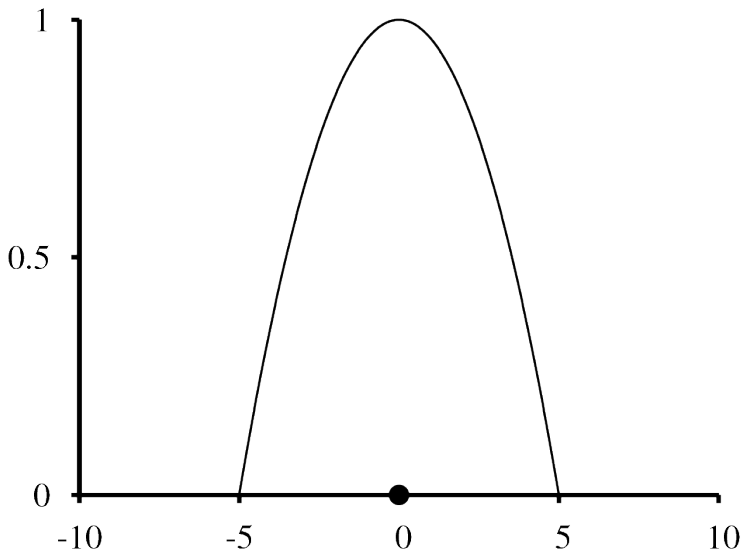
(Russell & Norvig)

- How do we choose the weight for each example.
- A kernel.

Non-parametric methods

- Not quite the same kind of kernel.
- Distance metric in this case.
- Symmetric about 0, and area under the kernel is bounded.

Non-parametric methods



(Russell & Norvig)

- Quadratic kernel

$$K(d) = \max(0, 1 - \left(\frac{2|d|}{k}\right)^2)$$

with kernel width $k = 10$, d is distance from the test point 0.

- k too large vs too narrow (choose using cross-validation).

Non-parametric methods

- General form of a locally weighted regression:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j K(\mathit{distance}(\mathbf{x}_q, \mathbf{x}_j))(y_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$

  where *distance* is any suitable metric.
- Solve this by gradient descent.
- Then the answer is:

$$h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$$

- Note that we need to solve this for every query point.
- However, we only solve for points with non-zero weights
  Those inside the kernel.

Summary

- We looked at various kernel-based methods.
- Started kerneless, finding optimal solutions to binary classification.
- Then saw how kernels allowed us to learn non-linear classifiers
- And soft-margin classifiers.
- Extended this to regression.
- Mentioned Gaussian process models.
- Ended with nearest-neighbour regression, which uses another kind of kernel.