Fondamenti di IA

03 - Linear Models, Non-parametric Models and SVM

SAPIENZA
UNIVERSITÀ DI ROMA

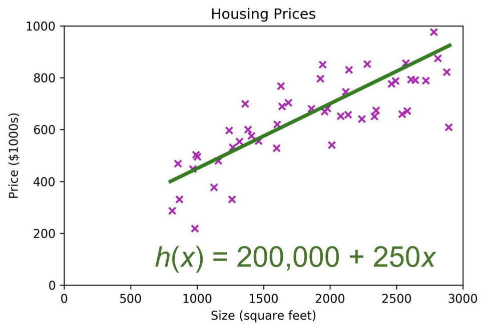
Fabrizio Silvestri

Linear Regression

Now it is time to move on from decision trees and lists to a different hypothesis space, one that has been used for hundreds of years: the class of linear functions of continuous-valued inputs.

Univariate Linear Function

- A univariate linear function (a straight line) with input x and output y has the form $y = h(x) = w_1x + w_0$
    - where w_0 and w_1 are real-valued coefficients to be learned.



- We use the letter w because we think of the coefficients as weights; *the value of y is changed by changing the relative weight of one term or another*.

Univariate Regression in vectorial form

- In the multivariate case we will define **w** to be the vector $\langle w_0, w_1 \rangle$, and define the linear function with those weights as

$$h_{\mathbf{w}}(x) = w_1 x + w_0 = \mathbf{w} \cdot \langle x, 1 \rangle$$

Linear Regression

- $h_{\mathbf{w}}(x)=w_1x+w_0 = \mathbf{w} \cdot \langle x,1 \rangle$
- The task of finding the $h_{\mathbf{w}}$ that best fits these data is called **linear regression**.
- What does “best fits” mean?

Best Fit

- To fit a line to the data, all we have to do is find the values of the weights $\langle w_0, w_1 \rangle$ that minimize the empirical loss.
- It is “traditional” (going back to Gauss) to use the **squared-error loss function**, L_2, summed over all the training examples:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

- Goal is to find $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\ Loss(h_{\mathbf{w}})$. The sum $\sum_{j=1..N}(y_j-(w_1x_j+w_0))^2$ is minimized when its partial derivatives with respect to w_0 and w_1 are 0.

Partial Derivatives Equal to 0

- The partial derivatives are as follows:

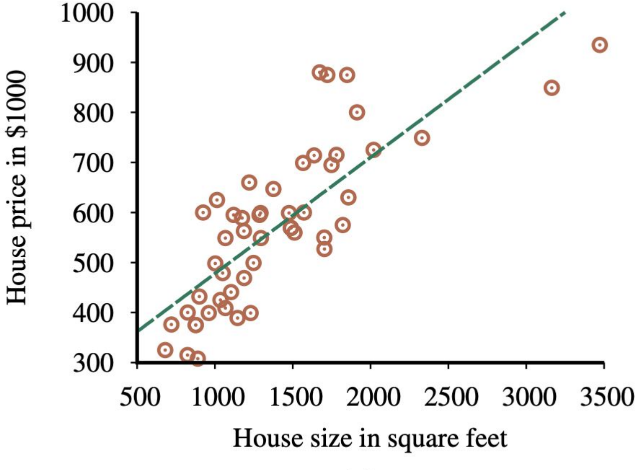
$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0$$

- Whose solution is

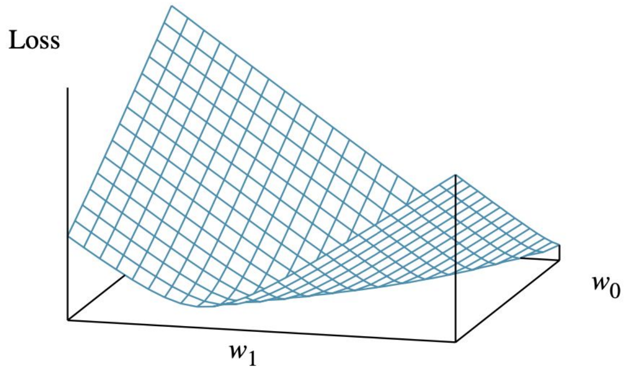
$$w_1=\frac{N(\sum x_jy_j)-(\sum x_j)(\sum y_j)}{N(\sum x_j^2)-(\sum x_j)^2};\;\; w_0=(\sum y_j-w_1(\sum x_j))/N$$

and it’s unique.

House Price



(a)



(b)

Figure 19.13 (a) Data points of price versus floor space of houses for sale in Berkeley, CA, in July 2009, along with the linear function hypothesis that minimizes squared-error loss: $y = 0.232x + 246$. (b) Plot of the loss function $\sum_j (y_j - w_1 x_j + w_0)^2$ for various values of w_0, w_1. Note that the loss function is convex, with a single global minimum.

Convex Loss Functions

- From the previous example we observe that the loss function is **Convex**.
- This is true for **every** linear regression problem with an L2 loss function.
  - ⇒ **No** Local Minima exist!
- End of the story for linear models: if we need to fit lines to data, we must compute:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \; w_0 = (\sum y_j - w_1(\sum x_j))/N$$

Gradient Descent

What if the Loss Function was not Convex?

- Take “baby steps” in the direction opposite to the gradient of the loss.
- We choose any starting point in weight space
  - In the examples above, a point in the (w_0, w_1) plane
- Compute an estimate of the gradient and move a small amount in the steepest downhill direction
- repeating until we converge on a point in weight space with (**local**) minimum loss.

```
$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged do
    for each $w_i$ in $\mathbf{w}$ do
        $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$
```

Learning Rate

```
$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged do
    for each $w_i$ in $\mathbf{w}$ do
        $\boxed{w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})}$
```

Update rule

- The parameter α is usually called the learning rate
  - when we are trying to minimize loss in a learning problem.
- It can be a fixed constant, or it can decay over time as the learning process proceeds.

SAPIENZA UNIVERSITÀ DI ROMA

Chain Rule

- For univariate regression, the loss is quadratic, so the partial derivative will be linear.
- (The only calculus you need to know is the chain rule:

  $\partial g(f(x))/\partial x = g'(f(x)) * \partial f(x)/\partial x$,

  plus the facts that $\partial x^2 = 2x$ and $\partial x = 1$.)
- Let's first work out the partial derivatives (slopes) in the simplified case of only one training example, (x, y):

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)).\end{aligned}$$

Chain Rule

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)).\end{aligned}$$

- Applying this to both w_0 and w_1 we get:

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

- Plugging this into the update rule of gradient descent, and folding the 2 into the unspecified learning rate α, we get the following learning rule for the weights:

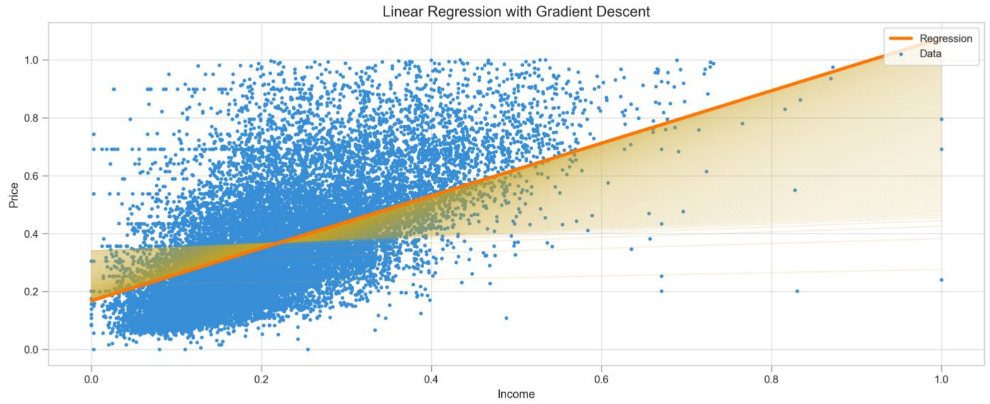
$$w_0 \leftarrow w_0 + \alpha\,(y - h_{\mathbf{w}}(x)); \quad w_1 \leftarrow w_1 + \alpha\,(y - h_{\mathbf{w}}(x)) \times x$$

SAPIENZA
UNIVERSITÀ DI ROMA

Meaning of the update rules for Linear Regression

$$w_0 \leftarrow w_0 + \alpha\,(y - h_{\mathbf{w}}(x))\,; \quad w_1 \leftarrow w_1 + \alpha\,(y - h_{\mathbf{w}}(x)) \times x$$

- These updates make intuitive sense: if $h_w(x) > y$ (i.e., the output is too large), reduce w_0 a bit, and reduce w_1 if x was a positive input but increase w_1 if x was a negative input.



Updates for Multiple Samples

- The preceding equations cover one training example.
- For N training examples, we want to minimize the sum of the individual losses for each example.
- The derivative of a sum is the sum of the derivatives, so we have:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

- These updates constitute the **batch gradient descent** learning rule for univariate linear regression.
- The loss surface is convex, which means that there are no local minima to get stuck in, and convergence to the global minimum is guaranteed
    - as long as we don't pick an α that is so large that it overshoots
- may be very slow: we have to sum over all N training examples for every step, and there may be many steps.

- A step that covers all the training examples is called an **epoch**.

Stochastic Gradient Descent

m becomes an *hyperparameter* to be tuned on a specific hardware architecture.

- A faster variant is called stochastic gradient descent or SGD
  - it randomly selects a small number of training examples at each step, and updates according to the update rule.
  - The original version of SGD selected only one training example for each step, but it is now more common to select a **minibatch** of m out of the N examples.
- Suppose we have N=10,000 examples and choose a minibatch of size m=100.
  - Then on each step we have reduced the amount of computation by a factor of 100
- The standard error of the estimated mean gradient is proportional to the square root of the number of examples
  - → the standard error increases by only a factor of 10.
  - So even if we have to take 10 times more steps before convergence, minibatch SGD is still 10 times faster than full batch SGD in this case.

SGD - General Considerations

- Convergence of minibatch SGD is not strictly guaranteed
    - it can oscillate around the minimum without settling down.
    - We will see how a schedule of decreasing the learning rate, α does guarantee convergence.
- SGD can be helpful in an online setting, where new data are coming in one at a time, and the stationarity assumption may not hold.
    - (In fact, SGD is also known as online gradient descent.)
    - With a good choice for α a model will slowly evolve, remembering some of what it learned in the past, but also adapting to the changes represented by the new data.
- SGD is widely applied to models other than linear regression, in particular **neural networks**.

- *Even when the loss surface is not convex, the approach has proven effective in finding good local minima that are close to the global minimum.*

Multivariate Linear Regression

Adding Dimensions

- We can easily extend to multivariable linear regression problems, in which each example $\mathbf{x}_j$ is an n-element vector.
- Our hypothesis space is the set of functions of the form

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

Including the Bias in the **w**’s vector

- Let us invent a dummy input attribute, $\mathbf{x}_{j,0}$, which is defined as always equal to 1.
- Then h is simply the dot product of the weights and the input vector
  - or, equivalently, the matrix product of the transpose of the weights and the input vector

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}$$

- The best vector of weights, **w***, minimizes squared-error loss over the examples:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

SAPIENZA
UNIVERSITÀ DI ROMA

Optimal **w***

- The best vector of weights, **w***, minimizes squared-error loss over the examples:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

- Gradient descent will reach the (unique) minimum of the loss function; the update equation for each weight w_i is

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}$$

Analytic Solution

- Let **y** be the vector of outputs for the training examples, and **X** be the data matrix
  - the matrix of inputs with one n-dimensional example per row.
- Then the vector of predicted outputs is **ŷ** = **Xw**
- The squared-error loss over all the training data is

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- We set the gradient to zero:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

- Rearranging, we find that the minimum-loss weight vector is given by the Normal Equation

pseudo-inverse

$$\mathbf{w}^* = \boxed{(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top}\mathbf{y}\,.$$

SAPIENZA UNIVERSITÀ DI ROMA

Overfitting in Multivariate Linear Regression

- With univariate linear regression we didn’t have to worry about overfitting.
- But with multivariable linear regression in high-dimensional spaces it is possible that some dimension that is actually irrelevant appears by chance to be useful, resulting in overfitting.
- Thus, it is common to use regularization on multivariable linear functions to avoid overfitting.

Regularizing Linear Regression

- With regularization we minimize the total cost of a hypothesis, counting both the empirical loss and the complexity of the hypothesis:

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$

- For linear functions the complexity can be specified as a function of the weights. We can consider a family of regularization functions:

$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

SAPIENZA
UNIVERSITÀ DI ROMA

L1 vs. L2 Regularization

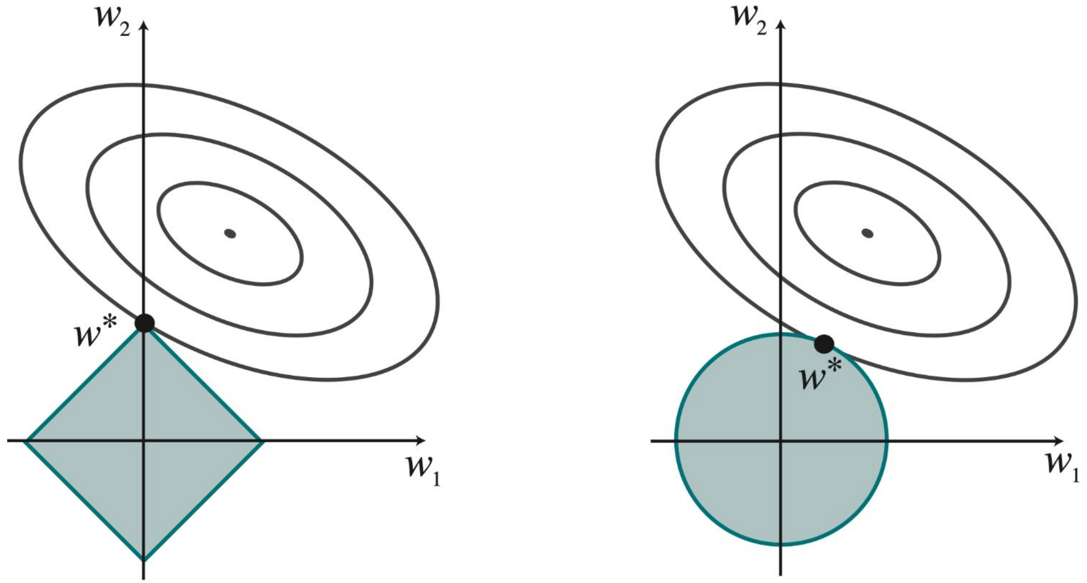


Figure 19.14 Why L_1 regularization tends to produce a sparse model. Left: With L_1 regularization (box), the minimal achievable loss (concentric contours) often occurs on an axis, meaning a weight of zero. Right: With L_2 regularization (circle), the minimal loss is likely to occur anywhere on the circle, giving no preference to zero weights.

Ridge Regression – a.k.a. L_2 Regularization

- Ridge is a method of estimating the coefficients of multiple-regression models in scenarios where linearly independent variables are highly correlated.
- The form of the hypothesis is $L(y, \hat{y}) = MSE(y, \hat{y}) + \lambda L_2(\mathbf{w})$
  - Where $L_2(w)$ is given by $L_2(\mathbf{w}) = \sum_i \mathbf{w}_i^2$
- There exist a closed form for the optimal value of **w**

$$\mathbf{w} = (X^T X + \lambda I_p)^{-1} X^T Y$$

SAPIENZA
UNIVERSITÀ DI ROMA

Lasso Regression – a.k.a. L_1 Regularization

- Lasso, or Least Absolute Shrinkage and Selection Operator, is quite similar conceptually to ridge regression.
- The form of the hypothesis is $L(y, \hat{y}) = MSE(y, \hat{y}) + \lambda L_1(\mathbf{w})$
  - Where $L_1(\mathbf{w})$ is given by $L_1(\mathbf{w}) = \sum_j |w_j|$
- For high values of λ, many coefficients are exactly zeroed under lasso, which is never the case in ridge regression.
- There exists a closed form for Lasso, too. But, it is not as elegant as the Ridge’s one

Elastic Net

- Limitations of Lasso:
  - If $p > n$, the lasso selects at most n variables.
    - The number of selected genes is bounded by the number of samples.
  - Grouped variables: the lasso fails to do grouped selection.
    - It tends to select one variable from a group and ignore the others.
- It combines Lasso and Ridge:
  - $L(y, \hat{y}) = MSE(y, \hat{y}) + \lambda_1 L_1(\mathbf{w}) + \lambda_2 L_2(\mathbf{w})$
  - By setting $\alpha = \lambda_1 \backslash (\lambda_1 + \lambda_2)$ we have that we can rewrite the above loss as:
    - $L(y, \hat{y}) = MSE(y, \hat{y}) + \alpha L_1(\mathbf{w}) + (1-\alpha) L_2(\mathbf{w})$
  - If you want to know more about Elastic Net go here.

Linear Classification as Regression

Linear classifiers with a hard threshold

$x_2 = 1.7x_1 - 4.9 \quad \text{or} \quad -4.9 + 1.7x_1 - x_2 = 0$

Decision Boundary

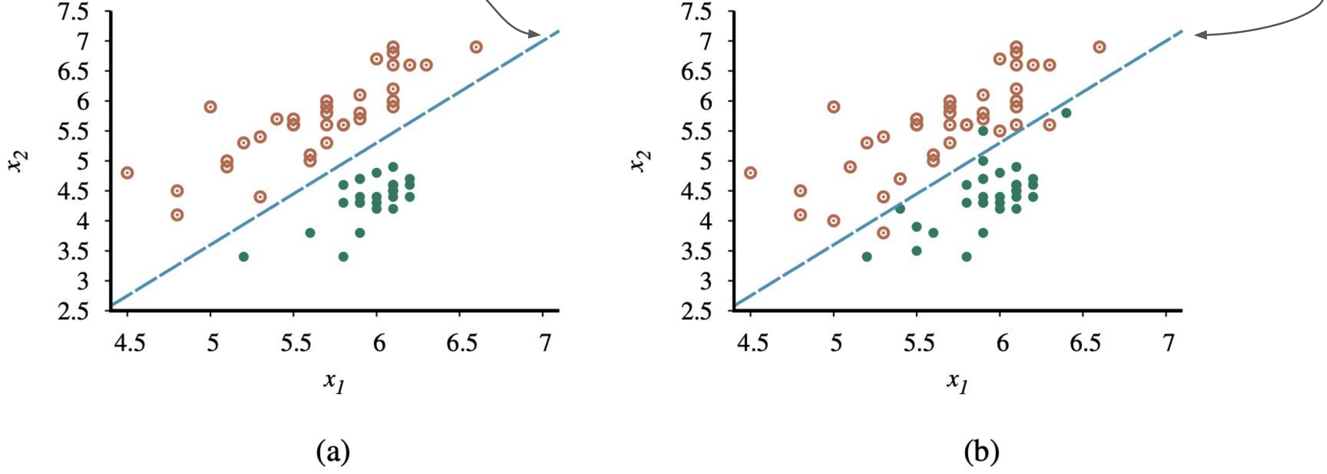


Figure 19.15 (a) Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

Classification Hypothesis

- The explosions, which we want to classify with value 1, are below and to the right of this line; they are points for which $-4.9 + 1.7x_1 - x_2 > 0$, while earthquakes have $-4.9 + 1.7x_1 - x_2 < 0$
- We can make the equation easier to deal with by changing it into the vector dot product form
  - with $x_0 = 1$ we have $-4.9x_0 + 1.7x_1 - x_2 = 0$, and we can define the vector of weights, $\mathbf{W} = \langle -4.9, 1.7, -1 \rangle$, and write the classification hypothesis

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ if } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ and } 0 \text{ otherwise}$$

Threshold Function

- we can think of h as the result of passing the linear function **w**·**x** through a threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w}\cdot\mathbf{x})$$

where $\text{Threshold}(z)=1$ if $z \geq 0$ and 0 otherwise

How to minimize the threshold function?

- Can we use Gradient Descent in weight space?
    - the gradient is zero almost everywhere in weight space except at those points where $\mathbf{w}\cdot\mathbf{x} = 0$, and at those points the gradient is **undefined**.
- A simple weight update rule that converges to a solution
    - that is, to a linear separator that classifies the data perfectly
    - provided the data are linearly separable.

for a single example ($\mathbf{x}$, y), we have

Perceptron update rule

$$\boxed{w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i}$$

Analyzing Perceptron Update Rule: $w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$

- Both the true value y and the hypothesis output $h_{\mathbf{w}}(\mathbf{x})$ are either 0 or 1, so there are three possibilities:

1. If the output is correct (i.e., $y=h_{\mathbf{w}}(\mathbf{x})$) then the weights are not changed.
2. If y is 1 but $h_{\mathbf{w}}(\mathbf{x})$ is 0, then w_i is increased when the corresponding input x_i is positive and decreased when x_i is negative.
    a. This makes sense, because we want to make $\mathbf{w}\cdot\mathbf{x}$ bigger so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 1.
3. If y is 0 but $h_{\mathbf{w}}(\mathbf{x})$ is 1, then w_i is decreased when the corresponding input x_i is positive and increased when x_i is negative.
    a. This makes sense, because we want to make $\mathbf{w}\cdot\mathbf{x}$ smaller so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 0.

SAPIENZA
UNIVERSITÀ DI ROMA

Learning Curves

- Typically the learning rule is applied one example at a time, choosing examples at random (as in stochastic gradient descent).

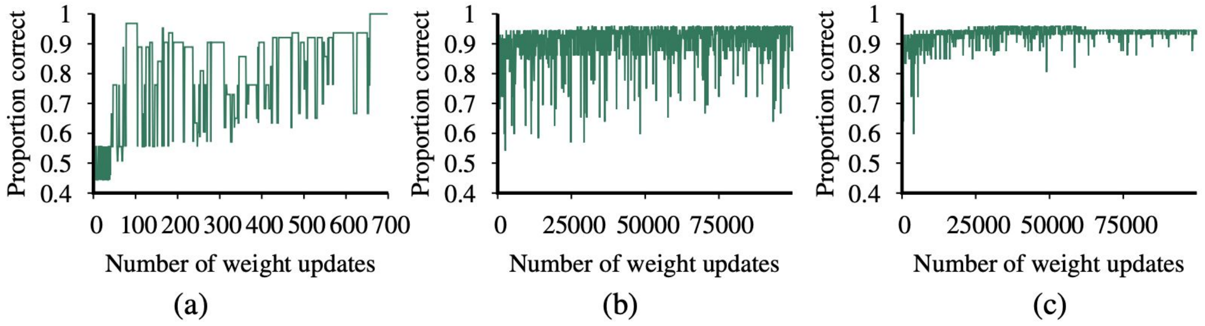


Figure 19.16 (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 19.15(a). (b) The same plot for the noisy, nonseparable data in Figure 19.15(b); note the change in scale of the *x*-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000+t)$.

Does it always converge?

- Adding more data points, the data might not be linearly separable anymore.

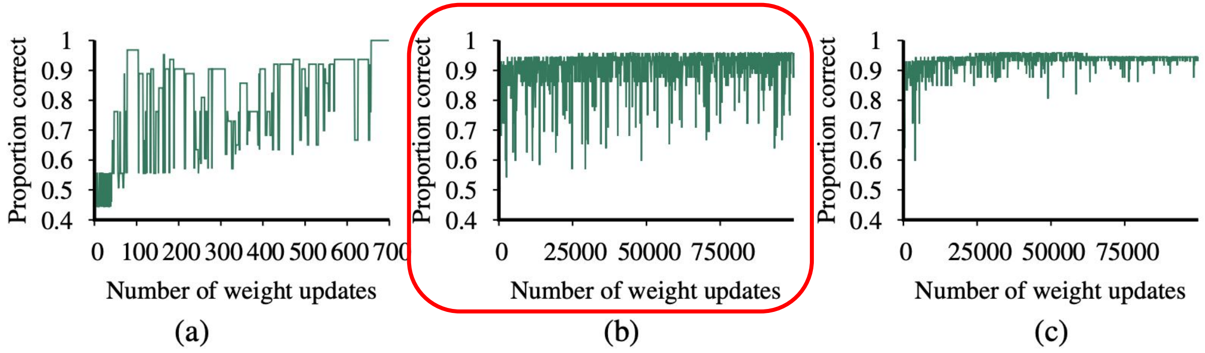
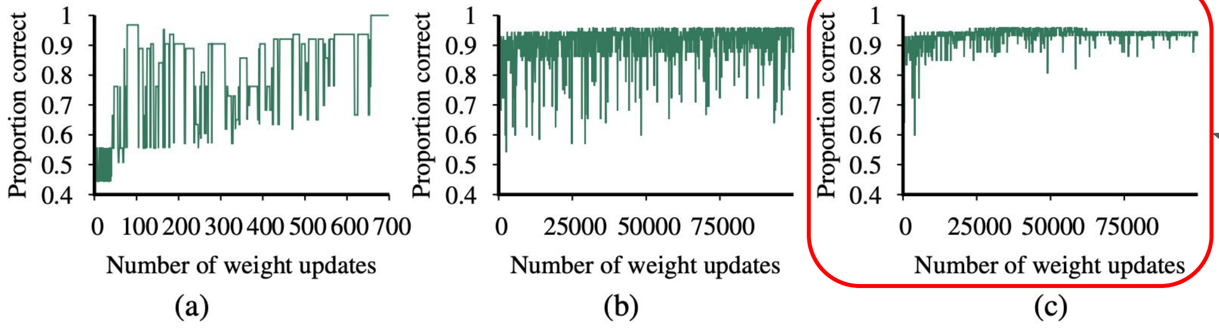


Figure 19.16 (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 19.15(a). (b) The same plot for the noisy, nonseparable data in Figure 19.15(b); note the change in scale of the *x*-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000 + t)$.

Scaling Learning Rate

- In general, the perceptron rule may not converge to a stable solution for fixed learning rate α
  - if α decays as O(1/t) where t is the iteration number, then the rule can be shown to converge to a minimum-error solution when examples are presented in a random sequence.

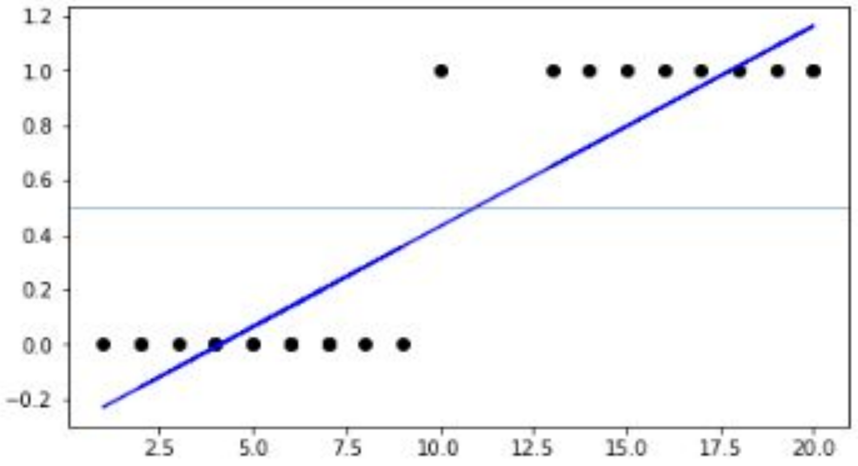


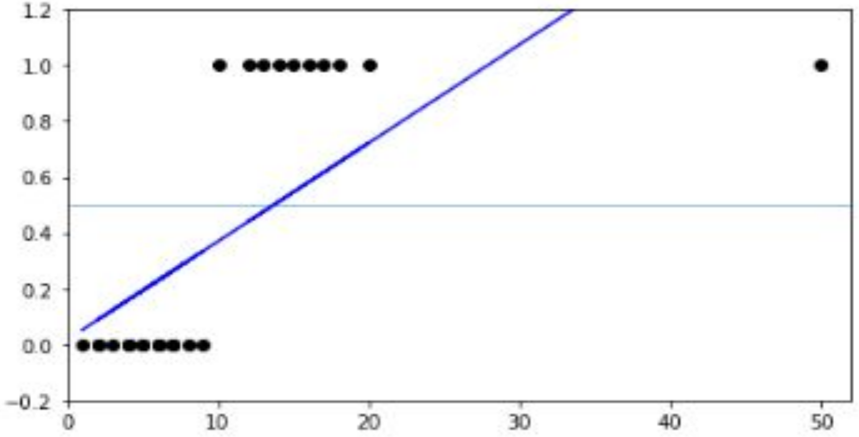
Not quite there yet, but much stabler than the second plot.

Figure 19.16 (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 19.15(a). (b) The same plot for the noisy, nonseparable data in Figure 19.15(b); note the change in scale of the *x*-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000+t)$.

Logistic Regression

Limitation of Threshold Functions





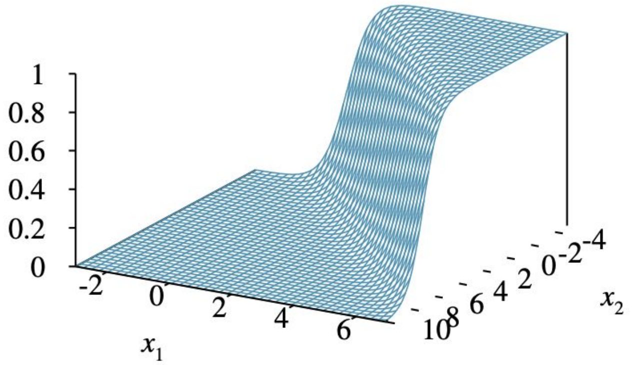
Logistic Function

- It is the function

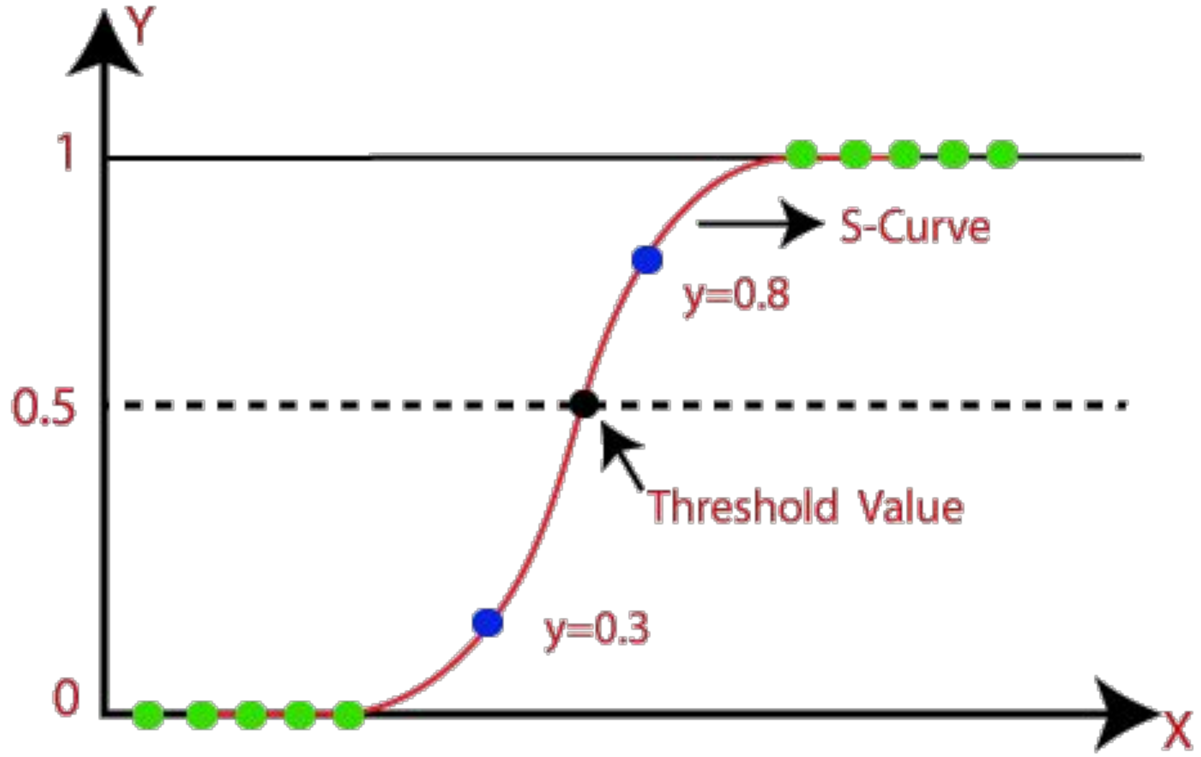
$$Logistic(z) = \frac{1}{1+e^{-z}}$$

It can replace the threshold function in the classification rule

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$



“New” Threshold Function



SAPIENZA
UNIVERSITÀ DI ROMA

Logistic Regression

- The process of fitting the weights of this model to minimize loss on a data set is called logistic regression.
- There is no easy closed-form solution to find the optimal value of w with this model
    - but the gradient descent computation is straightforward
- Because our hypotheses no longer output just 0 or 1, we will use the L2 loss function
- In the following, we'll use g to stand for the logistic function, with g′ its derivative.

Logistic Regression

- For a single example (**x**,y), the derivation of the gradient is the same as for linear regression up to the point where the actual form of h is inserted.

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w}\cdot\mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times x_i\,.
\end{aligned}$$

- The derivative g′ of the logistic function satisfies g′(z)=g(z)(1−g(z)), so we have

$$g'(\mathbf{w}\cdot\mathbf{x}) = g(\mathbf{w}\cdot\mathbf{x})(1 - g(\mathbf{w}\cdot\mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

Weights Update

- The weight update for minimizing the loss takes a step in the direction of the difference between input and prediction, $(y-h_{\mathbf{w}}(\mathbf{x}))$, and the length of that step depends on the constant α and g′:

$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

Binary Cross Entropy Loss

- Hypothesis with Logistic threshold can be seen as a probability score:
  - Values are in (0,1)
- For this kind of data there is a better loss function: binary cross entropy loss: $-(y \log(h_{\mathbf{w}}(\mathbf{x})) + (1 - y) \log(1 - h_{\mathbf{w}}(\mathbf{x})))$
- Where, we can pick p to be the output of $h_{\mathbf{w}}(\mathbf{x})$ when we use the logistic threshold

$$\frac{\partial L}{\partial \mathbf{w}} = (h_{\mathbf{w}}(\mathbf{x}) - y)\mathbf{x}$$

- The update rule in this case is then

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha(h_{\mathbf{w}}(\mathbf{x}) - y)\mathbf{x}$$

SAPIENZA UNIVERSITÀ DI ROMA

Why Is Logistic Regression a Linear Model?

- Let us keep our assumption that we want to model the probability of an event P(**x**).
- Doing regression directly on the probability won't work well:
    - Labels are 0/1
    - Aggregations and counting might not work
    - So what?
- Let us define a new quantity, let us call it odd ratio.
- The odd-ratio is

$$\frac{P(\mathbf{x})}{1 - P(\mathbf{x})}$$

Why Is Logistic Regression a Linear Model?

- Log-odds

$$\ln\left(\frac{P(\mathbf{x})}{1-P(\mathbf{x})}\right)$$

- Log-odds as a linear model

$$\ln\left(\frac{P(\mathbf{x})}{1-P(\mathbf{x})}\right)=\sum_{j=1}^{N} w_j x_j$$

Why Is Logistic Regression a Linear Model?

- Solve for P(x) to get:

$$P(\mathbf{x}) = \frac{1}{1 - exp\left(\sum_{j=1}^{N} w_j x_j\right)}$$

- Or, if we let σ be the sigmoid function, we can rewrite the above equation as

$$P(\mathbf{x}) = \sigma\left(\sum_{j=1}^{N} w_j x_j\right)$$

Non Parametric Models

Parameters in a Model

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

- In Linear Regression, the number of parameters coincides with the number of features used
  - In statistics, they often name them covariates.
- Neural Networks, which we haven’t seen, use billions of neurons (!!!) to take their decisions.
- So, it is legitimate to ask ourselves: Is it possible to learn a model without using any parameters?
  - Remember Decision Trees?

SAPIENZA
UNIVERSITÀ DI ROMA

Non-Parametric Models

- A nonparametric model is one that cannot be characterized by a bounded set of parameters.
- For example, a piecewise linear function retains all the data points as part of the model.
- Learning methods that do this have also been described as instance-based learning or memory-based learning.

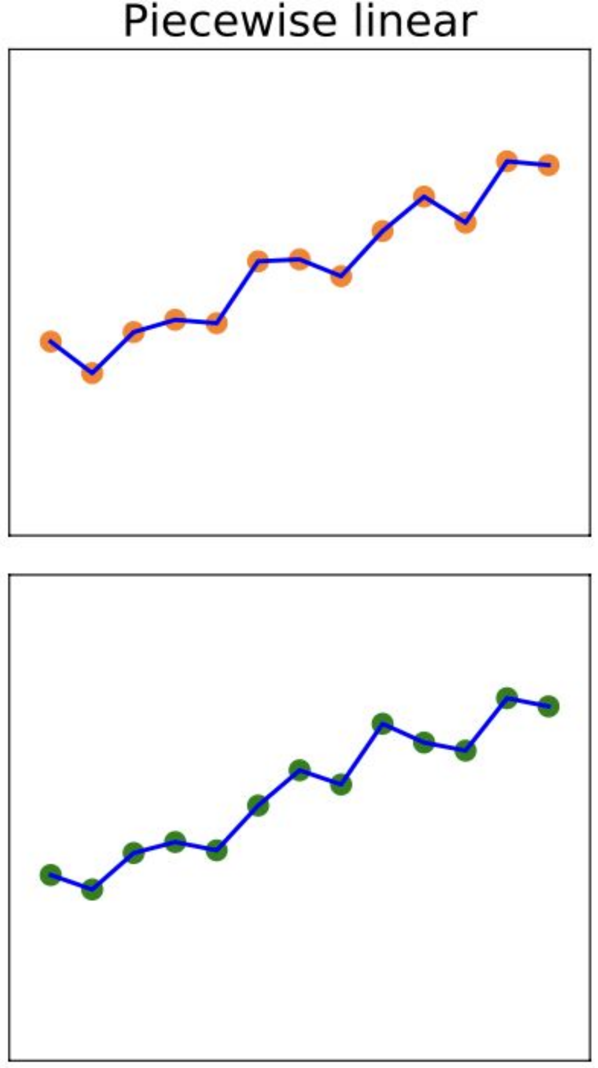


Table Lookup

- The simplest instance-based learning method is table lookup:
  - take all the training examples, put them in a lookup table
  - when asked for h(**x**), see if **x** is in the table
  - if it is, return the corresponding y.
- The problem with this method is that it does not generalize well: when x is not in the table we have no information about a plausible value.

Nearest-Neighbor Models

- We can improve on table lookup with a slight variation:
  - given a query $\mathbf{x}_q$, instead of finding an example that is equal to $\mathbf{x}_q$, find the k examples that are nearest to $\mathbf{x}_q$.
- This is called k-nearest-neighbors lookup.
- We'll use the notation **$NN(k,x_q)$** to denote the set of k neighbors nearest to $\mathbf{x}_q$.

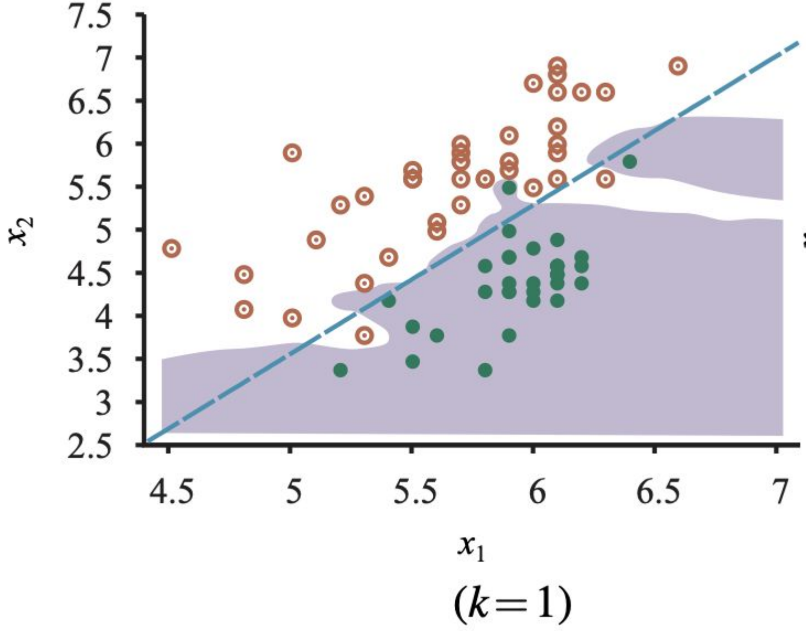
Nearest-Neighbor Classification

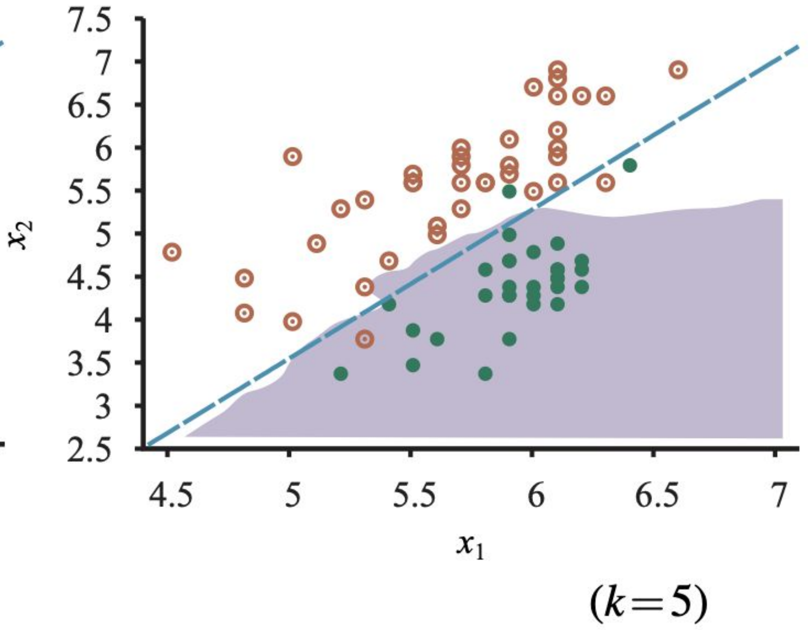
- To do classification, find the set of neighbors NN(k,$\mathbf{x}_q$) and take the most common output value
  - for example, if k=3 and the output values are ⟨Yes,No,Yes⟩, then the classification will be Yes.
- To avoid ties on binary classification, k should be chosen to be an odd number.

Nearest-Neighbor Regression

- To do regression, we can take the mean or median of the k neighbors,
    - or we can solve a linear regression problem on the neighbors.
- The piecewise linear function solves a (trivial) linear regression problem with the two data points to the right and left of $\mathbf{x}_q$.
    - When the $\mathbf{x}_i$ data points are equally spaced, these will be the two nearest neighbors.

Nearest-Neighbor Decision Boundaries (and Overfitting)





Distance Among Points

- The very word “nearest” implies a distance metric.
- How do we measure the distance from a query point $\mathbf{x}_q$ to an example point $\mathbf{x}_j$?
- Typically, distances are measured with a Minkowski distance or L_p norm, defined as

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p}$$

- With p=2 this is Euclidean distance and with p=1 it is Manhattan distance.

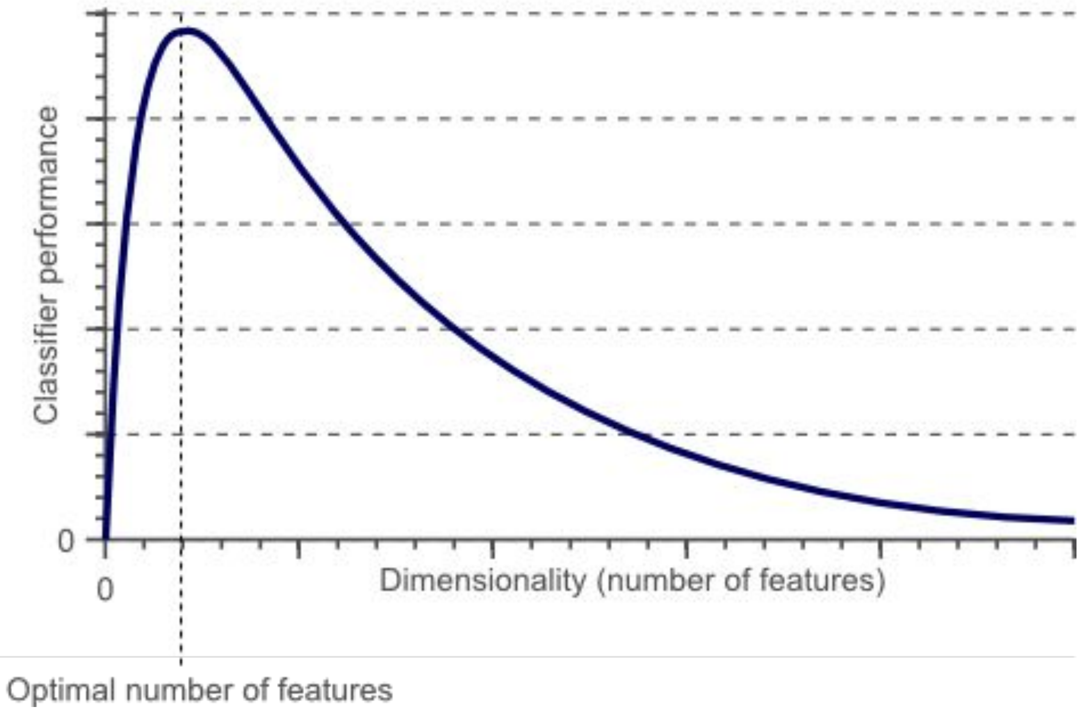
Data Normalization

- How do we compare a difference in age to a difference in weight?
- A common approach is to apply normalization to the measurements in each dimension.
- We can compute the mean μ_i and standard deviation σi of the values in each dimension, and rescale them so that $x_{j,i}$ becomes $(x_{j,i} − \mu_i) / \sigma_i$

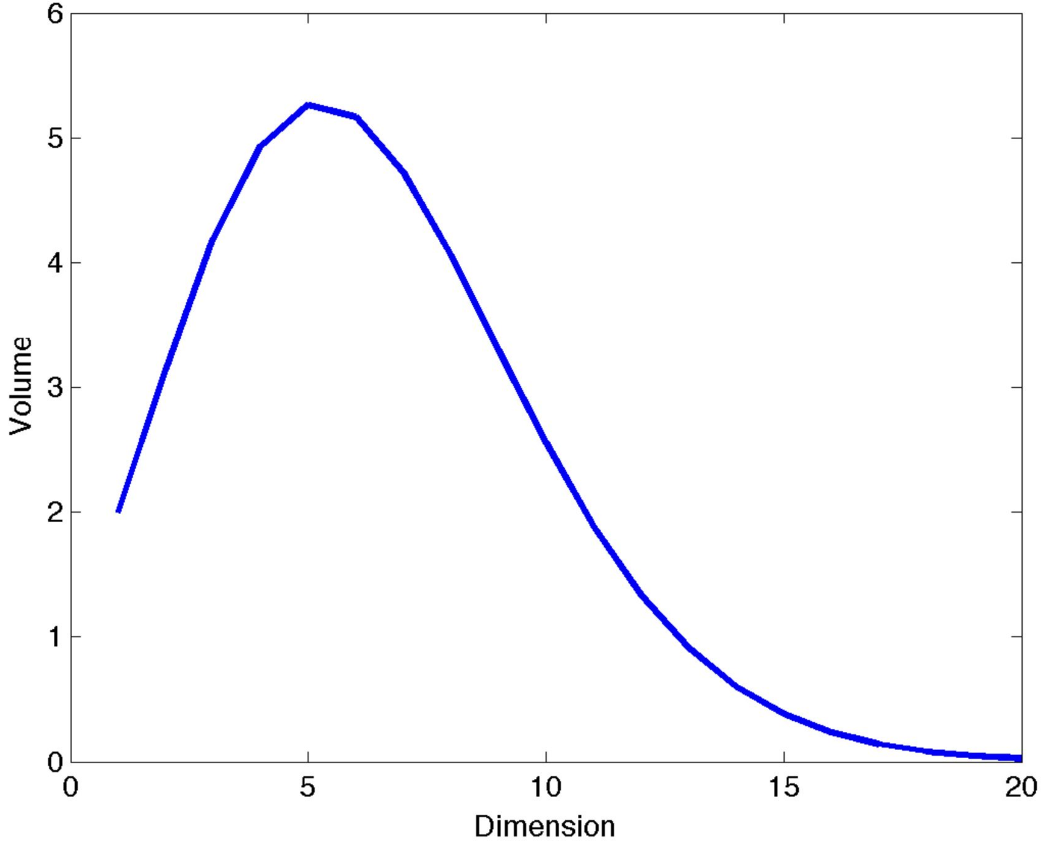
How Well NN Works?

- In low-dimensional spaces with plenty of data, nearest neighbors works very well: we are likely to have enough nearby data points to get a good answer.
- But as the number of dimensions rises we encounter a problem: the nearest neighbors in high-dimensional spaces are usually not very near!
- Consider k-nearest-neighbors on a data set of N points uniformly distributed throughout the interior of an n-dimensional unit hypercube.
- We'll define the k-neighborhood of a point as the smallest hypercube that contains the k nearest neighbors.
- Let l be the average side length of a neighborhood.
  - Then the volume of the neighborhood (which contains k points) is l^n and the volume of the full cube (which contains N points) is 1.
  - So, on average, $l^n = k/N$. Taking nth roots of both sides we get $l = (k/N)^{1/n}$.
- To be concrete, let k=10 and N=1,000,000.
- In two dimensions (n=2; a unit square), the average neighborhood has l=0.003, a small fraction of the unit square, and in 3 dimensions l is just 2% of the edge length of the unit cube.
- But by the time we get to 17 dimensions, l is half the edge length of the unit hypercube, and in 200 dimensions it is 94%. This problem has been called the curse of dimensionality.

Curse of Dimensionality Visualized



Curse of Dimensionality Visualized



The NN(k,$\mathbf{x}_q$) function

- The NN(k,$\mathbf{x}_q$) function is conceptually trivial:
  - given a set of N examples and a query $\mathbf{x}_q$, iterate through the examples
  - measure the distance to $\mathbf{x}_q$ from each one, and keep the best k.
- If we are satisfied with an implementation that takes O(N) execution time, then that is the end of the story.
- But instance-based methods are designed for large data sets, so we would like something faster.

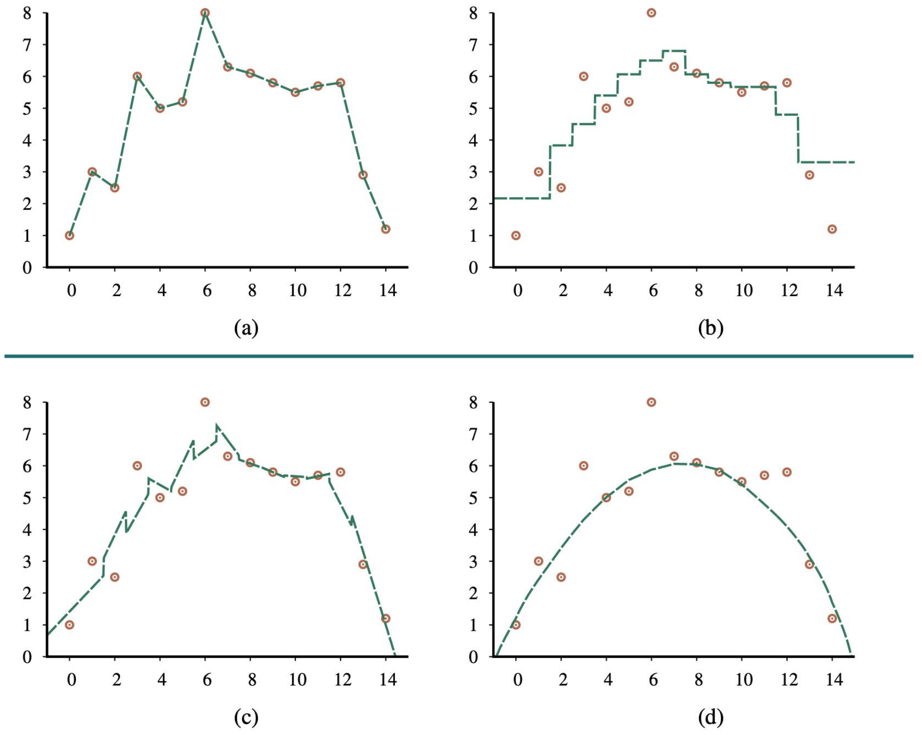
Locality-Sensitive Hashing (LSH)

- We can't use hashes to solve NN(k,xq) exactly, but with a clever use of randomized algorithms, we can find an approximate solution.
- First we define the approximate near-neighbors problem:
  - given a data set of example points and a query point $\mathbf{x}_q$, find, with high probability, an example point (or points) that is near $\mathbf{x}_q$.
  - To be more precise, we require that if there is a point $\mathbf{x}_j$ that is within a radius r of $\mathbf{x}_q$, then with high probability the algorithm will find a point $\mathbf{x}_j'$ that is within distance cr of $\mathbf{x}_q$.
  - If there is no point within radius r then the algorithm is allowed to report failure.
- The values of c and "high probability" are hyperparameters of the algorithm.

Locality-Sensitive Hashing (LSH)

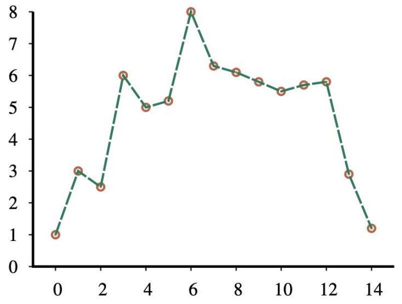
- To solve approximate near neighbors, we will need a hash function g(**x**) that has the property that, for any two points $\mathbf{x}_j$ and $\mathbf{x}_j'$, the probability that they have the same hash code is small if their distance is more than cr, and is high if their distance is less than r.
  - $\mathrm{dist}(\mathbf{x}_j, \mathbf{x}_j') = \Pr(g(\mathbf{x}_j) \neq g(\mathbf{x}_j'))$
- With large real-world problems, such as finding the near neighbors in a data set of 13 million Web images using 512 dimensions, LSH needs to examine only a few thousand images out of 13 million to find nearest neighbors.

Nonparametric Regression



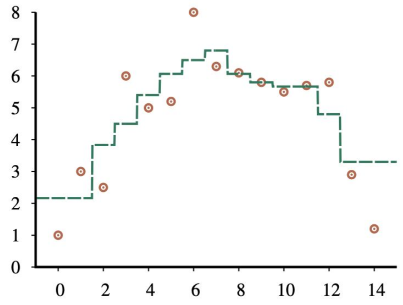
Connect The Dots

- This model creates a function h(**x**) that, when given a query $\mathbf{x}_q$, considers the training examples immediately to the left and right of $\mathbf{x}_q$, and interpolates between them.
- When noise is low, this trivial method is actually not too bad
  - which is why it is a standard feature of charting software in spreadsheets.
- When the data are noisy, the resulting function is spiky and does not generalize well.



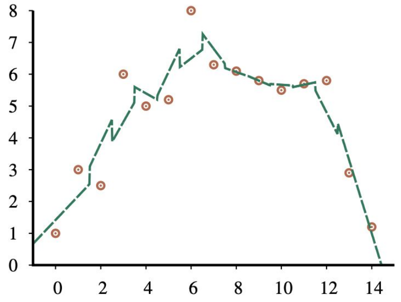
K-Nearest-Neighbors Regression

- Instead of using just the two examples to the left and right of a query point $\mathbf{x}_q$, we use the k nearest neighbors.
  - A larger value of k tends to smooth out the magnitude of the spikes, although the resulting function has discontinuities.
- k-nearest-neighbors average: h($\mathbf{x}$) is the mean value of the k points, $1/k \sum y_j$.



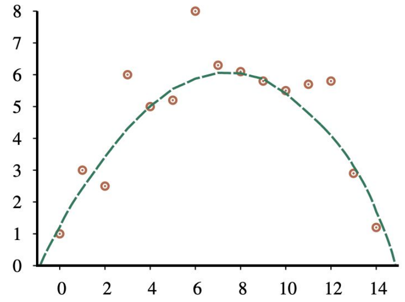
K-Nearest-Neighbor Linear Regression

- Finds the best line through the k examples.
- This does a better job of capturing trends at the outliers, but is still discontinuous.



Locally Weighted Regression

- Gives us the advantages of nearest neighbors, without the discontinuities.
- To avoid discontinuities in h(**x**), we need to avoid discontinuities in the set of examples we use to estimate h(**x**).
- The idea of locally weighted regression is that at each query point $\mathbf{x}_q$, the examples that are close to $\mathbf{x}_q$ are weighted heavily, and the examples that are farther away are weighted less heavily, and the farthest not at all.
- The decrease in weight over distance is typically gradual, not sudden.



Weighting Examples through Kernels

- We decide how much to weight each example with a function known as a kernel, whose input is a distance between the query point and the example.
- A kernel function $\mathcal{K}$ is a decreasing function of distance with a maximum at 0, so that $\mathcal{K}(\text{Distance}(\mathbf{x}_j,\mathbf{x}_q))$ gives higher weight to examples $\mathbf{x}_j$ that are closer to the query point $\mathbf{x}_q$ for which we are trying to predict the function value.
- The integral of the kernel value over the entire input space for $\mathbf{x}$ must be finite
    - and if we choose to make the integral 1, certain calculations are easier.
- Example:
    - quadratic kernel, $\mathcal{K}(d) = \max(0, 1 - (2|d|/w)^2)$, with kernel width $w=10$
    - Typically, the width is a hyperparameter of the model that is best chosen by cross-validation.

Locally Weighted Regression… Now With Kernels

- For a given query point $\mathbf{x}_q$ we solve the following weighted regression problem:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathbf{x}_j))\,(y_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$

- where Distance is any of the distance metrics discussed for nearest neighbors. Then the answer is $h(\mathbf{x}_q)=\mathbf{w}^* \cdot \mathbf{x}_q$.

Support Vector Machines

Why Support Vector Machines?

1. SVMs construct a maximum margin separator—a decision boundary with the largest possible distance to example points.
    - This helps them generalize well.
2. SVMs create a linear separating hyperplane, but they have the ability to embed the data into a higher-dimensional space, using the so-called kernel trick.
    - Often, data that are not linearly separable in the original input space are easily separable in the higher-dimensional space.
3. SVMs are nonparametric
    - the separating hyperplane is defined by a set of example points, not by a collection of parameter values. But while nearest-neighbor models need to retain all the examples, an SVM model keeps only the examples that are closest to the separating plane
        - usually only a small constant times the number of dimensions.

- Thus SVMs combine the advantages of nonparametric and parametric models: they have the flexibility to represent complex functions, but they are resistant to overfitting.

Max-Margin Separator

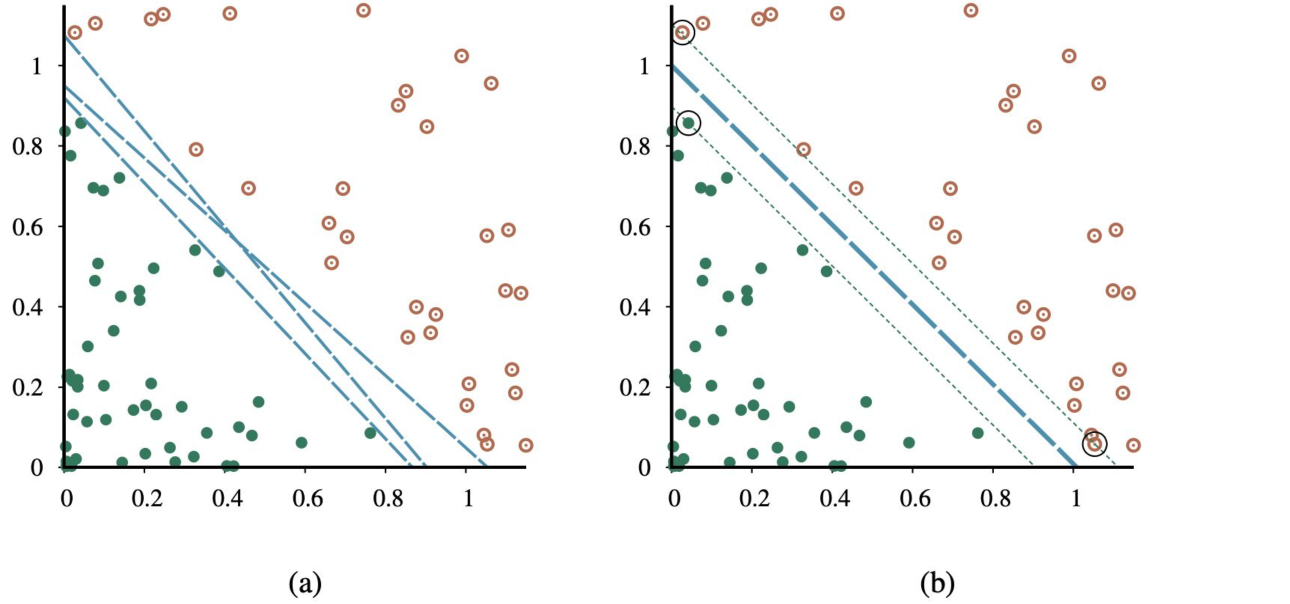


Figure 19.21 Support vector machine classification: (a) Two classes of points (orange open and green filled circles) and three candidate linear separators. (b) The maximum margin separator (heavy line), is at the midpoint of the **margin** (area between dashed lines). The **support vectors** (points with large black circles) are the examples closest to the separator; here there are three.

How Do We Find The Separator?

- Traditionally SVMs use the convention that class labels are +1 and -1, instead of the +1 and 0 we have been using so far.
- Also, whereas we previously put the intercept into the weight vector **w** (and a corresponding dummy 1 value into $\mathbf{x}_{j,0}$), SVMs do not do that; they keep the intercept as a separate parameter, **b**.
- The separator is defined as the set of points $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$.
    - We could search the space of **w** and b with gradient descent to find the parameters that maximize the margin while correctly classifying all the examples.

Can We Do “Better” Than SGD

- We won’t show how to get that, but will just say that there is an alternative representation called the dual representation, in which the optimal solution is found by solving

$$\underset{\alpha}{\operatorname{argmax}} \sum_{j} \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

- subject to the constraints α ≥ 0 and ∑ α y =0. This is a quadratic programming optimization problem, for which there are good software packages.
- Once we have found the vector α we can get back to **w** with the equation $\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$

Properties of the Dual Representation Equation

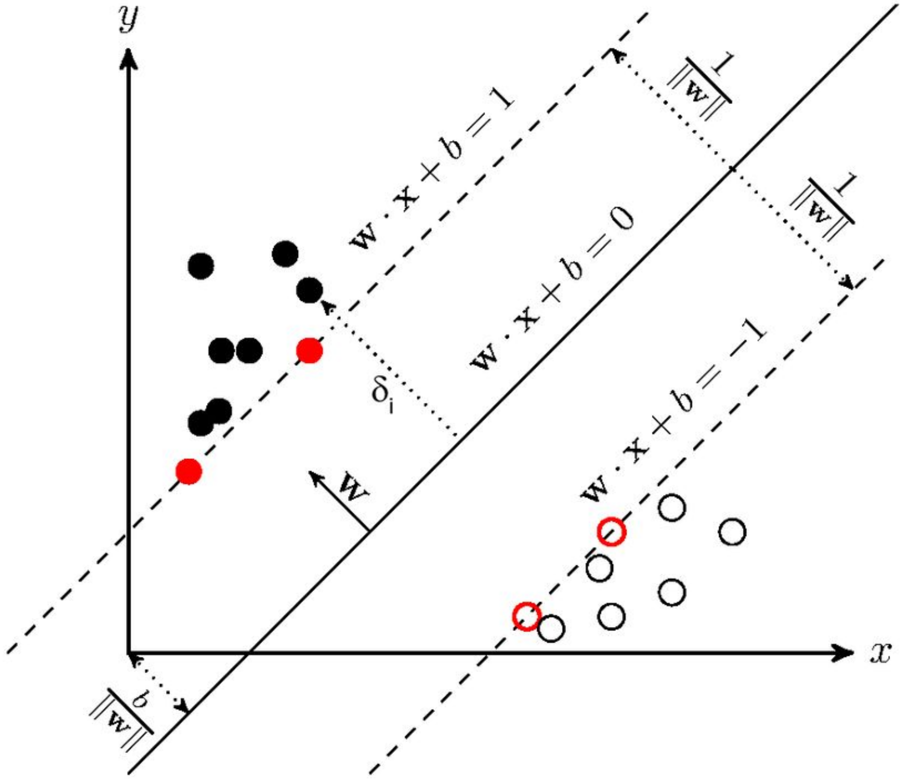
$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

- The expression is convex
  - it has a single global maximum that can be found efficiently.
- The data enter the expression only in the form of dot products of pairs of points.
  - This second property is also true of the equation for the separator itself; once the optimal α_j have been calculated, the equation is

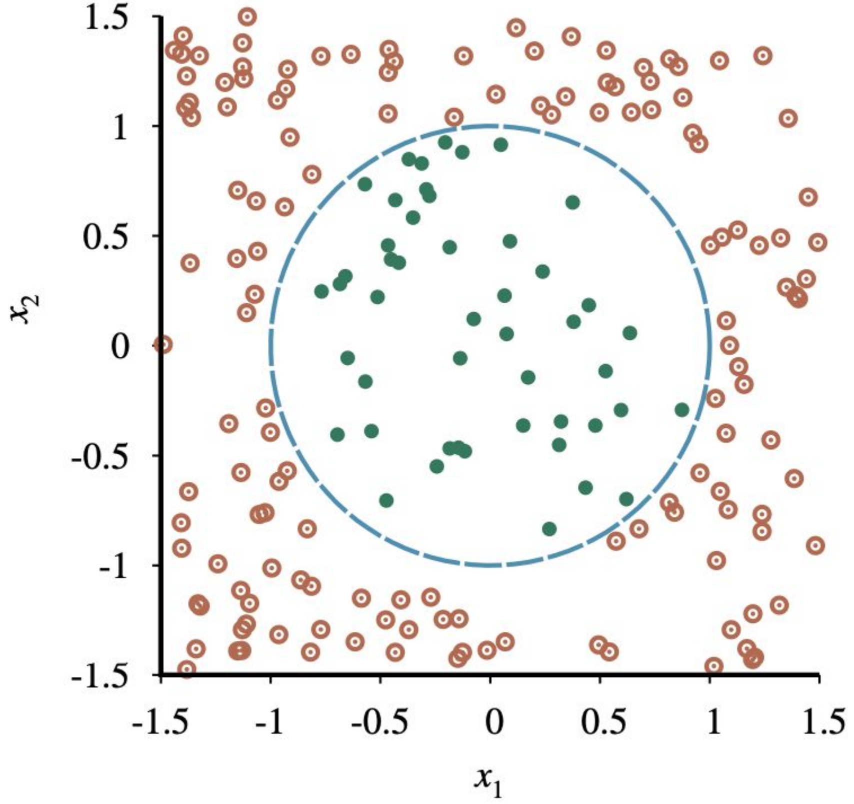
$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b\right)$$

- Weights α_j associated with each data point are zero except for the support vectors
  - the points closest to the separator.
  - They are called “support” vectors because they “hold up” the separating plane.
  - Because there are usually many fewer support vectors than examples, SVMs gain some of the advantages of parametric models.

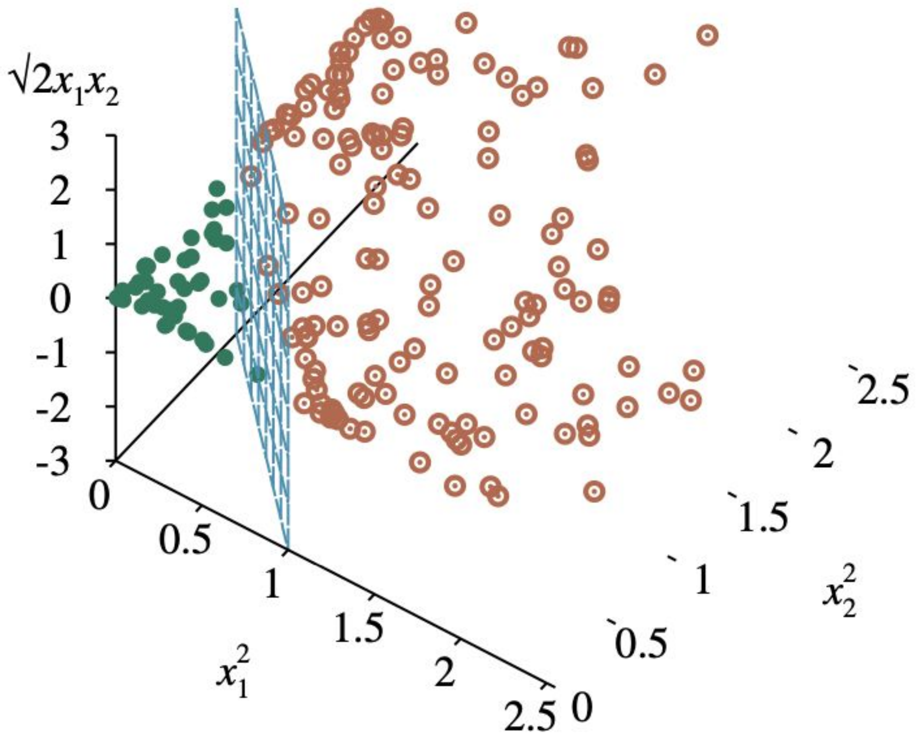
Max-Margin Separator - Meaning of the Constraints


y
x
$\mathbf{w}\cdot\mathbf{x}+b=1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$
$\frac{1}{\|\mathbf{w}\|}$
$\frac{1}{\|\mathbf{w}\|}$
δ_i
$\mathbf{w}$
$\frac{b}{\|\mathbf{w}\|}$

What if the examples are not Linearly Separable?



Mapping Points to a new Feature Space

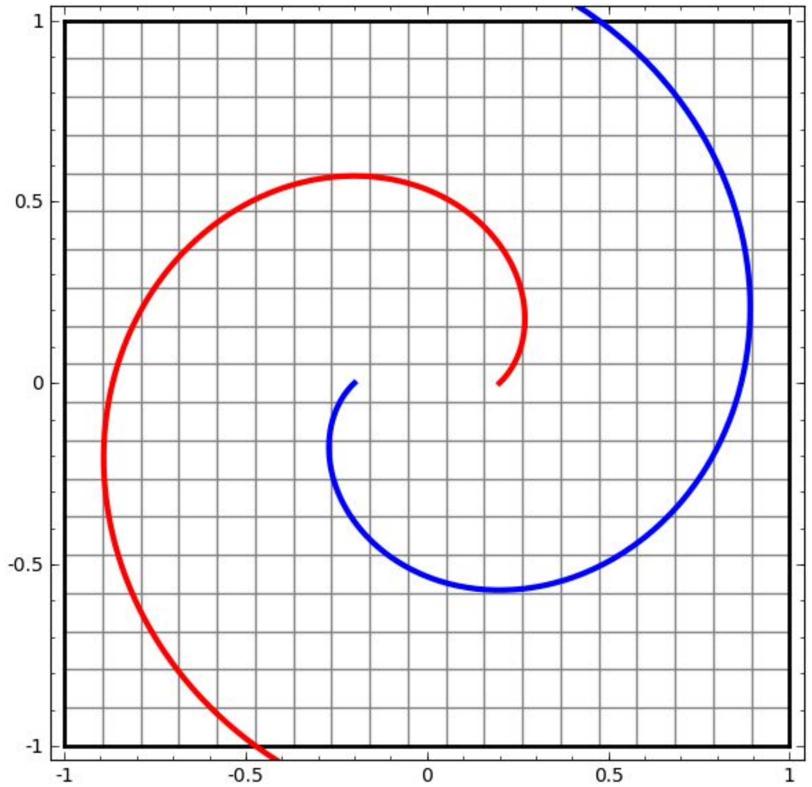


- We **map** each input vector **x** to a **new vector of feature values**, F(**x**).
- In particular, let us use the three features

$$f_1 = x_1^2, \qquad f_2 = x_2^2, \qquad f_3 = \sqrt{2}x_1x_2$$

- The data are linearly separable in this space
- If data are mapped into a space of sufficiently high dimension, then they will almost always be linearly separable

Transformations Can Be Complicated



Kernel Trick

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k \left(\boxed{\mathbf{x}_j \cdot \mathbf{x}_k}\right)$$

- $\mathbf{x}_j \cdot \mathbf{x}_k$ is measuring the distance among $\mathbf{x}_j$ and $\mathbf{x}_k$.
- When we map $\mathbf{x}_j$ and $\mathbf{x}_k$ to a new feature space $F(\mathbf{x}_j)$ and $F(\mathbf{x}_k)$ we can, of course, compute $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ but this won't be interesting.
- What if we could compute $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ directly as a function of $\mathbf{x}_j$ and $\mathbf{x}_k$?
- We can use a function $\mathscr{K}(\mathbf{x}_j, \mathbf{x}_k)$ of $\mathbf{x}_j$ and $\mathbf{x}_k$.
  - I.e., the kernel function can be applied to pairs of input data to evaluate dot products in some corresponding feature space.
- So, we can find linear separators in the higher-dimensional feature space F(x) simply by replacing $\mathbf{x}_j \cdot \mathbf{x}_k$ in the Equation above with $\mathscr{K}(\mathbf{x}_j, \mathbf{x}_k)$.

SAPIENZA
UNIVERSITÀ DI ROMA

What is a Kernel?

- A venerable result in mathematics, Mercer's theorem (1909), tells us that any "reasonable" kernel function corresponds to some feature space.
    - "reasonable" means that the matrix $\mathscr{K}_{jk}=\mathscr{K}(\mathbf{x}_j,\mathbf{x}_k)$ is **positive-definite**.
    - An n×n symmetric real matrix M is said to be positive-definite if $\mathbf{x}^T M\mathbf{x} > 0$ for all $\mathbf{x} \in \mathbb{R}^n$
- These feature spaces can be very large, even for innocuous-looking kernels.
- For example
    - the polynomial kernel, $\mathscr{K}(\mathbf{x}_j,\mathbf{x}_k)=(1+\mathbf{x}_j\cdot\mathbf{x}_k)^d$, corresponds to a feature space whose dimension is exponential in d.
    - A common kernel is the Gaussian: $\mathscr{K}(\mathbf{x}_j,\mathbf{x}_k)=\exp(-\gamma|\mathbf{x}_j-\mathbf{x}_k|^2)$, corresponding to a feature space whose dimension is infinite.

Kernelization

- The kernel method can be applied not only with learning algorithms that find optimal linear separators, but also with any other algorithm that can be reformulated to work only with dot products of pairs of data points.
- Once this is done, the dot product is replaced by a kernel function and we have a kernelized version of the algorithm.

Kernelization of Linear Regression

- If we let $X=[\mathbf{x}_1,\ldots,\mathbf{x}_n]$ and $\mathbf{y}=[y_1,\ldots,y_n]^\top$, the weights of a linear regression model can be written in closed form as $\mathbf{w}=(XX^\top)^{-1}X\mathbf{y}$
- We begin by expressing the solution **w** as a linear combination of the training inputs

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

  such a vector $\vec{\alpha}$ must always exist (the problem is convex)
- Similarly, during testing a test point is only accessed through inner-products with training inputs:

$$h(\mathbf{z}) = \mathbf{w}^\top \mathbf{z} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^\top \mathbf{z}$$

- $\mathbf{x}^\mathsf{T}\mathbf{z}$ can thus be replaced by any kernel k(**x**,**z**)

Kernelization of Linear Regression

Theorem. Kernelized Linear Regression has the solution $\vec{\alpha}$= $K^{-1}y$.

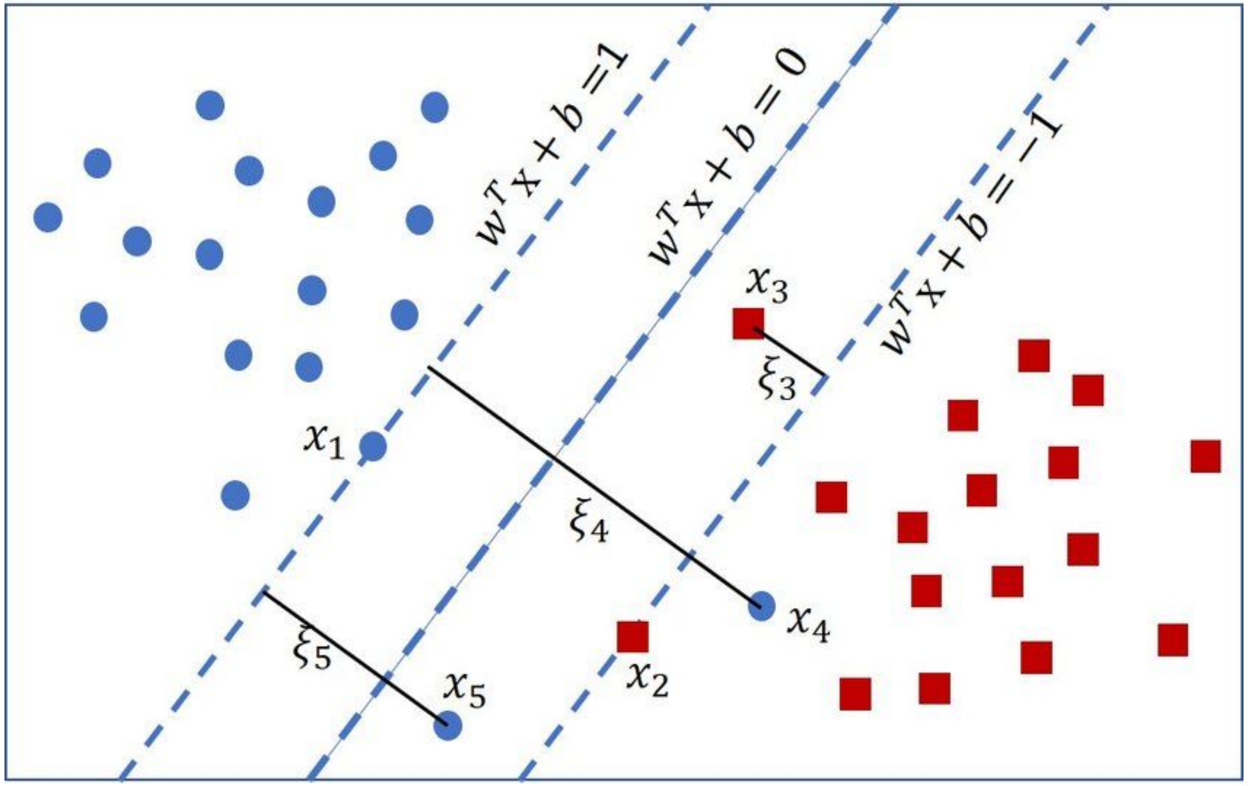
Proof

$$\begin{aligned}
\mathbf{X}\vec{\alpha} &= \mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y} \quad | \text{ multiply from left by } \mathbf{X}^\top\mathbf{X}\mathbf{X}^\top \\
(\mathbf{X}^\top\mathbf{X})(\mathbf{X}^\top\mathbf{X})\vec{\alpha} &= \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1})\mathbf{X}\mathbf{y} \quad |\text{substitute } \mathbf{K} = \mathbf{X}^\top\mathbf{X} \\
\mathbf{K}^2\vec{\alpha} &= \mathbf{K}\mathbf{y} \quad |\text{multiply from left by } (\mathbf{K}^{-1})^2 \\
\vec{\alpha} &= \mathbf{K}^{-1}\mathbf{y}
\end{aligned}$$

Soft Margin

- In the case of inherently noisy data, we may not want a linear separator in some high-dimensional space.
- Rather, we'd like a decision surface in a lower-dimensional space that does not cleanly separate the classes, but reflects the reality of the noisy data.
- That is possible with a **soft margin classifier**
  - which allows examples to fall on the wrong side of the decision boundary, but assigns them a penalty proportional to the distance required to move them back to the correct side.

Soft Margin



Soft Margin As Regularization

- Before, our margins were defined by the hard constraint:

$$y_i(x_i^T w + b) \geq 1 \quad y_i \epsilon \{-1, 1\}$$

- Allowing soft-margins we have the constraint:

$$y_i(x_i^T w + b) \geq 1 - \xi_i \qquad \xi_i \geq 0$$

- The new constraint permits a functional margin that is less than 1, and contains a penalty of cost $C\xi_i$ for any data point that falls within the margin on the correct side of the separating hyperplane (i.e., when $0 < \xi_i \leq 1$), or on the wrong side of the separating hyperplane (i.e., when $\xi_i > 1$).

Scikit-Learn’s SVM

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()
```

Scikit-Learn’s SVM

```
>>> clf.predict([[2., 2.]])
array([1])
```

Scikit-Learn’s SVM

```
>>> # get support vectors
>>> clf.support_vectors_
array([[0., 0.],
       [1., 1.]])
>>> # get indices of support vectors
>>> clf.support_
array([0, 1]...)
>>> # get number of support vectors for each class
>>> clf.n_support_
array([1, 1]...)
```

Fondamenti di IA

End of Lecture

03 - Linear Models, Non-parametric Models and SVM

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri