Machine learning

(521289S)

Prof. Tapio Seppänen
Center for Machine Vision and Signal Analysis
University of Oulu

Two-class supervised learning

- There are many applications in which some items / objects must be placed in either of two categories /classes automatically (classification)
  - For example our toy example in Chapter 1: automatically determining whether a given animal is a Cat or Dog
  - For example clinical decision making: is this patient infected or not
- To design such a classifier, we can apply a machine learning technique in which the input contains all the carefully designed features of the object and the output variable takes on only two discrete values, the class labels
- Compare with regression where the output variable is continuous valued
  - However, regression and classification are conceptually very similar
- A parametric model needs be established for the classifier and optimized through the cost function
- The training data containing lots of example cases must include the true class label of each data item:
  - The optimizer can then consider the two subsets of data to represent the two classes and find a model to distinguish them from each other
  - Classifier validation can be done to test the classifier accuracy by letting the classifier classify unforeseen data items and compare its decisions with the correct class labels
- This is called supervised learning: we inform the optimizer about the true class label of each data item and, thus, guide it to find an optimized solution

Two perspectives on classification

- We can solve our two-class classification problem as a regression problem and fit a regression model to our data to distinguish between the two classes (Regression perspective)
- Or, we can solve the problem by considering fitting linear or nonlinear decision surface in the feature space (Perceptron perspective)
- See Figure 6.1 for a graphical representation of the two perspectives

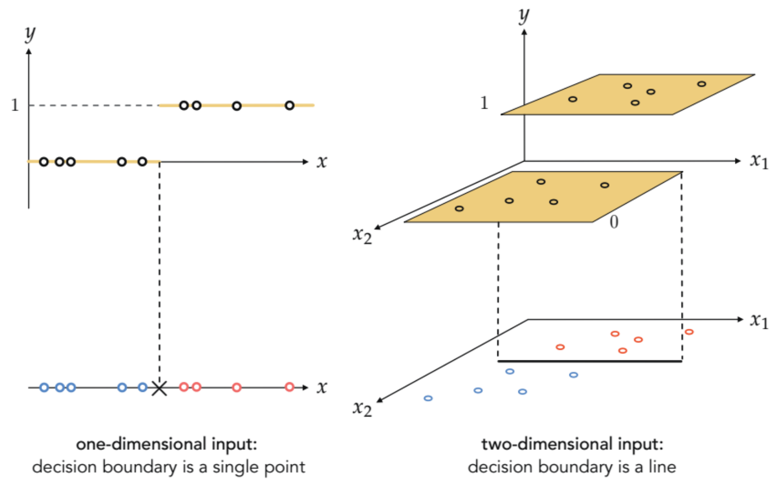

y
1
x
0
x_1
x_2
one-dimensional input:
decision boundary is a single point
two-dimensional input:
decision boundary is a line

Figure 6.1 Two perspectives on classification illustrated using single-input (left column) and two-input (right column) toy datasets. The regression perspective shown in the top panels is equivalent to the perceptron perspective shown in the bottom panels, where we look at each respective dataset from "above." In the Perceptron perspective we also mark the decision boundary. This is where the step function (colored in yellow in the top panels) transitions from its bottom to top step. See text for further details.

Perspective 1: Regression

Logistic regression

- We will first study logistic regression as it is used very commonly in various applications in which data items need be placed in two clategories automatically by considering the input values charactering the data item
- Training data is represented as input/output pairs as before:

$$\left\{\left(\mathbf{x}_p, y_p\right)\right\}_{p=1}^{P} \qquad \mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix} \qquad y_p \in \{0, +1\}$$

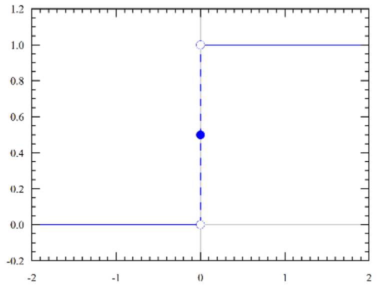
- Note here that the output variable y_p can take on only two values, 0 and 1, representing the two classes
  - Basically, any two discrete numbers would do, but this a common choice in literature
- It is important to have numerical values for the output variable so that we can apply the regression methods shown before in the course

Formulating the classifier

- In order to make decision about which class an object should be placed in, we need a function that yields two output values
- What is the function and the input to that function?
- We input the regression model to a step function

$$\text{step}\left(\mathring{\mathbf{x}}^T\mathbf{w}\right) \qquad \mathring{\mathbf{x}}^T\mathbf{w} = w_0 + x_1w_1 + x_2w_2 + \cdots + x_Nw_N \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \quad \text{and} \quad \mathring{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\text{step}(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t < 0 \end{cases}$$



https://en.wikipedia.org/wiki/Step_function

- Basically: if for any feature vector **x**, the regression line yields a value > 0.5, the step function outputs ’1’, otherwise ’0’
- Figure 6.2: red dashed line represents the step function placed according to the regression line

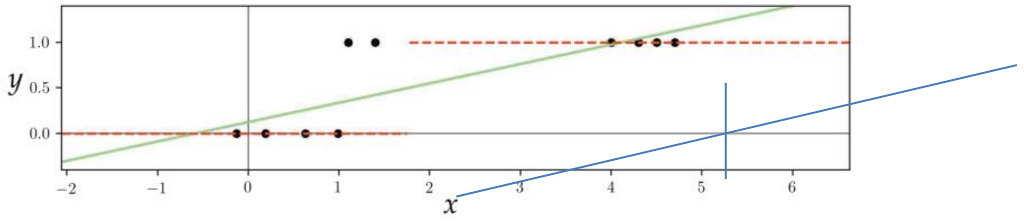


Figure 6.2 Figure associated with Example 6.1. See text for details.

Establishing a cost function

- Important note:
  - We don't do the linear regression on the data first and then apply the step function to make a decision for unforseen new data items
  - Instead: we take the step function including the regression model into our optimizer which then optimizes the performance of this step function (instead of the regression alone)

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{step}\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right) - y_p\right)^2$$

- We optimize regression model parameters **w** so that the step function yields correct estimates of the output variable y_p as often as possible
- This cost function is very problematic as a step function has only two values for a derivative: zero or infinite
  - Thus, a gradient-based optimization may fail

Logistic sigmoid cost function

- Due to the problem with derivatives, we can replace the step function with a ”smooth” version which is differentiable everywhere: sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- The steepness of the sigmoid function can be tuned with a parameter w
  - The larger the w parameter, the better the function approximates the step function

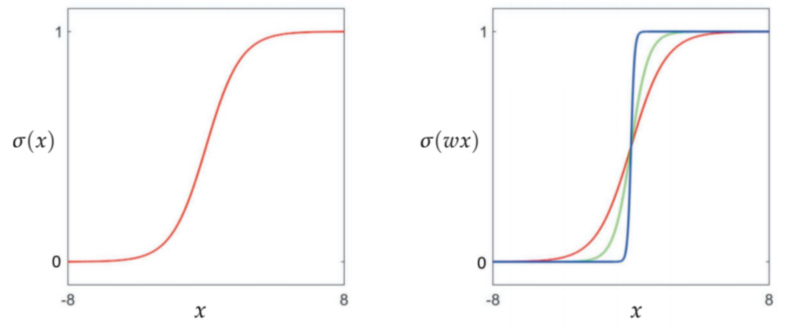


Figure 6.4 (left panel) Plot of the sigmoid function $\sigma(x)$. (right panel) By increasing the weight w in $\sigma(wx)$ from $w = 1$ (shown in red) to $w = 2$ (shown in green) and finally to $w = 10$ (shown in blue), the internally weighted version of the sigmoid function becomes an increasingly good approximator of the step function.

- Use of Least Squares cost function yields logistic regression:
  - Gradient search is now applicable
  - The **w** parameters are here used automatically to optimize the steepness, too

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right) - y_p\right)^2$$

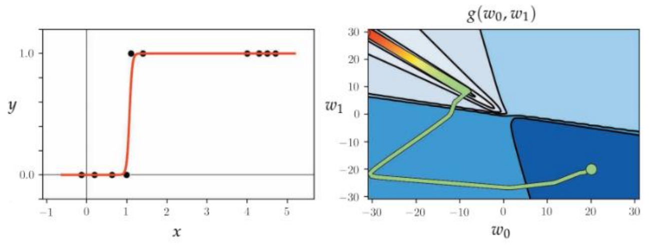


Figure 6.5 Figure associated with Example 6.3. See text for details.

Cross-entropy cost function

- We define point-wise log error as follows:

$$g_p(\mathbf{w}) = \begin{cases} -\log\left(\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right) & \text{if } y_p = 1 \\ -\log\left(1-\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right) & \text{if } y_p = 0 \end{cases}$$

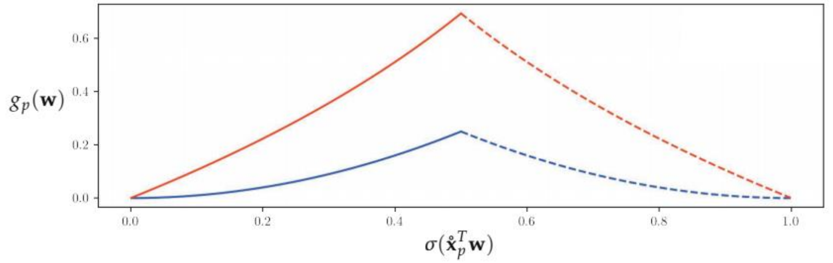


Figure 6.6 Visual comparison of the squared error (in blue) and the log error (in red) for two cases: $y_p = 0$ (solid curves) and $y_p = 1$ (dashed curves). In both cases the log error penalizes deviation from the true label value to a greater extent than the squared error.

- The decision is expected to be '0' or '1'
- But if σ() proposes '0.5', then the error is largest and the cost is largest
- Optimizer will update the parameters so that σ() would propose numbers close to '0' or '1' corresponding to cost function minima

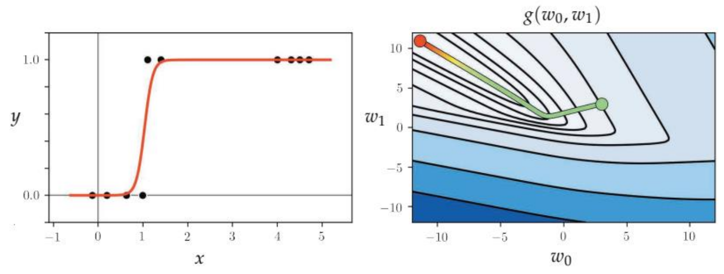


Figure 6.7 Figure associated with Example 6.4. See text for details.

- The point-wise cost function can be written also as:

$$g_p(\mathbf{w}) = -y_p\log\left(\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right) - \left(1-y_p\right)\log\left(1-\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right)$$

- Then, the overall cost function is (Cross Entropy cost function):

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p\log\left(\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right) + \left(1-y_p\right)\log\left(1-\sigma\left(\mathring{\mathbf{x}}_p^T\mathbf{w}\right)\right)$$

Softmax cost function

- Instead of $y_p \in \{0, +1\}$ let's have $y_p \in \{-1, +1\}$
- Let's use sign function for decision making:
- We have a new step function:

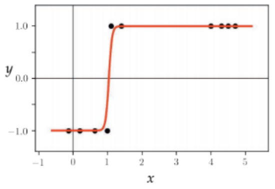
$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$

$$\text{sign}\left(\mathring{\mathbf{x}}^T \mathbf{w}\right)$$

- This has a linear decision boundary (hyperplane) between the two steps: $\mathring{\mathbf{x}}^T \mathbf{w} = 0$
- Let's use the point-wise log errors:

$$g_p(\mathbf{w}) = \begin{cases} -\log\left(\sigma\left(\mathring{\mathbf{x}}_p^T \mathbf{w}\right)\right) & \text{if } y_p = +1 \\ -\log\left(1 - \sigma\left(\mathring{\mathbf{x}}_p^T \mathbf{w}\right)\right) & \text{if } y_p = -1 \end{cases}$$

$$\tanh(x) = 2\,\sigma(x) - 1 = \frac{2}{1 + e^{-x}} - 1$$



- This can be easily manipulated to:

$$g_p(\mathbf{w}) = -\log\left(\sigma\left(y_p\, \mathring{\mathbf{x}}_p^T \mathbf{w}\right)\right)$$

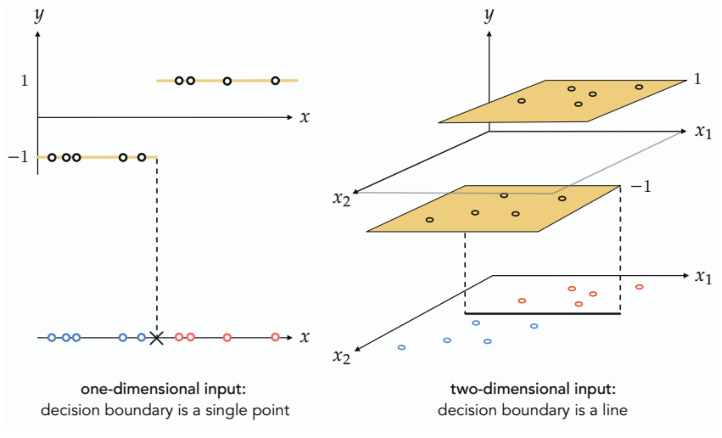
- Plug in sigmoid function to get:

$$g_p(\mathbf{w}) = \log\left(1 + e^{-y_p \mathring{\mathbf{x}}_p^T \mathbf{w}}\right)$$

- Softmax cost function is:

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^{P} \log\left(1 + e^{-y_p \mathring{\mathbf{x}}_p^T \mathbf{w}}\right)$$

  – Always convex!



y
1
x
−1
x
one-dimensional input:
decision boundary is a single point
y
1
x1
x2
−1
x1
x2
two-dimensional input:
decision boundary is a line

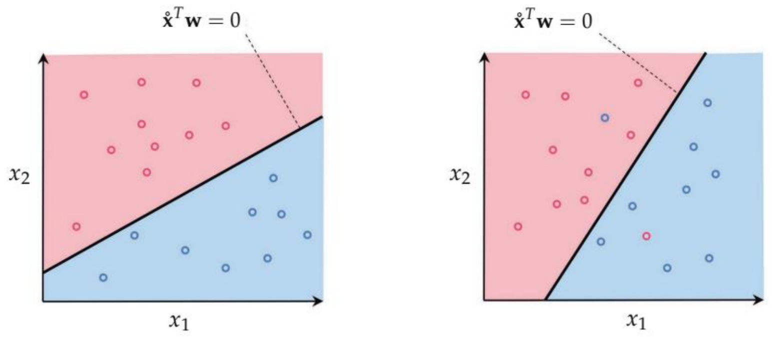
Figure 6.8 The analogous setup to Figure 6.1, only here we use label values $y_p \in \{-1, +1\}$.

Perspective 2: Perceptron

The linear decision boundary

- We can treat classification as a particular form of nonlinear regression with $y_p \in \{-1, +1\}$ and using the sign() function for decision rule in classification
- As we saw before, regression analysis yields a linear decision boundary (hyperplane) between the two classes of objects: $\mathring{\mathbf{x}}^T \mathbf{w} = 0$
- Ideally, the decision boundary separates the two classes perfectly (linearly separable case). In practice, the classes overlap partly

- (Intuitive clarification in 2-D case:
- A line in 2-D space: y = a+bx
- a+bx-y = 0
- a+b*x-1*y = 0
- $w_0 * 1 + w_1 * x_1 + w_2 * x_2 = 0$)



$\mathring{\mathbf{x}}^T \mathbf{w} = 0$
x_2
x_1
$\mathring{\mathbf{x}}^T \mathbf{w} = 0$
x_2
x_1

Figure 6.11 With the Perceptron we aim to directly learn the linear decision boundary $\mathring{\mathbf{x}}^T \mathbf{w} = 0$ (shown here in black) to separate two classes of data, colored red (class +1) and blue (class −1), by dividing the input space into a red half-space where $\mathring{\mathbf{x}}^T \mathbf{w} > 0$, and a blue half-space where $\mathring{\mathbf{x}}^T \mathbf{w} < 0$. (left panel) A linearly separable dataset where it is possible to learn a hyperplane to perfectly separate the two classes. (right panel) A dataset with two overlapping classes. Although the distribution of data does not allow for perfect linear separation, the Perceptron still aims to find a hyperplane that minimizes the number of misclassified points that end up in the wrong half-space.

Perceptron cost function

- All data points on the hyperplane have: $\mathring{\mathbf{x}}^T\mathbf{w} = 0$
- All data points ”above” the hyperplane have: $\mathring{\mathbf{x}}^T\mathbf{w} > 0$
- All data points ”below” the hyperplane have: $\mathring{\mathbf{x}}^T\mathbf{w} < 0$
- We can choose the class labels y_p as follows:

$$\mathring{\mathbf{x}}_p^T\mathbf{w} > 0 \longrightarrow y_p = +1$$
$$\mathring{\mathbf{x}}_p^T\mathbf{w} < 0 \longrightarrow y_p = -1.$$

- This can be rewritten more compactly as: $-y_p\mathring{\mathbf{x}}_p^T\mathbf{w} < 0$
- Let’s define a point-wise cost function: $g_p(\mathbf{w}) = \max\left(0, -y_p\mathring{\mathbf{x}}_p^T\mathbf{w}\right)$
  - Rectified Linear Unit, ReLU
  - All data points on the correct side of the hyperplane yield cost $g_p(w) = 0$
  - All data points on the wrong side of the hyperplane yield cost $g_p(w) > 0$
- The Perceptron cost function (ReLU cost function):

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\max\left(0, -y_p\mathring{\mathbf{x}}_p^T\mathbf{w}\right)$$

  - Always convex, but has discontinuities -> use zero-order and first-order optimization techniques only

- In the overlap case, this still yields optimal solution for the decision boundary

Softmax cost function

- Due to the nonlinearities of the Perceptron cost function, we can use ”soft” version of the ReLU-function: Softmax cost function

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} \log\left(1 + e^{-y_p \mathring{\mathbf{x}}_p^T \mathbf{w}}\right)$$

- This is infinitely many times differentiable -> all previously shown optimization techniques can be used

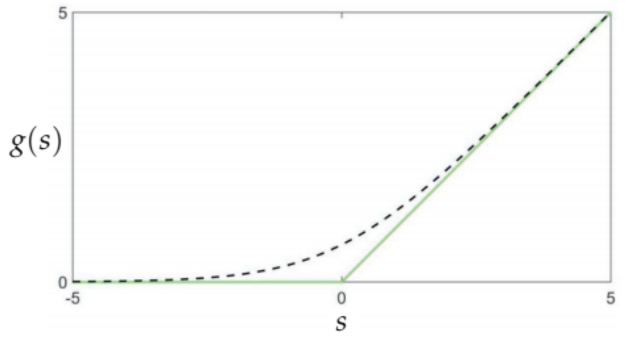
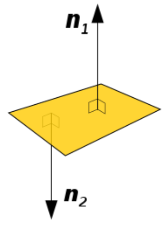


Figure 6.12 Plots of the Perceptron $g(s) = \max(0, s)$ (shown in green) as well as its smooth Softmax approximation $g(s) = \text{soft}(0, s) = \log(1 + e^s)$ (shown in dashed black).

Normalizing the feature-touching weights

- A problem with Softmax cost function optimization: As the exponential term gets minimum value at infinity, the optimizer keeps updating the parameter vector infinitely long with not much effect on the cost value
  - As the results the parameter values **w** tend towards infinity
  - (A geometric interpretation: the length of the hyperplane's normal vector defined by $[w_1,\dots,w_N]^T$ grows infinitely long! Note: parameter w_0 is left out here as it is merely the intercept of the hyperplane.)
- A solution is to normalize all the parameters at every iteration round:

$$w_i \leftarrow \frac{w_i}{\sqrt{\sum_{k=1}^{N} {w_k}^2}}$$



- The relative ratios between the parameters are maintained, but their squared sum is always equal to 1
  - hyperplane's normal vector direction is updated during iteration, but its length is kept constant (=1)

(Figure source: https://commons.wikimedia.org/wiki/File:Normal_vectors2.svg)

Regularizing two-class classification

- The Softmax cost function minimization problem with feature-touching weight normalization can be expressed also as:

$$\begin{aligned} \underset{b,\,\boldsymbol{\omega}}{\text{minimize}} \quad & \frac{1}{P}\sum_{p=1}^{P}\log\left(1+e^{-y_p\left(b+\mathbf{x}_p^T\boldsymbol{\omega}\right)}\right) \\ \text{subject to} \quad & \|\boldsymbol{\omega}\|_2^2 = 1. \end{aligned}$$

$$\text{(bias): } b = w_0 \qquad \text{(feature-touching weights): } \boldsymbol{\omega} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}$$

- Instead of direct feature-touching weight normalization, the optimization problem is often reformulated so that regularization is used:

$$g\left(b,\boldsymbol{\omega}\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1+e^{-y_p\left(b+\mathbf{x}_p^T\boldsymbol{\omega}\right)}\right) + \lambda\,\|\boldsymbol{\omega}\|_2^2$$

- The term $\|\boldsymbol{\omega}\|_2^2$ is referred to as a regularizer, and the parameter $\lambda \geq 0$ is called regularization parameter (to be defined experimentally)
  - Usually a very small value in order to emphasize the Softmax part of the cost function
- Both parts of the cost function have positive values. Thus, the optimizer minimizes the function by searching for small values for both of them.
  - Thus, parameters w_i get small values (and do not tend to infinity!)

Support Vector Machines (SVM)

- SVM is a special form of Perceptron in which the optimizer searches for the decision border that has the largest margin between the two classes
  - Maximum margin decision boundary
  - All data points on both sides are maximally far away from the decision boundary
  - Note: this condition occurs only if the classes are linearly separable!
  - The linear decision surface (hyperplane) is in the middle and the margin/buffer is defined by two parallel hyperplanes at equal distance from the middle
  - The parallel hyperplanes touch the closest data points of both sides

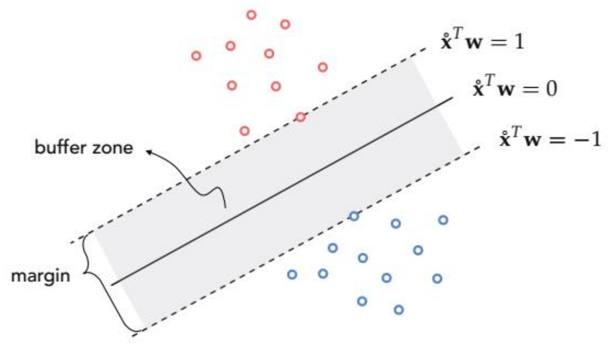

$\mathring{\mathbf{x}}^T\mathbf{w} = 1$
$\mathring{\mathbf{x}}^T\mathbf{w} = 0$
$\mathring{\mathbf{x}}^T\mathbf{w} = -1$
buffer zone
margin

Figure 6.17 For linearly separable data the width of the buffer zone (in gray) confined between two evenly spaced translates of a separating hyperplane that just touch each respective class, defines the margin of that separating hyperplane.

Margin-Perceptron

- Let’s first extend basic Perceptron model to include margins
- In general, the parallel hyperplanes on both sides are at distance β from the linear decision surface
  - The hyperplane equations are defined by: $\mathring{\mathbf{x}}^T\mathbf{w} = +\beta$ and $\mathring{\mathbf{x}}^T\mathbf{w} = -\beta$
- However, we can divide both sides by β and get: $\mathbf{w} \longleftarrow \frac{\mathbf{w}}{\beta}$
- Thus, it is mathematically equivalent to state that: $\mathring{\mathbf{x}}^T\mathbf{w} = \pm 1$
  - The optimizer will find optimal values for **w** in both cases
- We have now more convenient mathematical formulation for the data points in the two classes:

$$\begin{array}{ll} \mathring{\mathbf{x}}^T\mathbf{w} \geq 1 & \text{if } y_p = +1 \\ \mathring{\mathbf{x}}^T\mathbf{w} \leq -1 & \text{if } y_p = -1 \end{array}$$

which can be expressed in the familiar form as: $y_p\,\mathring{\mathbf{x}}^T\mathbf{w} \geq 1$

- The Margin-Perceptron cost can be written now in a similar way as before:

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\max\left(0,\, 1 - y_p\,\mathring{\mathbf{x}}_p^T\mathbf{w}\right)$$

SVM cost function

- It can be shown that the maximization of the margin can be done by minimizing the length of the normal vector $\boldsymbol{\omega}$ while minimizing the Margin-Perceptron cost
  - constrained optimization problem (hard-margin support vector problem):

$$\begin{aligned}&\underset{b,\,\boldsymbol{\omega}}{\text{minimize}}\;\; \|\boldsymbol{\omega}\|_2^2 \\ &\text{subject to}\;\; \max\left(0,\, 1 - y_p\left(b + \mathbf{x}_p^T\boldsymbol{\omega}\right)\right) = 0, \quad p = 1, \ldots, P.\end{aligned}$$

- We can solve this by applying regularization approach as before:

$$g\left(b, \boldsymbol{\omega}\right) = \frac{1}{P}\sum_{p=1}^{P}\max\left(0,\, 1 - y_p\left(b + \mathbf{x}_p^T\boldsymbol{\omega}\right)\right) + \lambda\left\|\boldsymbol{\omega}\right\|_2^2$$

- Softmax version of SVM cost function:

$$g\left(b, \boldsymbol{\omega}\right) = \frac{1}{P}\sum_{p=1}^{P}\max\left(0,\, \log\left(1 + e^{1 - y_p \mathring{\mathbf{x}}_p^T\mathbf{w}}\right)\right) + \lambda\left\|\boldsymbol{\omega}\right\|_2^2$$

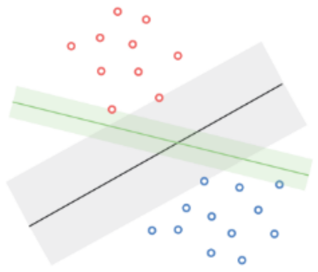
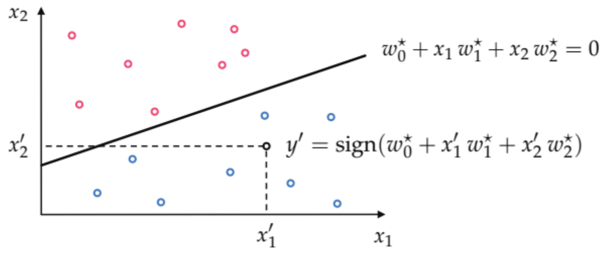


Figure 6.18 Infinitely many linear decision boundaries can perfectly separate a dataset like the one shown here, where two linear decision boundaries are shown in green and black. The decision boundary with the maximum margin – here the one shown in black – is intuitively the best choice. See text for further details.

Perceptron and SVM in classification

- For any new data point ***x***', we can classify it into one of the two classes by computing the sign of the model output *y*': $\text{sign}\left(\mathring{\mathbf{x}}'^T \mathbf{w}^\star\right)$



x_2
$w_0^\star + x_1 w_1^\star + x_2 w_2^\star = 0$
x_2'
$y' = \text{sign}(w_0^\star + x_1' w_1^\star + x_2' w_2^\star)$
x_1'
x_1

Figure 6.21 Once a decision boundary has been learned for the training dataset with optimal parameters $w_0^\star$ and $\mathbf{w}^\star$, the label y of a new point $\mathbf{x}$ can be predicted by simply checking which side of the boundary it lies on. In the illustration shown here $\mathbf{x}$ lies below the learned hyperplane, and as a result is given the label $\text{sign}\left(\mathring{\mathbf{x}}'^T \mathbf{w}^\star\right) = -1$.

- Often in practical applications, the two classes are not linearly separable but overlap partly
  - One cannot find a hyperplane to separate the classes perfectly
  - However, the minimization of cost functions will drive the solution towards optimal separation of the classes
- In practise, logistic regression, Perceptron and SVM yield very similar solutions!

Nonlinear two-class classification

- The linear model of classification can be straight forwardly extended to nonlinear decision surfaces
  - From linear model to nonlinear model (choose proper nonlinearity!):

$$\text{model}(\mathbf{x}, \Theta) = \mathring{\mathbf{f}}^T \mathbf{w}$$

  - Softmax cost function:

$$g(\Theta) = \frac{1}{P} \sum_{p=1}^{P} \log\left(1 + e^{-y_p \mathring{\mathbf{f}}_p^T \mathbf{w}}\right)$$

  - Classification:

$$y = \text{sign}\left(\mathring{\mathbf{f}}^T \mathbf{w}\right)$$

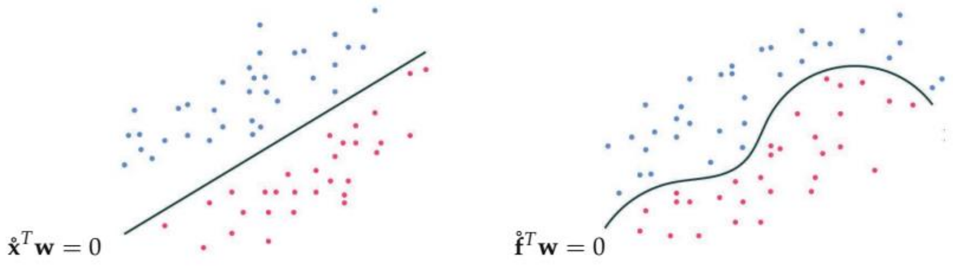


Figure 10.8 Figurative illustrations of linear and nonlinear two-class classification. (left panel) In the linear case the separating boundary is defined as $\mathring{\mathbf{x}}^T \mathbf{w} = 0$. (right panel) In the nonlinear case the separating boundary is defined as $\mathring{\mathbf{f}}^T \mathbf{w} = 0$. See text for further details.

Example of nonlinear classification: 1-D case

- Linear on the left, nonlinear on the right
- Quadratic model is a sufficient nonlinearity here

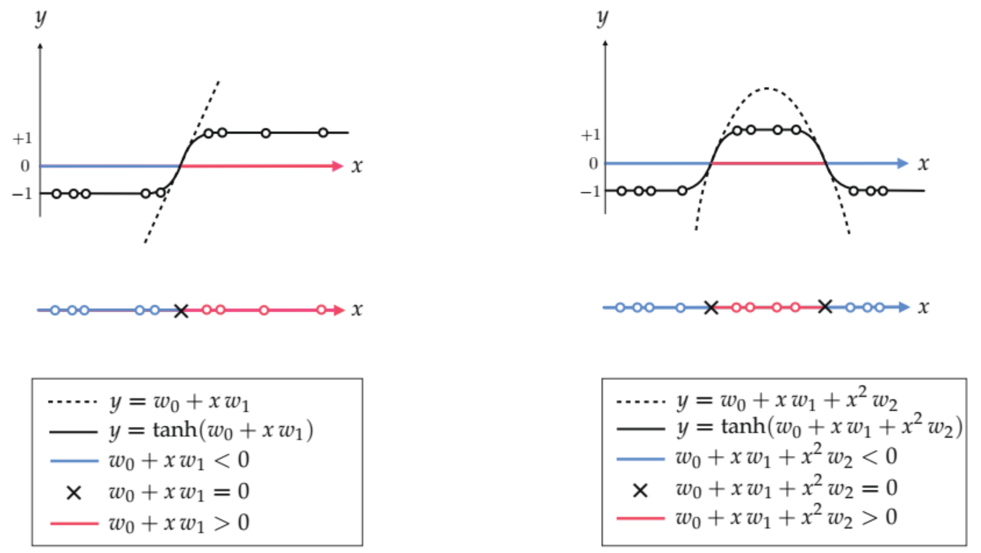


Figure 10.9 Figure associated with Example 10.4. (left column) A prototypical linear two-class classification dataset with fully tuned linear model shown from the regression perspective (top), and from the perceptron perspective (bottom) where label values are encoded as colors (red for +1 and blue for −1). (right column) A simple nonlinear two-class classification dataset that requires a decision boundary consisting of two points, something a linear model cannot provide. As can be seen here, a quadratic model can achieve this goal (provided its parameters are tuned appropriately).

$$f_1(x) = x$$

$$\text{model}(x,\Theta) = w_0 + \underset{\uparrow\, f_2(x) = x^2}{\overset{f_1(x) = x\,\downarrow}{x w_1 + x^2}} w_2$$

$$f_2(x) = x^2$$

Example of nonlinear classification: 2-D case

- Ellipsoidal model is a sufficient nonlinearity here
- On the right: problem is linear in the transformed feature space!

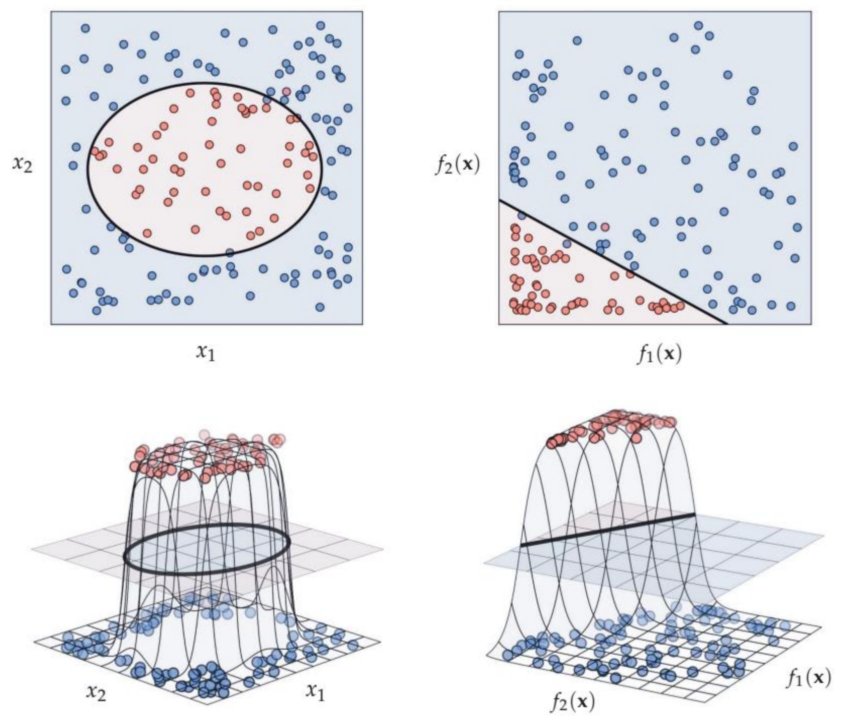


Figure 10.11 Figure associated with Example 10.5. See text for details.

$$f_1(\mathbf{x}) = x_1^2$$

$$\text{model}(\mathbf{x}, \Theta) = \underbrace{w_0 + x_1^2 w_1 + x_2^2 w_2}$$

$$f_2(\mathbf{x}) = x_2^2$$

Balancing of different class sizes

- If one class contains much more data than the other, it will dominate the optimization procedure and the hyperplane is ”pushed” towards/over the smaller class
  - the model will not generalize well but produces lot of misclassifications with new data
- Weighing can be used to balance the point-wise costs. For example, the Softwax cost function becomes

$$g(\mathbf{w}) = \sum_{p=1}^{P} \beta_p \log\left(1 + e^{-y_p \text{model}(x_p, \mathbf{w})}\right)$$

- The β_p values are fixed for all data items of each class. Usually, they are the inverse of the class sizes:

$$\beta_{+1} \propto \frac{1}{|\Omega_{+1}|}$$
$$\beta_{-1} \propto \frac{1}{|\Omega_{-1}|}.$$

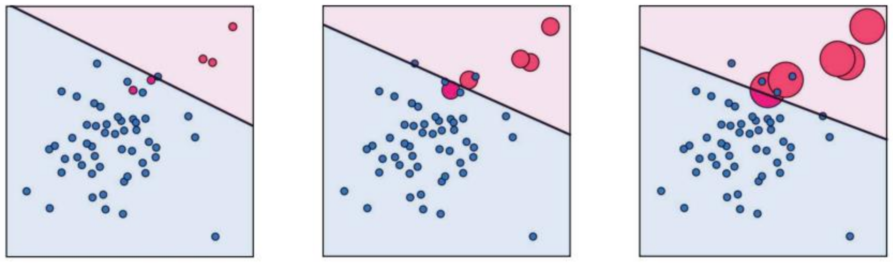


Figure 6.25 Figure associated with Example 6.12. See text for details.

Simple quality metrics for classification

- The accuracy of classification needs be estimated from the data
- The data is run through the classifier and a confusion matrix is computed:

$$\begin{array}{ccc} & & \text{predicted label} \\ & & \begin{array}{cc} +1 & \quad -1 \end{array} \\ \text{actual label} & \begin{array}{c} +1 \\ -1 \end{array} & \begin{bmatrix} A & B \\ C & D \end{bmatrix} \end{array}$$

  - A: number of cases with actual label ’+1’ and predicted class ’+1’
  - B: number of cases with actual label ’+1’ and predicted class ’-1’
  - C: number of cases with actual label ’-1’ and predicted class ’+1’
  - D: number of cases with actual label ’-1’ and predicted class ’-1’
- Sensitivity = A/(A+B) (computed over the ’+1’ class data)
- Specificity = D/(C+D) (computed over the ’-1’ class data)
- Balanced accuracy = $\frac{1}{2}\frac{A}{A+B} + \frac{1}{2}\frac{D}{C+D}$
  - (Balancing over classes with potentially different number of data items)