Machine Learning

Lecture 2: Linear/Polynomial/Logistic Regression

Marc Sebban & Amaury Habrard

Hubert Curien lab, UMR CNRS 5516
University of Jean Monnet Saint-Étienne (France)

Academic year 2016-2017

The Regression Problem

How to optimize the parameters of the hyperplane

$$h_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_0$$

which fits the best the following set of training examples $S = \{(\mathbf{x}, y)\}_{i=1}^{5}$?

x_1: Living area ($feet^2$)	x_2: # bedrooms	x_3: Garden ($feet^2$)	y: House Price (1000$)
2104	3	10050	400
1416	2	7534	232
1534	3	4305	315
852	2	2152	178
1990	4	9850	240

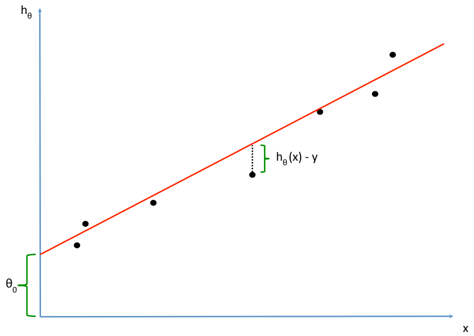
Outline

Notations

- m : # of training examples
- $x \in \mathbb{R}^n$: input feature vector of n variables.
- y : output variable/target
- (x, y): training example
- $(x^{(i)}, y^{(i)})$: i^{th} training example.
- h : hypothesis that maps from input x to output y.
- $h(x)$ is of the linear form: $h(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$. For conciseness, let us define $x_0 = 1$ s.t.

$$h(x) = h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

where $\theta = (\theta_0, ..., \theta_n) \in \mathbb{R}^{n+1}$.



Goal

How to choose the parameters θ so that the hypothesis h will make accurate predictions?

Non regularized Least Squares Problem

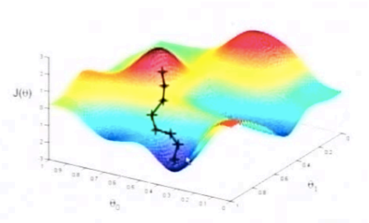
$$\min_\theta J(\theta) = \min_\theta \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

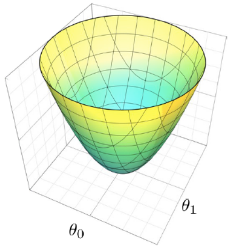
Learning algorithms for minimizing $J(\theta)$

How to minimize $J(\theta)$?

There exist 3 main solutions for minimizing $J(\theta)$:

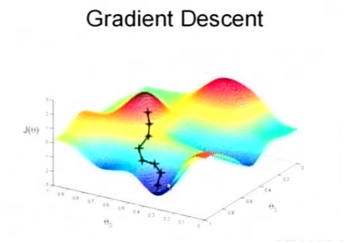
- Batch Gradient Descent
- Stochastic Gradient Descent
- Closed-form solution





Batch Gradient Descent

Gradient Descent



Basic Idea

Let us assume that $J(\theta)$ is differentiable:

- Start with some **initialization** of θ (e.g. $\vec{\theta} = 0$ or some randomly chosen vector.)
- **Update** θ's values so that to reduce $J(\theta)$ (by computing partial derivatives of $J(\theta)$ w.r.t. θ).
- **Repeat** the process till convergence to the minimum of $J(\theta)$.

Gradient Descent

Update rule

Gradient descent is based on the observation that if $J(\theta)$ is **differentiable** in a neighborhood of x, then $J(\theta)$ decreases fastest if one goes from x in the direction of the negative gradient of $J(\theta)$. Therefore, gradient descent updates each parameter θ_i as follows:

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$$

where:

- $\frac{\partial}{\partial \theta_i} J(\theta)$ gives us the direction of the **deepest descent**.
- α is the **learning rate** which controls how large a step you take in the direction of the steepest descent.

Gradient Descent

Using linear algebra notations

Let $\nabla_\theta J = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{n+1}$ be the gradient of J w.r.t. θ.

We can rewrite the update rule as follows:

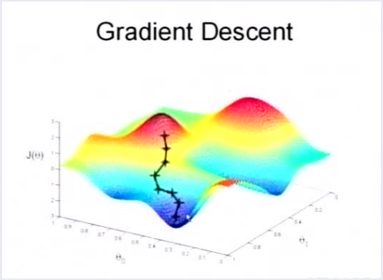
$$\boxed{\theta := \theta - \alpha \nabla_\theta J}$$

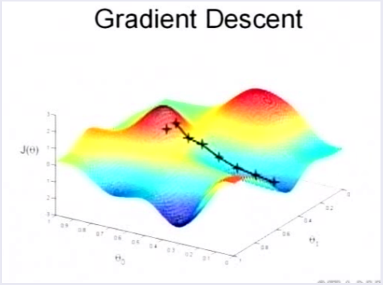
where both θ and $\nabla_\theta J$ are $(n+1)$-dimensional feature vectors.

Batch Gradient Descent Algorithm

Impact of the initialization

Note that with a slightly different initial starting point, you can actually end up at a completely different local optimum.





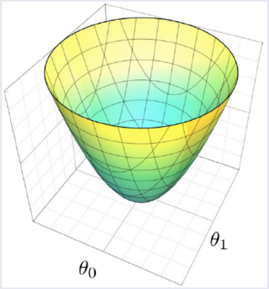
However, the function $J(\theta)$ actually does not look like this nasty one.

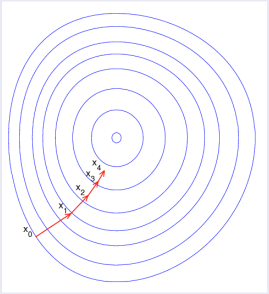
Batch Gradient Descent Algorithm

Convex Problem

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

It turns out that $J(\theta)$ is a **quadratic function** with only one **(global) optimum**.





Update of the i^{th} parameter of θ

Assume we have only one training example x (i.e. $m = 1$):

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2}\left(h_\theta(x) - y\right)^2$$

$$\begin{aligned}
\frac{\partial}{\partial\theta_i}J(\theta) &= \frac{\partial}{\partial\theta_i}\frac{1}{2}(h_\theta(x) - y)^2 \\
&= 2 \times \frac{1}{2}(h_\theta(x) - y) \times \frac{\partial}{\partial\theta_i}(h_\theta(x) - y) \\
&= (h_\theta(x) - y) \times \frac{\partial}{\partial\theta_i}(\theta_0 x_0 + \ldots + \theta_i x_i + \ldots + \theta_n x_n - y) \\
&= (h_\theta(x) - y)x_i
\end{aligned}$$

Therefore, we get the following update rule:

$$\theta_i := \theta_i - \alpha(h_\theta(x) - y)x_i$$

Batch Gradient Descent Algorithm

More generally, with m training examples, we get:

Batch Gradient Descent Algorithm

```
Initilization of $\vec{\theta}$
Repeat
{
    $$\forall \theta_i \text{ of } \theta \quad \theta_i := \theta_i - \alpha \frac{1}{m} \sum_{j=1}^{m} \left( h_\theta(x^{(j)}) - y^{(j)} \right) x_i^{(j)}$$
}
until convergence ($\approx$ ``stabilization'') of $J(\theta)$
```

Note that we use “Batch” because at each gradient descent, we are going to look at the entire training set and performing a sum over the m examples.

Stochastic Gradient Descent

From Batch to Stochastic Gradient Descent

Remark

If m is huge - say millions of examples - then if you are running batch gradient descent, you have to perform at each step a sum over one million of examples. Therefore, we need an alternative algorithm.

Stochastic (or incremental) Gradient Descent

```
Initialization of $\vec{\theta}$
Repeat
{
    For $j = 1$ to $m$
        $\forall \theta_i$ of $\theta$,   $\theta_i := \theta_i - \alpha \left( h_\theta(x^{(j)}) - y^{(j)} \right) x_i^{(j)}$
}
until convergence of $J(\theta)$
```

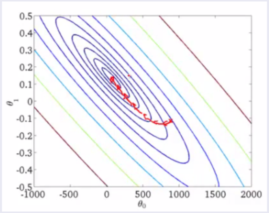
Stochastic Gradient Descent

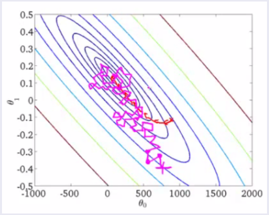
Advantages of the Stochastic Gradient Descent

- The update of the parameters θ's starts with the first training example. A second update is achieved with the second one...
- Much faster for large datasets than the batch gradient descent.

Disadvantages of the Stochastic Gradient Descent

- It won't converge to the global minimum exactly but...



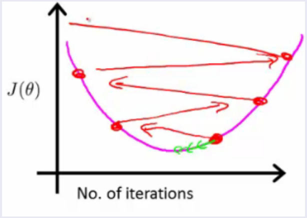


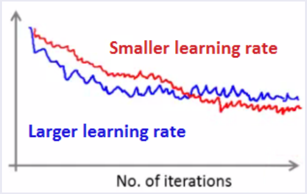
...it tends to wander around some regions close to the global minimum.

A few words about the learning rate α

Large versus small learning rate α

For some specific examples, $J(\theta)$ may increase. This may occur in such following situations where α is large ("zigzag" effect).





To prevent the parameters θ's from oscillating around the global minimum we can take a smaller learning rate.

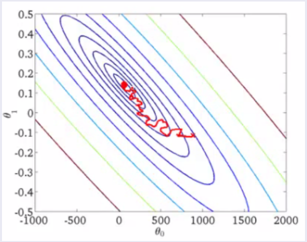
A few words about the learning rate α

Large versus small learning rate α

In most implementations, the learning rate is held constant. However, if you want to converge to "a minimum" you can slowly decrease α over time, such that at iteration i we get:

$$\alpha_i = \frac{C_1}{i + C_2},$$

which means you're guaranteed to converge "somewhere".



Example

Let x be the number of hours of training the day before an exam and y the grade obtained by the students. What is the equation of the linear regression $h_\theta(x) = \theta_0 + \theta_1 x$ after the first three iterations of the stochastic gradient descent algorithm (with $\alpha = 0.1$)?

x : Nb of hours	y : Grade $\in [0, 10]$
5	9
3	6
0	1

Solution

- **Step 1**: with $x^{(1)} = (1,5)$ and $y^{(1)} = 9$
  - Initialization: $\theta_0 = 0$, $\theta_1 = 0$ and $h_\theta(x^{(1)}) = 0$
  - Update of θ_0: $\theta_0 = 0 - 0.1(0-9) \times 1 = \mathbf{0.9}$.
  - Update of θ_1: $\theta_1 = 0 - 0.1(0-9) \times 5 = \mathbf{4.5}$.
  - Update of $h_\theta(x)$: $\mathbf{h_\theta(x) = 0.9 + 4.5x}$.
- **Step 2**: with $x^{(2)} = (1,3)$ and $y^{(2)} = 6$ given $h_\theta(x^{(2)}) = 0.9 + 4.5 \times 3 = 14.4$
  - Update of θ_0: $\theta_0 = 0.9 - 0.1(14.4-6) \times 1 = \mathbf{0.06}$.
  - Update of θ_1: $\theta_1 = 4.5 - 0.1(14.4-6) \times 3 = \mathbf{1.98}$.
  - Update of $h_\theta(x)$: $\mathbf{h_\theta(x) = 0.06 + 1.98x}$.
- **Step 3**: with $x^{(3)} = (1,0)$ and $y^{(3)} = 1$ given $h_\theta(x^{(3)}) = 0.06 + 1.98 \times 0 = 0.06$
  - Update of θ_0: $\theta_0 = 0.06 - 0.1(0.06-1) \times 1 = \mathbf{0.154}$.
  - Update of θ_1: $\theta_1 = 1.98 - 0.1(0.06-1) \times 0 = \mathbf{1.98}$.
  - Update of $h_\theta(x)$: $\mathbf{h_\theta(x) = 0.154 + 1.98x}$.

X	y	$h_\theta(x)$
5	9	10.05
3	6	6,09
0	1	0.154

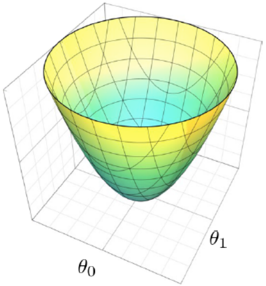
Mini-batch gradient descent

Mini-batch gradient descent

A compromise between a batch gradient descent and a stochastic gradient descent, is to compute the gradient only on a (randomly selected) bunch of training examples (called a "**mini-batch**") at each step.

It may result in smoother convergence, as the gradient computed at each step uses more training examples than in the stochastic setting.

Closed-Form Solution



Notations

- Let $\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{n+1}$ be the gradient of J w.r.t. θ.
- More generally, if you have a fonction $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & & \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \text{ where } A \in \mathbb{R}^{m \times n}.$$

Closed-form Solution

To solve the linear regression problem in closed form, we are going to make use of the following 7 properties on the **trace** of a $n \times n$ square matrix A.

The **trace** of an $n \times n$ square matrix A is defined to be the sum of the elements on the main diagonal: $tr(A) = a_{11} + \ldots + a_{nn} = \sum_{i=1}^{n} a_{ii}$

Properties of the trace

1. $tr(A + B) = trA + trB$
2. $trAB = trBA$ (while in general, $AB \neq BA$)
3. $trABC = trCAB = trBCA$
4. Let $f(A) = trAB$, then $\nabla_A f(A) = B^T$
5. if $X, Y \in \mathbb{R}^n$ $X^T Y = Y^T X = a$ where $a \in \mathbb{R}$
6. if $a \in \mathbb{R}$, $tr(a) = a$
7. $\nabla_A trABA^T C = CAB + C^T AB^T$

Let us remind that $h_\theta(x^{(j)}) = \sum_{i=0}^{n} \theta_i x_i^{(j)} = \theta^T x^{(j)} = (x^{(j)})^T \theta$, where $(x^{(j)})^T = (1, x_1^{(j)}, ..., x_n^{(j)})$ and $\theta^T = (\theta_0, ..., \theta_n)$ are two vectors in $\mathbb{R}^n$.

Rewriting of J with matrices and vectors

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- $X\theta = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \theta = \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(1)}) \\ \vdots \\ h_\theta(x^{(m)}) \end{bmatrix}$
- $y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$
- We deduce that: $X\theta - y = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix} \in \mathbb{R}^m$

Closed-form Solution

Rewriting of $J(\theta)$

Recall that if $z \in \mathbb{R}^m$ then $z^T z = \sum_{i=1}^m z_i^2$. Applying this property on $X\theta - y \in \mathbb{R}^m$ we get:

$$\frac{1}{2}(X\theta - y)^T(X\theta - y) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 = J(\theta)$$

Minimizing $J(\theta)$ w.r.t. θ boils down to solving:

$$\nabla_\theta J(\theta) \overset{set}{=} \vec{0}$$

Therefore,

$$\nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y) \overset{set}{=} \vec{0}$$

Closed-form Solution

Closed-form Solution

$$\nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

expanding the quadratic function, we get

$$= \frac{1}{2}\nabla_\theta(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y)$$

using property (6) ($tr(a) = a$ if $a \in \mathbb{R}$), we get

$$= \frac{1}{2}\nabla_\theta tr(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y)$$

using property (1) ($tr(A + B) = trA + trB$), we get

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - \nabla_\theta tr(\theta^T X^T y) - \nabla_\theta tr(y^T X\theta) - \nabla_\theta tr(y^T y)]$$

The last term does not depend on θ. Therefore, we get:

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - \nabla_\theta tr(\theta^T X^T y) - \nabla_\theta tr(y^T X\theta)]$$

Closed-form Solution

Closed-form Solution

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - \nabla_\theta tr(\theta^T X^T y) - \nabla_\theta tr(y^T X\theta)]$$

On the 2nd term we can apply (5) ($X^T Y = Y^T X$)

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - \nabla_\theta tr(y^T X\theta) - \nabla_\theta tr(y^T X\theta)]$$

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - 2\nabla_\theta tr(y^T X\theta)]$$

Closed-form Solution

Closed-form Solution

$$= \frac{1}{2}[\nabla_\theta tr(\theta^T X^T X\theta) - 2\nabla_\theta tr(y^T X\theta)]$$

Using property (3) ($trABC = trCAB$) and inserting I, we get:

$$= \frac{1}{2}[\nabla_\theta tr(\theta I\theta^T X^T X) - 2\nabla_\theta tr(y^T X\theta)]$$

As $\theta I\theta^T X^T X$ is of the form $ABA^T C$, we can use property (7) ($\nabla_A trABA^T C = CAB + C^T AB^T$)

$$= \frac{1}{2}[X^T X\theta I + X^T X\theta I - 2\nabla_\theta tr(y^T X\theta)]$$

$$= X^T X\theta - \nabla_\theta tr(y^T X\theta)$$

Closed-form Solution

Closed-form Solution

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y) = X^TX\theta - \nabla_\theta tr(y^TX\theta)$$

Using property (2) ($trAB = trBA$), we get:

$$= X^TX\theta - \nabla_\theta tr(\theta y^TX)$$

Applying (4) ($\nabla_A trAB = B^T$), we get:

$$= X^TX\theta - X^Ty$$

Closed-form Solution

Closed-form Solution

Therefore, solving $\nabla_\theta J(\theta) \overset{set}{=} \vec{0}$ boils down to solving:

$$\begin{aligned} X^T X\theta - X^T y &= 0 \\ X^T X\theta &= X^T y \qquad \textbf{(Normal Equation)} \\ \text{we get:} & \\ \theta &= \boxed{(X^T X)^{-1} X^T y} \qquad \textbf{(closed-form solution)} \end{aligned}$$

Exercise

Exercise

Let x be the number of hours of training the day before an exam and y the grade obtained by the students. Find the equation of the linear regression $h_\theta(x) = \theta_0 + \theta_1 x$ using the closed form solution.

x : Nb of hours	y : Grade $\in [0, 10]$
5	9
3	6
0	1

Inverse of a 2×2 matrix

Let $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. If $det(A) \neq 0$, then the inverse of A is

$$A^{-1} = \frac{1}{det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

What if $(X^TX)^{-1}$ is non-invertible?

Context where $(X^TX)^{-1}$ is non-invertible ($n \times n$ matrix)

- Redundant features (linearly dependent)
  $\rightarrow$ Solution: **perform a PCA**.
- Too many features
  $\rightarrow$ Solution: **delete some irrelevant features** (feature selection algorithms).

Gradient descent versus Normal Equation

Gradient descent	**Normal Equation**
Need to choose α	No need to choose α
Needs many iterations	Don't need to iterate
Works well even when n is large ($n \sim 10^6$)	Need to compute $(X^T X)^{-1}$
Allows an online acquisition of training data	Slow if n is large

Probabilistic interpretation of Linear Regression

Objective function in linear regression

Objective function

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Why least square? Why not minimize the absolute value of the errors?

Error term

Until now, we assumed that $y^{(i)}$ could be approximated by $\hat{y}^{(i)} = \theta^T x^{(i)}$

$$y^{(i)} \approx \theta^T x^{(i)} \qquad (1)$$

Equation (1) can be rewritten as follows:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \qquad (2)$$

where $\epsilon^{(i)}$ is an **error term** which captures unmodelled effects, like the absence of relevant features, random noise, etc. Assume that $\epsilon^{(i)} \sim N(0, \sigma)$ (this can be justified by the **Central Limit Theorem**).

$$P(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

From (2), we get $\epsilon^{(i)} = y^{(i)} - \theta^T x^{(i)}$ and therefore,

$$P(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

Likelihood

Let $x_1, x_2, ..., x_m$ be a random set of m i.i.d. observations, coming from an unknown density function $f(x|\theta)$ where θ is a parameter. For an i.i.d. sample, the **likelihood** is:

$$L(\theta) = f(x_1, ..., x_m|\theta) = f(x_1|\theta) \times f(x_2|\theta) \times \cdots \times f(x_m|\theta) = \prod_i f(x_i|\theta).$$

Remarks

1. For the sake of simplicity, it may be useful to make use of the log likelihood $\ell(\theta) = lnL(\theta)$.
2. Many common probability distributions are log-concave.

Maximum Likelihood Estimation

It is desirable to find an estimate $\hat{\theta}$ that makes the data as probable as possible. Therefore, we get an estimate of θ by solving

$$\frac{\partial \ell(\theta)}{\partial \theta} = 0|_{\theta=\hat{\theta}}$$

Maximum Likelihood Estimation

Example: ML estimation of parameter p of a binomial distribution

Let $x_1, x_2, ..., x_m$ be a random set of m i.i.d. observations (where $\forall i, x_i \in \{0, 1\}$), coming from a bernouilli distribution of parameter p. Compute the estimate of p by the Maximum Likelihood method.

$$\begin{aligned} f(x_1, ..., x_m|p) &= f(x_1|p) \times f(x_2|p) \times \cdots \times f(x_m|p) \\ &= p^{x_1}(1-p)^{1-x_1} \times \ldots \times p^{x_m}(1-p)^{1-x_m} \\ &= p^{\sum_i x_i}(1-p)^{m-\sum_i x_i} \\ &= p^X(1-p)^{m-X}, \end{aligned}$$

where $X = \sum_i x_i$ is a binomial variable.

Example: ML estimation of parameter p of a binomial distribution (ctd)

$$
\begin{aligned}
\frac{\partial ln f(x_1, ..., x_m|\theta)}{\partial \theta} &= 0|_{\theta=\hat{\theta}} \\
\Leftrightarrow \frac{\partial X ln\hat{p} + (m - X) ln(1 - \hat{p})}{\partial \theta} &= 0|_{\theta=\hat{\theta}} \\
\Leftrightarrow \frac{X}{\hat{p}} &= \frac{m - X}{1 - \hat{p}} \\
\Leftrightarrow \hat{p} &= \frac{X}{m}
\end{aligned}
$$

Likelihood in Regression

$$L(\theta) = P(\vec{y}|x;\theta) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$\begin{aligned}
\ell(\theta) &= lnL(\theta) \\
&= ln\prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} ln\left(\frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\right) \\
&= \sum_{i=1}^{m} ln(\frac{1}{\sqrt{2\pi}\sigma}) + \sum_{i=1}^{m} ln\left(exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\right) \\
&= m.ln\frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} -\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}
\end{aligned}$$

It turns out that...

$$\begin{aligned}\ell(\theta) &= lnL(\theta)\\ &= m.ln\frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} -\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\end{aligned}$$

So, maximize $\ell(\theta)$ is the same as minimizing

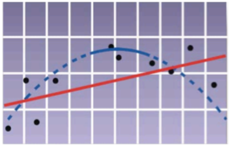
$$\sum_{i=1}^{m} \frac{(\theta^T x^{(i)} - (y^{(i)})^2}{2} = J(\theta)$$

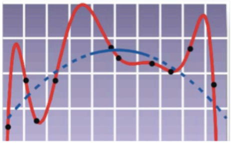
which the original objective function of linear regression.

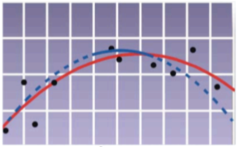
Conclusion

The ordinary least square algorithm that we worked on is just maximum likelihood assuming i.i.d gaussian errors on the data.

How to learn non linear models? Polynomial regression


capacity too low
⇨under-fitting


capacity too high
⇨over-fitting


optimal capacity
⇨good generalisation

Polynomial regression

In polynomial regression, the relationship between x and y is modelled as an n^{th} degree polynomial that can be generalized to any set of monomials.

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n.$$

How to learn non linear models? Polynomial regression

Polynomial regression

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_n x^n.$$

Note that the regression function is still linear in terms of the unknown parameters $\theta_0, ..., \theta_n$.

Therefore, we can address the problem by using a standard multiple regression model where $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$, etc. are distinct variables.

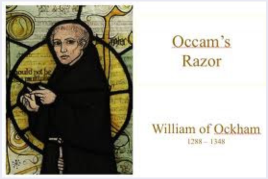
$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n.$$

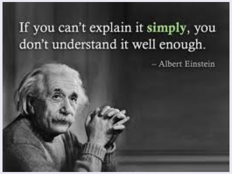
$$\theta = \boxed{(X^T X)^{-1} X^T y} \qquad \textbf{(closed-form solution)}$$

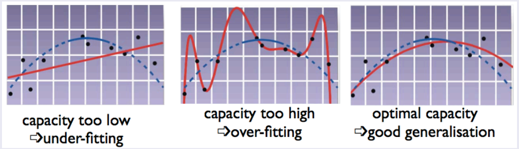
How to prevent overfitting in Linear Regression?

Occam's razor principle

"Choose the simplest explanation consistent with data"...




If you can't explain it simply, you don't understand it well enough.
– Albert Einstein

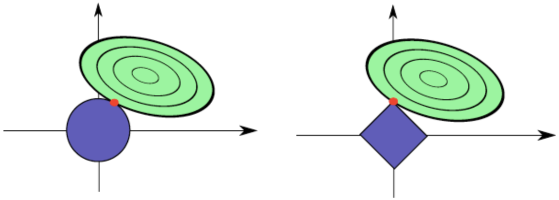


Solution: Use of **regularization** to control the complexity of the model.

Methods to prevent overfitting in Linear Regression

Regularized versions of Linear Regression

1. Ridge Regression (using the L_2-norm).
2. LASSO (using the L_1 norm).



Ridge Regression

Ridge regression is based on the L_2 norm and penalizes the size of the regression coefficients. The optimization problem is the following:

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left({x^{(i)}}^T \theta - y^{(i)} \right)^2 + \lambda ||\theta||_2^2$$

We can show that the closed-form solution is given by:

$$\theta = \boxed{(X^T X + \lambda I)^{-1} X^T y} \qquad \textbf{(closed-form solution)}$$

Note the similarity to the ordinary least squares solution, but with the addition of a "ridge" down the diagonal

Ridge Regression

Uniform Stability of the Ridge Regression

The ridge regression comes with a generalization bound on the true risk.

$$\mathcal{R}_{h_\theta} \leq \hat{\mathcal{R}}_{h_\theta} + \frac{4B^2}{\lambda m} + \left(\frac{8B^2}{\lambda} + 2B\right)\sqrt{\frac{ln 1/\delta}{2m}},$$

where we consider the bounded case $\mathcal{Y} = [0, B]$.

LASSO (Tibshirani 1996)

LASSO (for *Least Absolute Shrinkage and Selection Operator*) makes use of the L_1 norm to constrain the algorithm to remove irrelevant features. It **bounds the sum of the absolute values of the coefficients**.
The optimization problem is defined as follows:

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left({x^{(i)}}^T \theta - y^{(i)} \right)^2 + \lambda ||\theta||_1$$

It turns out that increasing the least square penalty will cause more and more of the parameters to be driven to zero.

Closed-form solution of LASSO

Because the LASSO penalty has the absolute value operation in it, the objective function is not differentiable and as a result, lacks a closed form in general.

However, in the special case of an orthonormal matrix ($X^T X = I$), it is possible to obtain closed form solutions for the LASSO:

$$\theta_{LASSO} = S(\theta, \lambda)$$

where

$$S(\theta, \lambda) = \begin{cases} \theta - \lambda \; if \; \theta > \lambda \\ 0 \; if \; |\theta| \leq \lambda \\ \theta + \lambda \; if \; \theta < -\lambda \end{cases}$$

Limitations of Linear Regression

From Linear to Logistic Regression

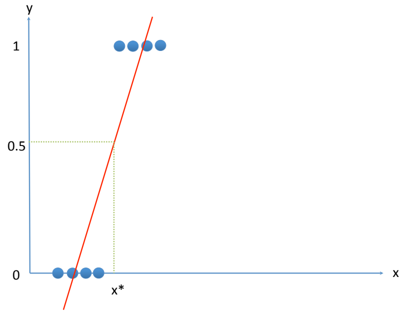
Limitations of Linear Regression

From Regression to Classification

Until now, we assumed that $y \in \mathbb{R}$ (continuous variable). Let's assume now that $y \in \{0, 1\}$ (discrete variable).

Predicting $y \approx$ classification task.

Example: $y = 1$ when a patient has a disease and $y = 0$ otherwise.

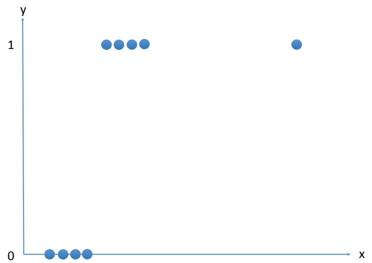


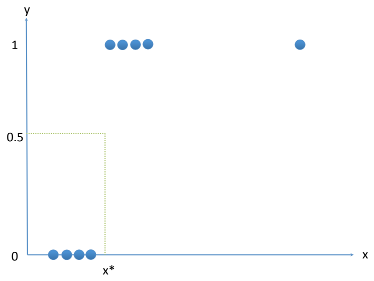
Limitations of Linear Regression

Linear regression for classification

Sometimes it will work OK but in general it is actually a pretty bad idea to apply linear regression to classification problems.

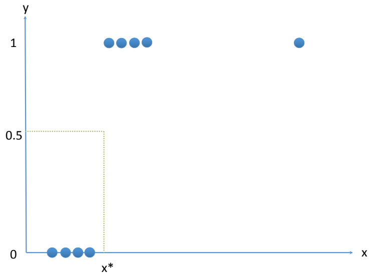
By giving you an additonal example, it's still obvious what the relationship between x and y is $\rightarrow$ It wouldn't change anything.

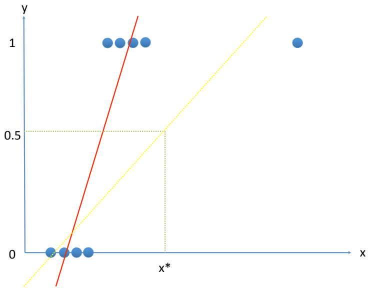




Limitations of Linear Regression

But if we now fit linear regression to this data, we end up with a different line and a different threshold.





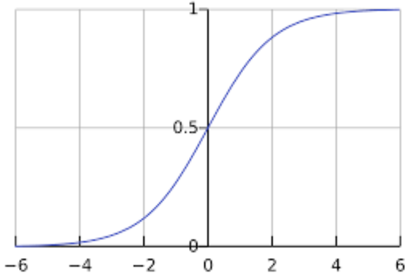
To fix this problem, instead of choosing a linear function, we are going to take something slightly different: the **logistic function**.

Logistic Regression

Assume $y \in \{0,1\}$ and $h_\theta(x) \in [0,1]$. Let us define $h_\theta(x)$ as follows:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is the **sigmoid (or logistic)** function.



Logistic Regression

Assume we aim at learning $h_\theta(x)$ such that:

$$P(y=1|x,\theta) = h_\theta(x)$$
$$P(y=0|x,\theta) = 1 - h_\theta(x)$$

Thus, $\forall y \in \{0,1\}$ we get:

$$P(y|x,\theta) = h_\theta(x)^y \left(1 - h_\theta(x)\right)^{1-y}$$

We deduce that the likelihood is:

$$L(\theta) = P(\vec{y}|x;\theta) = \prod_i P(y^{(i)}|x^{(i)},\theta) = \prod_i h_\theta(x^{(i)})^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

Logistic Regression

Likelihood Maximization

As done before, it is much easier to maximize the log of the likelihood $\ell(\theta) = lnL(\theta)$ rather than the likelihood $L(\theta)$.

$$\ell(\theta) = lnL(\theta) = \sum_i y^{(i)} ln\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) ln\left(1 - h_\theta(x^{(i)})\right)$$

To maximize $\ell(\theta)$, we can apply the same algorithm we run for minimizing the quadratic function in linear regression.

For every step, gradient ~~descent~~ ascent updates each parameter θ_j as follows:

$$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} \ell(\theta)$$

Logistic Regression

Likelihood Maximization

If we compute the partial derivatives of $\ell(\theta)$ w.r.t. θ_j, we get:

$$\frac{\partial}{\partial\theta_j}\ell(\theta) = \sum_i \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}.$$

Therefore,

$$\theta_j := \theta_j + \alpha \sum_i \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}.$$

Gradient Ascent Algorithm

We get exactly the same solution as least square regression.

$$\theta_j := \theta_j - \alpha \sum_i \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}.$$

Does it mean that it is the same algorithm?

Logistic Regression

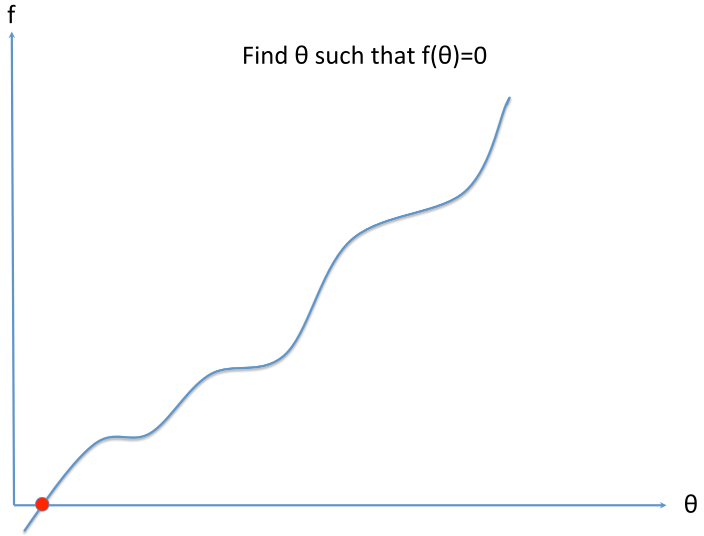
Gradient Ascent Algorithm

$$\theta_j := \theta_j - \alpha \sum_i \left(h_\theta(x^{(i)} - y^{(i)}) \right) x_j^{(i)}.$$

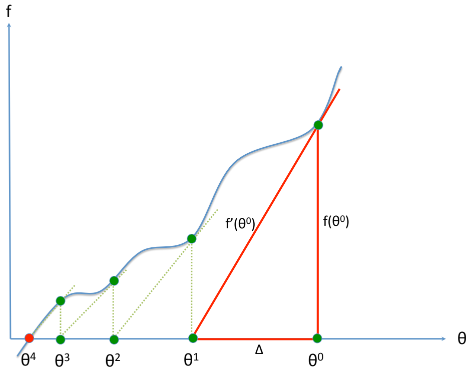
It turns out that there exists another optimization method that often runs much faster than gradient ascent (batch or stochastic) $\rightarrow$ Newton's method.

Newton's Method

Let's assume that we have a function $f(\theta)$. We aim at finding a value of θ s.t. $f(\theta) = 0$.


f
Find θ such that f(θ)=0
θ

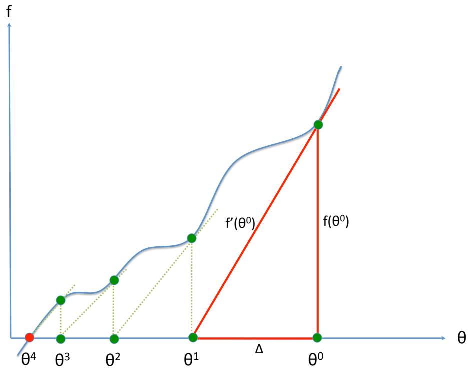
Newton's Method



By the definition of the gradients, $f'(\theta)$ at a given iteration is equal to the vertical length $f(\theta)$ divided by the horizontal length Δ. Therefore, at iteration 1, we get:

$$f'(\theta^0) = \frac{f(\theta^0)}{\Delta} \text{ and } \Delta = \frac{f(\theta^0)}{f'(\theta^0)}$$

Newton's Method



$$\theta^1 = \theta^0 - \Delta = \theta^0 - \frac{f(\theta^0)}{f'(\theta^0)} \text{ (quadratic convergence towards the solution)}$$

More generally, one iteration of Newton's method updates θ^t as follows:

$$\theta^{t+1} = \theta^t - \frac{f(\theta^t)}{f'(\theta^t)}.$$

Newton's Method for Logistic Regression

Let's apply this idea to maximizing the log-likelihood $\ell(\theta)$ in logistic regression. To maximize $\ell(\theta)$ we want to find θ s.t. $\ell'(\theta) = 0$. Therefore, applying Newton's method, we get:

$$\theta^{t+1} = \theta^t - \frac{\ell'(\theta^t)}{\ell''(\theta^t)}.$$

When θ is no longer a raw number but a feature vector, we get:

$$\theta^{t+1} = \theta^t - H^{-1}\nabla_\theta \ell$$

where

- H is the Hessian matrix.
- $\nabla_\theta \ell$ is the gradient.

Newton's Method versus Gradient Ascent Method

Advantage

For a reasonable number of features and training examples, Newton's method converges more quickly than gradient ascent in far fewer iterations.

Disadvantage

At each iteration, we need to invert the $n \times n$ Hessian matrix (where n is the number of features). Therefore, if n is very large, it is expensive. Otherwise, this method works very well for logistic regression.