Classification and Representation Learning

Course 3&4 : Linear Machine Learning

Hoel Le Capitaine

Academic year 2017-2018

Motivation

Machine learning is all about optimization

- different settings and objectives
  - supervised learning minimizes the error between the hypothesis and the data.
  - unsupervised learning maximizes the fit between the model and the data
  - reinforcement learning maximizes the obtention of rewards.
- the function to be optimized is called the objective function.
- ML searches values of free parameters which optimize the objective function on the data set.
- the simplest optimization method is the gradient descent (or ascent) method.

Gradient descent

Basic derivation

- let consider the following function $f(x) = x^2 + 2x + 1$
- this function has a minimum in

Gradient descent

Basic derivation

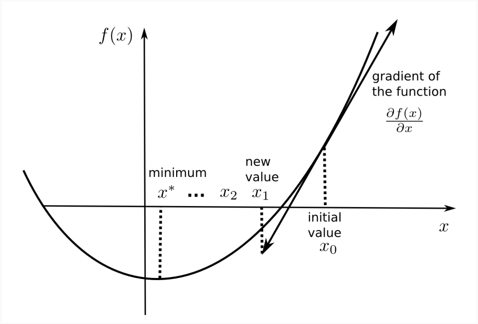
- let consider the following function $f(x) = x^2 + 2x + 1$
- this function has a minimum in $x = -1$, because $f'(x) = 2x + 2$, which is equal to 0 when $x = -1$
- optimum values of a function are obtained at points where the derivative function is 0
- to know if it is a maximum or minimum, one has to look at the second derivative $f''(x) = 2 > 0$
- if the second derivative is positive, the optimum is a minimum, otherwise a maximum.

Gradient descent

Principle

- it is not always possible to obtain the analytical form of the derivative
    - too complex to compute
    - we know only certain values of the function
    - numerical instability
- gradient descent is an iterative method allowing to minimize the objective function by iteratively modifying the estimation of the free parameter :
    - start with a wrong estimate of x: $x_0 = 0$
    - compute the derivative of the objective function in x_0: $f'(x_0) = 2$
    - compute a new value x_1 for the estimate using the gradient descent update rule: $x_1 = x_0 - \eta f'(x_0)$
    - η is called the learning rate (usually between 0 and 1).
    - iterate the update rule until we are close from the minimum -1.

Gradient descent



Sign of gradient

- when the gradient is positive, the function is increasing, so the minimum of the function is smaller than the estimate.
- when the gradient is negative, the function is decreasing, so the minimum of the function is higher.
- when the gradient is close to zero, we are around the minimum, we can stop.

Gradient descent

Multivariate functions

- gradient descent can be applied to multivarite functions

$$\min_{x,y,z} f(x,y,z)$$

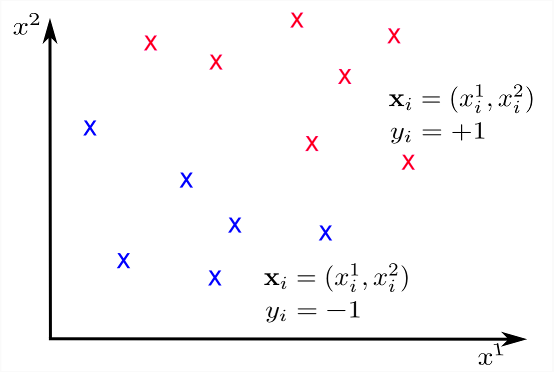
- each variable is updated independently using partial derivatives

$$\Delta x = x_{t+1} - x_t = -\eta \frac{\partial f(x,y,z)}{\partial x}$$

$$\Delta y = y_{t+1} - y_t = -\eta \frac{\partial f(x,y,z)}{\partial y}$$

$$\Delta z = z_{t+1} - z_t = -\eta \frac{\partial f(x,y,z)}{\partial z}$$

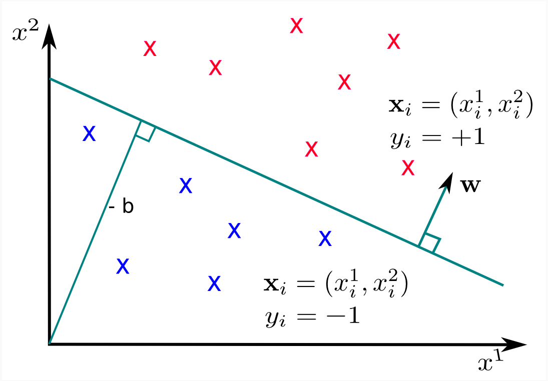
Linear classification



Binary linear classification

- the training data $\mathcal{D}$ contains N examples $(\mathbf{x}_i, y_i)_{i=1,\cdots,N}$, with a d-dimensional input vector $\mathbf{x}_i \in \mathbb{R}^d$, and a binary output $y_i \in \{-1, +1\}$
- the part of the data where $y = +1$ is called the positive class (red), the other is the negative one (blue).

Binary linear classification



x^2
x^1
- b
$\mathbf{w}$
$\mathbf{x}_i = (x_i^1, x_i^2)$
$y_i = +1$
$\mathbf{x}_i = (x_i^1, x_i^2)$
$y_i = -1$

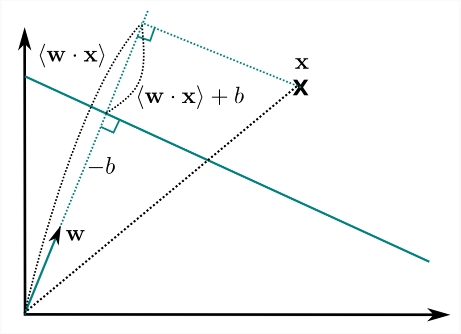
Looking for an hyperplane

- in the case data is linearly separable, we want to find the hyperplane $(\mathbf{w}, b)$ of $\mathbb{R}^d$ that correctly separates the two classes

$$\forall \mathbf{x}_i \in \mathcal{D},\ y_i = f(\mathbf{x}_i) = \text{sign}(h_{\mathbf{w},b}(\mathbf{x}_i)) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$$

- $\mathbf{w}$ is called the weight vector, and b is called the bias.

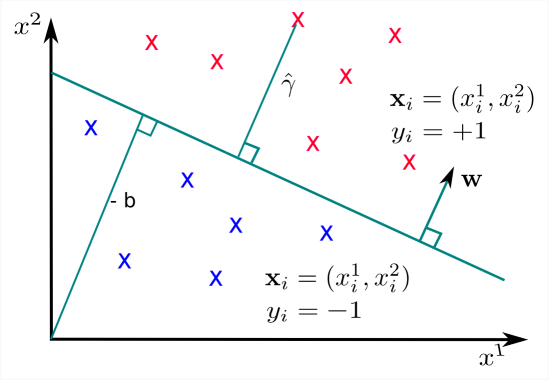
Binary linear classification



$\langle \mathbf{w} \cdot \mathbf{x} \rangle$
$\mathbf{x}$
$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b$
$-b$
$\mathbf{w}$

- for a point $\mathbf{x}_i \in \mathcal{D}$, $\mathbf{w} \cdot \mathbf{x}_i + b$ is the projection of $\mathbf{x}_i$ on the hyperplance $(\mathbf{w}, b)$.
- if $\mathbf{w} \cdot \mathbf{x}_i + b > 0$, the point is above the hyperplane, $y_i = +1$
- if $\mathbf{w} \cdot \mathbf{x}_i + b < 0$, the point is below the hyperplane, $y_i = -1$
- if $\mathbf{w} \cdot \mathbf{x}_i + b = 0$, the point is on the hyperplane, $y_i = 0$

Functional margin



x^2
$\hat{\gamma}$
$\mathbf{x}_i = (x_i^1, x_i^2)$
$y_i = +1$
$\mathbf{w}$
- b
$\mathbf{x}_i = (x_i^1, x_i^2)$
$y_i = -1$
x^1

- the relative distance of a point $\mathbf{x}_i$ to the hyperplane allows to compute its functional margin $\hat{\gamma}_i = y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$, that determines how well is the example classified by the hyperplane. Why?

Functional margin

$$\hat{\gamma}_i = y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

- if $\hat{\gamma}_i > 0$, the example is well classified (both y_i and $\mathbf{w} \cdot \mathbf{x}_i + b$ have the same sign)
- if $\hat{\gamma}_i < 0$, the example is badly classified
- if $\hat{\gamma}_i = 0$, the example is on the hyperplane
- the largest $|\hat{\gamma}_i|$, the further away from the hyperplane is the example
- a good linear classifier is an hyperplane where each example in the training set has a positive functional margin
- the functional margin $\hat{\gamma}$ of the training set with respect to the hyperplane is the minimum functional margin of all examples

Objective of linear classification

- linear classification is the process of finding an hyperplane $(\mathbf{w}, b)$ giving a positive functional margin for the training set.
- if such an hyperplane can be found, the training set is said linearly separable.
- otherwise, the problem is non-linearly separable and other methods have to be applied.
- problem: every vector colinear to a suitable $\mathbf{w}$ would also satisfy this condition.
- therefore we consider the geometric margin of the training set

$$\gamma = \min_i \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{\|\mathbf{w}\|}$$

Avoiding the bias

- taking systematically the bias into account is not necessary
- we look for an hypertplane $(\mathbf{w}, b)$ in $\mathbb{R}^d$ where

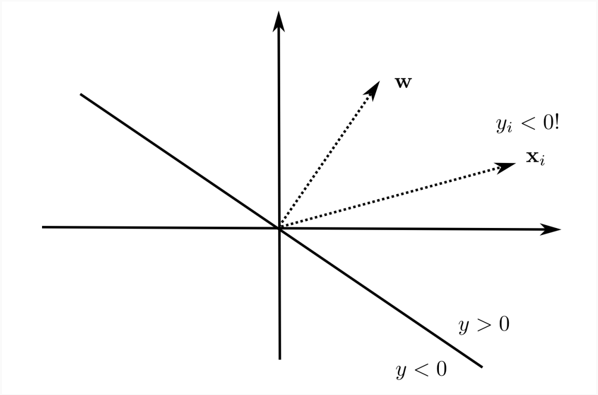
$$\forall (\mathbf{x}, y) \in \mathcal{D},\ y(\mathbf{w} \cdot \mathbf{x} + b) = y(w_1x_1 + w_2x_2 + \cdots + w_dx_d + b) > 0$$

- we can rewrite it as

$$\forall (\mathbf{x}, y) \in \mathcal{D},\ y(w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d) > 0$$

  with $w_0 = b$
- finally we have $\mathbf{w} = [b, w_1, \cdots, w_d]$ and $\mathbf{x} = [1, x_1, \cdots, x_d]$, and the problem stays the same, except that we work in a space of dimension $d + 1$

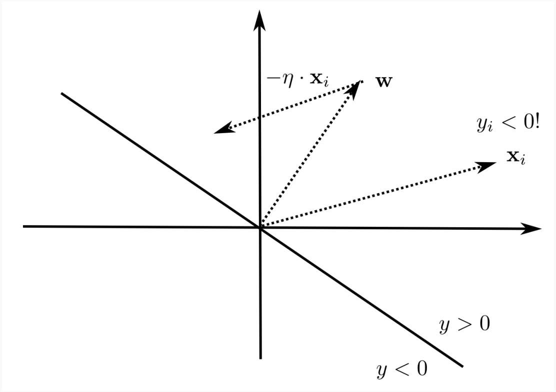
How to find such an hyperplane



Problem written, how to get the solution?

- let us suppose we have an initial vector $\mathbf{w}$, and a new example from the training set $\mathbf{x}_i$
- with the current hyperplane, $\mathbf{x}_i$ is badly classified ($y_i < 0$, but $\mathbf{w} \cdot \mathbf{x}_i$ so $\hat{\gamma}_i < 0$)

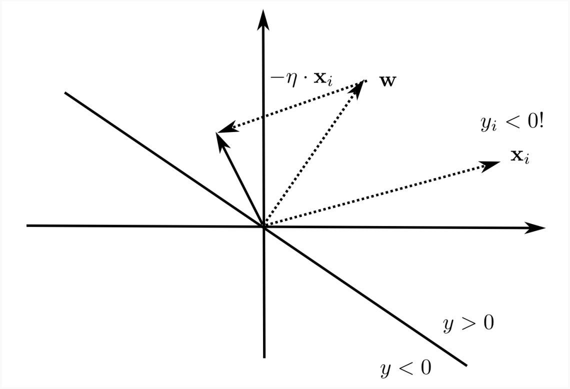
How to find such an hyperplane



Problem written, how to get the solution?

- as it is a mistake, the weight vector should move away from the actual position $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{x}_i$
- after this learning step, we have the new hyperplane

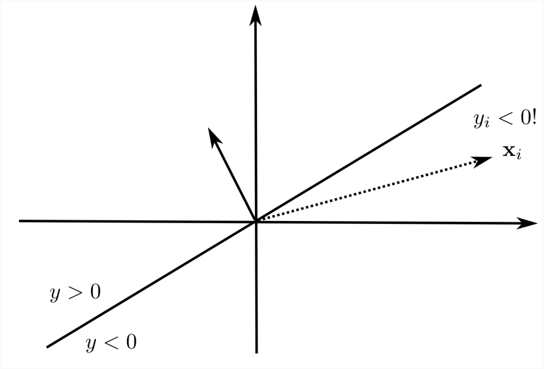
How to find such an hyperplane



Problem written, how to get the solution?

- as it is a mistake, the weight vector should move away from the actual position $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{x}_i$
- after this learning step, we have the new hyperplane

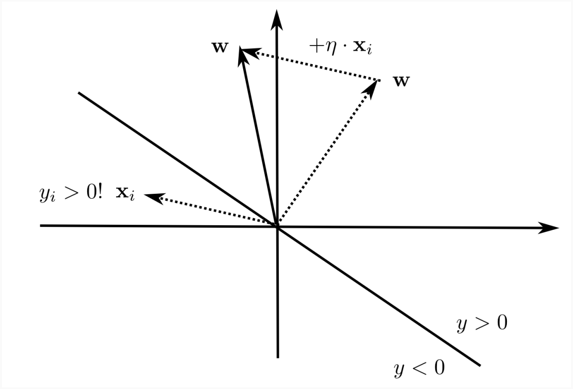
How to find such an hyperplane



Problem written, how to get the solution?

- with this new hyperplane, the example is correctly classified
- only valid if the error was made on a negative example: $y_i < 0$ and $\mathbf{w} \cdot \mathbf{x}_i > 0$

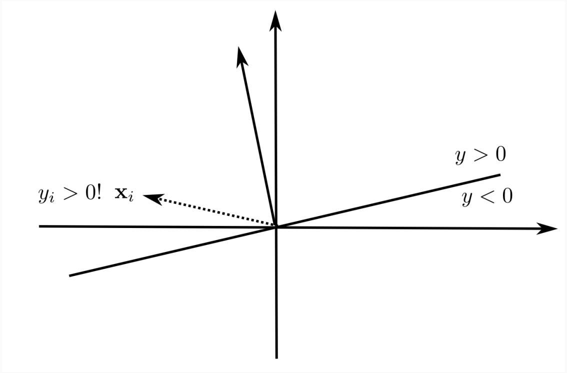
How to find such an hyperplane



Problem written, how to get the solution?

- on the opposite, if $y_i > 0$ and $\mathbf{w} \cdot \mathbf{x}_i < 0$, the learning rule should be reversed $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i$
- the weight vector is attracted by the misclassified example

How to find such an hyperplane



Problem written, how to get the solution?

- on the opposite, if $y_i > 0$ and $\mathbf{w} \cdot \mathbf{x}_i < 0$, the learning rule should be reversed $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i$
- the weight vector is attracted by the misclassified example

Perceptron algorithm

Principle

- when a positive example is misclassified $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i$
- when a negative example is misclassified $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{x}_i$
- when an example is well classified $\mathbf{w} \leftarrow \mathbf{w}$
- all of this can be summarized as

  $$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i,$$

  where $f_{\mathbf{w}}(\mathbf{x}_i)) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$
- this is called the Perceptron learning rule, or delta rule

Perceptron algorithm

The beginning of artificial intelligence and machine learning

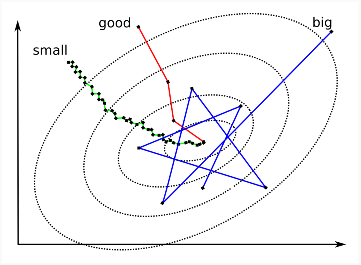
The Perceptron learning rule was invented by the psychologist Frank Rosenblatt in 1958. It was the first algorithmic neural network able to learn linear

Algorithm 1: Perceptron learning

```
while $\hat{\gamma} = \min_i y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 0$
    forall examples $(\mathbf{x}_i, y_i)$
        $f_{\mathbf{w}}(\mathbf{x}_i) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x}_i)$
        $\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$
```

- this algorithm iterates over all examples of the training set and applies the delta rule to each of them.
- if there are still mistakes on the training set ($\hat{\gamma} < 0$), the algorithm is applied again.
- the parameter is called the learning rate and regulates the speed of convergence.

On the learning rate



Choosing η is critical

- if it is too small, the algorithm will need a lot of iterations to converge.
- if it is too big, the algorithm can oscillate around the desired values without ever converging.

The perceptron algorithm is not optimal: it stops learning whenever there are no errors on the training set, but does not try to optimize the geometric margin.

Linear classification casted as an optimization problem

No gradient and optimization so far ...

- perceptron algorithm aims to find the weights minimizing the quadratic error function

$$E = \frac{1}{2}\sum_{i=1}^{N}(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

- when the prediction $f_{\mathbf{w}}(\mathbf{x}_i)$ is the same as y_i for all i, the quadratic error is minimal and equal to 0
- applying gradient descent,

$$\Delta\mathbf{w} = -\eta\frac{\partial E}{\partial \mathbf{w}}$$

Exercise 1: $\mathbf{w}$ update

Using gradient descent given above, find the update rule of $\mathbf{w}$

Batch perceptron algorithm

Algorithm 2: Batch Perceptron learning

```
while $\hat{\gamma} = \min_i y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 0$
    $\Delta\mathbf{w} = 0$
    forall examples $(\mathbf{x}_i, y_i)$
        $f_{\mathbf{w}}(\mathbf{x}_i) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$
        $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} + \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
```

Batch and online learning

Stability vs speed

- online learning (also called Stochastic Gradient Descent - SGD)

$$\Delta \mathbf{w} = \eta (y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \; \forall i$$

- batch learning

$$\Delta \mathbf{w} = \eta \sum_{i=1}^{N} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$$

- batch learning is mathematically correct and less sensible to noise in the data. But slow if data is simple.
- online learning converges faster, but can be instable. Allows adaptation during usage.
- a common trade-off is mini-batches: the training set is split into small chunks of data (usually 100 or 200 examples) and batch learning is applied on each. Good for stability and parallelism.

Linear regression

Continuous output

- in linear regression, the output is linearly dependent on the projection of the input onto the hyperplane

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

- with the same notation trick, we can get rid of the bias:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

- the goal of the linear regression (or least mean square - LMS) is to minimize the quadratic error:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

Analytical solution (closed form)

Exercise 2: Closed form solution

In this exercise, we consider only one variable x, such that we try to explain y by $ax + b$. The sum of square errors is defined by $L = \frac{1}{2}\sum_{i=1}^{N}(y_i - ax_i - b)^2$.

- write the necessary conditions w.r.t. a and b for L to be minimum, and write it as a system of linear equations
- Give the analytical value of a and b.

Multivariate closed form

Univariate case

Computationally light for univariate problem (only correlation, standard deviation and mean need to be computed)

Multivariate case : A simple and elegant formula

Objective is to minimize (see previous lecture)

$$||X\mathbf{w} - \mathbf{y}||^2$$

- setting its derivative $2X^T(X\mathbf{w} - \mathbf{y})$ to 0 gives

$$X^T X\mathbf{w} = X^T\mathbf{y}$$

- i.e. $\mathbf{w} = (X^T X)^{-1} X^T\mathbf{y}$

From analytical to gradient based solutions

but ...

- computational complexity (size of X) of inversion
- numerical stability
- a gradient descent version saves a lot of time (number of iterations, parallelism) and memory (minibatch loaded)

Gradient descent for linear regression

How to get w ?

- gradient descent leads to an update rule similar to the Perceptron rule:

$$\begin{aligned}
\Delta \mathbf{w} &= -\eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \\
&= -\eta \frac{\partial}{\partial \mathbf{w}} \frac{1}{2} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 \\
&= -\eta \frac{1}{2} \sum_{i=1}^{N} \frac{\partial}{\partial \mathbf{w}} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 \\
&= -\eta \frac{1}{2} \sum_{i=1}^{N} 2(y_i - \mathbf{w} \cdot \mathbf{x}_i) \frac{\partial}{\partial \mathbf{w}} (y_i - \mathbf{w} \cdot \mathbf{x}_i) \\
&= -\eta \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)(-\mathbf{x}_i) \\
&= \eta \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)\mathbf{x}_i
\end{aligned}$$

Gradient descent for the least mean-square method of linear regression

Batch and online version

- online

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \ \forall i$$

- batch

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_i (y_i - f_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$$

- there is no functional margin to compute: there will always be a remaining error
- the convergence criterium has to be defined by the user
  - lower limit on the variation of weights
  - threshold on the improvement of the quadratic error
  - validation set

Comparison of linear classfication and regression

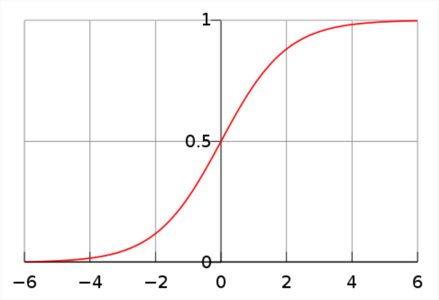
Discrete/continous

- classification and regression differ in the nature of their outputs: in classification they are discrete, in regression they are floating values.
- however, when trying to minimize the error between a model $f_{\mathbf{w}}(\mathbf{x})$ and the real data y, we have found the same learning rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i)) \cdot \mathbf{x}_i$$

- regression and classification are therefore the same problem. Methods which apply to one can almost automatically be applied to the other problem.
- logistic regression is an example of such a bridge between the two problems.

Logistic regression



Non linearity

- We can use a logistic function instead of a linear function (sign) in order to transform the projection on the hyperplane into an output:

$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x})}$$

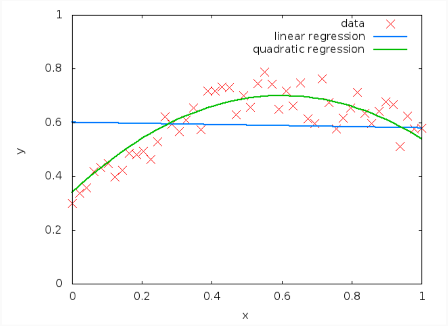
- the logistic function g has the following interesting property $g'(x) = g(x)g(1-x)$

Logistic regression

Exercise 3: Logistic regression learning rule

Given the following quadratic error $L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - g(\mathbf{w}\cdot\mathbf{x}_i))^2$, where g is the logistic function, give the update rule of $\mathbf{w}$ using gradient descent and the perceptron algorithm.

Polynomial regression



- the functions underlying real data are rarely linear plus some noise around the ideal value.
- in the figure, the input/output function is better modelled by a second-order polynome:

$$y = f_{\mathrm{w}}(x) = w_1 x + w_2 x^2 + b$$

Polynomial regression

$$y = f_{\mathbf{w}}(x) = w_1 x + w_2 x^2 + b$$

- we can transform the input into a vector of coordinates

$$\mathbf{x} = [1, x, x^2], \quad \mathbf{w} = [b, w_1, w_2]$$

- the problem becomes $y = \mathbf{w} \cdot \mathbf{x}$
- we can simply apply linear regression methods to find $\mathbf{w}$
- Problem solved! The only unknown is which order for the polynome matches best the data.
- One can perform regression with any kind of parameterized function using gradient descent.

Use of logistic regression for classification

Same problem

- logistic regression can be used for classification
- applies a non-linear function on the scalar product

$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x})}$$

- the continuous output $f_{\mathbf{w}}(\mathbf{x})$ is interpreted in terms of the probability of belonging to one class or another

$$P(y = 1|\mathbf{x}, \mathbf{w}) = f_{\mathbf{w}}(\mathbf{x}) \qquad P(y = 0|\mathbf{x}, \mathbf{w}) = 1 - f_{\mathbf{w}}(\mathbf{x})$$

- the update rule is similar to the perceptron rule, using the class as target and the probability as a prediction

$$\Delta \mathbf{w} = \eta(y_i - f_{\mathbf{w}}(\mathbf{x}_i) \cdot \mathbf{x}_i$$

- logistic regression also provides a confidence estimation: the closer $g(\mathbf{w} \cdot \mathbf{x}_i)$ is from 0 or 1, the more confident we are that the classfication is correct.