High-Dimensional Data Analysis

Lecture 5 - Shades of Regression

Fall semester - 2024

Dr. Eng. Valentin Leplat
Innopolis University
September 25, 2024

Outline

Gradient Descent for Least squares problems

Recalls - Training set

- Recall the notations:
  1. $x^{(i)}$ to denote the "input" variables (living area in this example), also called input **features**,
  2. and $y^{(i)}$ to denote the "output" or target variable that we are trying to predict (price).
- A pair $(x^{(i)}, y^{(i)})$ is called a **training example/sample** .
- and the dataset that we'll be using to learn—a list of m training examples $\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$—is called a **training set**. [1]
- We will also use $\mathcal{X}$ denote the space of input values, and $\mathcal{Y}$ the space of output values. In this example, $\mathcal{X} = \mathcal{Y} = \mathbb{R}$.

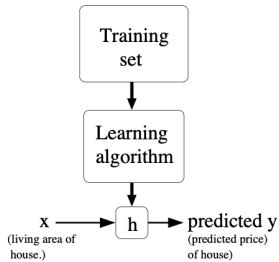
[1] Note that the superscript "(i)" in the notation is simply an index into the training set, and has nothing to do with exponentiation.

Recalls - h-hypothesis

- Formal description of a supervised learning problem, our goal is: *given a training set, to learn a function*

$$h : \mathcal{X} \to \mathcal{Y}$$

  so that $h(x)$ is a "good" predictor for the corresponding value of y.
- For historical reasons, this function h is called a **hypothesis**.
- Seen pictorially, the process is therefore like this:



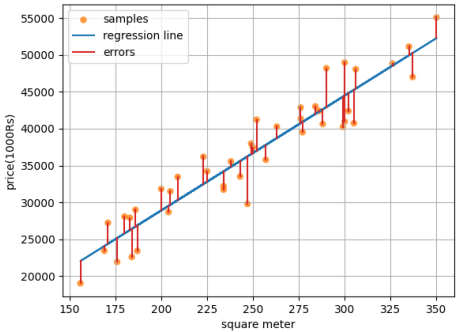
Recalls - one dimensional linear regression

We have a set of m points in the plane

$$p^{(i)} = (x^{(i)}, y^{(i)}) \in \mathbb{R}^2 \qquad i = 1, 2, \ldots, m,$$

and we want to approximate them with a line[2]

$$d = \{(x, y) \in \mathbb{R}^2 \mid y = \theta_1 x + \theta_0\} \subset \mathbb{R}^2.$$



[2] We assume that the data is linearly related

Recalls - linear regression

In the exact case, we would have

$$y^{(i)} = \theta_1 x^{(i)} + \theta_0 \text{ for all } i.$$

In matrix form, this is equivalent to solving a linear system (with two variables: θ_1 and θ_0)

$$\begin{pmatrix} x^{(1)} & 1 \\ x^{(2)} & 1 \\ \vdots & \vdots \\ x^{(m)} & 1 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_0 \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix},$$

In the n-dimensional case, it boils down to solve:

$$A\theta = b$$

where $A \in \mathbb{R}^{m \times n}$ (m, n resp. denotes the number of samples and the dimension of the vector of unknowns θ), and $b \in \mathbb{R}^m$.

Recalls - General least squares problems

Due to the presence of noise within the data, or in the the *overdetermined* case, no (exact) solution exists:

- we would often like to find the solution θ that minimizes some error measured by the function

$$\|A\theta - b\|$$

- for some vector norm $\|.\|$ in $\mathbb{R}^m$.

Recalls - General least squares problems

Due to the presence of noise within the data, or in the the *overdetermined* case, no (exact) solution exists:

- we would often like to find the solution θ that minimizes some error measured by the function

$$\|A\theta - b\|$$

- for some vector norm $\|.\|$ in $\mathbb{R}^m$.
- a common choice is the squared ℓ_2-norm of $A\theta - b$, that is $\|A\theta - b\|_2^2$.
- The *General least squares problems* boils down to solve:

$$\min_{\theta \in \mathbb{R}^n} \quad \|A\theta - b\|_2^2$$

or equivalently:

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

with $h\left(x^{(i)}\right) := \langle \theta, x^{(i)} \rangle$.

Recalls - General least squares problems

- In *Lecture 3*, we have seen how to solve this optimization problem using SVD, we called such a technique a *direct* method.
- In this section, we briefly present an *iterative* method, the (stochastic) *Gradient Descent* method.
- To ease a bit the notation, let us denote $f(\theta) := \|A\theta - b\|_2^2$.

Recalls - General least squares problems

- In *Lecture 3*, we have seen how to solve this optimization problem using SVD, we called such a technique a *direct* method.
- In this section, we briefly present an *iterative* method, the (stochastic) *Gradient Descent* method.
- To ease a bit the notation, let us denote $f(\theta) := \|A\theta - b\|_2^2$.
- Why Gradient Descent ?

Recalls - General least squares problems

- In *Lecture 3*, we have seen how to solve this optimization problem using SVD, we called such a technique a *direct* method.
- In this section, we briefly present an *iterative* method, the (stochastic) *Gradient Descent* method.
- To ease a bit the notation, let us denote $f(\theta) := \|A\theta - b\|_2^2$.
- Why Gradient Descent ? **A**: answers will come, we need first to gather some knowledge.

What is gradient descent ?

- An *iterative numerical* scheme that generates a sequence $\{\theta_k\}_{k=1}^{\infty}$[3] that we hope will converge towards a "good" solution of the optimization problem $\min_\theta f(\theta)$.
- The most well-known and simple first-order method for a differentiable function f.

Iteration ($k \geqslant 0$):

Initialization: Set $\theta_0 \in \mathbb{R}^n$.

1. Choose a step size α_k (s.t. $f(\theta_{k+1}) < f(\theta_k)$)
2. Compute

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla f(\theta_k) \tag{1}$$

[3] k designates the iteration counter.

What is gradient descent ?

- An *iterative numerical* scheme that generates a sequence $\{\theta_k\}_{k=1}^{\infty}$[3] that we hope will converge towards a "good" solution of the optimization problem $\min_\theta f(\theta)$.
- The most well-known and simple first-order method for a differentiable function f.

Iteration ($k \geqslant 0$):

Initialization: Set $\theta_0 \in \mathbb{R}^n$.

1. Choose a step size α_k (s.t. $f(\theta_{k+1}) < f(\theta_k)$)
2. Compute

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla f(\theta_k) \tag{1}$$

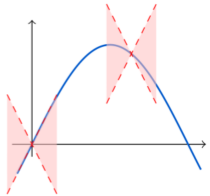
Many variants of this method: differ from the *step size strategy*

- The sequence $\{\alpha_k\}_{k=0}^{\infty}$ is chosen in advance, ex: $\alpha_k = \alpha$ (constant) or $\alpha_k = \frac{\alpha_0}{\sqrt{k+1}}$
- Full relaxation: $\alpha_k = \arg\min_{\alpha \geqslant 0} f(\theta_k - \alpha \nabla f(\theta_k))$
- *Armijo* rule

[3] k designates the iteration counter.

Why is it working ?

We need to have *convenient* functions :).



A function f is Lipschitz continuous on its domain if there exists a constant $L > 0$ such that for any $x, y \in \mathrm{dom} f$, we have:

$$|f(x) - f(y)| \leqslant L||x - y||$$

Insights:

- Such a function is limited in how fast it can change, L is an upper bound to the maximum steepness of $f(x)$
- Stronger than continuous, weaker than continuously differentiable
  Example: $f(x) = |x|$ is Lipschitz continuous but not continuously differentiable (in 0).

Why is it working ?

Even more *convenient* functions for us :).

A differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is L-smooth with respect to a norm $\|.\|$ if

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n$$

Insights:

- smoothness assures us that the gradient cannot change too quickly
- therefore we have an assurance that the gradient information is informative within a region around where it is taken.
- Implication and intuition: we may decrease the function's value by moving the direction opposite of the gradient.

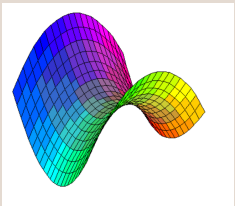
How good is it working ?

Stationary point

A stationary point of f at $x^\star$ satisfy the first-order optimality condition, in the unconstrained case we have: $\nabla f(x^\star) = 0$.

Saddle point

A saddle point of f is a stationary point that is neither a local max neither a local min. [a]



$\nabla^2 f(x^\star)$ has at least one strictly neg. and one strictly pos. eigenvalue.

[a] Intuitively, it exists a descent direction and an ascent direction

How good is it working ?

Global convergence: starting faraway from a stationary point $x^\star$

- convergence to a stationary point given that the step sizes are properly chosen.
- The function is supposed to be L-smooth and bounded below :).

How good is it working ?

Global convergence: starting faraway from a stationary point $x^\star$

- convergence to a stationary point given that the step sizes are properly chosen.
- The function is supposed to be L-smooth and bounded below :).
- We can obtain a rate of convergence of the sequence $\{\|\nabla f(x_k)\|\}_{k=0}^{\infty}$ as the following:

$$\min_{0 \leqslant i \leqslant k} \|\nabla f(x_i)\| \leqslant \frac{1}{\sqrt{k+1}} \left[C \left(f(x_0) - f(x^\star)\right)\right]^{\frac{1}{2}} \sim O(k^{-1/2})$$

where C is some positive constant.

How good is it working ?

Global convergence: starting faraway from a stationary point $x^\star$

- convergence to a stationary point given that the step sizes are properly chosen.
- The function is supposed to be L-smooth and bounded below :).
- We can obtain a rate of convergence of the sequence $\{\|\nabla f(x_k)\|\}_{k=0}^{\infty}$ as the following:

$$\min_{0 \leqslant i \leqslant k} \|\nabla f(x_i)\| \leqslant \frac{1}{\sqrt{k+1}} \left[C\left(f(x_0) - f(x^\star)\right)\right]^{\frac{1}{2}} \sim O(k^{-1/2})$$

where C is some positive constant.
Hence:

$$\|\nabla f(x_k)\| \to 0 \text{ as } k \to \infty$$

- Our goal is modest here: approach a stationary point (local minimum if possible) !

How good is it working ?

Global convergence: starting faraway from a stationary point $x^\star$

- convergence to a stationary point given that the step sizes are properly chosen.
- The function is supposed to be L-smooth and bounded below :).
- We can obtain a rate of convergence of the sequence $\{\|\nabla f(x_k)\|\}_{k=0}^{\infty}$ as the following:

$$\min_{0 \leqslant i \leqslant k} \|\nabla f(x_i)\| \leqslant \frac{1}{\sqrt{k+1}} \left[C \left(f(x_0) - f(x^\star)\right)\right]^{\frac{1}{2}} \sim O(k^{-1/2})$$

where C is some positive constant.
Hence:

$$\|\nabla f(x_k)\| \to 0 \text{ as } k \to \infty$$

- Our goal is modest here: approach a stationary point (local minimum if possible) !
- *Remark*: there is no dependence in the dimension n !!

Why gradient descent ?

m training samples, and n features.

Gradient Descent Method

- Need to choose α_k.
- Needs many iterations.
- Works well even if n is large.

Direct Method (solving Normal Equations)

- No need to choose α_k.
- Don't need to iterate.
- Need to compute $(A^T A)^{-1}$ ($O(n^3)$) or the SVD ($O(\min(mn^2, m^2 n))$)
  Slow if n is very large (starting from $n \approx 10^5$).

Small illustration

- To illustrate how to proceed in practice, consider the simple two-dimensional this function

$$f(x, y) := x^2 + 3y^2$$

where $\nabla f(x, y) = \begin{pmatrix} 2x \\ 6y \end{pmatrix}$

- For this example, we can compute the optimal step size with full relaxation, that is

$$\alpha_k := \text{argmin}_{\alpha_k} f((x_k, y_k) - \alpha \nabla f(x_k, y_k))$$

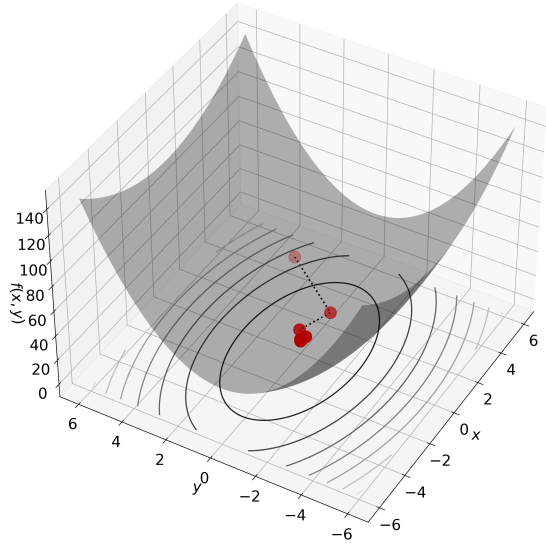
Pose $g(\alpha) = f((x_k, y_k) - \alpha \nabla f(x_k, y_k)) = (x_k(1 - 2\alpha))^2 + 3(y_k(1 - 6\alpha))^2$.
Compute $\alpha \geqslant 0$ such that $g^{'}(\alpha) = 0$, that is solving

$$x_k^2(2\alpha - 1) + 3y_k^2(18\alpha - 3) = 0$$

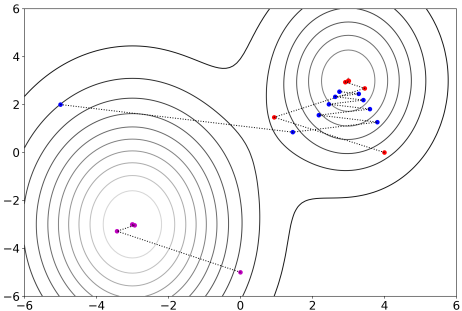
We get $\alpha = \frac{x_k^2 + 9y_k^2}{2x_k^2 + 54y_k^2} \geqslant 0$.

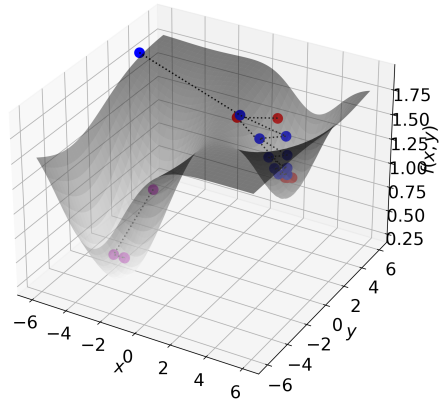
- Demo - Section 1

Small illustration - convex



Small illustration - non-convex





Step size ? - we are so lucky :)

- For our problem at hand

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

with $f(\theta) := \frac{1}{2}\|A\theta - b\|_2^2$.

Step size ? - we are so lucky :)

- For our problem at hand

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

with $f(\theta) := \frac{1}{2}\|A\theta - b\|_2^2$.

- We are lucky here since it is possible to find the best *constant* step size, that is:

$$\alpha_k = \frac{1}{L} \quad \forall k$$

where L is the Lipschitz constant for the gradient.
Proof on the board :).

Step size ? - we are so lucky :)

- For our problem at hand

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

  with $f(\theta) := \frac{1}{2}\|A\theta - b\|_2^2$.
- We are lucky here since it is possible to find the best *constant* step size, that is:

$$\alpha_k = \frac{1}{L} \quad \forall k$$

  where L is the Lipschitz constant for the gradient.
  Proof on the board :).
- Our gradient here ? $\nabla f(\theta) = A^T A\theta - A^T b$.
- What is L then ? By definition, choosing $\|.\| := \|.\|_2$, we have

Step size ? - we are so lucky :)

- For our problem at hand

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

with $f(\theta) := \frac{1}{2}\|A\theta - b\|_2^2$.

- We are lucky here since it is possible to find the best *constant* step size, that is:

$$\alpha_k = \frac{1}{L} \quad \forall k$$

where L is the Lipschitz constant for the gradient.
Proof on the board :).

- Our gradient here ? $\nabla f(\theta) = A^T A\theta - A^T b$.
- What is L then ? By definition, choosing $\|.\| := \|.\|_2$, we have

$$\|\nabla f(x) - \nabla f(x)\|_2 = \|A^T A(x - y)\|_2 \leqslant \|A^T A\|_2 \|x - y\|_2$$

for all $x, y \in \mathbb{R}^n$, hence $\|A^T A\|_2 = L = (\sigma_1(A))^2$

Stochastic Gradient Descent - Least Squares Lin. Reg.

- Very[very] roughly speaking: when m is way too big, computing the *full* gradient $\nabla f(\theta)$ becomes too expansive.
- We may decide at each iteration to update θ_k by using only **one** sample at the time and picked **randomly**[4] instead of the full training set.
- For a selected $x^{(i_k)}$, the stochastic gradient at iteration k, denoted g_k, is computed as follows:

$$g_k := (\langle \theta_k, x^{(i_k)} \rangle - y^{(i_k)}) x^{(i_k)}. \tag{2}$$

[4] based on a uniform distribution or by cycling over all i sequentially.

Stochastic Gradient Descent - Least Squares Lin. Reg.

Iteration ($k \geqslant 0$):

Initialization: Set $\theta_0 \in \mathbb{R}^n$.

1. Choose $i_k \sim \mathcal{U}(1, ..., m)$
2. Choose a step size $\alpha_k > 0$
3. Compute $g_k := (\langle \theta_k, x^{(i_k)} \rangle - y^{(i_k)}) x^{(i_k)}$
4. Compute

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k g_k \tag{3}$$

Stochastic Gradient Descent - Least Squares Lin. Reg.

- The convergence guarantees are significantly different, out of the scope of this class to discuss them.
- In particular: the step size should be a decreasing sequence such as

$$\alpha_k := \frac{\beta}{\gamma + k}$$

  with some positive constants β and γ.
- For those interested: ISP - Skoltech (2023) - Part 1

Logistic Regression

Some context and motivation

- In this section: we talk about the classification problem.
- This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values.
- For now, we will focus on the **binary** classification problem in which y can take on only two values, 0 and 1.[5]
- For instance, if we are trying to build a spam classifier for email, then $x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise.
- Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

[5] Most of what we say here will also generalize to the multiple-class case.

Recall - Bernoulli MLE Estimation

Some basic things first:

- the **Bernoulli distribution**
  - is the discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability $1 - p$.
  - can be used to represent a (possibly biased) coin toss where 1 and 0 would represent "heads" and "tails", respectively, and p would be the probability of the coin landing on heads. [6]
- **Parameters**:
  - given a model, the parameters are the numbers that yield the actual distribution.
  - In the case of a Bernoulli random variable, the single parameter was the value p.
- **Maximum Likelihood Estimation** (MLE): algorithm for estimating parameters !
  - Central idea: select that parameters θ that make the observed data the most likely.
  - use data: m independent and identically distributed (IID) samples $X^{(1)}, ..., X^{(m)}$.

[6] In particular, unfair coins would have $p \neq 1/2$.

Recall - Bernoulli MLE Estimation

- Goal here: use MLE to estimate the p parameter of a Bernoulli distribution.
- $X^{(i)} \sim \text{Ber}(p)$. We want to find out what that p is.
- Step one of MLE is to write the likelihood of a Bernoulli as a function that we can maximize.
- Since a Bernoulli is a discrete distribution, the likelihood is the probability mass function.
- The probability mass function of a Bernoulli X can be written as

$$f(X; p) = p^X (1-p)^{(1-X)}$$

- Cool :D: Its an equation that allows us to say that the probability that $X = 1$ is p and the probability that $X = 0$ is $1 - p$.
- Exercise: Convince yourself that when $X^{(i)} = 0$ and $X^{(i)} = 1$ the PMF returns the right probabilities.

Recall - Bernoulli MLE Estimation

Now let's do some MLE estimation:

- First write the likelihood function:

$$L(p) = \prod_{i=1}^{m} p^{X^{(i)}} (1-p)^{(1-X^{(i)})}$$

- Then write the log likelihood function:

$$\begin{aligned} l(p) &= \log L(p) \\ &= \sum_{i=1}^{m} p^{X^{(i)}} (1-p)^{(1-X^{(i)})} \\ &= \sum_{i=1}^{m} X^{(i)} \log(p) + \sum_{i=1}^{m} (1-X^{(i)}) \log(1-p) = \sum_{i=1}^{m} X^{(i)} \log\left(\frac{p}{1-p}\right) + m \log(1-p) \end{aligned}$$

- one way to find the value which maximizes a function that is to find the first derivative of the function and set it equal to 0:

$$\frac{dl(p)}{dp} = \frac{\sum_i^m X^{(i)} - mp}{p(1-p)} = 0$$

then $p = \frac{\sum_i^m X^{(i)}}{m}$.

All that work and find out that the MLE estimate is simply the sample mean...

Back to business

- For logistic regression, p will be a little bit more complicated, it has to be linked to the data samples and the parameters θ :).
- **Spoiler**: we will *learn* p given the training set, i.e. p will be approximated by some $h_\theta(x)$.
- To achieve this, we first highlight key points and introduce some useful objects:
  - we could approach the classification problem ignoring the fact that y is discrete-valued, and use our "old" linear regression algorithm to try to predict y given x.
  - However, it is easy to construct examples where this method performs very poorly.
  - Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$.

Choice for $h_\theta(x)$

Key idea

To fix this, lets change the form for our hypotheses $h_\theta(x)$. We will choose

$$h_\theta(x) := g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{4}$$

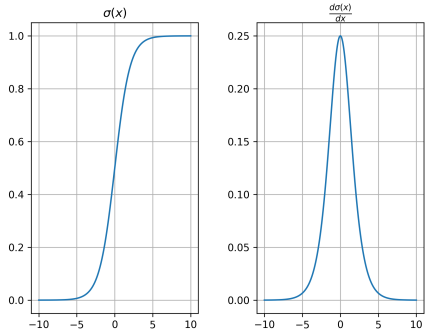
where

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

is called the **logistic function** or the **sigmoid function**.

Choice for $h_\theta(x)$

Here is a plot showing $g(z)$:



- Notice that $g(z)$ tends towards 1 as $z \to +\infty$, and $g(z)$ tends towards 0 as $z \to -\infty$.
- Moreover, $g(z)$, and hence also $h_\theta(x)$, is always bounded between 0 and 1.

A bit more on Sigmoid function

- For now, lets take the choice of g as given.
- Other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons, the choice of the logistic function is a fairly natural one.
- Before moving on, here's a useful property of the derivative of the sigmoid function, which we write a $g^{'}$:

$$\begin{aligned} g^{'}(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{1+e^{-z})}(1-\frac{1}{1+e^{-z}}) \\ &= g(z)(1-g(z)) \end{aligned} \tag{5}$$

Estimation of θ

- So, given the logistic regression model, how do we fit θ for it ?

[7] More formally, we consider a **conditional** maximum likelihood estimation

Estimation of θ

- So, given the logistic regression model, how do we fit θ for it ?
- It could be derived as the maximum likelihood estimator under a set of assumption ![7]
- Ok, lets endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood

[7] More formally, we consider a **conditional** maximum likelihood estimation

Estimation of θ

(conditional) PMF

Let us assume

$$\begin{aligned} \mathbb{P}(y = 1|x; \theta) &= \underset{\text{"}p\text{"}}{h_\theta(x)} \\ \mathbb{P}(y = 0|x; \theta) &= 1 - h_\theta(x) \end{aligned} \tag{6}$$

Note that this can be written more compactly as *a conditional probability mass function* (PMF):

$$\mathbb{P}(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Estimation of θ

- Assuming that the m training examples were generated independently, we can then write down the (conditional) likelihood of the parameters as

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} \mathbb{P}(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned} \tag{7}$$

- As before, it will be easier to maximize the log likelihood:

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + \sum_{i=1}^{m} (1 - y^{(i)}) \log\Big(1 - h_\theta(x^{(i)})\Big) \tag{8}$$

- In order to turn this into a *loss function* (something that we need to minimize), we'll just flip the sign on Equation (8):

$$f(\theta) := -l(\theta) = \sum_{i=1}^{m} \Big(-y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log\Big(1 - h_\theta(x^{(i)})\Big)\Big) \tag{9}$$

Estimation of θ

- Note that Equation (9) is the result of the sum of the so-called *cross-entropy loss* for each sample.
- For a single sample (x, y), the *cross-entropy loss*, denoted $L_{CE}(.,.)$, is defined as follows

$$L_{CE}(\hat{y}, y) = -\log \mathbb{P}(y|x;\theta) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) \tag{10}$$

  where $\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$.
- We want the loss to be smaller if the model's estimate is close to correct, and bigger if the model is confused.
  Exercise: try to convince yourselves that if this loss function does the right thing.
- For information: It's called the *cross entropy loss*, because Equation (10) is also the formula for the *cross-entropy* between the true probability distribution y and our estimated distribution $\hat{y}$.

Estimation of θ

How do we maximize the likelihood ?

- Similar to our derivation in the case of least squares linear regression, we can use Gradient **Descent**. Written in vectorial notation, our updates will therefore be given

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla_\theta f(\theta_k)$$

- Lets start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient descent rule

$$\begin{aligned}
\nabla_\theta f(\theta) &= -(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})\nabla_\theta g(\theta^T x) \\
&= -(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})g(\theta^T x)(1-g(\theta^T x))\nabla_\theta(\theta^T x) \\
&= -(y - g(\theta^T x))x = (h_\theta(x) - y)x
\end{aligned} \tag{11}$$

Above, we used the fact that $g^{'}(z) = g(z)(1-g(z))$.

Stochastic Gradient Descent - Logistic Regression

Iteration ($k \geqslant 0$):

Initialization: Set $\theta_0 \in \mathbb{R}^n$.

1. Choose $i_k \sim \mathcal{U}(1, ..., m)$
2. Choose a step size $\alpha_k > 0$
3. Compute $g_k := (h_{\theta_k}(x^{(i_k)}) - y^{(i_k)})x^{(i_k)}$
4. Compute

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k g_k \tag{12}$$

- If we compare this to the update rule (3) for Least Squares Linear Regression, we see that it looks identical; but this is not the same algorithm, because $h_\theta(x^{(i)})$ is now defined as a non-linear function of $\theta^T x^{(i)}$.
- Nonetheless, it's a little surprising that we end up with the same update rule for a rather different algorithm and learning problem. Is this coincidence, or is there a deeper reason behind this ? Answers can be found when studying the Generalized Linear Models (GLM).

Estimation of θ

- The full gradient, that is taking into account all the data points, is

$$\nabla f(\theta) = X^T(h_\theta(X) - y)$$

where X is the matrix of size $m \times n$ with $X[i,:]^T = x^{(i)} \in \mathbb{R}^n$ (each row of X is a sample/data point), with $y \in \mathbb{R}^m$ the vector of labels, $h_\theta(X) = \frac{1}{1+e^{\cdot(X\theta)}} \in \mathbb{R}^m$ with $e^{\cdot()}$ the element-wise exponential. [8]

- **Demo**: Section 2 - subsection 2
  Data set

[8] exponential applied to each component of the vector $X\theta$

Another algorithm for maximizing $l(\theta)$

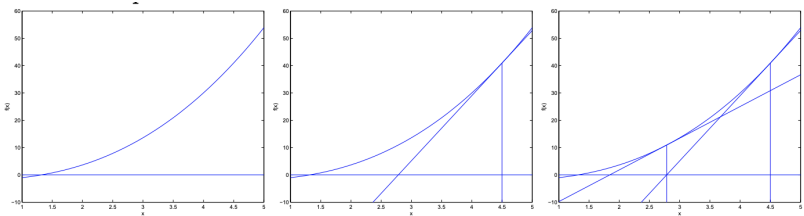
- To get us started, lets consider Newton's method for finding a zero of a function.
- Specifically, suppose we have some function $F : \mathbb{R} \to \mathbb{R}$, and we wish to find a value of θ so that $F(\theta) = 0$.
- Here, $\theta \in \mathbb{R}$ is a real number. Newton's method performs the following update:

$$\theta_{k+1} \leftarrow x_k - \frac{F(\theta_k)}{F'(\theta_k)}$$

- This method has a natural interpretation in which we can think of it as approximating the function F via a linear function that is tangent to F at the current guess θ, solving for where that linear function equals to zero, and letting the next guess for θ be where that linear function is zero.

Another algorithm for maximizing $l(\theta)$

- Here's a picture of the Newton's method in action:



- In the *leftmost* figure, we see the function F plotted along with the line $y = 0$. We're trying to find θ so that $F(\theta) = 0$; the value of θ that achieves this is about 1.3.
- Suppose we initialized the algorithm with $\theta = 4.5$. Newton's method then fits a straight line tangent to F at $\theta = 4.5$, and solves for the where that line evaluates to 0. (*Middle* figure.)
- This give us the next guess for $\theta = 4.5$, which is about 2.8. The *rightmost* figure shows the result of running one more iteration, which the updates θ to about 1.8. After a few more iterations, we rapidly approach $\theta = 1.3$.

Another algorithm for maximizing $l(\theta)$

- Newton's method gives a way of getting to $F(\theta) = 0$.
- What if we want to use it to minimize some function $f(\theta) = -l(\theta)$?
- The minima of $f(\theta)$ correspond to points where its first derivative is zero.
- So, by letting $F(\theta) = f^{'}(\theta)$, we can use the same algorithm to minimize $f(.)$, and we obtain update rule:

$$\theta_{k+1} := \theta_k - \frac{f^{'}(\theta_k)}{f^{''}(\theta_k)}$$

- Something to think about: How would this change if we wanted to use Newton's method to maximize rather than minimize a function?

Another algorithm for maximizing $l(\theta)$

- Lastly, in our logistic regression setting, θ is vector-valued, so we need to generalize Newton's method to this setting.
- The generalization of Newton's method to this multidimensional setting (also called the Newton-Raphson method) is given by

$$\theta_{k+1} \leftarrow \theta_k - H^{-1} \nabla f(\theta_k)$$

where:

  - $\nabla f(\theta_k)$ is our usual gradient (vector of partial derivatives),
  - and H is an n-by-n matrix (actually, $n+1$-by-$n+1$, assuming that we include an intercept term θ_0) called the Hessian, whose entries are given by

$$H[i,j] = \frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j}$$

Another algorithm for maximizing $l(\theta)$

- Newton's method typically enjoys faster convergence than (batch) gradient descent, and requires many fewer iterations to get very close to the minimum.
- One iteration of Newton's can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an n-by-n Hessian; but so long as n is not too large, it is usually much faster overall.
- When Newton's method is applied to maximize/minimize the logistic regression log likelihood function $l(\theta)$, the resulting method is also called **Fisher scoring**.

A small Digression

The perceptron learning algorithm

- We now digress to talk briefly about an algorithm that's of some historical interest.
- Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1 or exactly.
- To do so, it seems natural to change the definition of g to be the threshold function

$$g(z) = \begin{cases} 1 \text{ if } z \geqslant 0 \\ 0 \text{ if } z < 0 \end{cases} \tag{13}$$

- If we then let $h_\theta(x) = g(\theta^T x)$ as before but using this modified definition of g, and if we use the update rule

$$\theta_{k+1} := \theta_k - \alpha_k g_k$$

with $g_k := (h_{\theta_k}(x^{(i_k)}) - y^{(i_k)})x^{(i_k)}$, then we have the **perceptron learning algorithm**.

The perceptron learning algorithm

- In the 1960s, this "perceptron" was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression and least squares linear regression.
- In particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.

Regularized Regressions

Context

- In this section, we limit the presentations of so-called *regularization's* for the **logistic** regression,
- however, the latter are typically considered as well for linear regressions.
- **Key observation**: There is a problem with learning weights that make the model perfectly match the training data.
- If a feature is perfectly predictive of the outcome because it happens to only occur in one class, it will be assigned a very high weight.
- The weights for features will attempt to perfectly fit details of the training set, in fact too perfectly, modeling noisy factors that just accidentally correlate with the class. This problem is called **overfitting**.

How to deal with overfitting ?

- To avoid overfitting, a new *regularization* term $R(\theta)$ is added to the loss function in Problem (9) resulting in the following Problem for a batch of m data points/samples:

$$\min_{\theta \in \mathbb{R}^n} \sum_{i=1}^{m} \left(-y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log\Big(1 - h_\theta(x^{(i)})\Big) \right) + \alpha R(\theta) \tag{14}$$

- The new regularization term $R(\theta)$ is used to *penalize* large weights.
- **Practical effect**: a setting of the weights that matches the training data perfectly—but uses many weights with high values to do so—will be penalized more than a setting that matches the data a little less well, but does so using smaller weights.
- **Drawback**: how to choose a proper α ?

How to deal with overfitting ?

- There are two (+1 combination) common choices for $R(\theta)$.
- First one - the ℓ_2 regularization [9] : a quadratic function of the weight values, named because it uses the (square of the) ℓ_2 norm of the weight values.
- If θ consists of n weights, then:

$$R(\theta) := \|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2 \tag{15}$$

- **Advantages**:
  1. it is differentiable, hence allowing to use Gradient Descent Methods effectively (including stochastic version).
  2. for Linear Regression
     - "optimal solution" can be found with SVD for *underdetermined* systems (lecture 3),
     - a fix when $A^T A$ is not invertible (why ?)

[9] also referred to as the *Ridge* regularization

How to deal with overfitting ?

- Second common choice: the ℓ_1 regularization.
- a linear function of the weight values, named after the ℓ_1-norm $\|\theta\|_1$, the sum of the absolute values of the weights, or *Manhattan distance* [10]:
- The ℓ_1 regularized Problem becomes:

$$\min_{\theta \in \mathbb{R}^n} \sum_{i=1}^{m} \Big(-y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log \Big(1 - h_\theta(x^{(i)}) \Big) \Big) + \alpha \sum_{i=1}^{n} |\theta_i| \qquad (16)$$

[10] the Manhattan distance is the distance you'd have to walk between two points in a city with a street grid like New York

Comparison of regularizations

- These kinds of regularization come from statistics, see for instance Tibshirani, R. J. *Regression shrinkage and selection via the lasso.* Journal of the Royal Statistical Society. Series B (Methodological), 58(1):267–288. 1996
- both are commonly used in language processing.

ℓ_2 (Ridge) regularization

- easy to optimize because of its simple derivative
- prefers weight vectors with many small weights

ℓ_1 (Lasso) regularization

- more complex to optimize ($|\theta|$ is not differentiable in zero).
- prefers sparse solutions with some larger weights but many more weights set to zero → leads to much sparser weight vectors, that is, far fewer features.

Remark - A third one regularization is also popular, and is called the ▸ elastic net model, combines the two regularizations.

Probabilistic Interpretation

- Both ℓ_1 and ℓ_2 regularization have Probabilistic (Bayesian) interpretations as constraints on the prior of how weights should look.
  1. ℓ_1 regularization can be viewed as a Laplace prior on the weights,
  2. ℓ_2 regularization corresponds to assuming that weights are distributed according to a Gaussian distribution with mean $\mu = 0$.
- In a Gaussian or normal distribution, the further away a value is from the mean, the lower its probability (scaled by the variance σ).
- By using a Gaussian prior on the weights, we are saying that weights prefer to have the value 0. A Gaussian distribution for a weight θ_j has probability density

$$\frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2}} \tag{17}$$

Probabilistic Interpretation

- If we multiply each weight by a Gaussian prior on the weight, we are thus maximizing the following

$$\theta^\star = \text{argmax}_\theta \prod_{i=1}^{m} \mathbb{P}(y^{(i)}|x^{(i)}) \times \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2}} \tag{18}$$

- which in log space, with $\mu_j = 0$, and assuming $2\sigma_j^2 = 1$ for all j, corresponds to

$$\theta^\star = \text{argmax}_\theta \sum_{i=1}^{m} \log \mathbb{P}(y^{(i)}|x^{(i)}) - \alpha \sum_{j}^{n} \theta_j^2 \tag{19}$$

  which is equivalent to the ℓ_2 regularized Logistic Regression Problem.

Summary

Summary

We have seen :

- An *iterative Method* for solving the Least squares problems: the **Gradient Descent** Method
- Key parameter of the Gradient Descent Method: choose the step size [11] α_k.
- The *Logistic Regression*: an hypothesis, or model, $h_\theta(x)$ which predict the probability of an input sample x to belong to one class $y \in \{0, 1\}$.
- How to estimate the parameters θ:
  1. Build the optimization problem with MLE,
  2. Use Gradient Descent Method to numerically solve this problem.
- The Newton's Method for solving the logistic regression problems.
- The ℓ_2 and ℓ_1 regularizations.

[11] also called *learning rate by AI specialists*

Preparation for the lab

- Review the lecture :).
- Practice the stochastic Gradient Descent for logistic Regression Problems;
  1. see how the loss function evolves along iterations.
  2. Does it always decrease ?
  3. What happens if you choose a constant step size vs a decreasing step size α_k ?

Next topics are outside the scope of this class, but are good things to look into next:

1. **Generalized Linear Models**:
   Michael I. Jordan, *Learning in graphical models* (unpublished book draft). ▸ Extract
   McCullagh and Nelder, *Generalized Linear Models* (2nd ed.). ▸ pdf
2. **Multinomial Logistic Regression**: The loss function for multinomial logistic regression generalizes the loss function for binary logistic regression from 2 to K classes. ▸ Stanford Notes

Goodbye, So Soon

THANKS FOR THE ATTENTION

- v.leplat@innopolis.ru
- sites.google.com/view/valentinleplat/