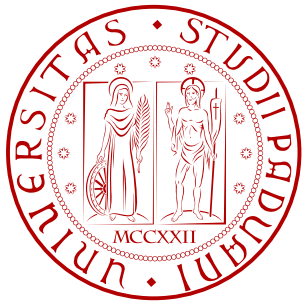
A PRIMER ON REGRESSION AND CLASSIFICATION MODELS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



Outline

- Regression
  - Linear space
  - Non-linear (feature) space
  - Logistic regression problem
  - Regularization
- Classification, class conditional probabilities
  - 2 classes
  - K>2 classes

REGRESSION

Regression

- Fitting a given set of N data points

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$$

- Outputs are (here) scalar
- Input points are (column) vectors of D elements

$$\boldsymbol{x}_i = [x_{1,i} \;\; x_{2,i} \;\; \ldots \;\; x_{D,i}]^T$$

- Linear regression models
  - Finding a best *fitting line* in 2D (D=1), a best fitting *plane* in 3D (D=2) or a *hyper-plane* in D+1 dimensions when input vectors have D elements, with D>2

Linear regression objective

- We introduce a (column) vector of weights

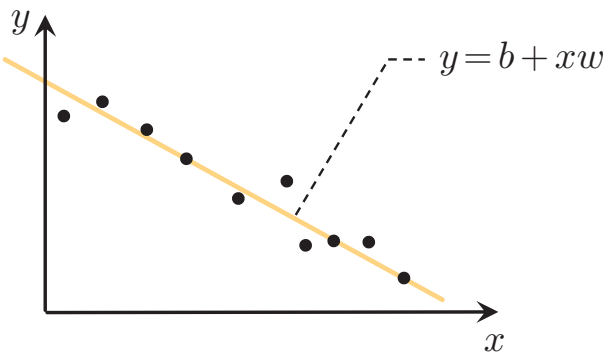
$$\boldsymbol{w} = [w_1 \;\; w_2 \;\; \ldots \;\; w_D]^T$$

- Linear regression
  - amount to finding the *best fitting hyper-plane*

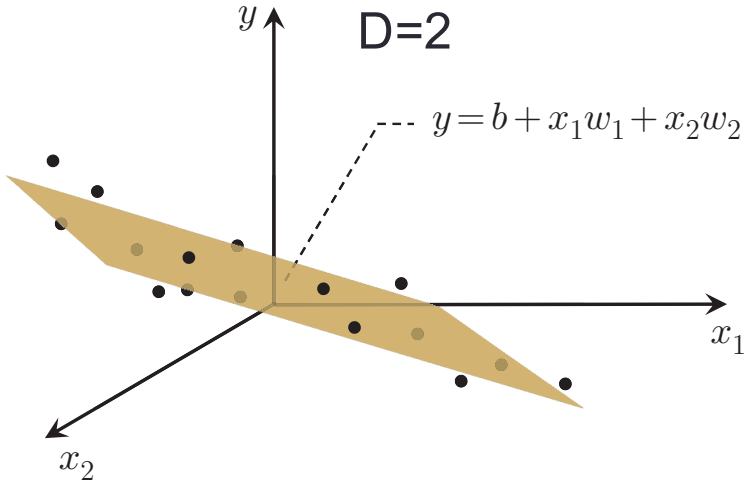
$$b + \boldsymbol{x}_i^T \boldsymbol{w} \approx y_i \,, \;\; i = 1, 2, \ldots, N$$

  - Input $\mathbf{x}_i$ are referred to as input features
  - b is a constant, referred to as the bias

D=1



D=2



Common cost function

- To find the best fitting hyper-plane
  - it is common to use the Least squares cost function

$$g(b, \boldsymbol{w}) = \sum_{i=1}^{N} \left(b + \boldsymbol{x}_i^T \boldsymbol{w} - y_i\right)^2$$

  - Minimize sum of squared differences between
    - Input features and output points
    - Across the entire dataset

- Objective:

$$\underset{b,\boldsymbol{w}}{\text{minimize}} \sum_{i=1}^{N} \left(b + \boldsymbol{x}_i^T \boldsymbol{w} - y_i\right)^2$$

Compact notation

$$\tilde{\boldsymbol{x}}_i = \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix} \qquad \tilde{\boldsymbol{w}} = \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}$$

- This notation trick is used all the times with ANNs
- With this notation, we can rewrite the cost function as:

$$g(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{N} (\tilde{\boldsymbol{x}}_i^T \tilde{\boldsymbol{w}} - y_i)^2$$

- This function is (quadratic) *always convex*
  - Regardless of the input data points

The gradient

- Applying the chain rule for derivatives, we get:

$$\nabla g(\tilde{\boldsymbol{w}}) = 2\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i(\tilde{\boldsymbol{x}}_i^T\tilde{\boldsymbol{w}} - y_i) = 2\left(\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i\tilde{\boldsymbol{x}}_i^T\right)\tilde{\boldsymbol{w}} - 2\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i y_i$$

- We can perform gradient descent to minimize the cost
- However, in this (rare) instance
  - We can set the gradient equal to zero and solve for $\tilde{\boldsymbol{w}}$
  - This leads to the following system of equations:

$$\left(\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i\tilde{\boldsymbol{x}}_i^T\right)\tilde{\boldsymbol{w}} = \sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i y_i$$

Direct solution

- If matrix $\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T$ is invertible, we obtain:

$$\tilde{\boldsymbol{w}} = \left(\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T \right)^{-1} \sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i y_i$$

- This is, in practice, more computationally demanding than solving the system of equations

The efficacy of a learned model

- With the optimal parameters

$$\tilde{\boldsymbol{w}}^\star = \begin{bmatrix} b^\star \\ \boldsymbol{w}^\star \end{bmatrix}$$

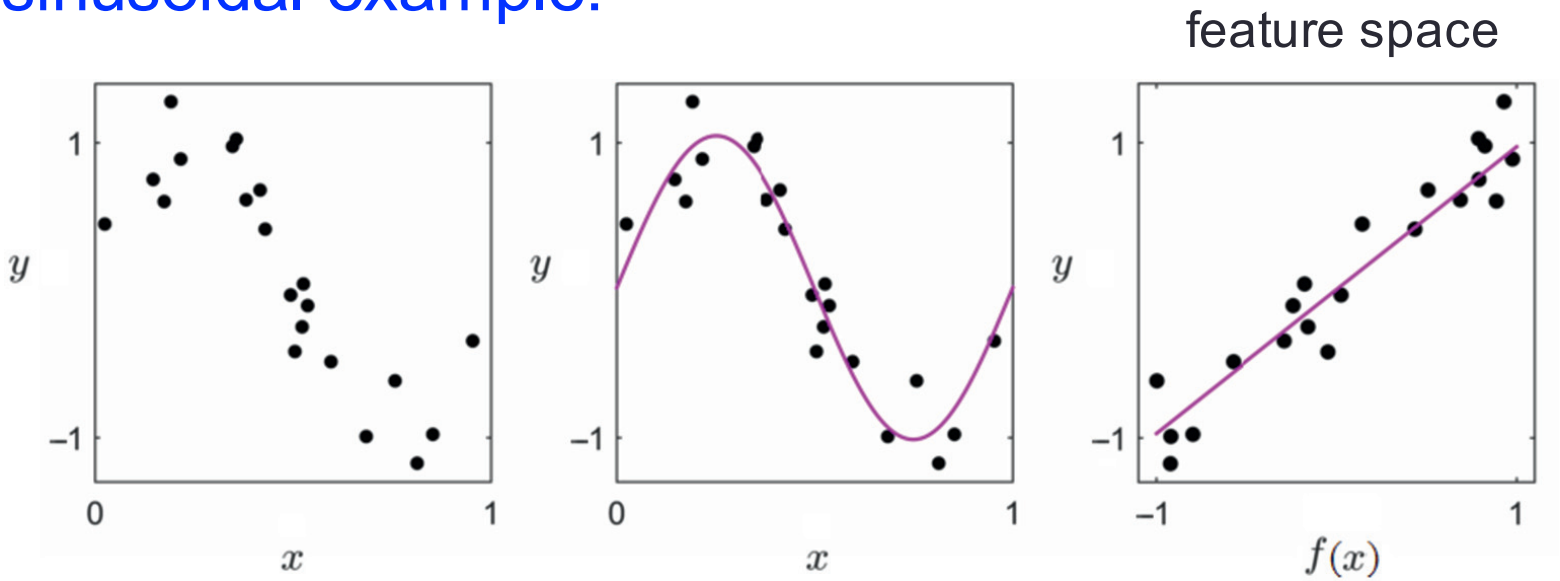
- We can compute the *efficacy* of a learned model
- Using the Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(b^\star + \boldsymbol{x}_i^T\boldsymbol{w}^\star - y_i)^2$$

- Sometimes it is a good idea to compute the MSE with *new testing data* (not used for training): to provide assurance that the model works well on future data points

Non linear data (1/2)

- Often, data do not follow a linear regression model
- Solution: represent the data in a suitable feature space
- (nosy) sinusoidal example:



(left panel) A simulated regression dataset. (middle panel) A weighted form of a simple sinusoid $y = b + f(x)\,w$ (in magenta), where $f(x) = \sin(2\pi x)$ and where b and w are tuned properly, describes the data quite well. (right panel) Fitting a sinusoid in the original feature space is equivalent to fitting a line in the transformed feature space where the input feature has undergone feature transformation $x \longrightarrow f(x) = \sin(2\pi x)$.

Non linear data (2/2)

- This amounts to using a non-linear transformation

$$\boldsymbol{f}_i \stackrel{\Delta}{=} f(\boldsymbol{x}_i) \qquad \tilde{\boldsymbol{f}}_i = \begin{bmatrix} 1 \\ \boldsymbol{f}_i \end{bmatrix}$$

- Use this for the linear fitting in place of the original data
- The new cost function is:

$$g(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{N} (\tilde{\boldsymbol{f}}_i^T \tilde{\boldsymbol{w}} - y_i)^2$$

- Is still *quadratic* and *convex*

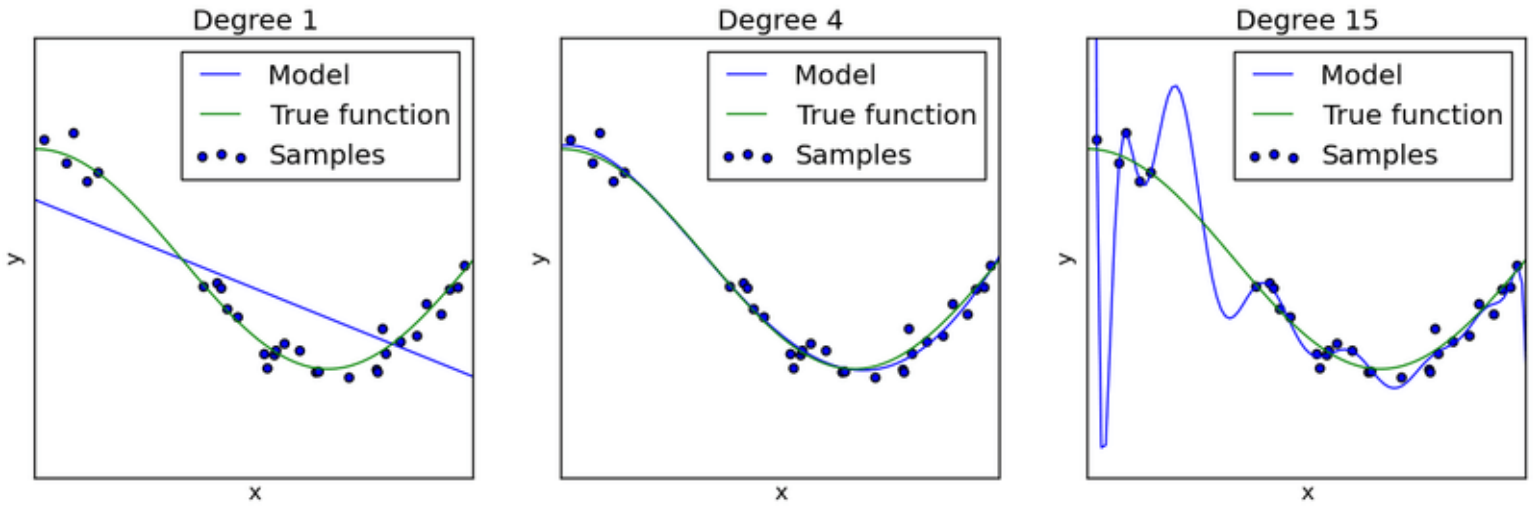
Take away message

A properly designed feature (or set of features) for linear regression provides a good *nonlinear* fit in the original feature space and, simultaneously, a good *linear* fit in the transformed feature space.

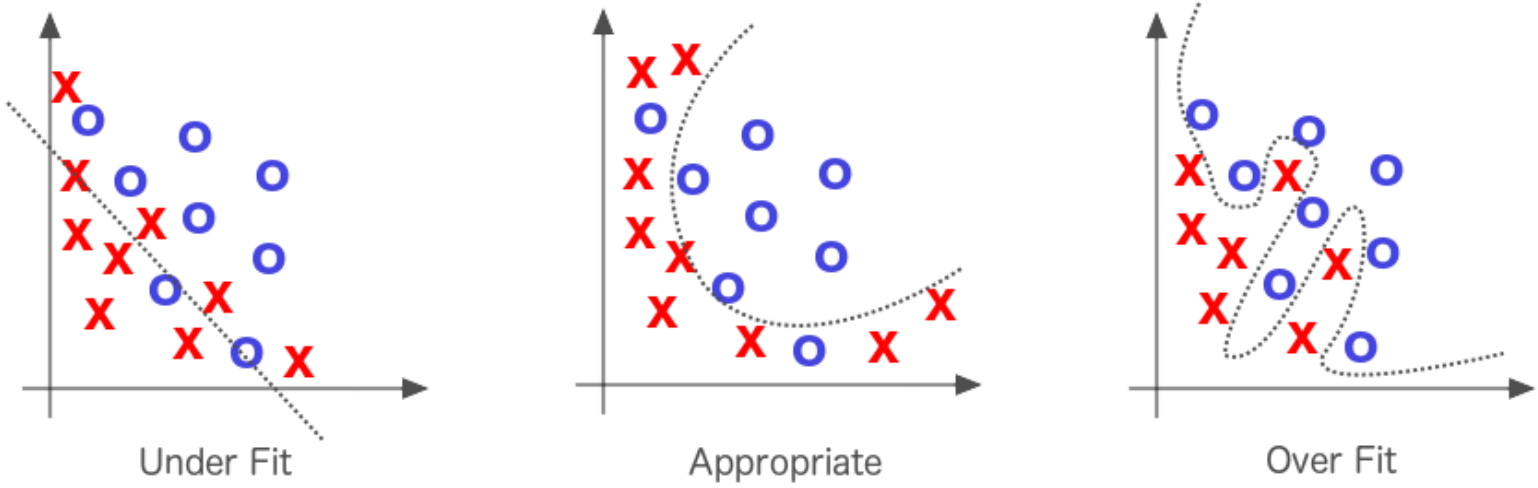
- Meaning of "good" deserves some attention
  - We do not want to "overfit" the data
  - Overfitting
    - production of a model that corresponds *closely* or *exactly* to a particular set of data, and may therefore fail to fit additional (new) data or *predict future observations reliably*
    - essence of overfitting - having extracted some of the residual variation (representing noise), as if that variation represented *underlying model structure*. This produces models with *too many parameters*

Overfitting: visual examples

Regression:



Classification:



Non-linear regression

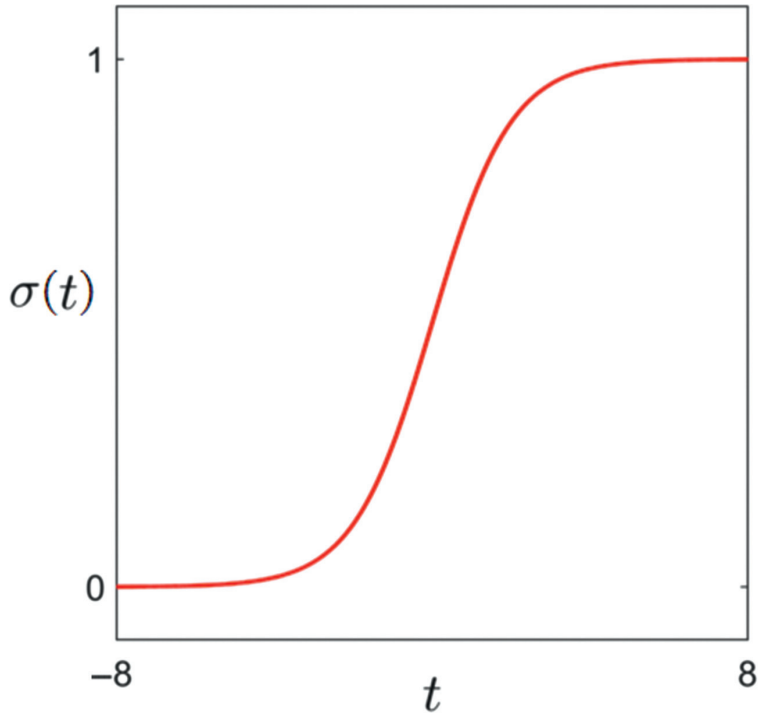
- Previous model
  - Nonlinear relation was easy to assess
  - Cost function remained convex and
  - Input (x) / output (y) relationship was linear the parameters (b, **w**):
  - *It still is a linear regression* problem

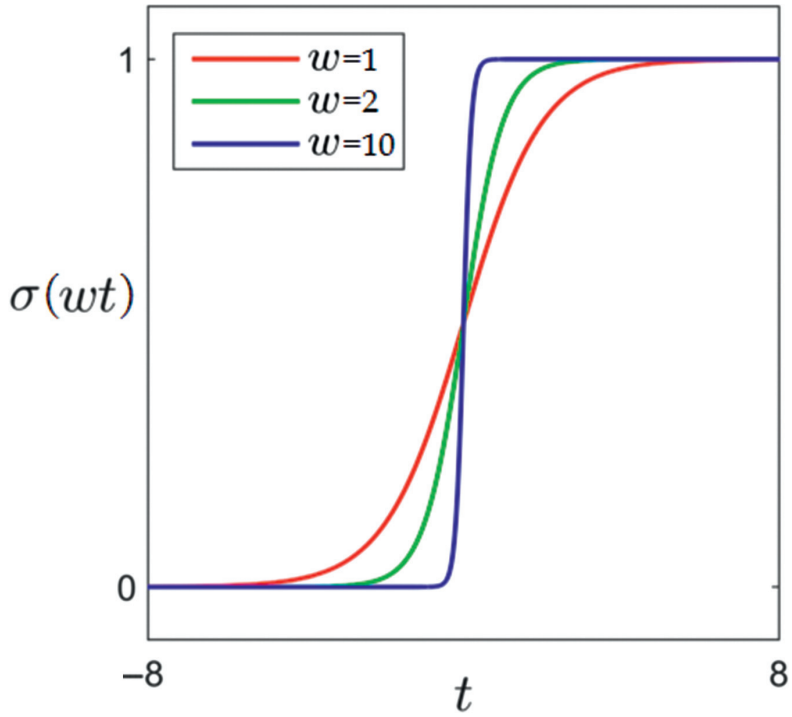
$$b + \underbrace{f(\boldsymbol{x}_i)^T \boldsymbol{w}}_{\text{linear}} \approx y_i$$

- Next, we explore
  - Nonlinear models for regression where
    - Cost function is no longer convex
    - Input (x) / output (y) relationship nonlinear in its parameters
    - ℓ_2 regularization

- To be concrete: we focus on *logistic regression*, but the discussion applies to a broader set of problems

Logistic sigmoid function $\sigma(t) = \frac{1}{1+e^{-t}}$





- Devised by Pierre François Verhulst in the early 19th century
  - To model how a population (bacteria, animal species, etc.) grows in time
  - Intuition: regardless of the environment, the system only has a finite amount of resources (space, food, etc.) → there must be a cap on the total population in any biological system → saturation point

Logistic regression problem (1/2)

- If a dataset is distributed according to the logistic function
- Then, this data must satisfy:

$$\sigma(b + \boldsymbol{x}_i^T \boldsymbol{w}) \approx y_i \ , \ i = 1, 2, \ldots, N$$

- So we have a nonlinear system of P equations
- Non-linearity leads to a non-convex Least Squares cost function

$$g(b, \boldsymbol{w}) = \sum_{i=1}^{N} (\sigma(b + \boldsymbol{x}_i^T \boldsymbol{w}) - y_i)^2$$

$$g(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{N} (\sigma(\tilde{\boldsymbol{x}}_i^T \tilde{\boldsymbol{w}}) - y_i)^2$$ compact notation

Logistic regression problem (2/2)

- Note that (key property):

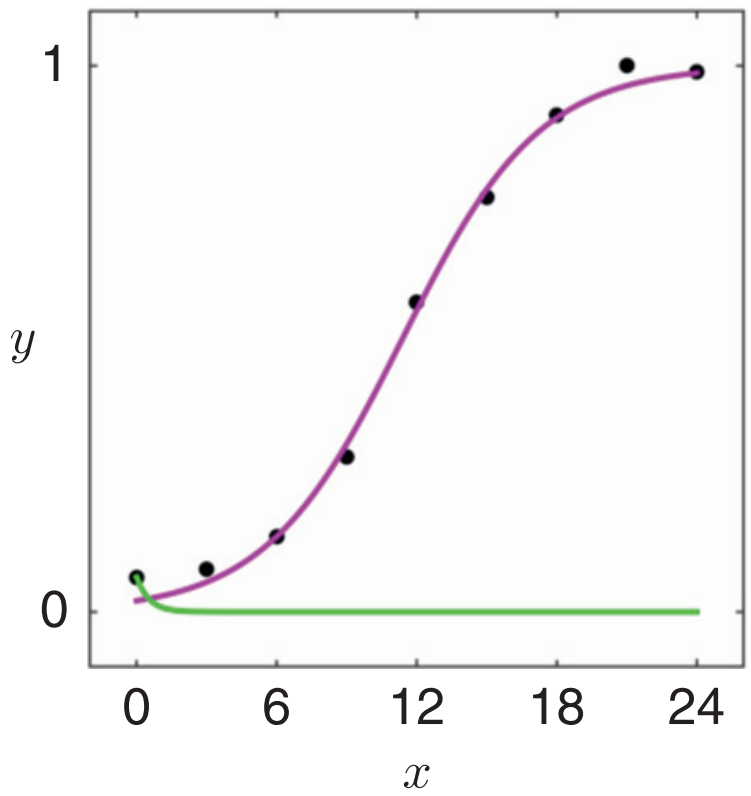
$$\sigma'(t) \triangleq \frac{d\sigma(t)}{dt} = -\frac{(1+e^{-t})'}{(1+e^{-t})^2} = \frac{e^{-t}}{(1+e^{-t})^2} = \sigma(t)(1-\sigma(t))$$

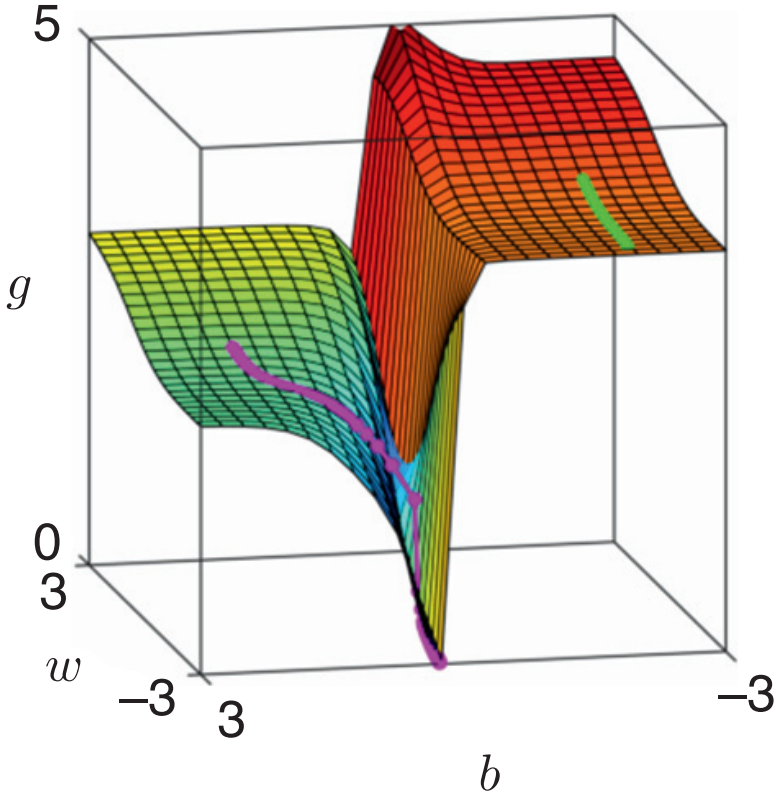
- Using this and the chain rule for derivatives,

$$\nabla g(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{N} 2\left(\sigma(\tilde{\boldsymbol{x}}_i^T\tilde{\boldsymbol{w}}) - y_i\right)\sigma(\tilde{\boldsymbol{x}}_i^T\tilde{\boldsymbol{w}})(1-\sigma(\tilde{\boldsymbol{x}}_i^T\tilde{\boldsymbol{w}}))\tilde{\boldsymbol{x}}_i$$

- Due to the involved non-linearities
  - Finding directly a minimum of the cost function *is a fruitless venture*
  - A numerical technique (e.g., Newton's method) may be used

Finding the minimum …





- Two fits found minimizing the least square cost
- Using gradient descent with a different initialization
- The cost function is *no longer convex*

Regularization

- The problematic flat areas *of non-convex functions*
  - Can be ameliorated using a regularizer
  - Is a simple convex function that is added to the original cost function
  - Has the effect of *slightly convexifying the cost function*
  - The most common regularizer is the squared ℓ_2 norm

$$\|\boldsymbol{w}\|^2 = \sum_{i=1}^{D} w_i^2$$

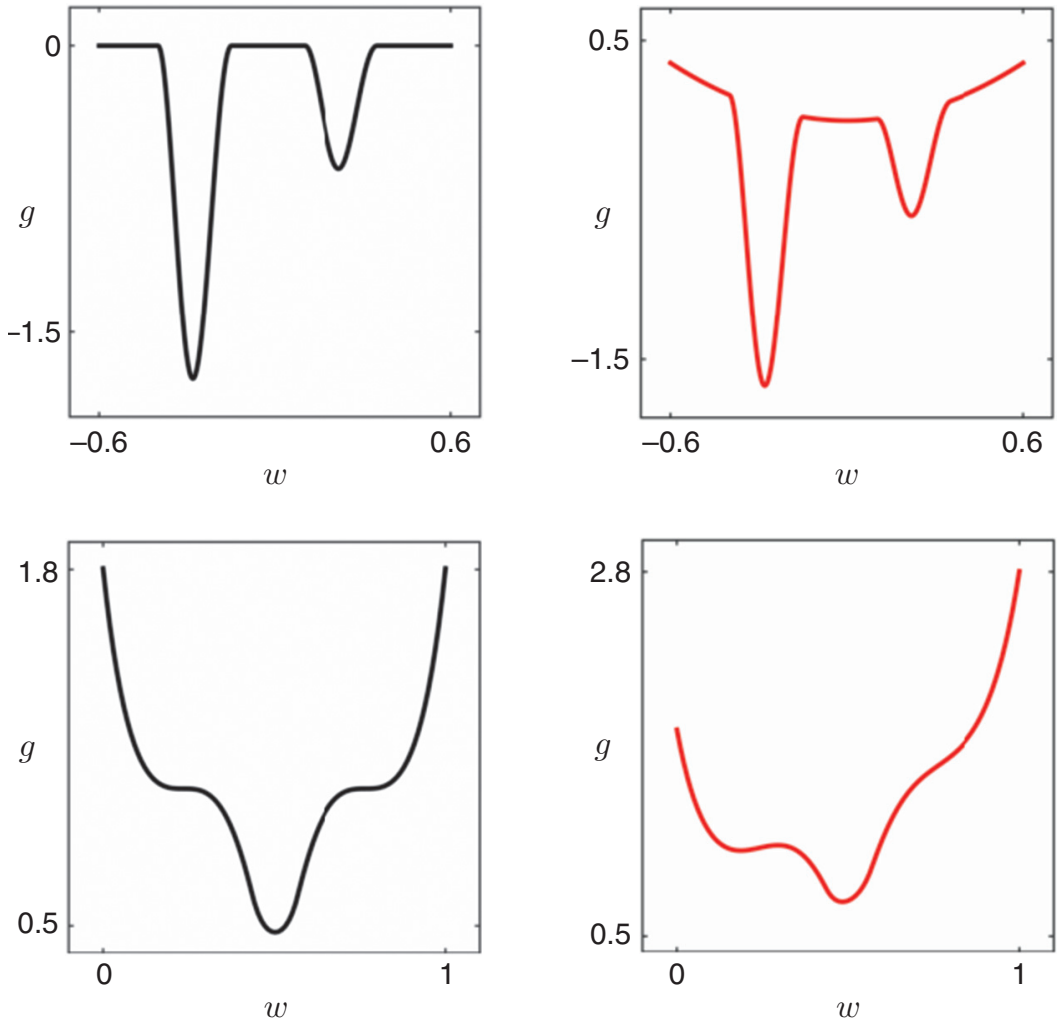
  - The regularized cost function is:

$$g(b, \boldsymbol{w}) + \lambda \|\boldsymbol{w}\|^2$$

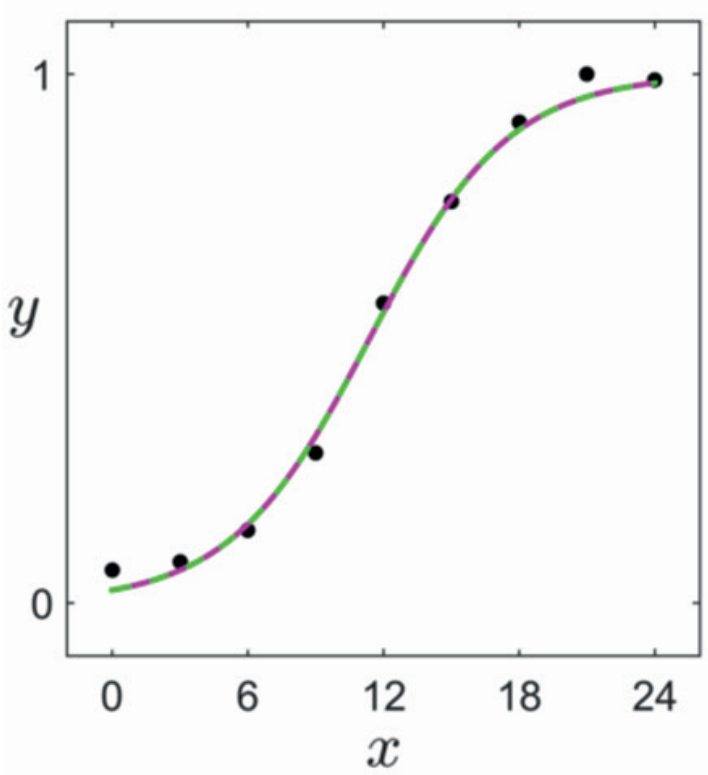
  - λ is set by the user and weighs the importance of the two terms

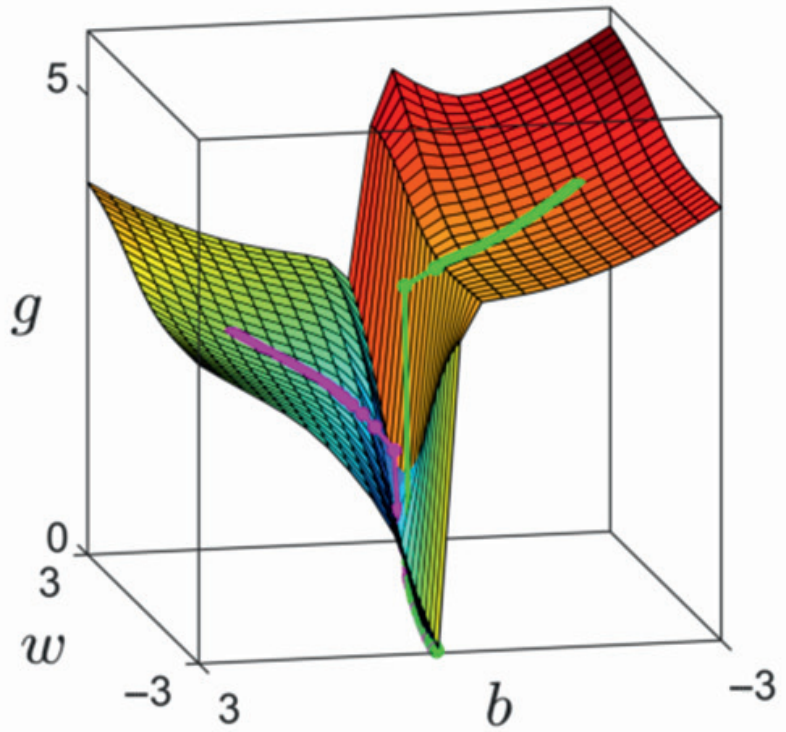
Regularization - examples

- λ=1 (pretty large)
- The regularizer helps
  - remove flat areas
  - remove saddle points
  - remove local minima



Regularized logistic regression





- λ=0.1
- Now, both initializations lead to the same global minimum
- Surface is still non-convex, but the large flat area that led to a poor solution for the green path has been curved upwards

CLASSIFICATION

Two Classes (1/2)

- Classes are: C1 and C2
- Probabilistic view of clustering
  - we model the class conditional densities $p(\boldsymbol{x}|\mathrm{C}_i)\ ,\ i=1,2$
  - and the class priors $p(\mathrm{C}_i)$
- Posterior probability of class C1 can be written as:

$$p(\mathrm{C}_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathrm{C}_1)p(\mathrm{C}_1)}{p(\boldsymbol{x}|\mathrm{C}_1)p(\mathrm{C}_1) + p(\boldsymbol{x}|\mathrm{C}_2)p(\mathrm{C}_2)} =$$

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

- where we have defined:

$$a \triangleq \ln\left(\frac{p(\boldsymbol{x}|\mathrm{C}_1)p(\mathrm{C}_1)}{p(\boldsymbol{x}|\mathrm{C}_2)p(\mathrm{C}_2)}\right)$$

Two Classes (2/2)

- Important property of logistic function: $\sigma(-a) = 1 - \sigma(a)$
- Inverse of logistic function:

$$a = \ln\left(\frac{\sigma}{1-\sigma}\right) = \ln\left(\frac{p(\mathrm{C}_1|\boldsymbol{x})}{p(\mathrm{C}_2|\boldsymbol{x})}\right)$$

- It it called the logit function
- Represents the log of the ratio of probs. for the two classes
- To see this:

$$a = \ln\left(\frac{p(\boldsymbol{x}|\mathrm{C}_1)p(\mathrm{C}_1)}{p(\boldsymbol{x}|\mathrm{C}_2)p(\mathrm{C}_2)}\right) = \ln\left(\frac{\sigma(a)}{\sigma(-a)}\right) = \ln\left(\frac{\sigma(a)}{1-\sigma(a)}\right)$$

Multiple Classes

- For the case of K>2 classes, we have:

$$p(\mathrm{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathrm{C}_k)p(\mathrm{C}_k)}{\sum_j p(\boldsymbol{x}|\mathrm{C}_j)p(\mathrm{C}_j)} = \\ = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

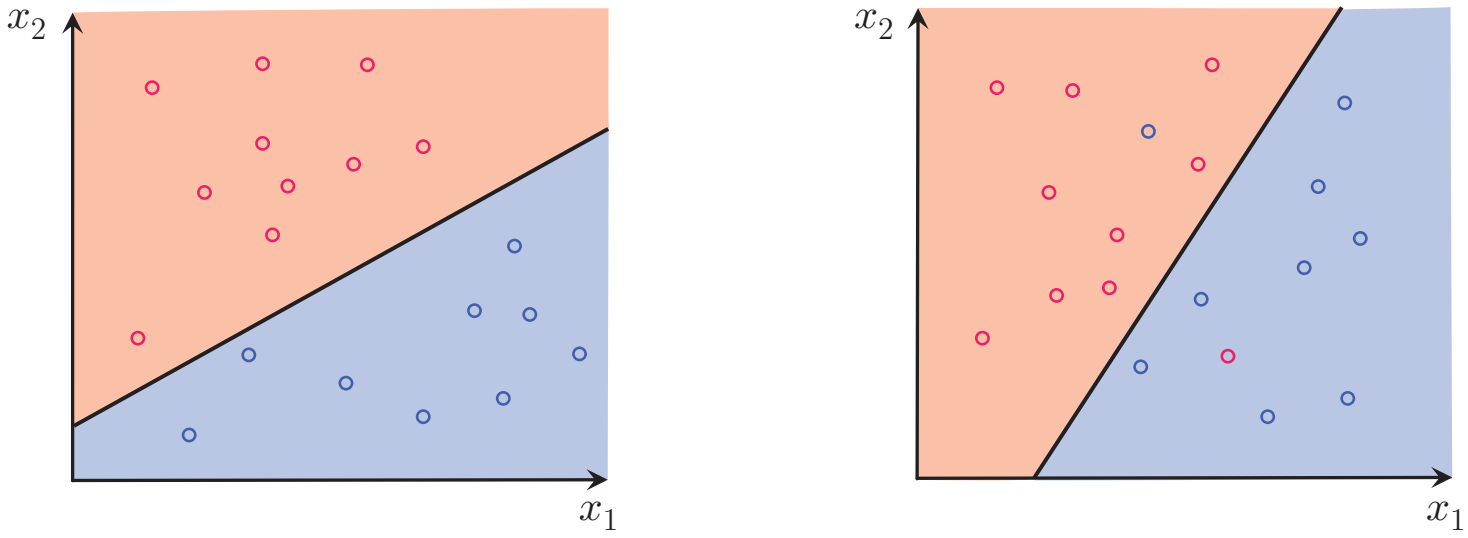
- which is known as the normalized exponential
- is a multiclass generalization of the logistic function, with:

$$a_k = \ln p(\boldsymbol{x}|\mathrm{C}_k)p(\mathrm{C}_k)$$

- It is also known as the softmax function
  - generalizes the max function as if: $a_k \gg a_j \text{ for all } j \neq k$

$$\Rightarrow p(\mathrm{C}_k|\boldsymbol{x}) = 1\,,\; p(\mathrm{C}_j|\boldsymbol{x}) = 0$$

From regression to classification



With linear classification we aim to learn a hyperplane $b + \mathbf{x}^T\mathbf{w} = 0$ (shown here in black) to separate feature representations of the two classes, colored red (class "+1") and blue (class "−1"), by dividing the feature space into a red half-space where $b + \mathbf{x}^T\mathbf{w} > 0$, and a blue half-space where $b + \mathbf{x}^T\mathbf{w} < 0$. (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, we can still find a hyperplane that minimizes the number of misclassified points that end up in the wrong half-space.

Classification *vs* regression

- Linear regression: aim is to represent (approximate) a dataset
- Classification
  - Aim is to separate two (or more) classes of the input/output data through a learned hyperplane. In other words, we want to learn a hyperplane of the form:

$$b + \boldsymbol{x}_p^T \boldsymbol{w} = 0$$

  - That separates the two classes of points as much as possible with
    - one class (C1) of points lying “above” the hyperplane with

$$b + \boldsymbol{x}_p^T \boldsymbol{w} > 0$$

    - another class (C2) of points lying “below” the hyperplane with:

$$b + \boldsymbol{x}_p^T \boldsymbol{w} < 0$$

Formally (1/2)

- We have a dataset $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$
  - where: $\boldsymbol{x}_i = [x_{1,i} \;\; x_{2,i} \;\; \ldots \;\; x_{D,i}]^T$
  - With each entry representing a feature (as with regression)

- For classification
  - The labels y_i are no longer continuous, but take values on a discrete set
  - They indicate the class membership of the corresponding point $\mathbf{x}_i$
  - For two classes C1 and C2

$$y_i = +1 \text{ if } b + \boldsymbol{x}_i^T \boldsymbol{w} > 0 \; (\boldsymbol{x}_i \text{ belongs to class C1})$$

$$y_i = -1 \text{ if } b + \boldsymbol{x}_i^T \boldsymbol{w} < 0 \; (\boldsymbol{x}_i \text{ belongs to class C2})$$

Formally (2/2)

- These relations

$$y_i = +1 \text{ if } b + \boldsymbol{x}_i^T \boldsymbol{w} > 0 \ (\boldsymbol{x}_i \text{ belongs to class C1})$$
$$y_i = -1 \text{ if } b + \boldsymbol{x}_i^T \boldsymbol{w} < 0 \ (\boldsymbol{x}_i \text{ belongs to class C2})$$

- Can be expressed compactly: by multiplying both expressions by their relative label value $-y_i$, leading to the equivalent expression for a correctly classified point $\mathbf{x}_i$:

$$-y_i(b + \boldsymbol{x}_i^T \boldsymbol{w}) < 0$$

- By taking the maximum between this expression and zero, a point (feature vector) $\mathbf{x}_i$ is classified correctly if

$$\max(0, -y_i(b + \boldsymbol{x}_i^T \boldsymbol{w})) = 0$$

- In fact, if a point is classified correctly, this max must give zero, and it is >0 only for a misclassified point

Cost function

- Since for a correctly classified point $\mathbf{x}_i$, we have:

$$\max(0, -y_i(b + \boldsymbol{x}_i^T \boldsymbol{w})) = 0$$

- For a misclassified point $\mathbf{x}_i$, we have:

$$\max(0, -y_i(b + \boldsymbol{x}_i^T \boldsymbol{w})) > 0$$

- We can define a cost function as:

$$g(b, \boldsymbol{w}) = \sum_{n=1}^{N} \max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w}))$$

  - which amounts to "weighing" (somewhat counting) the number of *misclassified points*, as only these points generate a value strictly greater than zero

Optimization problem

- The cost function is

$$g(b, \boldsymbol{w}) = \sum_{n=1}^{N} \max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w}))$$

- We can use it to formulate an optimization problem as:

$$\underset{(b,\boldsymbol{w})}{\text{minimize}} \sum_{n=1}^{N} \max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w}))$$

- Problem: this function is continuous, but *is not everywhere differentiable*, preventing the use of gradient and Newton's methods. Moreover has a trivial minimum at (b=0, **w=0**)

Solution

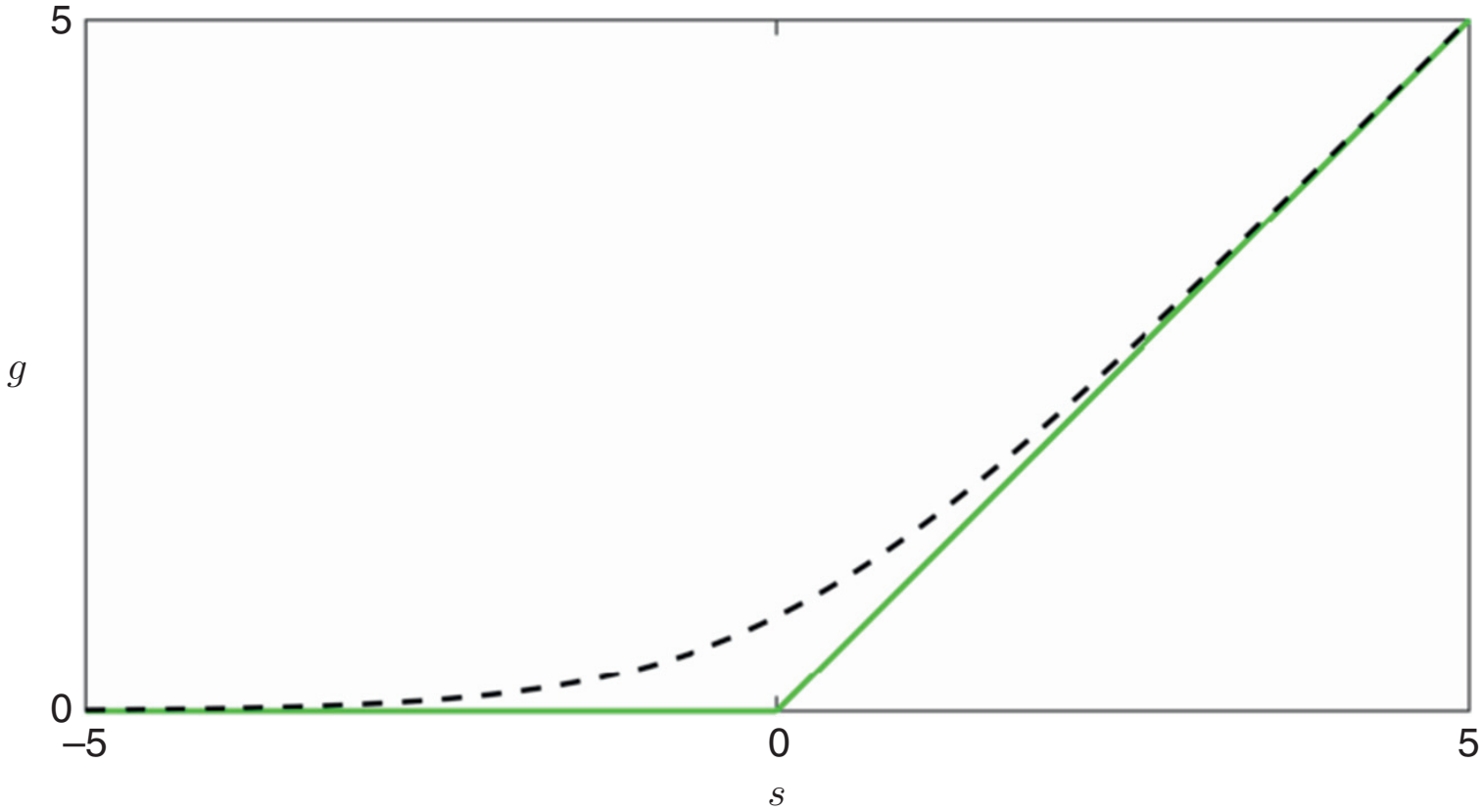
- Use a *smooth version* of the cost function

$$g(b, \boldsymbol{w}) = \sum_{n=1}^{N} \max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w}))$$

- In particular, we replace the non-differentiable max function
  - with the differentiable softmax function

$$\text{soft}(s_1, s_2) \stackrel{\text{def}}{=} \log(e^{s_1} + e^{s_2})$$

Softmax example



Plots of the non-differentiable perceptron or hinge cost $g(s) = \max(0, s)$ (shown in green) as well as its smooth softmax approximation $g(s) = \text{soft}(0, s) = \log\left(1 + e^{s}\right)$ (shown in dashed black).

$$\text{soft}(s_1, s_2) \overset{\text{def}}{=} \log(e^{s_1} + e^{s_2}) \approx \max(s_1, s_2)$$

Cost function revisited

- Using the *softmax* function we approximate the max as

$$\max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w})) \approx \text{soft}(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w})) = \\ = \log(1 + e^{-y_n(b + \boldsymbol{x}_n^T \boldsymbol{w})})$$

- The cost function can thus be rewritten as:

$$g(b, \boldsymbol{w}) = \sum_{n=1}^{N} \max(0, -y_n(b + \boldsymbol{x}_n^T \boldsymbol{w})) \\ \approx \sum_{n=1}^{N} \log(1 + e^{-y_n(b + \boldsymbol{x}_n^T \boldsymbol{w})})$$

Softmax function for K inputs

- The *softmax* function generalizes to K inputs as:

$$\max(s_1, s_2, \ldots, s_K) \approx \operatorname{soft}(s_1, s_2, \ldots, s_K) = \log\left(\sum_{k=1}^{K} e^{s_k}\right)$$

Sources

- Chapter 3 and Chapter 4 (Section 4.1) of [1]

[1] J. Watt, R. Borhani, A. K. Katsaggelos, “Machine learning refined: foundations, algorithms and applications,” Cambridge University Press, 2016.

- Section 4.2 of [2]

[2] C. Bishop, “Pattern Recognition and Machine Learning,” Springer Verlag, 2006.

A PRIMER ON REGRESSION AND CLASSIFICATION MODELS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT