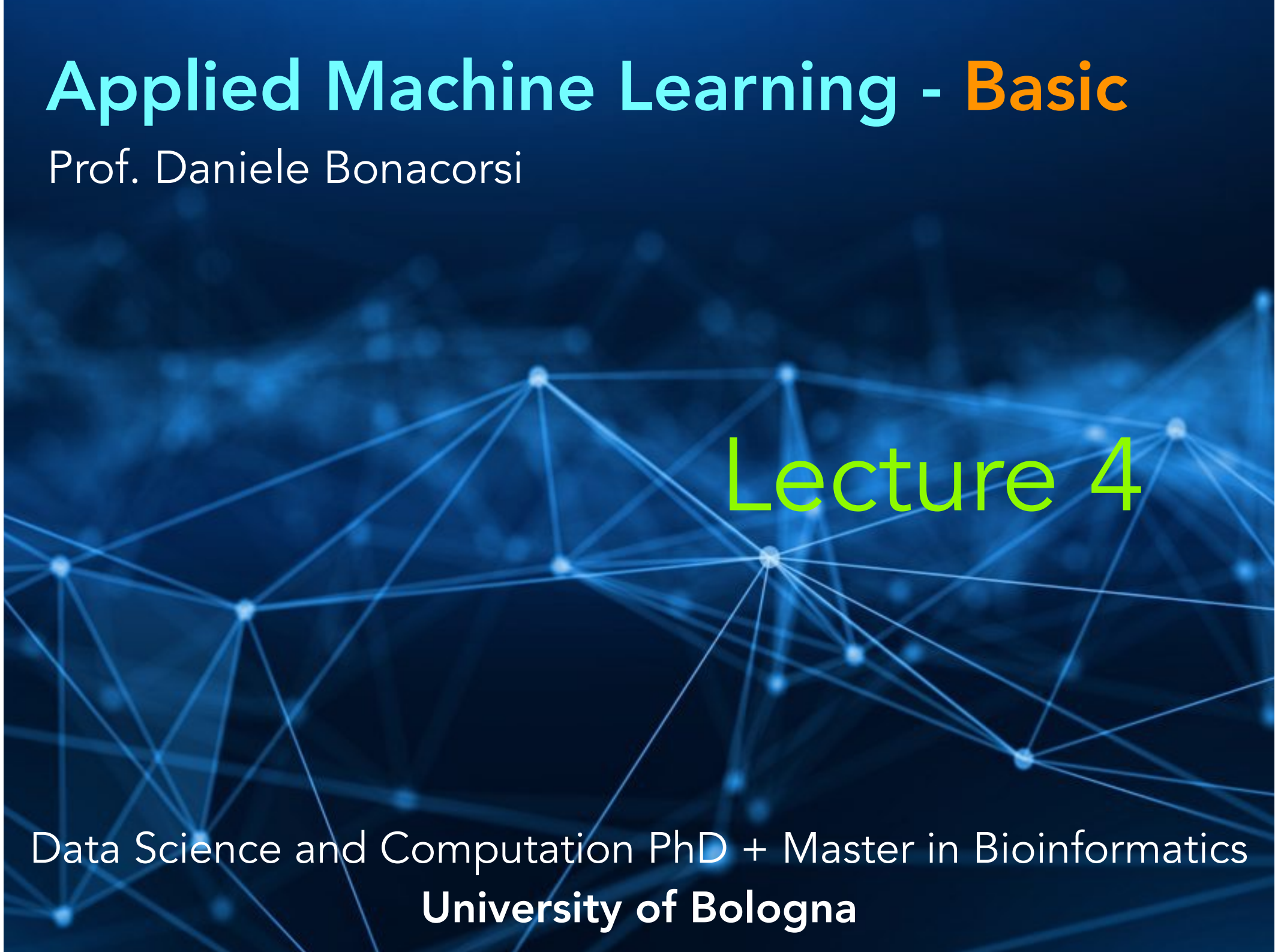
Applied Machine Learning - Basic

Prof. Daniele Bonacorsi



Lecture 4

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Communications

Where we are with the lectures' calendar

AML - Basic: 6 March - 17 April

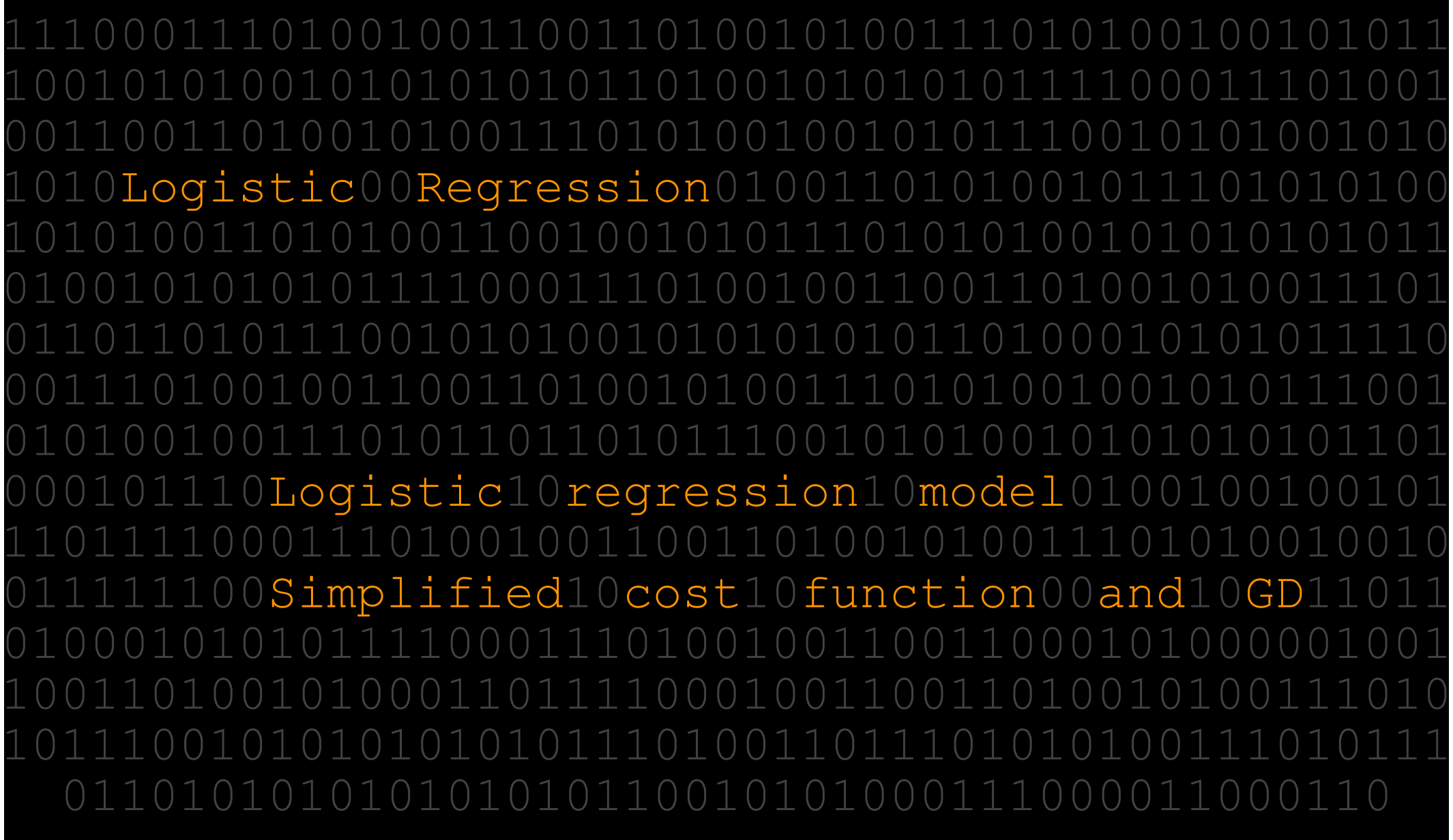
- 8 lectures → 8 * 4hrs = 32 hrs → **4 CFU**

mon Mar 6 14-18 (DONE)
fri Mar 10 14-18 (DONE)
mon Mar 13 14-18 (DONE)
fri Mar 17 14-18 (in progress..)
mon Mar 27 14-18 (to do)
Fri Mar 31 14-18 (to do)
mon Apr 14 14-18 (to do)
fri Apr 17 14-18 (to do)

Dates are susceptible to change:
in case, you will be notified in advance.

At the end of today's lecture **we will be at 50% of AML-Basic**

- Next week we do not have lectures.. time to recap, get your environment ready, play with colab, etc.. and get ready for the second half !


Logistic00Regression
Logistic10regression10model
Simplified10cost10function00and10GD

Simplified cost function and GD

We have a cost function for logistic regression (at least for one example in the training set).

We will work on it further, namely:

- (easy) we will extend it to the whole training set
- (easy) we will figure out a slightly simpler way to write it
- we will figure out how to apply GD to fit the parameters of logistic regression.

In a nutshell, steps above will soon enable us to **implement a fully working version of logistic regression**.

Rewrite and simplify the logistic regression cost function

[NOTE: we drop "(i)"]

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

$$\text{cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) & \text{if } y = 0 \end{cases} \qquad y \in \{0, 1\}$$

Rewrite:

$$\text{cost}\left(h_\theta(x), y\right) = -y \log\left(h_\theta(x)\right) - (1-y) \log\left(1 - h_\theta(x)\right)$$

Indeed:

if $y = 1$ $\Rightarrow$ $\text{cost} = -\log\left(h_\theta(x)\right)$

if $y = 0$ $\Rightarrow$ $\text{cost} = -\log\left(1 - h_\theta(x)\right)$

Logistic regression cost function (final form)

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

$$\text{cost}\left(h_\theta(x), y\right) = -y \log\left(h_\theta(x)\right) - (1-y) \log\left(1 - h_\theta(x)\right)$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) \right]$$

(and now we have put back the indexes, too..)

This is called **cross-entropy** or **log loss**. Why do we choose this particular function?

- it looks like there could be other cost functions we could have chosen…
- (no details here, but) this can be derived from statistics using the principle of maximum likelihood estimation (i.e. an idea in statistics for how to efficiently find parameters' data for different models).
- And it also has this nice property that it is convex as we need.

<u>This cost function is what essentially everyone uses in training logistic regression to build a good ML classification model.</u>

Recap: comparing the J's

Regression:

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

Logistic regression:

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

$$\text{cost}\left(h_\theta(x), y\right) = -y \log\left(h_\theta(x)\right) - (1-y)\log\left(1-h_\theta(x)\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right]$$

What's next? (in words, first)

Given this cost function, we need to:

- fit the θ parameters, i.e. try to find the θ parameters that minimise $J(\theta)$. This would give us some set of θ parameters.
- given a new example with some set of features x, we can then take the θs that we fit to our training set and output our prediction h
- the output of my hypothesis must then be interpreted as the probability that y=1, given the input x and parameterised by θs

What's next? (cont'd)

1. $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right]$

2. $\min_\theta J(\theta)$ => fit parameters and get θ

how to do this ?

3. make a prediction given new x => output: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

4. interpret the output as $P(y=1 \mid x;\theta)$

Now we must figure out how to actually minimise J(θ) so that we can actually fit the parameters to our training set.

Minimising J(θ) for logistic regression using GD

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

use GD => want $\min_\theta J(\theta)$

Repeat {

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

compute it...

Ready?

} (simultaneously update all θ_j)

Minimising J(θ) for logistic regression using GD

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

use GD => want $\min_\theta J(\theta)$

Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

}

(simultaneously update all θ_j)

Anything familiar?!

GD for linear regr. vs **GD for logistic regr.**

Compare what you get now for logistic regression to what you got for linear regression…

GD for linear rear vs **GD for logistic regr.**

Compare what you get now for logistic regression to what you got for linear regression…

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

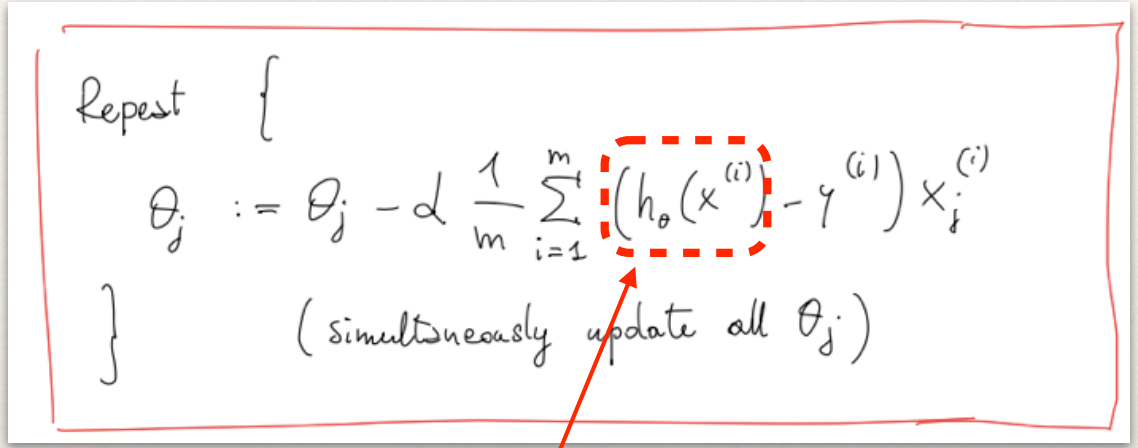
} (simultaneously update all θ_j)

The GD algorithm expression looks exactly the same!

So, are linear regression and logistic regression different algorithms or not?

GD for linear rear vs **GD for logistic regr.**

Compare what you get now for logistic regression to what you got for linear regression…



The GD algorithm expression looks exactly the same!

So, are linear regression and logistic regression different algorithms or not?

- From linear regression to logistic regression, what has changed is that **the definition for this hypothesis has changed**!
- So even though the GD update rule looks cosmetically identical, the definition of the hypothesis has changed, so **this is actually *not* the same thing as GD for linear regression: it looks like it, but differences are hidden in the h!**

GD for linear rear vs **GD for logistic regr.**

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

} (simultaneously update all θ_j)

Same GD algorithm… different hypotheses.

linear regression ⟶ $h_\theta(x) = \theta^T x$

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

Feature scaling for logistic regression

Same as before!

In linear regression we talked about feature scaling and mean normalisation.

We saw how it can help GD converge faster for linear regression.

The idea of feature scaling also applies to GD for logistic regression.

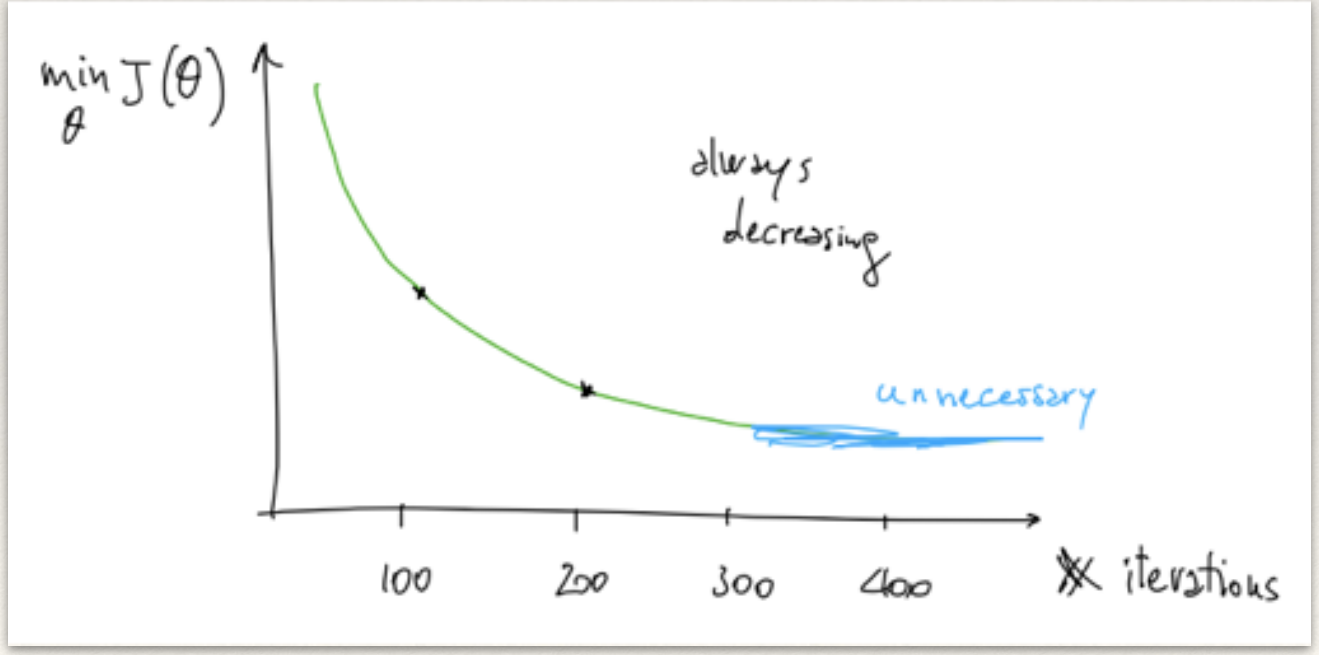
- When we have features that are on very different scale, then applying feature scaling can make GD run faster also for logistic regression.

Monitor your GD in logistic regression

Same as before!

Hint: in GD for linear regression, we talked about how to monitor a GD algo to make sure that it was converging. Apply that same method to logistic regression, too.

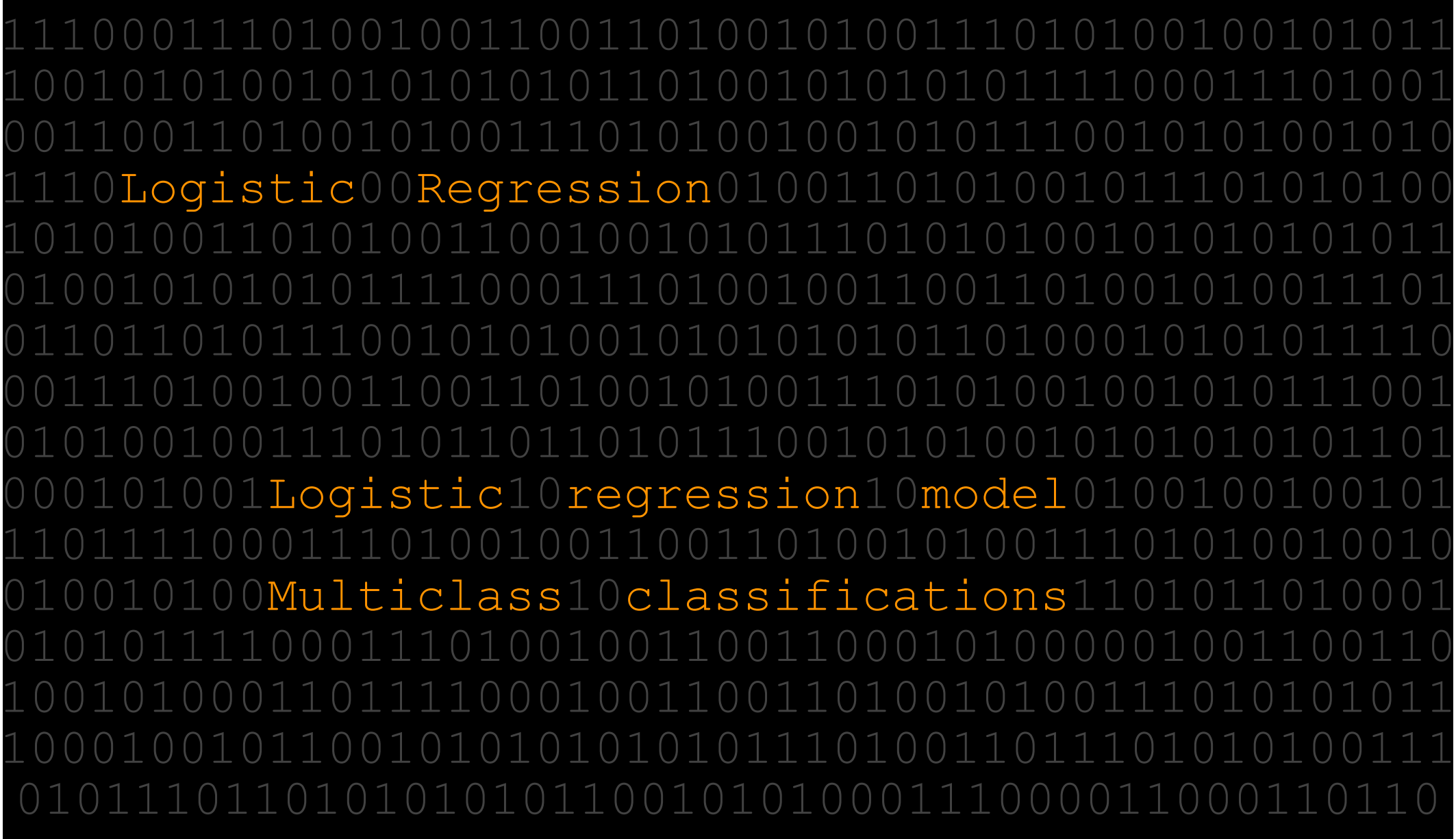
Remember:



Summary

We now know **how to implement GD for logistic regression**.

This is a very powerful, and probably the most widely used, classification algorithm in the ML world.


Logistic00Regression
Logistic10regression10model
Multiclass10classifications

We now need to get **logistic regression** to work for **multi-class classification** problems.

We will see in particular one algorithm for this: the **one-versus-all classification** algorithm.

Multi-class classification

What's a **multi-class classification** problem?

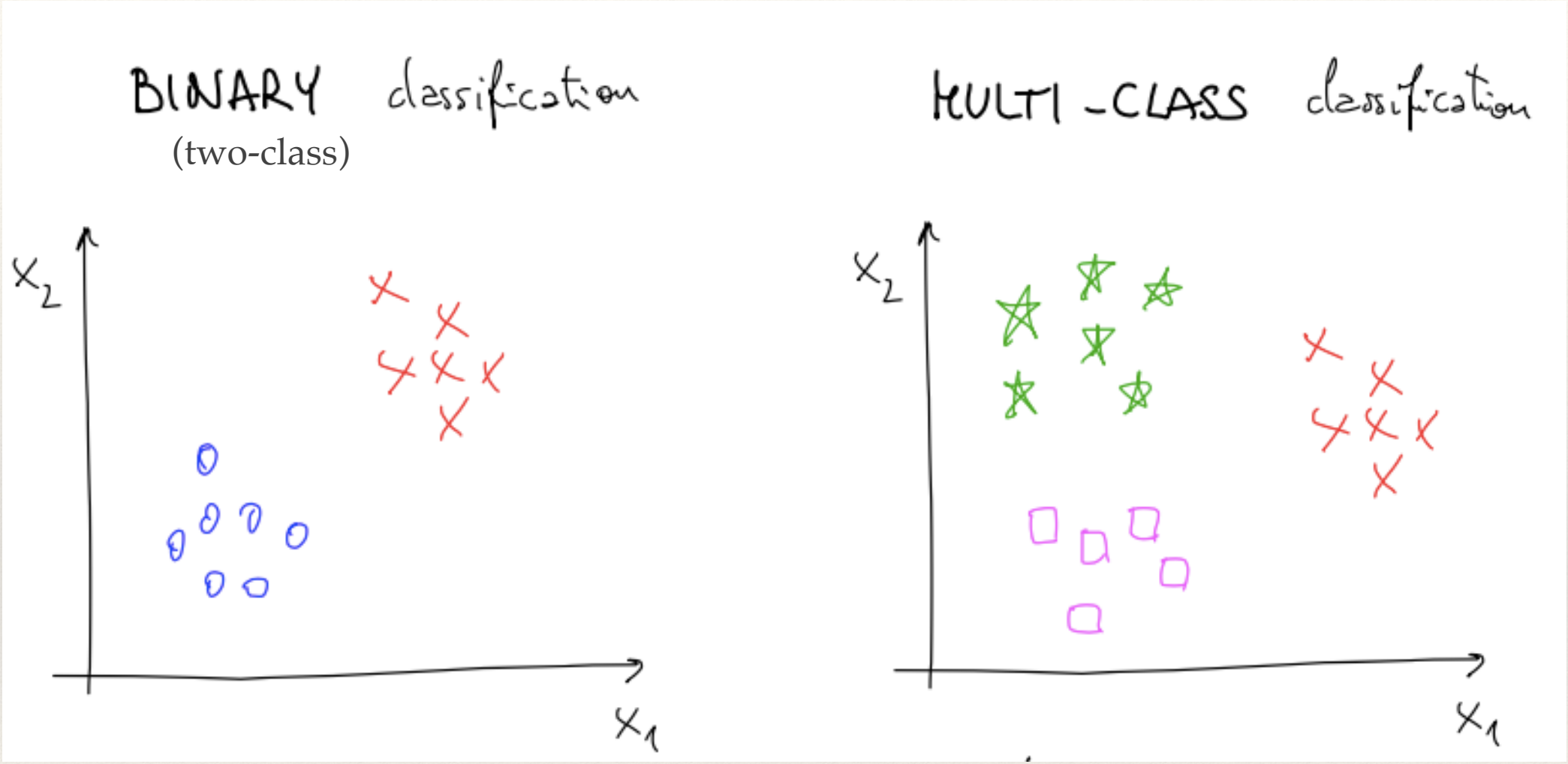
Examples:

- email foldering/tagging (home, work, family, hobby)
- medical diagnosis (not ill, cold, flu, pneumonia)
- weather (sunny, rainy, snowy, windy)

All these are multi-class classification problems.

Think of classes as if you tag each option y=1,2,3,.. or y=0,1,2,3

- again, it doesn't really matter how you index



How do we get a learning algorithm that works for the multi-class case, too?

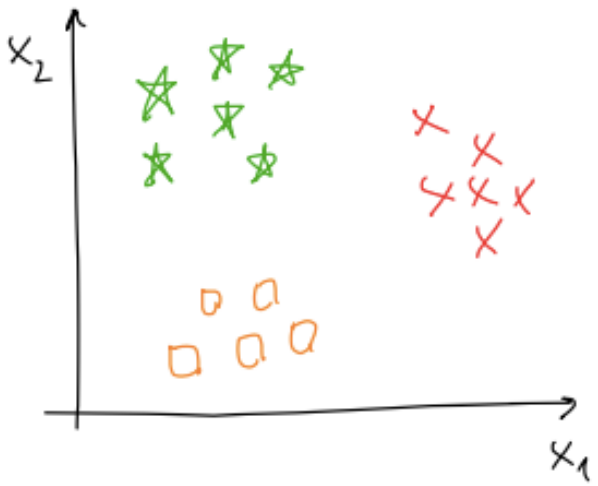
One-vs-all

For multi-class, we will adopt an approach called **one-vs-all** (one-vs-rest) **classification**

Basic idea is very simple: *take your training set and turn your problem into more (e.g. 3) separate binary (i.e. 2 classes) classification problems*

- create a new sort of fake training set where classes 2 and 3 get assigned to the negative class, and class 1 gets assigned to the positive class
- train a standard logistic regression classifier (number 1) to distinguish 1 and not-1. That (*as we learned already!*) will give a decision boundary
- same for class 2
- same for class 3

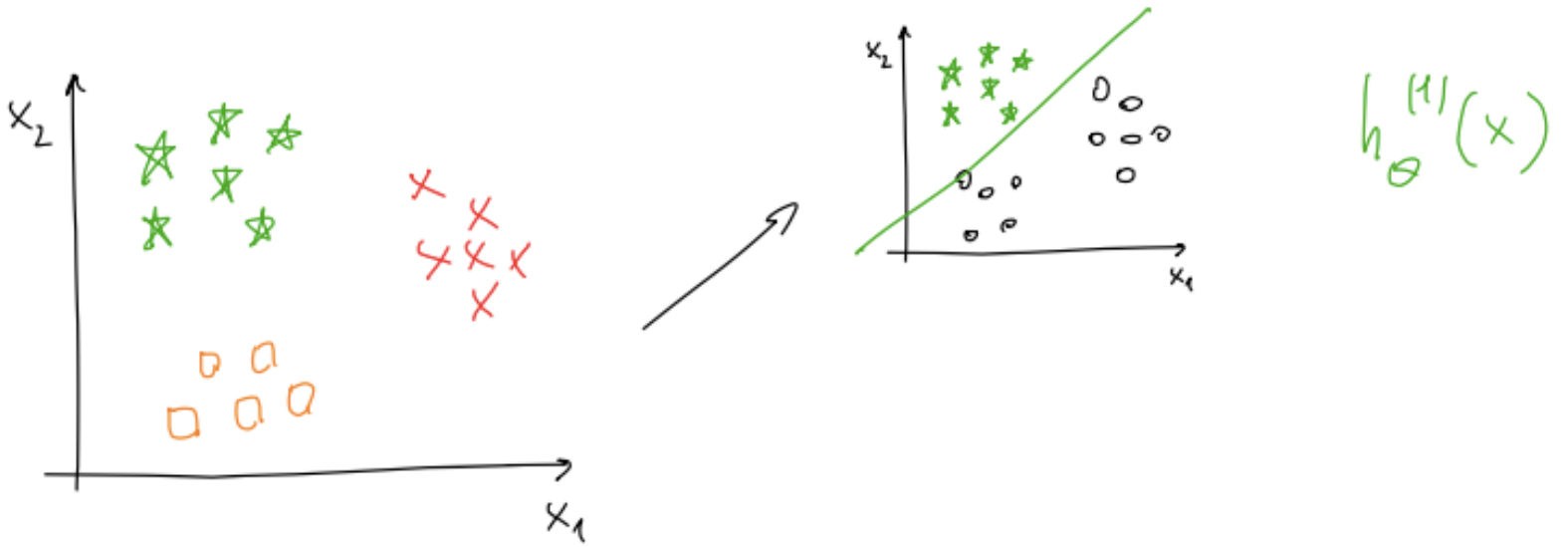
Intuitive? Not enough? Let's visualise it..



CLASS 1 ☆

CLASS 2 □

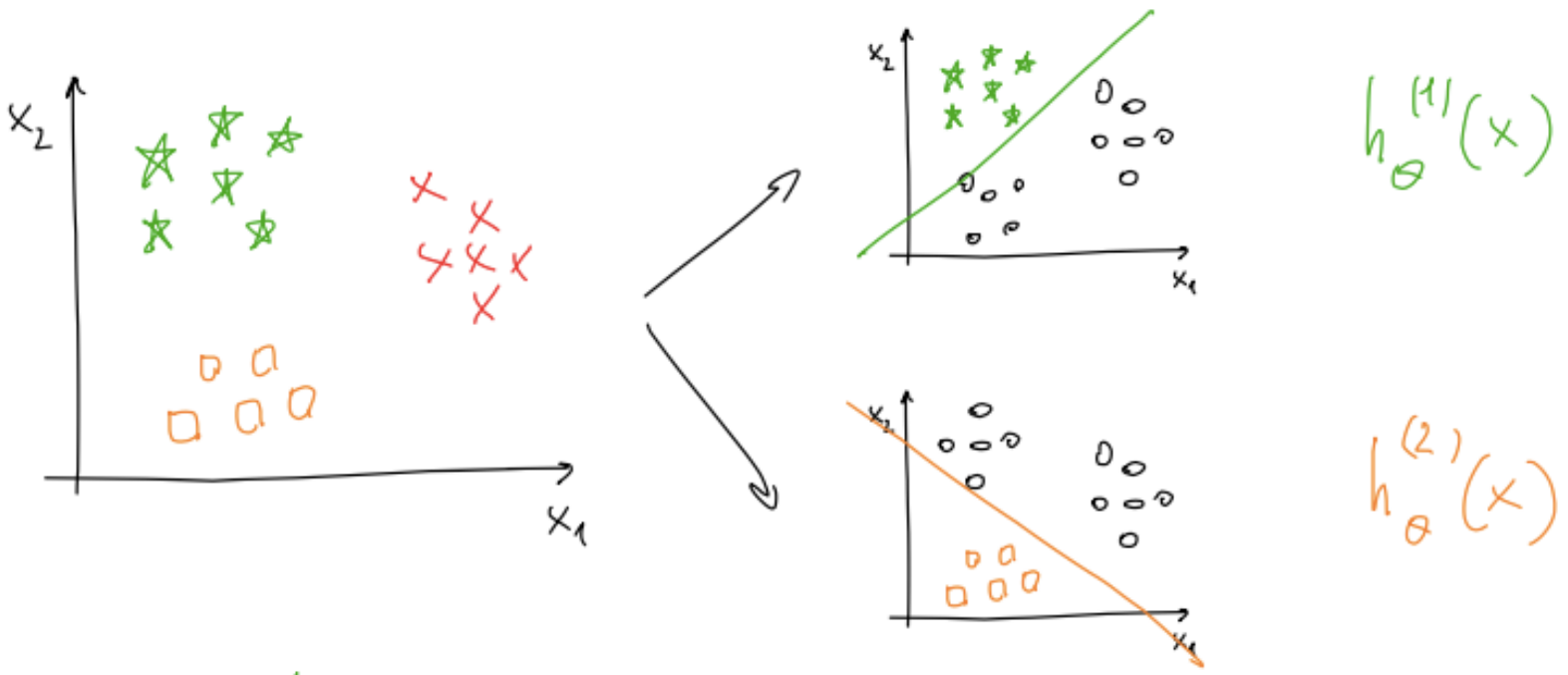
CLASS 3 ×

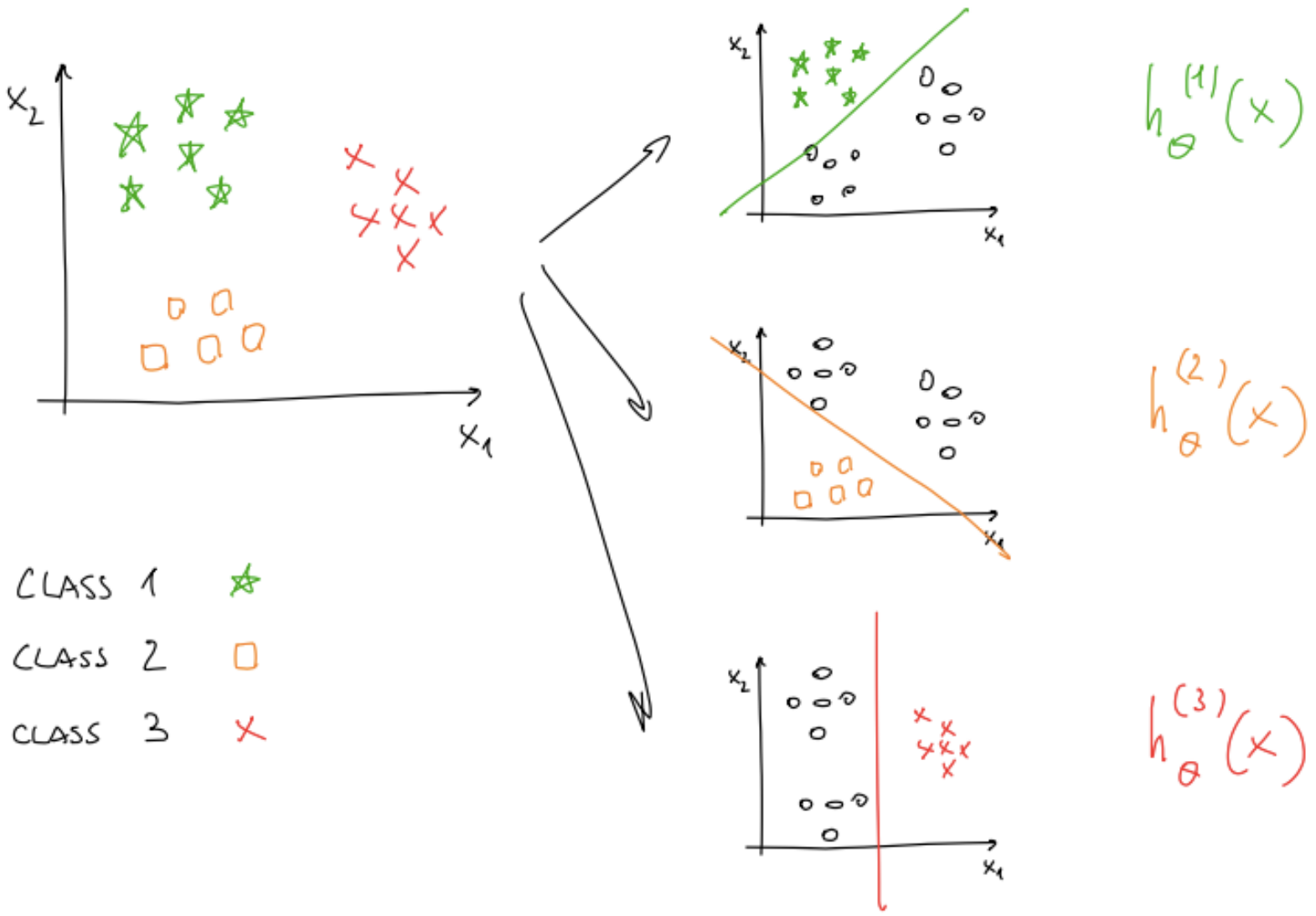


CLASS 1 ☆

CLASS 2 □

CLASS 3 X


x_2
x_1
$h_\theta^{(1)}(x)$
$h_\theta^{(2)}(x)$
CLASS 1
CLASS 2
CLASS 3


x_2
x_1
CLASS 1
CLASS 2
CLASS 3
$h_\theta^{(1)}(x)$
$h_\theta^{(2)}(x)$
$h_\theta^{(3)}(x)$

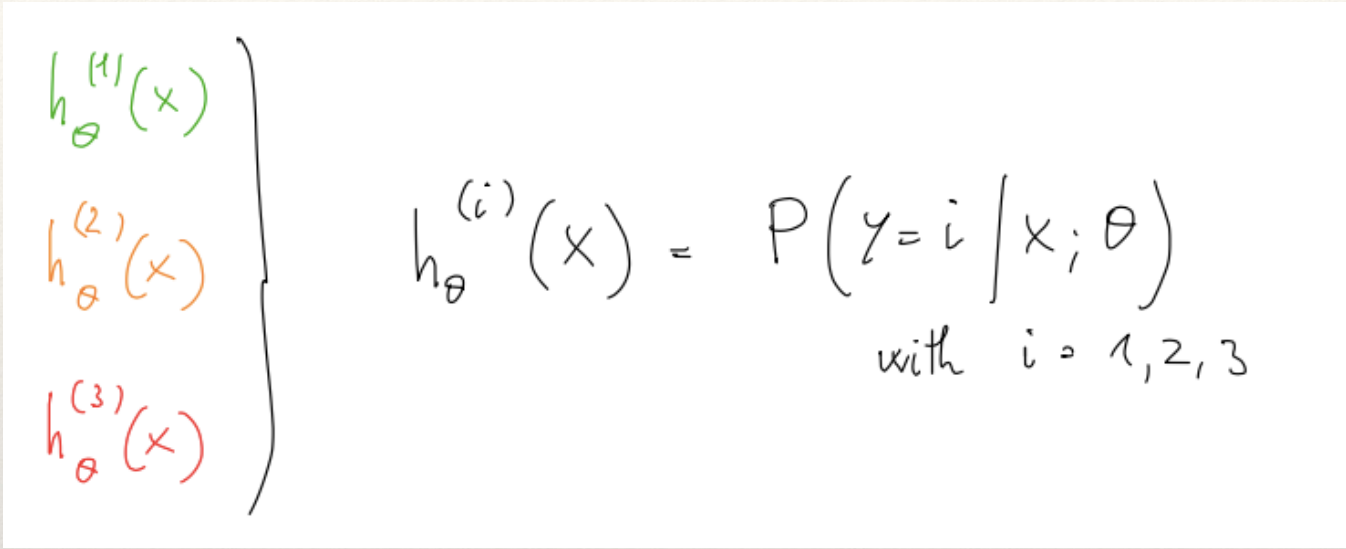
Almost ready to go…

In summary, we have built 3 classifiers.

1-vs-rest

2-vs-rest

3-vs-rest



Classifiers 1/2/3 are learning to recognise green/orange/red symbols w.r.t the rest, respectively

In words: for i = 1, 2, 3, we have fitted a classifier that estimates what is the probability that y is equal to class i, given x parametrised by θ

And now?

.. ready to classify

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y=i$.

e.g. 3 times

On a new input x, to make a prediction, run each classifier on the input and pick the class i that maximises:

$$\max_i h_\theta^{(i)}(x)$$

i.e. whichever i gives the highest probability, we predict y to be that value

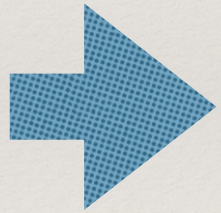
Summary

One-vs-all method for multi-class classification problems:

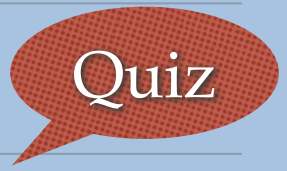
- train a logistic classifiers $h_\theta(x)$ for each class, to predict the probability that y=1
- to make a prediction on a new x, pick the class that maximises $h_\theta(x)$

With this method, you can now take the logistic regression classifier and make it work on multi-class classification problems as well.

Ready for Quiz 4



C.Y.U.



Question Q4

Suppose you have a multi-class classification problem with **k** classes, so **y**∈{1,2,..,k}. Using the 1-vs-all method, how many different logistic regression classifiers will you need to train?

1. k-1
2. k
3. k+1
4. 1


Logistic00Regression
Logistic10regression10model
Alternative10to10GD

We discussed about GD for minimising the cost function $J(\theta)$ for logistic regression.

There are alternative concepts (and consequently algorithms) - even more optimised than "basic" GD.

- In principle, using some of these one might be able to get logistic regression to run more quickly than it would be possible with GD.
- And this might also let the algorithm scale better to very large datasets and ML problems, i.e. very large number of features.

Different approaches to optimise $J(\theta)$

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$

GRADIENT DESCENT

CONJUGATE GRADIENT

BFGS

L-BFGS

A first glance to the zoo of algos for the GD task

(DISCLAIMER: details of each of these algo are beyond the scope of this course, but they are rarely quoted in an introductory course, so good to know at least these exist!)

Conjugate gradient: another iterative and search-direction based algo

- GD iteratively searches for a better minimum by looking in the gradient direction. Conjugate gradient is similar, but the search directions is differently specified (also required to be orthogonal to each other)

BFGS: Broyden–Fletcher–Goldfarb–Shanno. It is a quasi-Newtonian iterative algorithm to solve non-linear optimisation problems

- usable (instead of Newton-Raphson methods) when Jacobian/Hessian is not available or too complex to compute at every iteration

L-BFGS: L="limited-memory". It is a limited-mem version of BFGS

- particularly suited to problems with large (>thousands) numbers of variables

How to approach them in this course

They are optimisation algorithms too, namely different ways to perform the same task, sometimes with more sophisticated choice or implementations, with pros and cons..

- task is always the same, i.e. minimise $J(\theta)$, compute the derivatives..

Around ML theory, one may end up in spending days or weeks studying these algorithms (mainly, in advanced numerical computing courses)

In this course we will not dig into them

- NOTE: many experts admit they used these for years, and successfully, before actually admitting they got into their details only years after. It is actually entirely possible to use these algorithms successfully and apply to lots of different learning problems without actually understanding the inter-loop of what these algorithms do

.. and it will suffice to have a quick comparison between GD and not-GD (see next slide)

GD or not-GD, this is the question

The mentioned non-GD algos have some advantages w.r.t GD…

- with any of these you usually **do not need to manually pick a learning rate** α
  - they have a clever inter-loop called a "line search algo" that automatically tries out different values for α and automatically picks a good α so that it can even pick a different α for every iteration. No need to choose it yourself
- often **faster than GD**
  - implementation choices make them converge much faster than GD

.. and disadvantages:

- quite a lot **more complex than GD**
  - in particular, you probably should not implement these algorithms - conjugate gradient, L-BGFS, BFGS - yourself unless you're an expert in numerical computing. But you can use them!
- **hard to debug**
  - you're using a sophisticated optimisation library, this makes all a little bit more opaque and so harder to debug. But it could be a good trade-off for large ML problems as they run faster..

How to use them?

Just as one wouldn't recommend to write your own code to compute everything (down to the square roots of numbers.. or to compute inverses of matrices..), for these algorithms too what one should recommend is obvious: **just use existing software libraries**

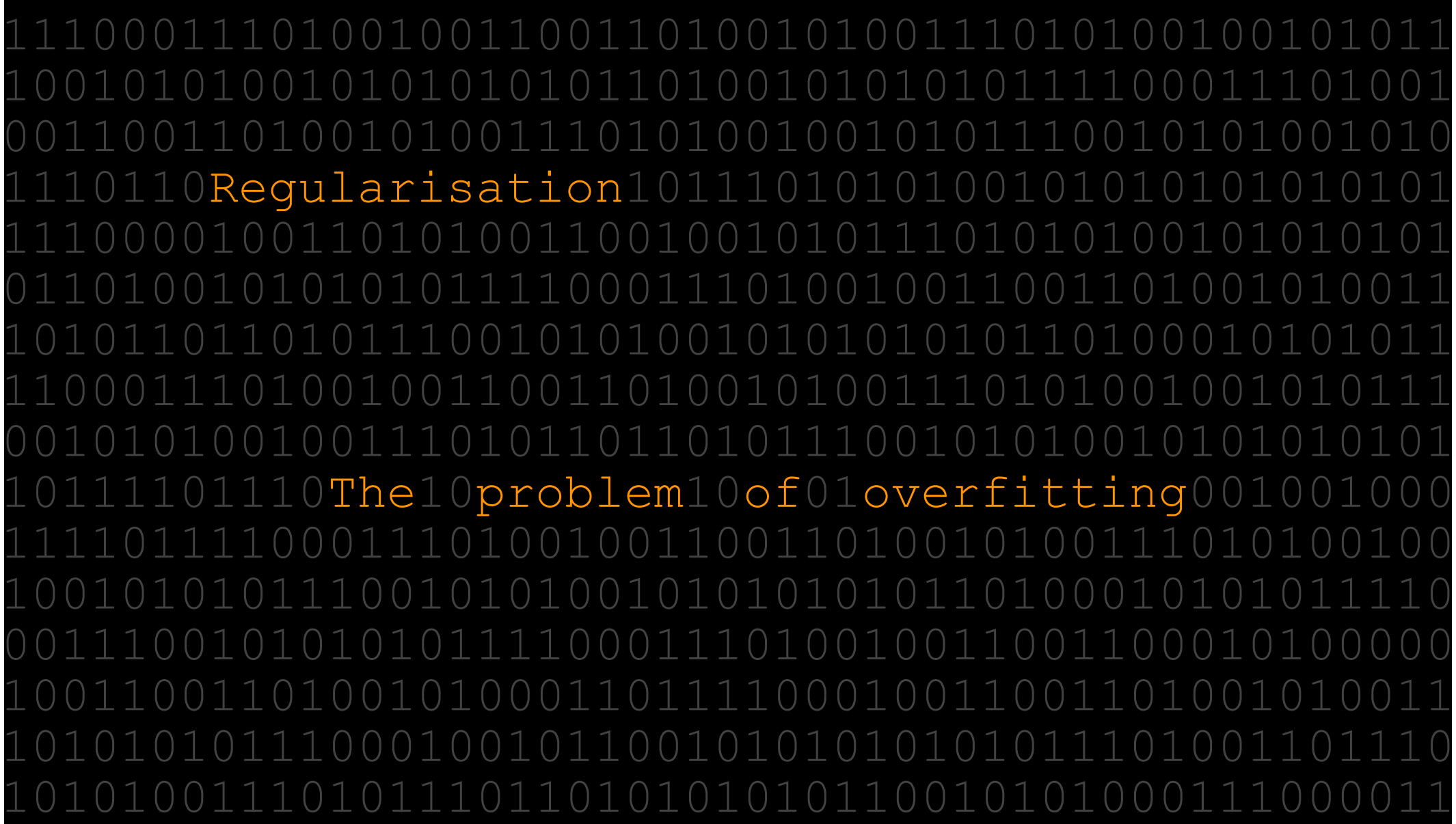
Caveats:

1. you do not find *everything* in *every* tool written in *every* language
2. you need a good algo implementation in the library of your choice
   - if you're using a given programming language for your ML application, you might want to try out a couple of different libraries to make sure that you find a good library implementing them
   - .. and newest versions sometimes exist, sometimes don't, some are better some aren't..

In a nutshell, **in applied ML a lot depends on what tools you use**.

- but be reassured: **(variations of) GD itself, in a Python ecosystem, will suffice all your applied ML needs for a pretty long time**


Regularisation


Regularisation
The problem of overfitting

Overfitting

We have discussed 2 different learning algos:

- linear regression
- logistic regression

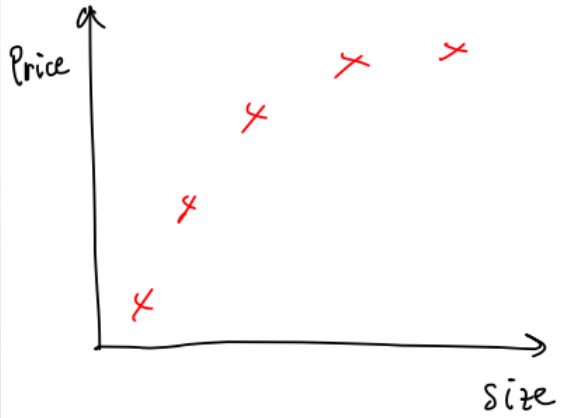
They work well for many problems. It might happen that when you apply them to certain ML applications of yours, they run into a problem called **overfitting**, with the consequence to perform very poorly.

Let's discuss what overfitting is, and what is a technique called **regularisation**, to ameliorate/reduce the overfitting problem

- *the idea of covering this topic as follows - as we did for GD - is to have some intuitive but solid way to really understand how regularisation acts!*

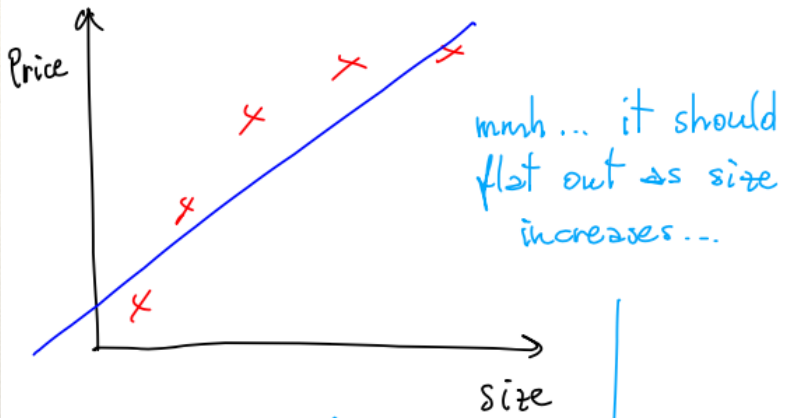
Concept of **overfitting** for linear regression

Example : predicting housing prices with linear regression as a function of the house size



Concept of **overfitting** for linear regression

Example : predicting housing prices with linear regression as a function of the house size



mmh... it should flat out as size increases...

fit a linear function:

$\theta_0 + \theta_1 x$

this algo doesn't fit the training set very well

"UNDERFITTING"

or

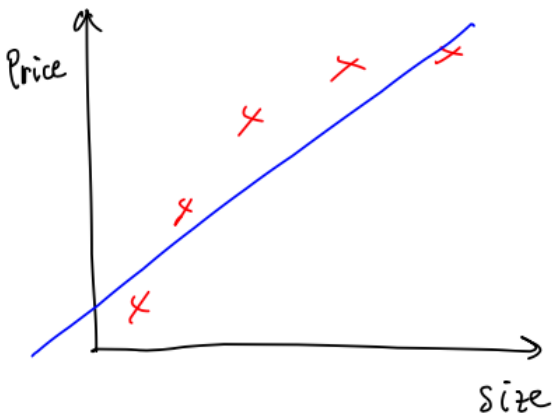
"HIGH-BIAS"

historical

"very strong preconception" despite what the data actually says

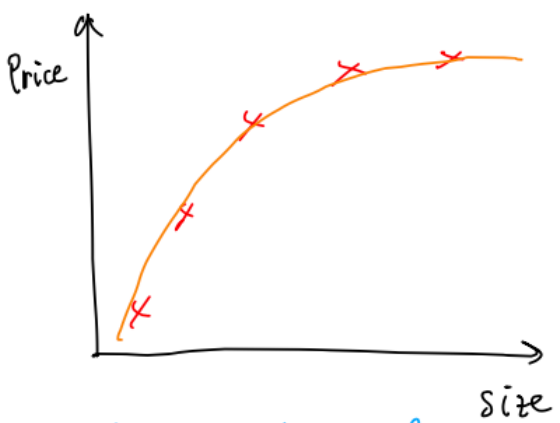
Concept of **overfitting** for linear regression

Example : predicting housing prices with linear regression as a function of the house size



$\theta_0 + \theta_1 x$

"UNDERFITTING"
or
"HIGH-BIAS"



fit a quadratic function:

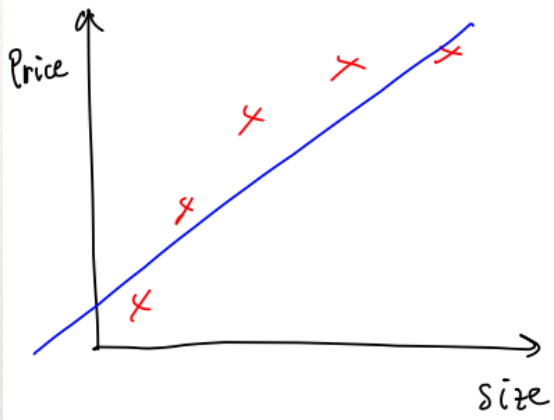
$\theta_0 + \theta_1 x + \theta_2 x^2$

it fits quite well (in this range)

"JUST RIGHT"

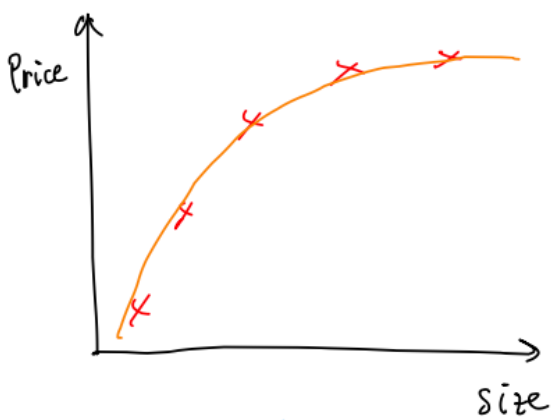
Concept of **overfitting** for linear regression

Example : predicting housing prices with linear regression as a function of the house size



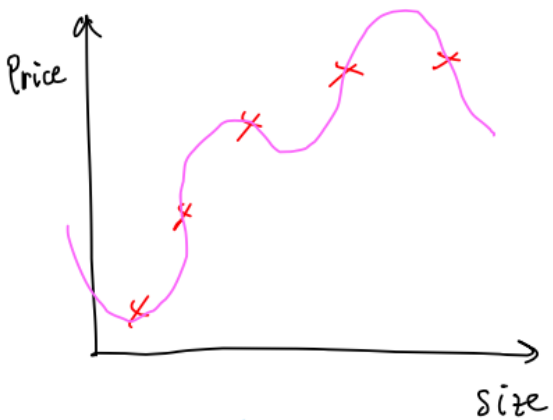
$\theta_0 + \theta_1 x$

"UNDERFITTING" or "HIGH-BIAS"



$\theta_0 + \theta_1 x + \theta_2 x^2$

"JUST RIGHT"



fit a 4th order polynomial

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

it "seems" to do a good job, but this is NOT a good model for our predictions! (e.g. Too many ups and downs!)

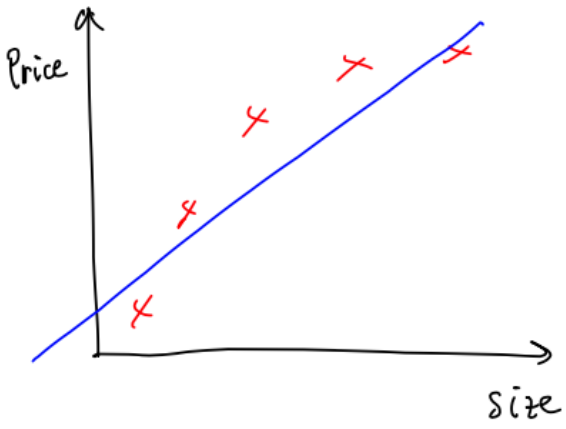
"OVERFITTING" or "HIGH-VARIANCE"

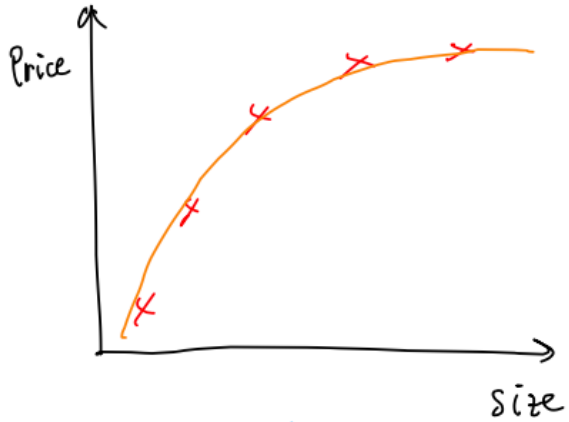
high-order polynomials : you can fit almost ANY function. The space of possible hypotheses is too large, too variable!

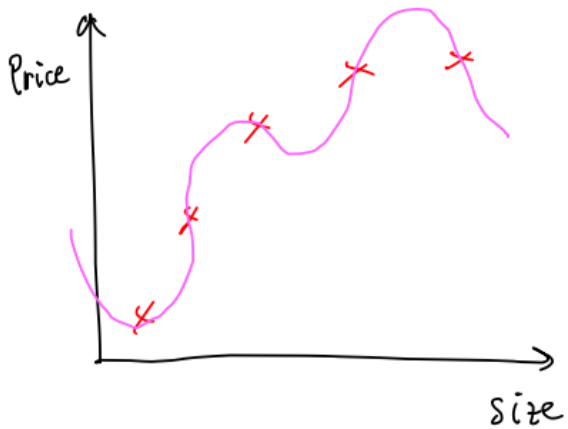
historical

Concept of **overfitting** for linear regression

Example : predicting housing prices with linear regression as a function of the house size







$\theta_0 + \theta_1 x$

"UNDERFITTING" or "HIGH-BIAS"

$\theta_0 + \theta_1 x + \theta_2 x^2$

"JUST RIGHT"

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"OVERFITTING" or "HIGH-VARIANCE"

Overfitting and lack of generalisation

The problem of **overfitting**: if we have too many features, the learned hypothesis may fit the training set too well

- the cost function $J(\theta)$ may be very close to 0, even exactly 0!

You may end up with a curve that tries too hard to fit the training set, so that **it eventually fails to generalise to new examples**

- e.g. in our example: it would fail to predict prices on new houses
- The "**generalisation**" here refers to *how well a hypothesis applies to new examples that you have not ever seen in your training set*

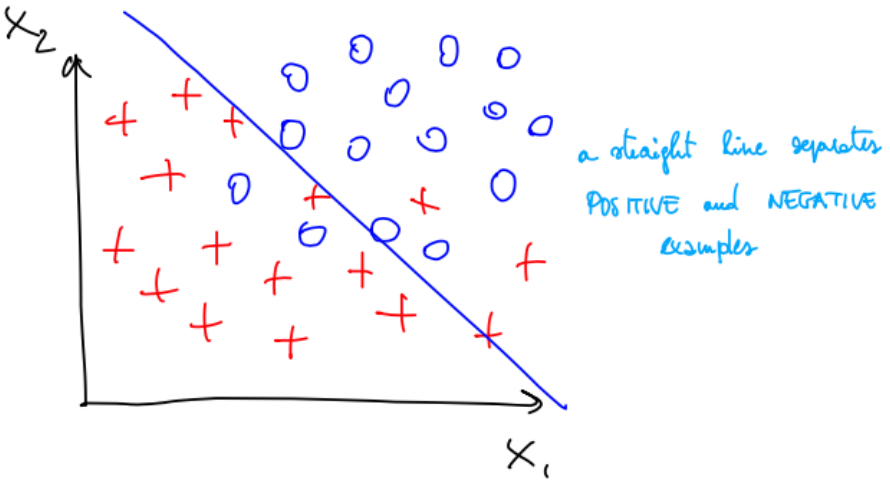
Food for thoughts: **optimisation vs generalisation** in ML

This is overfitting for linear regression.

It similarly applies to logistic regression.

Concept of **overfitting** for logistic regression

Example : logistic regression example , 2 features (x_1, x_2)



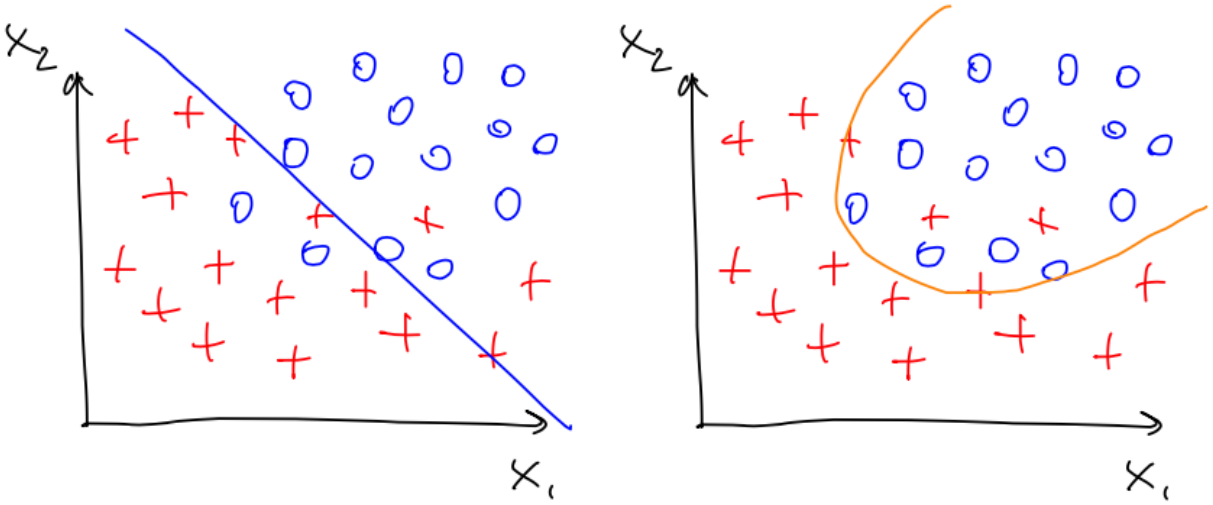
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

g ↓ sigmoid

UNDERFITTING

Concept of **overfitting** for linear regression

Example : logistic regression example, 2 features (x_1, x_2)



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

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

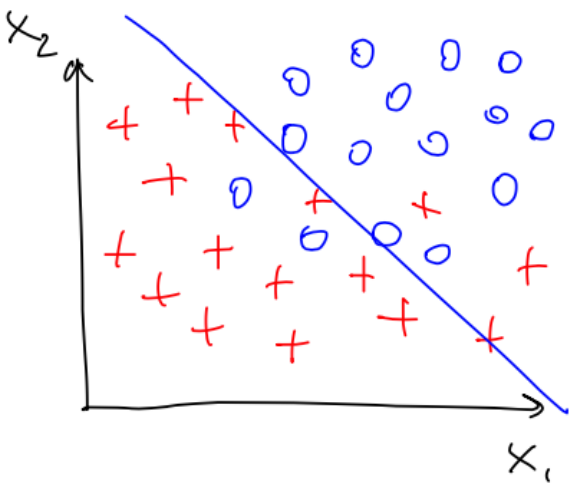
add quadratic terms →

UNDERFITTING

pretty good ! (as good as you could get ?)

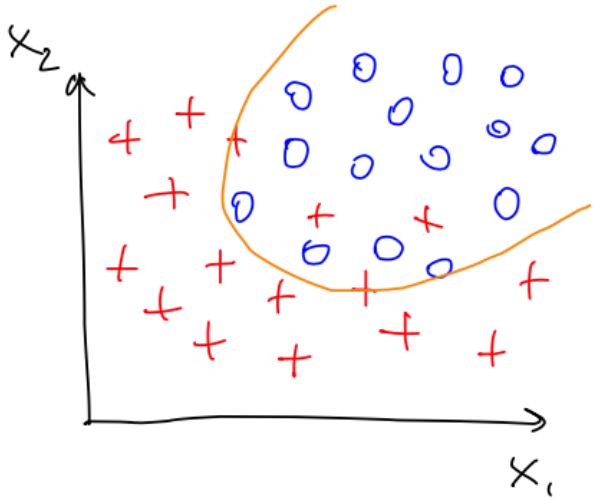
Concept of **overfitting** for linear regression

Example : logistic regression example , 2 features (x_1, x_2)



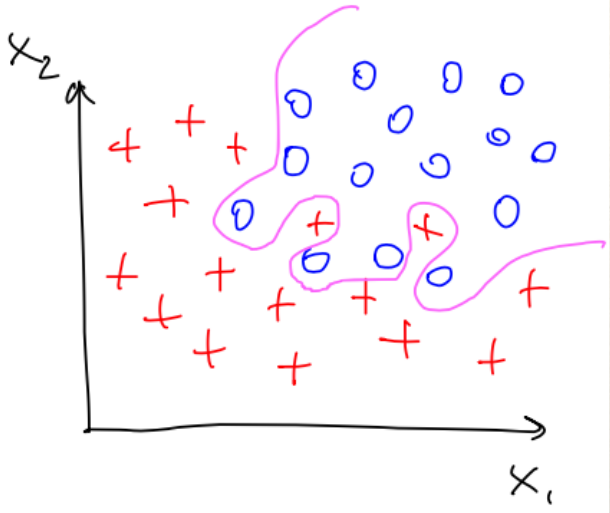
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

UNDERFITTING



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

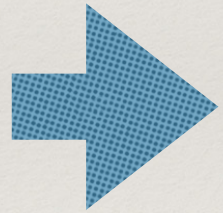
ok



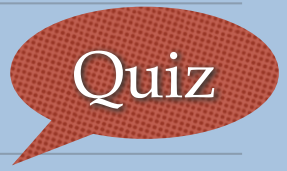
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \ldots)$

OVERFITTING

Ready for Quiz 5



C.Y.U.



Question Q5

Suppose you are facing the medical diagnosis problem of telling if a tutor is malignant or benign, and the hypothesis $h_\theta(x)$ you built is overfitting the training set. This means that:

1. it makes accurate predictions on examples from your training set, and it generalises well to make accurate predictions on new, formerly unseen examples
2. it does NOT make accurate predictions on examples from your training set, but it generalises well to make accurate predictions on new, formerly unseen examples
3. it makes accurate predictions on examples from your training set, but it does NOT generalise well to make accurate predictions on new, unseen examples
4. it does NOT make accurate predictions on examples from your training set, and does NOT generalise well to make accurate predictions on new, unseen examples

The curse of overfitting

Naively:

- "why should I worry? I just plot hypotheses and use common sense..". Plot the *h*, see what's going on, select the appropriate degree polynomial (even high orders), check decision boundaries, ..

Not so simple.

- you might have/need **a lot of features**: all what we said might work for 1/2-dim data, but if many features you will definitely not find help in 3D dataviz!
  - e.g. same (housing) example, but with size, # bedrooms, # floors, age of house, average income in neighbourhood, size of the kitchen, presence of garage or not, etc
- also, you might have a **small training dataset with just few examples** (overfitting will be your curse..)

When we have many features, and perhaps relatively little training data, then **over-fitting can become a serious problem**.

How to attack it?

How to address overfitting

Basically, 2 main options:

1. **reduce the number of features**
2. **tune the importance of features**

We will go through both

- NOTE: these are basic ideas and techniques. More refined approaches exist.

Fight overfitting: # of features

First option: **reduce the number of features**

Manually look through the list of features, and select which ones to keep, drop the others

This option comes with a major drawback: you risk to loose useful information

- throwing away some features = throwing away (potentially rich) info about "the problem"
  - those features might have been rich in describing the pattern you would have wanted to highlight in your data-driven model..

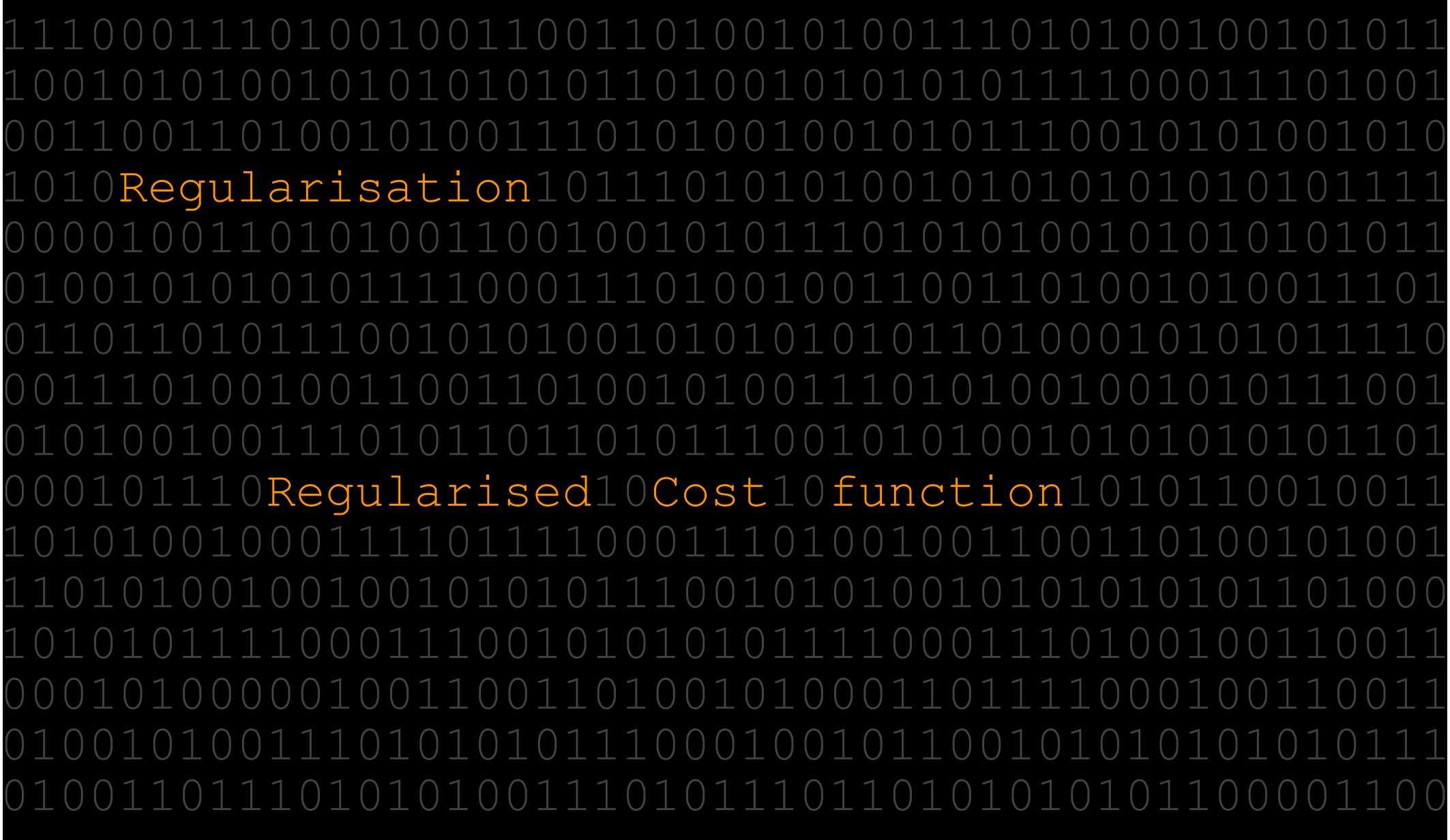
Fight overfitting: importance of features

Second option: **tune the <u>importance of features</u>**

This is usually referred to as the **"regularisation"** process.

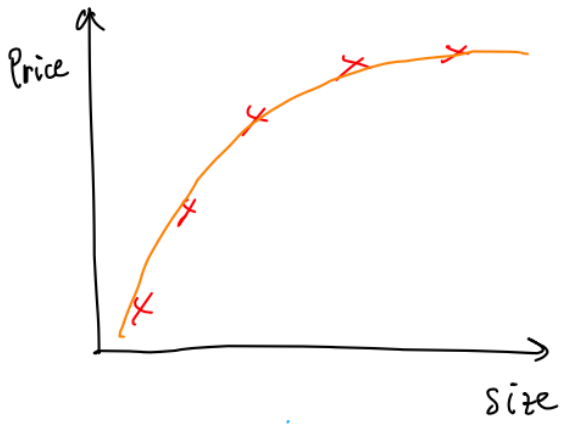
- we are going to *keep all the features*, but (*neat trick!*) tune the magnitude of the parameters θ_j
  - perfectly suited method when we have a lot of features, each of which contributes a little bit to predicting the value of y, and we can fine tune how much

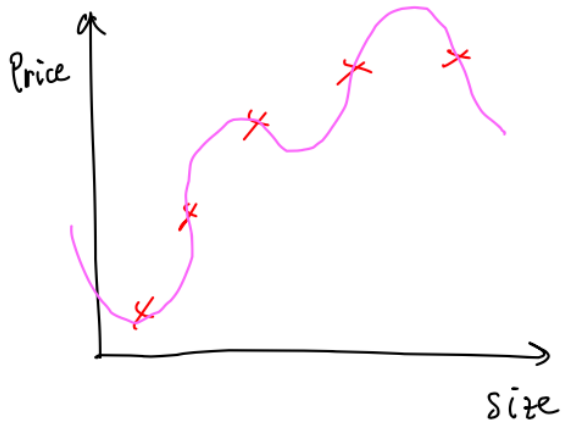
The advantage of this second option is that you do not need to throw away any feature


Regularisation
Regularised Cost function

Let's see how to apply **regularisation** - i.e. reduce the magnitude of some of the parameters θ_j - to address **overfitting**.

We start with a linear regression case.



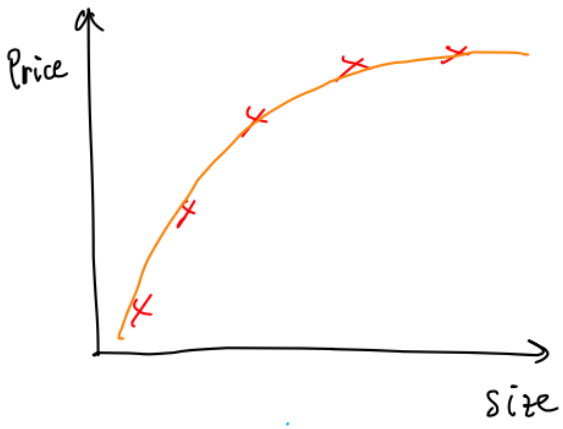


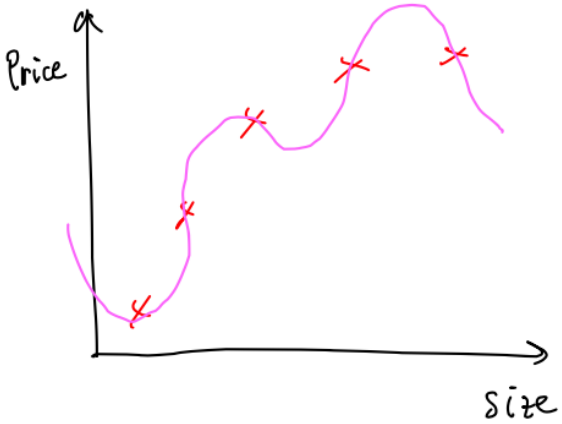
$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize θ_3, θ_4 by making them very small

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



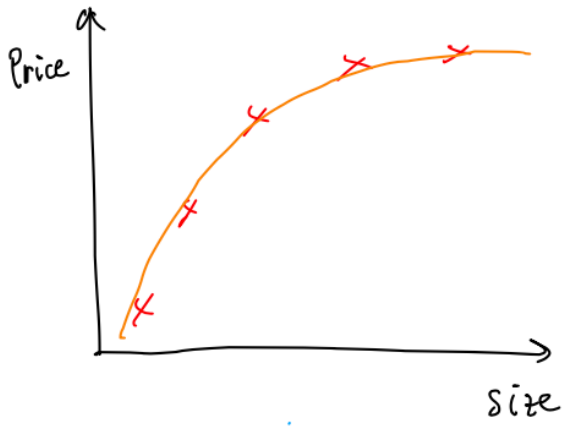


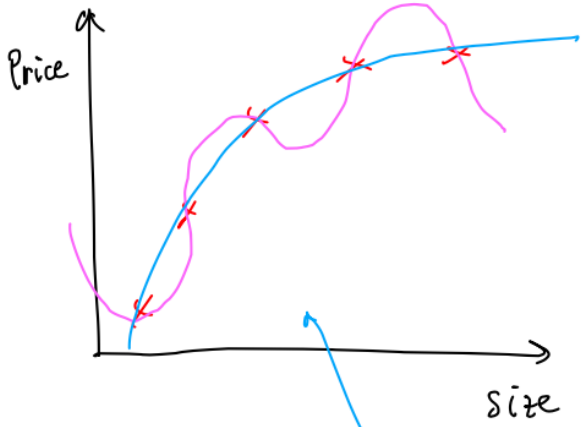
$\theta_0 + \theta_1 x + \theta_2 x^2$ $\qquad$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize θ_3, θ_4 by making them very small

$$\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + 1000\, \theta_3^2 + 1000\, \theta_4^2$$

The only way, now, to $\min_\theta$ ↑ is if θ_3, θ_4 are SMALL





DONE !

$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ (or tiny contributions)

Suppose we penalize θ_3, θ_4 by making them very small

$$\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + 1000\, \theta_3^2 + 1000\, \theta_4^2$$

The only way, now, to $\min_\theta$ ↑ is if θ_3, θ_4 are SMALL

$\Rightarrow$ minimization gives $\theta_3 \approx 0$, $\theta_4 \approx 0$

Regularised cost function

Suppose :

features : $x_1, x_2, \dots, x_{500}$

parameters : $\theta_0, \theta_1, \dots, \theta_{500}$

which are the θs that are less likely to be relevant ?!?

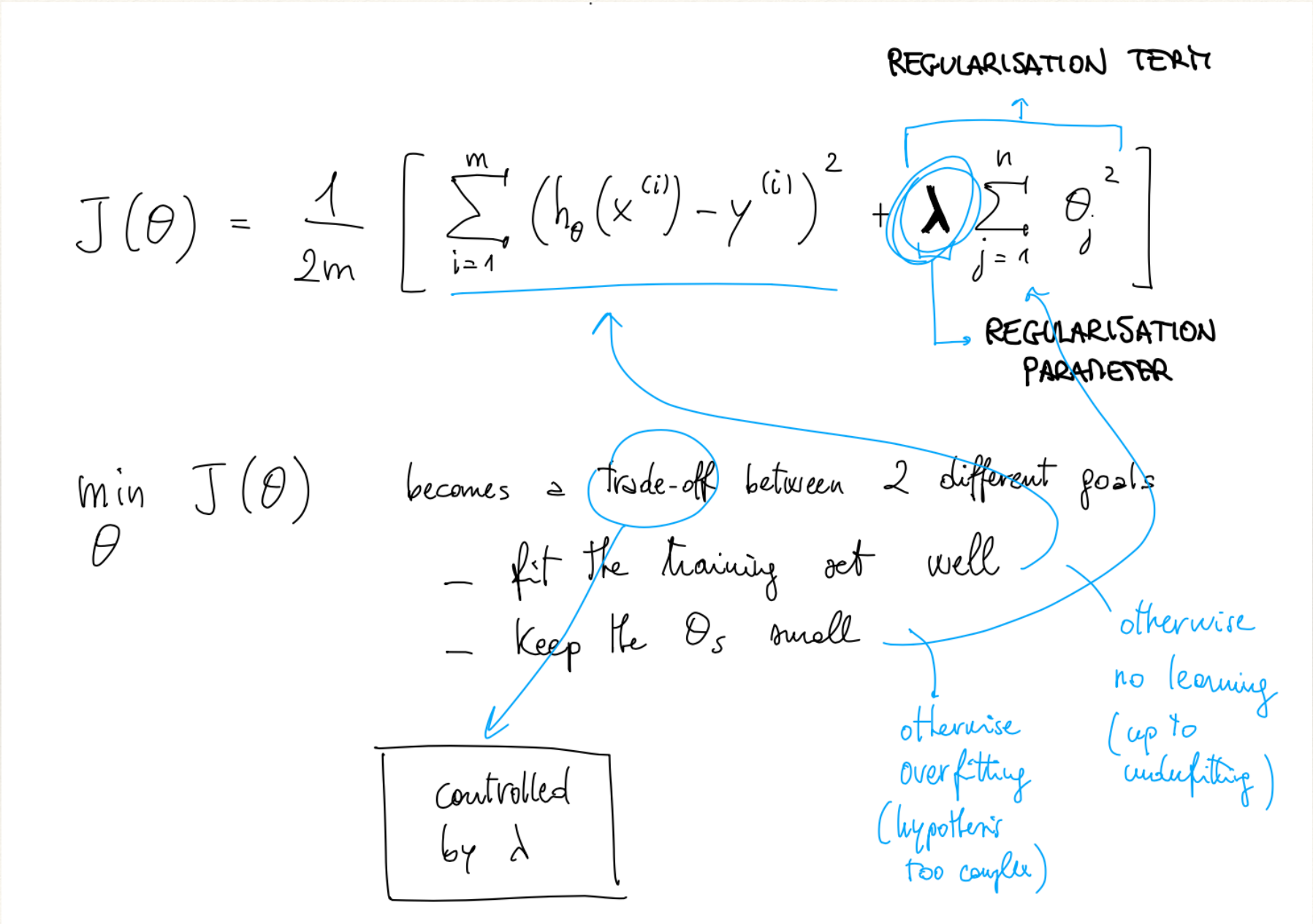
Regularization modifies the cost function : to shrink all params !!!

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

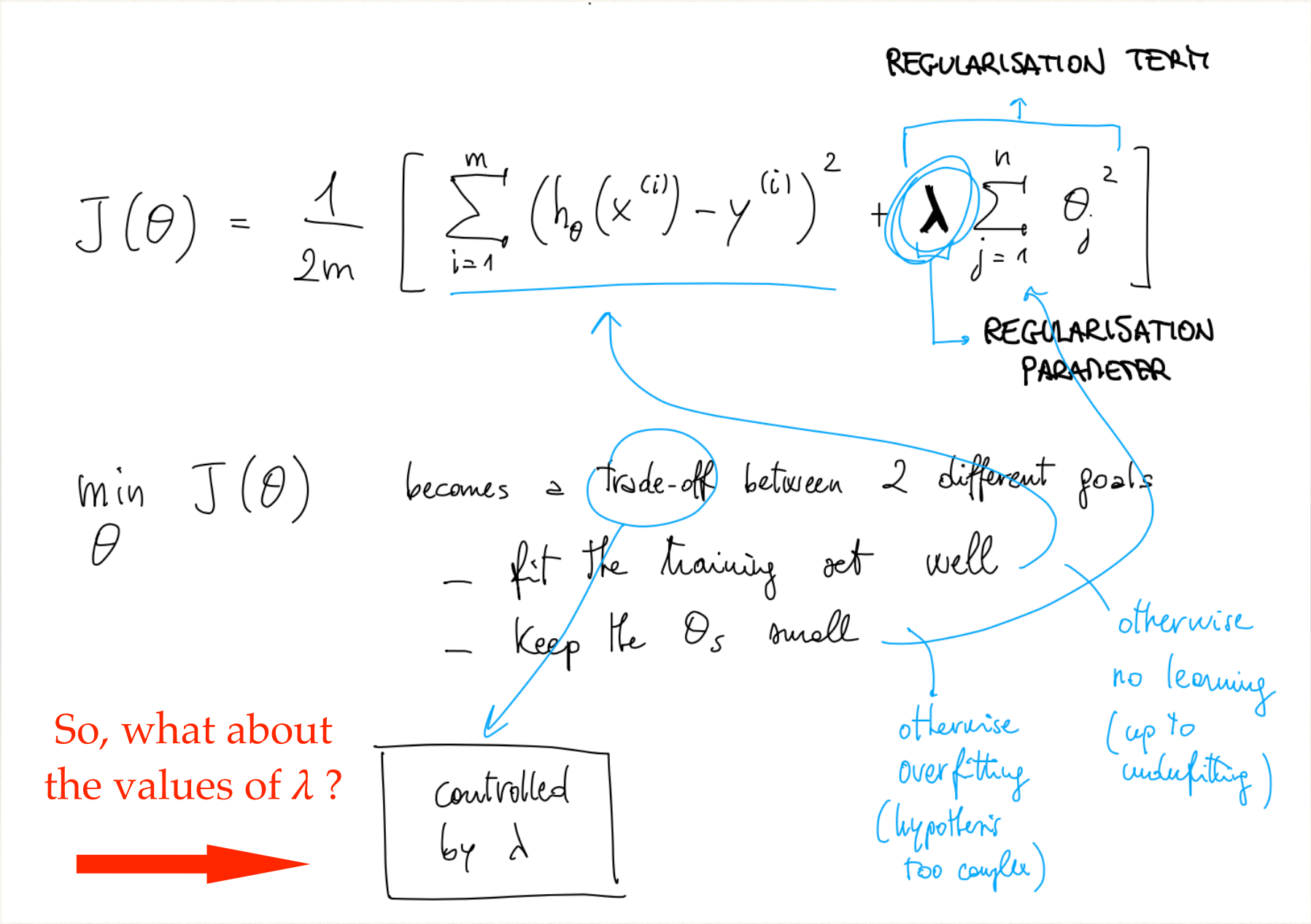
(*) NOTE : $\sum_{j=1}^{n}$: no penalization on θ_0 (more a convention..)

a REGULARISATION TERM to shrink ALL θs (*)

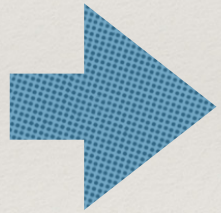
λ: a new hyper-parameter



λ: a new hyper-parameter



Ready for Quiz 6



C.Y.U.

Quiz

Question Q6

I am applying regularised linear regression, and I choose a very high value of λ (perhaps too large for our problem, e.g. 10^{10}). What happens?

1. gradient descent will fail to converge
2. the algorithm will cause underfitting
3. the algorithm will fail to eliminate overfitting, but I am safe against underfitting
4. the algorithm will be very aggressive and should be monitored closely, but it will eventually eliminate overfitting with no major drawbacks

The quiz, discussed..

what if λ

too small ? => no regularization → overfitting

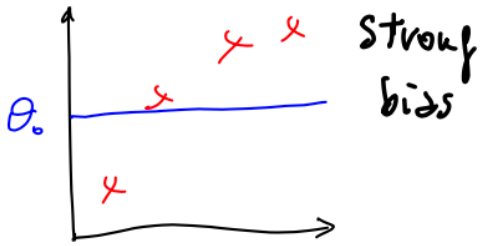
too large ?

e.g. $\lambda \sim 10^{10}$ => penalize all θ_s

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots$ ⇒ $\theta_1 \approx 0$, $\theta_2 \approx 0$, $\theta_3 \approx 0$, $\theta_4 \approx 0$ ⇒ $h_\theta(x) = \theta_0$

UNDERFITTING

of course you kept the θ_s small, but at what price !!!



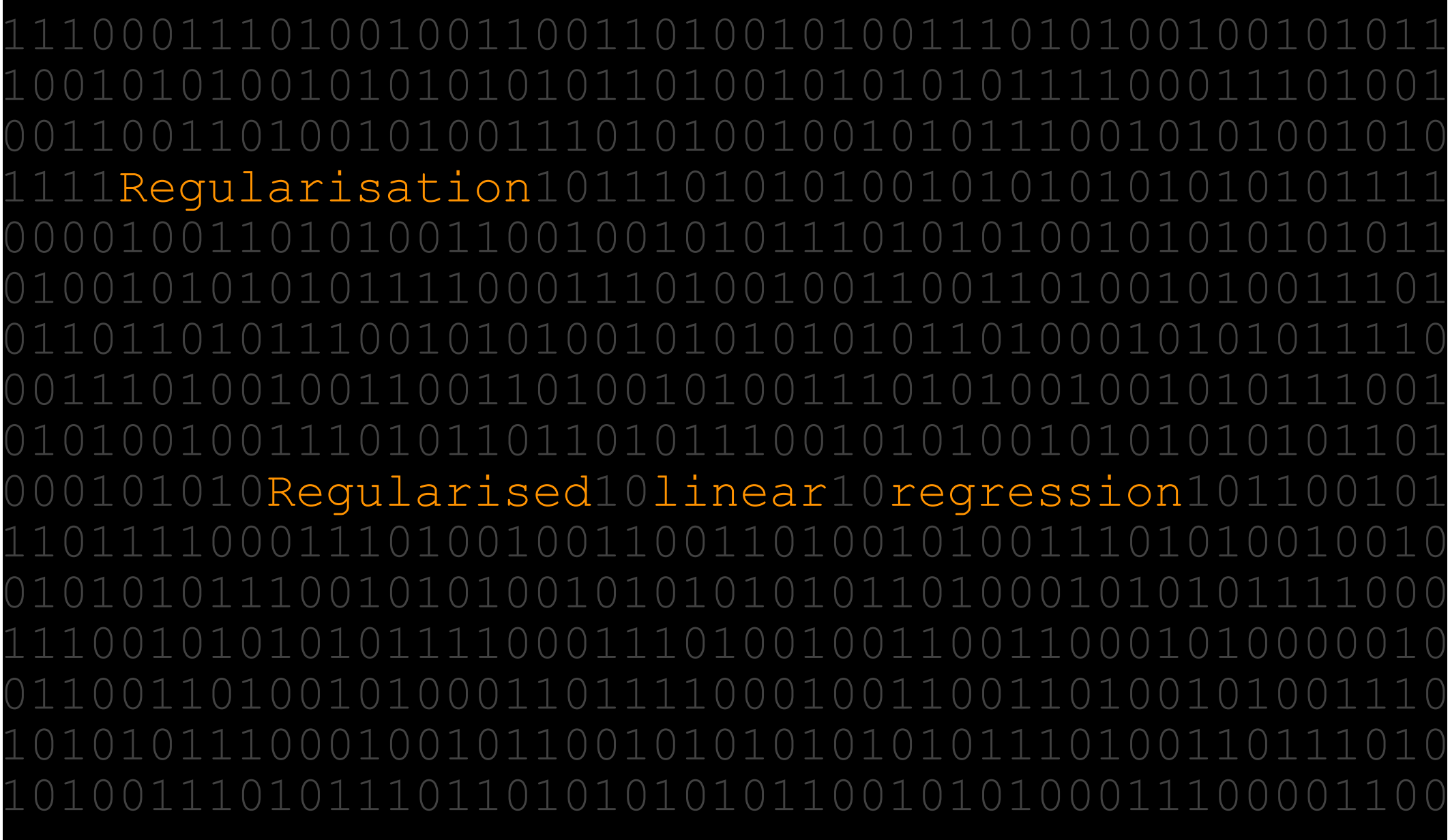
Summary

We discussed the **cost function** in case of **regularisation**, in the example of linear regression.

More generally and simply, the idea behind regularisation - that may explain (even by common sense) why it works - can be expressed as follows, by thinking the entire process viceversa:

*If you take small values for the θ parameters, this will usually correspond to having a simpler hypothesis and smoother functions. Which are therefore, also, less prone to overfitting. But, on the other hand, underfitting is not an option: fine-tune your regularisation and it will increase the **weights** only for the actually important features.*

Slowly, we will stop calling them the θ parameters..


Regularisation
Regularised linear regression

For **linear regression**, we have previously worked out two learning algorithms

- one based on **GD**
- one based on the **normal equation**

We now take those two algorithms and generalise them to the case of **regularised linear regression**

- *admittedly: we will be brief for GD, and even more brief for the normal equation. Namely, just a glance (and no summary of concepts at the end of this section)*

Starting point :

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 - \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

our usual objective

regularization term

$$\min_\theta J(\theta)$$

← find θs that minimize $J(\theta)$

GD: for regular linear regression without Regul.

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$(j = 0, 1, \ldots, n)$

} ← update the θ_j

Separate out the θ_0 part:

(remember we did not penalize θ_0 in regularization, so let's separate θ_0 and $\theta_1, \dots, \theta_n$)

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$$(j = 0, 1, \dots, n)$$

}

All the same as before

Modify it as follows :

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

$$(j = 0, 1, \ldots, n)$$

}

This is a GD algo for regularized linear regression, that tries to minimize the regularized cost function $J(\theta)$.

Modify it as follows:

Repeat {

$$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)} \right] = \frac{\partial}{\partial \theta_0} J(\theta)$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

$(j = \cancel{0}, 1, \dots, n)$

}

(not dimostrated) $= \frac{\partial}{\partial \theta_j} J(\theta)$ → regularized

This is a GD algo for regularized linear regression, that tries to minimize the regularized cost function $J(\theta)$.

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

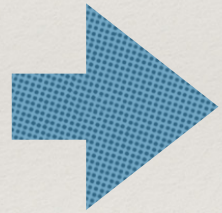
}

can be re-written as

$$\theta_j := \theta_j \boxed{\left(1 - \alpha \frac{\lambda}{m}\right)} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

interesting ...

Ready for Quiz 7



C.Y.U.

Quiz

Question Q7

Suppose you are doing GD on a large training set of **m** examples, using a fairly small learning rate **α** and some regularisation parameter λ. In the θ_j update rule of GD, which of the following is true?

1. $(1-\alpha\lambda/m) > 1$
2. $(1-\alpha\lambda/m) = 1$
3. $(1-\alpha\lambda/m) < 1$
4. the θ_j update rule is not defined

The quiz, discussed..

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$1 - \alpha \frac{\lambda}{m} < 1$ (e.g. 0.99)

α: small; m: large; $\frac{\lambda}{m} > 0$

shrinking θ_j

exactly the same as original GD (pre-regularisation)

↳ on every iteration, GD for regularized linear regression shrinks θ_j and does the same as before to converge.

[note: this is intuition ... mathematically, it is GD on a changed $J(\theta)$]

GD was just one of our 2 algorithms for fitting a linear regression model.

The second algorithm was the one based on the **normal equation**, where what we did was we created the design matrix X.

How would regularisation impact the normal equation method?

normal equation (w/o regularization)

$$X = \begin{pmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix}$$

$(x^{(1)})^T$: a training example

$m \times (n+1)$ matrix

$$y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

m-dim vector ($\mathbb{R}^m$)

contains the label for the training set

How to $\min_\theta J(\theta)$? $\longrightarrow$ $\theta = (X^T X)^{-1} X^T y$

$\hookrightarrow$ $\frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{=} 0 \rightarrow$ (... some math ...)

normal equation (w regularization)

$$X = \begin{pmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix} \qquad y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

$$\min_{\theta} J(\theta) \; ?$$

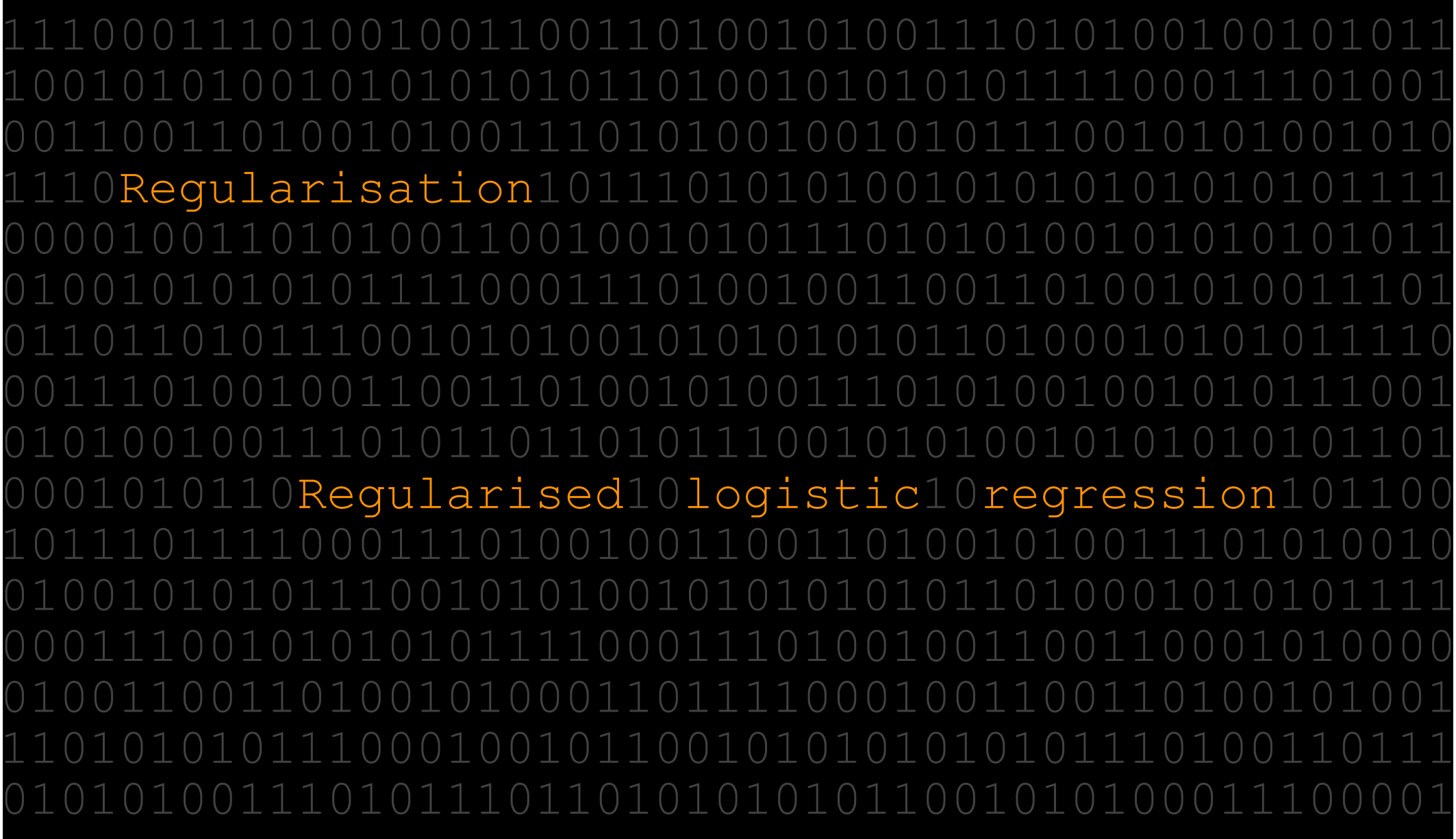
$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

$(n+1) \times (n+1)$ matrix

in addition, while $X^T X$ has invertibility issues, the additional term makes it non-singular!

(no derivation)

e.g. $n = 2 \Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$


Regularisation
Regularised logistic regression

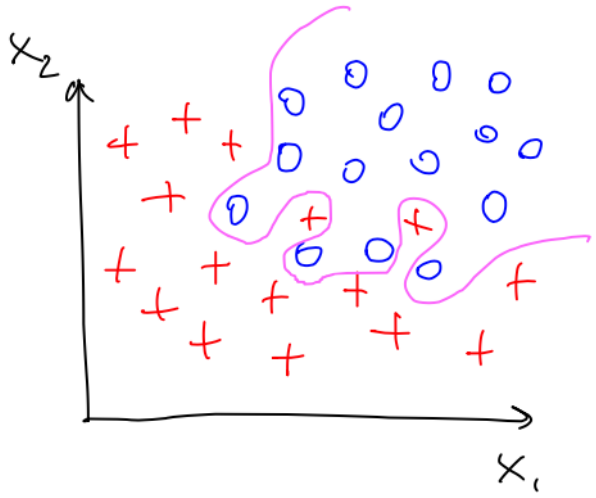
For **logistic regression**, we previously talked mostly about one algorithm:

- GD

We can adapt any technique above and make them work for **regularised logistic regression**.

- *again: very brief and just intuitive and informative..*

Starting point: **logistic regression** (w/o regularization)



$$h_\theta(x) = g\left(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots\right)$$

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

$$\text{cost}\left(h_\theta(x), y\right) = -y \log\left(h_\theta(x)\right) - (1-y) \log\left(1 - h_\theta(x)\right)$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) \right]$$

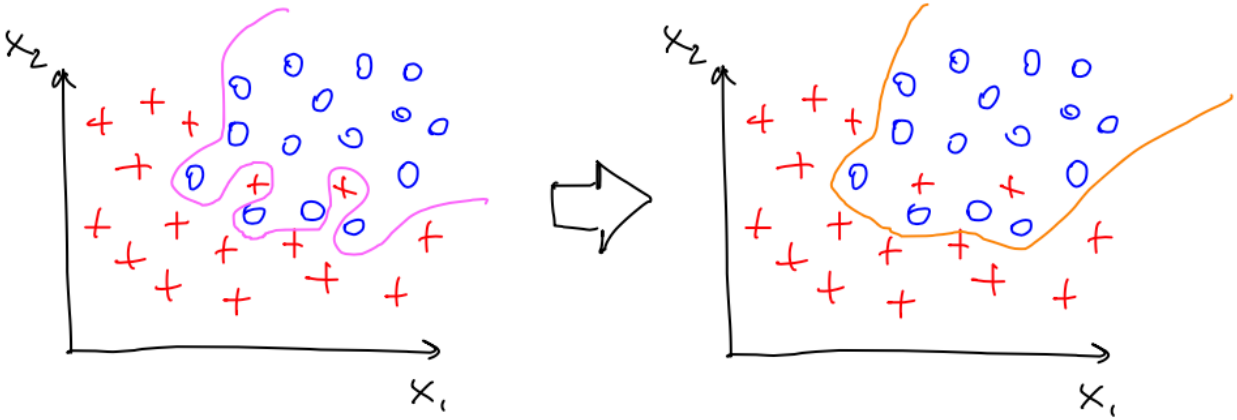
Starting point: logistic regression (w regularization)

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

$$+\underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

it penalizes $\theta_1, \dots, \theta_n$ from being too large

more reasonable!



How to implement it?

GD: Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$(j = 0, 1, \dots, n)$

}

for regular logistic regression without Regul.

update the θ_j

GD: Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$$(j = 0, 1, \dots, n)$$

}

for regular logistic regression without Regul.

↑ update the θ_j

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$$(j = \cancel{0}, 1, \dots, n)$$

}

GD: Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$(j = 0, 1, \dots, n)$

} update the θ_j

for regular logistic regression without Regul.

⇩

and this is GD for regularized logistic regression

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

$(j = 1, \dots, n)$

}

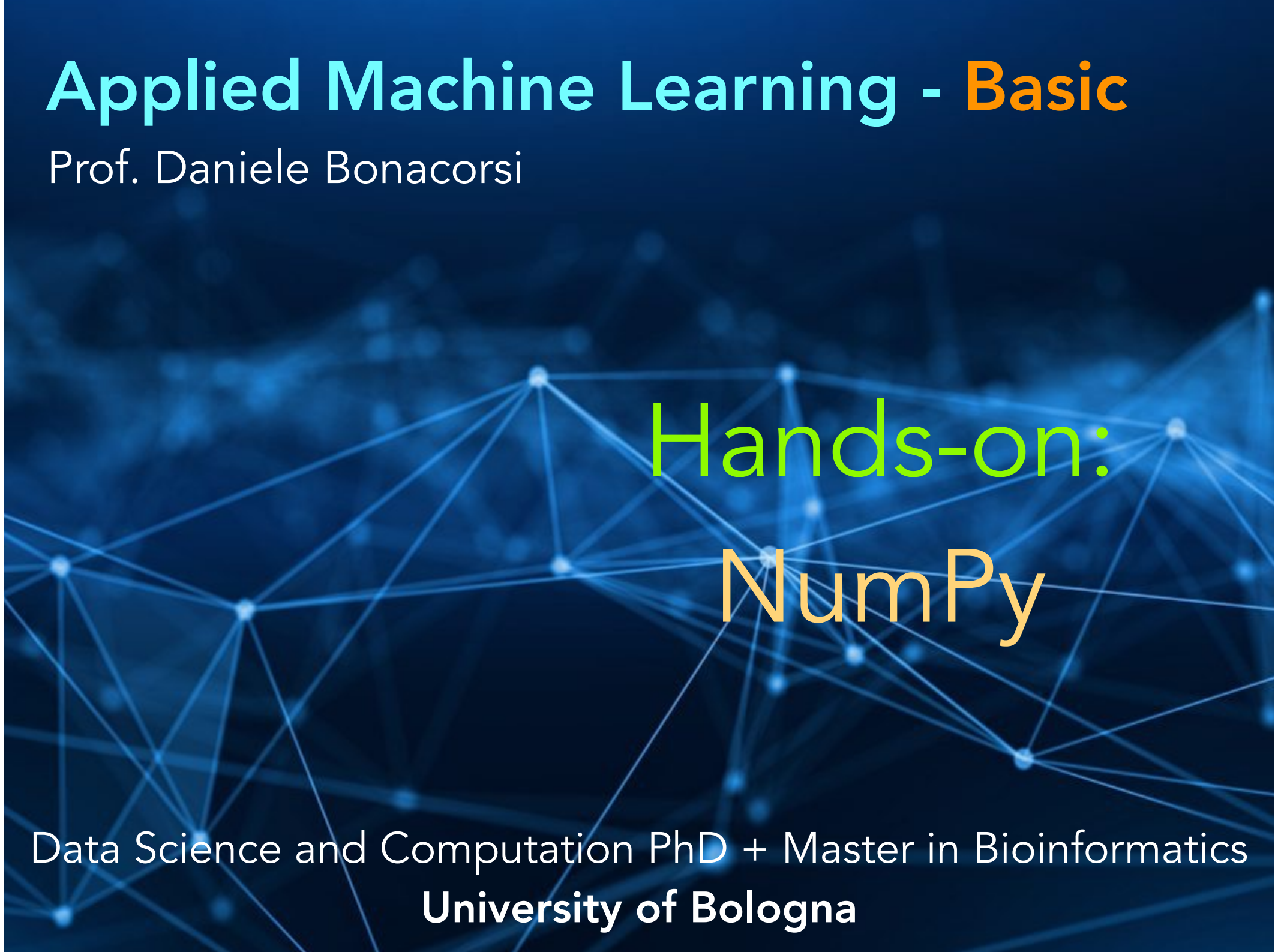
$\frac{\partial}{\partial \theta_j} J(\theta)$ the one that use regularization

as for linear regression!

NOTE: cosmetically identical, but now $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi



Hands-on: NumPy

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Python light overview:

some useful packages → **NumPy**

DISCLAIMER: “This is not a programming course”, remember?

NumPy

https://numpy.org/

A third-party package added to Python to support scientific computing

- in particular, it provides you with **multi-dimensional array objects**
  - i.e. support matrix manipulation, linear algebra, all operations you might want to do on large collection of numbers (e.g. plenty in ML!)

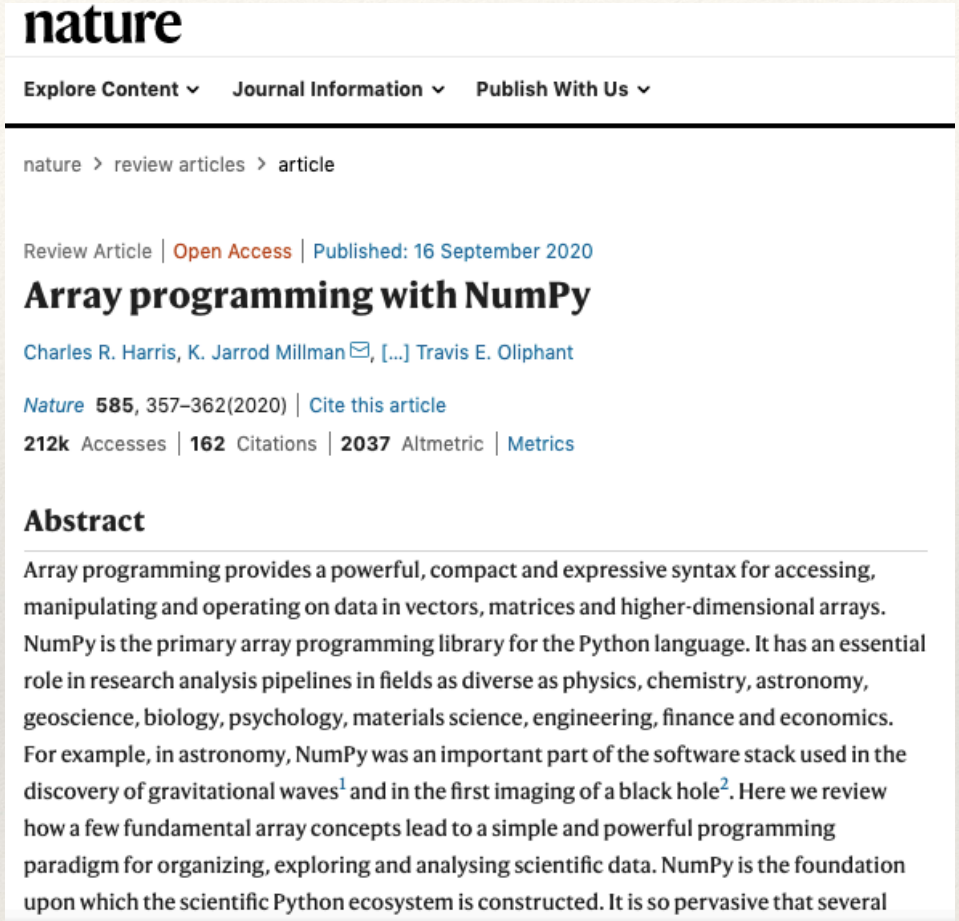
Official NumPy paper

The NumPy paper on **Nature**

- https://www.nature.com/articles/s41586-020-2649-2

To be formally cited as:

- Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* **585,** 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2



nature

Explore Content | Journal Information | Publish With Us

nature > review articles > article

Review Article | Open Access | Published: 16 September 2020

Array programming with NumPy

Charles R. Harris, K. Jarrod Millman, [...] Travis E. Oliphant

Nature **585**, 357–362(2020) | Cite this article

212k Accesses | **162** Citations | **2037** Altmetric | Metrics

Abstract

Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. For example, in astronomy, NumPy was an important part of the software stack used in the discovery of gravitational waves[1] and in the first imaging of a black hole[2]. Here we review how a few fundamental array concepts lead to a simple and powerful programming paradigm for organizing, exploring and analysing scientific data. NumPy is the foundation upon which the scientific Python ecosystem is constructed. It is so pervasive that several

Not to be discussed deeply in this course, but you should be aware of the gender unbalance controversy behind this paper. More details at the lecture.

NumPy 1D array

```
import numpy as np
```

Just a convention, but almost always followed

```
a = np.array( [1, 2, 3] )
```

1-D array object

```
print(a)
```

```
[1 2 3]
```

Printed out as a 1D array

```
print( a[0] )
```

```
1
```

Supports matrix manipulation and linear algebra

NumPy 2D array

We can construct 2D arrays, as multi-dimensional arrays with any number N of dimensions, otherwise known (despite not mathematically precise) as "tensors"

```
import numpy as np

a = np.array( [[1, 2, 3], [4, 5, 6]] )

print(a)
```

a list of lists..

2-D array object

```
[[1 2 3]
 [4 5 6]]
```

Printed out in a form that suggest rows and columns of a matrix. This is intentional: NumPy is intended for matrix manipulation so indeed it effectively understands 2D objects as a matrix and prints it out in rows and columns

```
print( a.ndim )
print( a.size )
print( a.shape )
print( a.dtype )
```

```
2
6
(2, 3)
int32
```

Tuple

a tuple is a read-only list

size = total nb of elements in the arrays

shape = nb elements in each dimension

type: all NumPy array elements must be of the same type (lists instead can contain mixed data types)

NumPy 2D array

```
print(a)
```

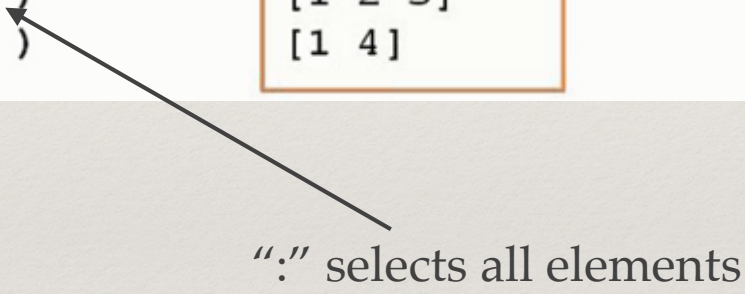
```
[[1 2 3]
 [4 5 6]]
```

```
print( a[1, 0] )
print( a[0, 0:3] )
print( a[0, :] )
print( a[:, 0] )
```

```
4
[1 2 3]
[1 2 3]
[1 4]
```

slices

[row, col]
Row
Row
Column



Initialising/building arrays

NumPy

```
a = np.zeros((2, 3))
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

```
a = np.arange(4)
```

```
[0 1 2 3]
```

```
a = np.arange(1, 25).reshape(2, 3, 4)
```

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

Elementwise operations

NumPy

```
a = np.arange(9).reshape(3,3)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
a + a
```

```
[[ 0  2  4]
 [ 6  8 10]
 [12 14 16]]
```

```
a * a
```

```
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
```

Combining arrays and scalars

NumPy

```
a = np.arange(9).reshape(3,3)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
print( a + 1 )
```

add a scalar to EACH element of the matrix

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
print( a * 4 + 1 )
```

```
[[ 1  5  9]
 [13 17 21]
 [25 29 33]]
```

Broadcasting

NumPy

NumPy arrays also support broadcasting, a cool feature.

2D array

```
a = np.arange(9).reshape(3,3)
```

```
a
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

1D array

```
b = np.array([10, 20, 30])
```

```
b
[10 20 30]
```

Matrix + Vector ???

Yes! The elements of the vector are broadcasted to have the same shape of the matrix we are adding them to, and a 3x3 matrix is created

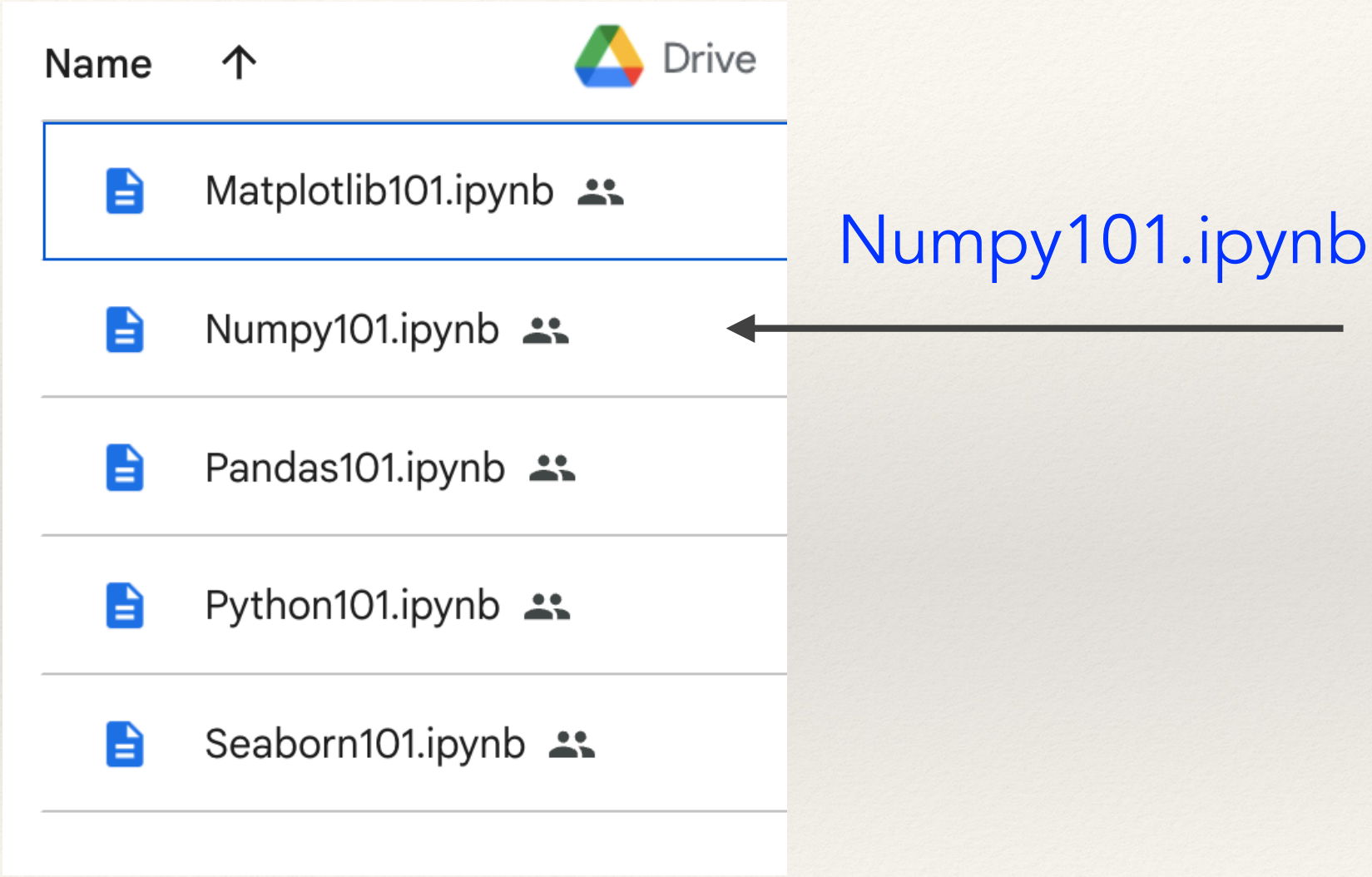
```
b is broadcast
[[10 20 30]
 [10 20 30]
 [10 20 30]]
```

```
print( a + b )
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

Hands-on:

NumPy


Name
Drive
Matplotlib101.ipynb
Numpy101.ipynb
Pandas101.ipynb
Python101.ipynb
Seaborn101.ipynb
Numpy101.ipynb

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on: Pandas

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Python light overview:

some useful packages → **Pandas**

DISCLAIMER: “This is not a programming course”, remember?

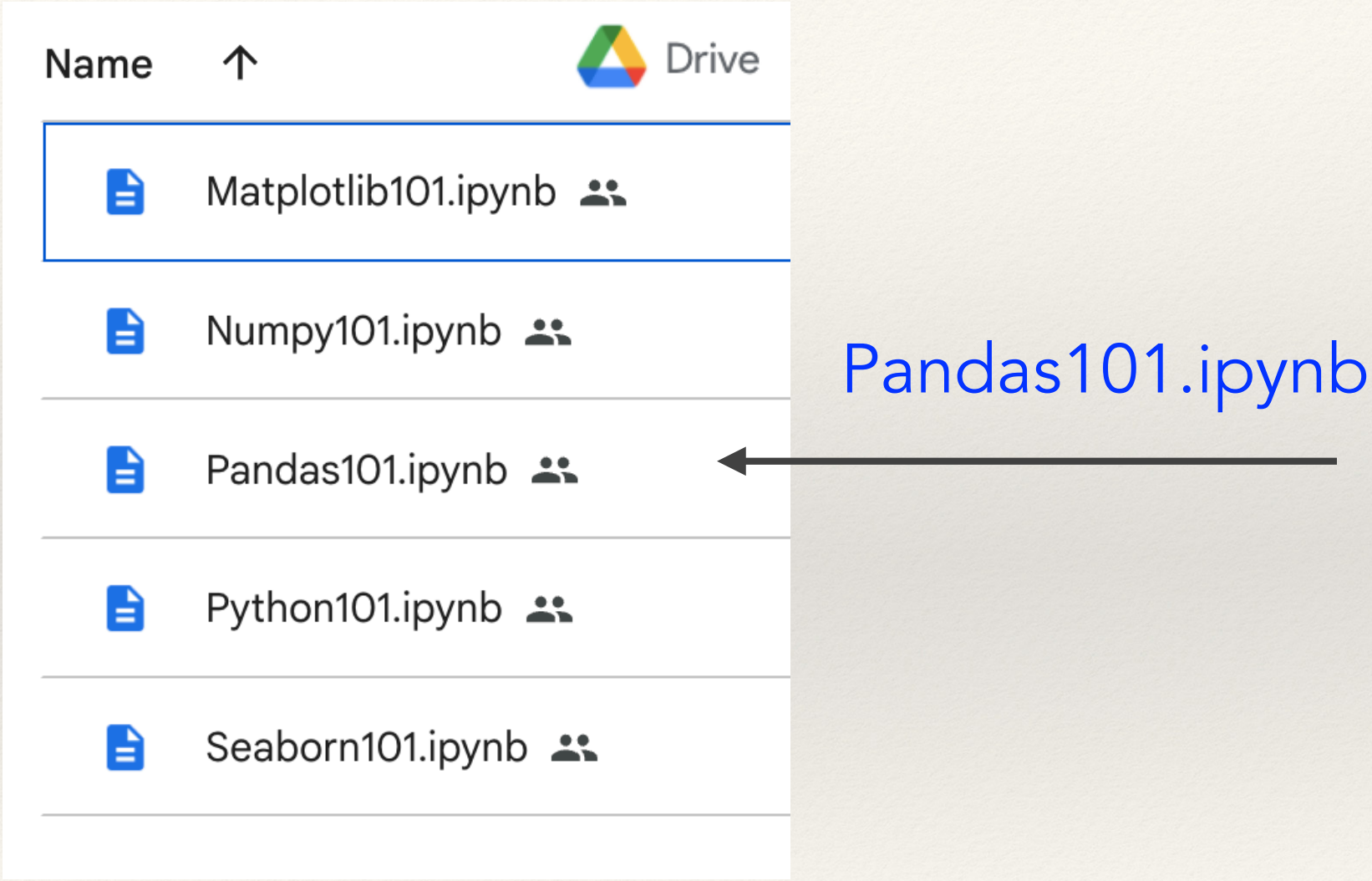
Pandas

https://pandas.pydata.org/

Pandas is an open source library providing **high-performance, easy-to-use data structures and data analysis tools** for Python.

Hands-on:

Pandas


Name
Drive
Matplotlib101.ipynb
Numpy101.ipynb
Pandas101.ipynb
Python101.ipynb
Seaborn101.ipynb
Pandas101.ipynb

2 virtual lectures coming!

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on: Matplotlib

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Python light overview:

some useful packages → **Matplotlib**

DISCLAIMER: “This is not a programming course”, remember?

Matplotlib

https://matplotlib.org/

A package that offers a huge range of predefined functions to plot and **visualise your data**.

Plot a 2D array

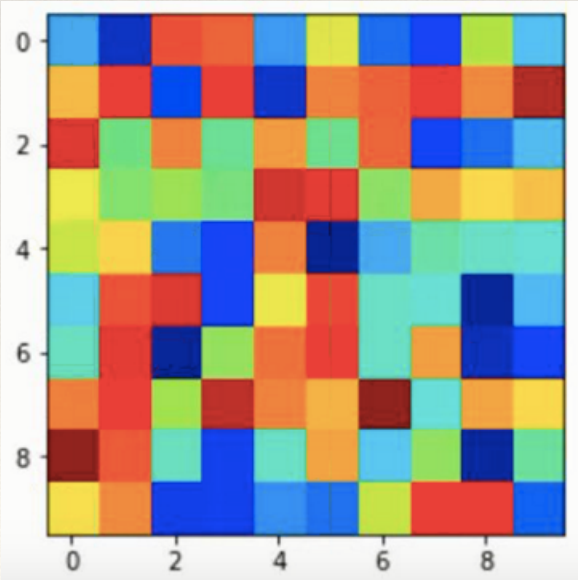
Suppose you want to make a 2D plot of a 2D NumPy array organised to some sort of color map.

```
import matplotlib.pyplot as plt

a = np.random.rand(10, 10)

plt.imshow(a, cmap='jet')
plt.show()
```

a 10x10 matrix in which each element is initialised to be 0 to 1

$0 \leq r < 1$

create a plot and show it



rand = Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

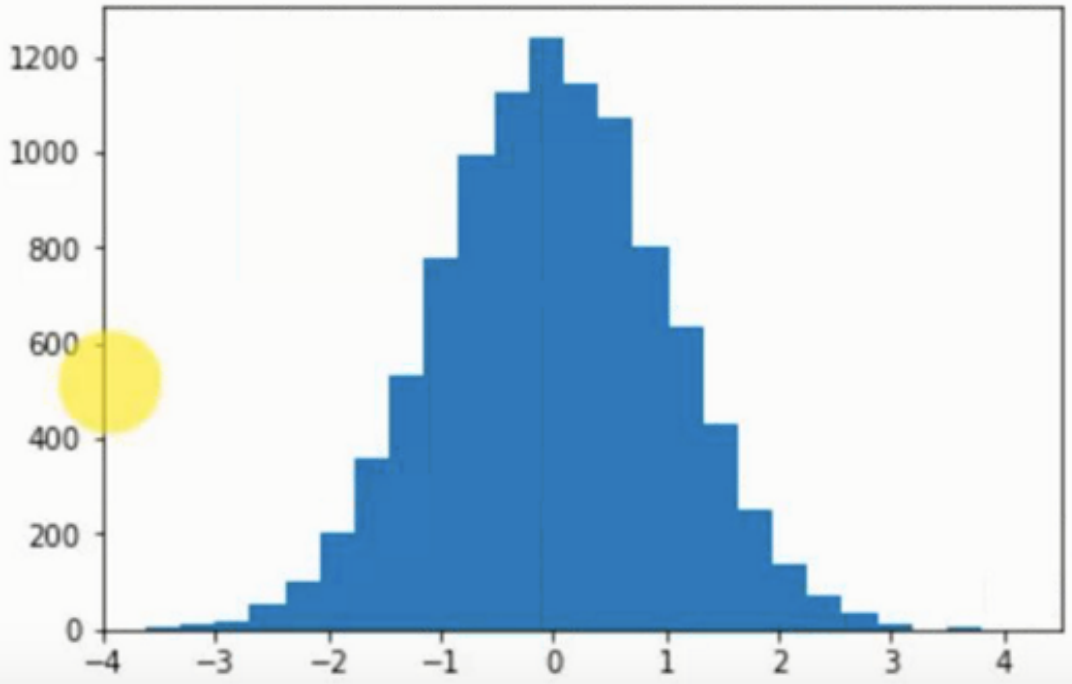
Plot a histogram

E.g. plot a standard normal distribution

```
import matplotlib.pyplot as plt

a = np.random.randn(10000)

plt.hist(a, 25)
plt.show()
```

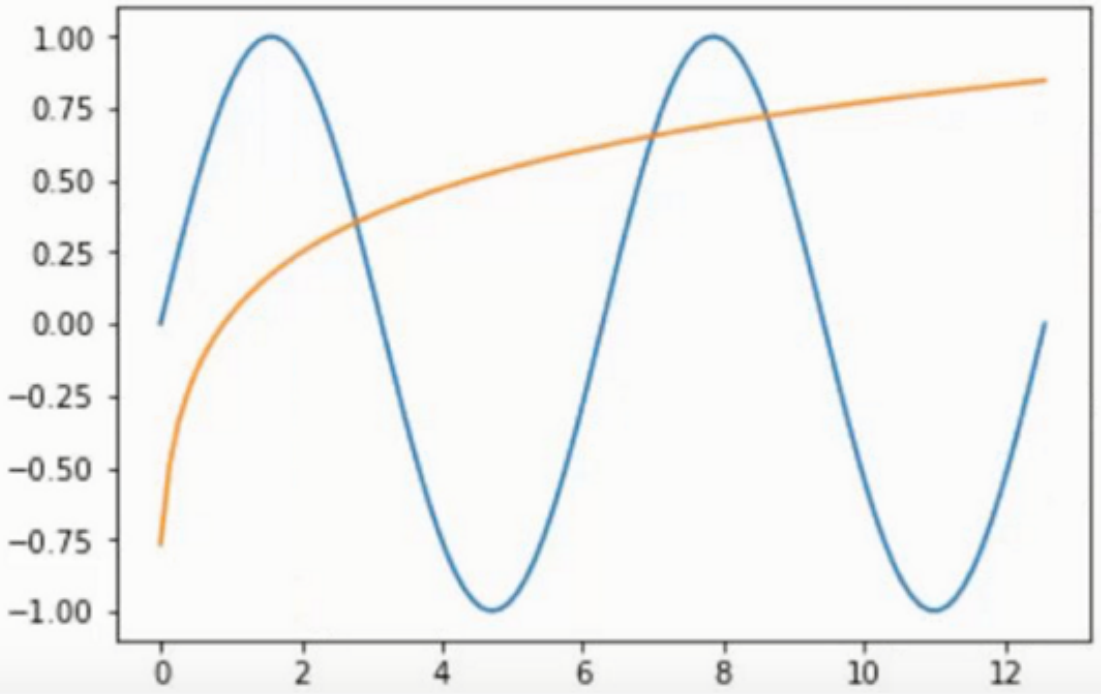
25 buckets

randn = Return a sample from the "standard normal" distribution.



Plot a function

```
import matplotlib.pyplot as plt

x = np.linspace(0.0, 4 * np.pi, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.log(x + .1) / 3)
plt.show()
```

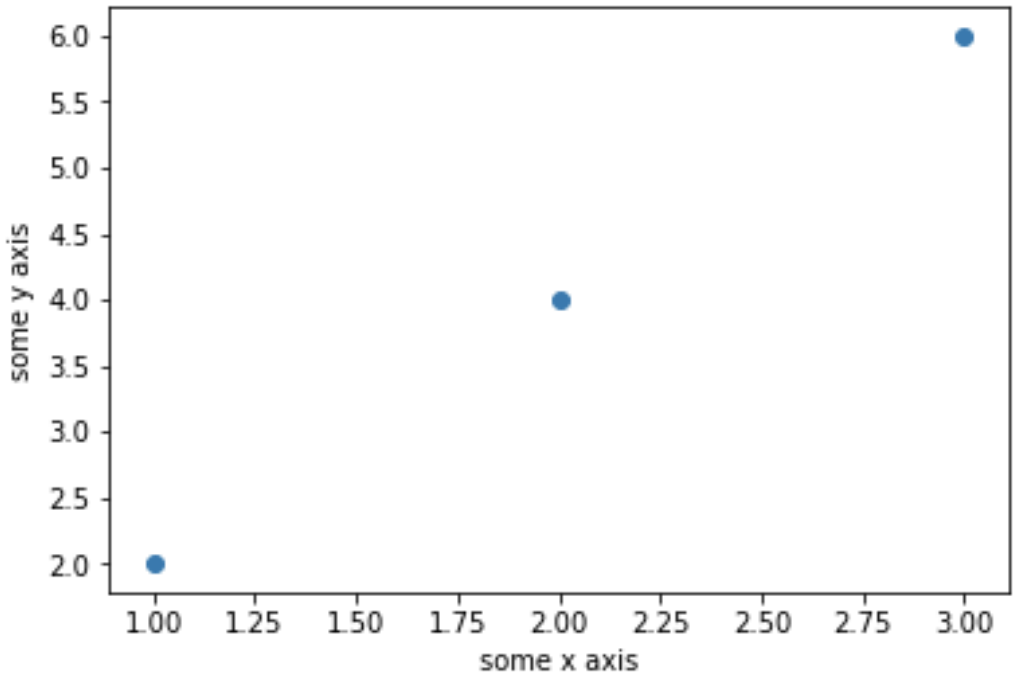


Scatter plot

matplotlib

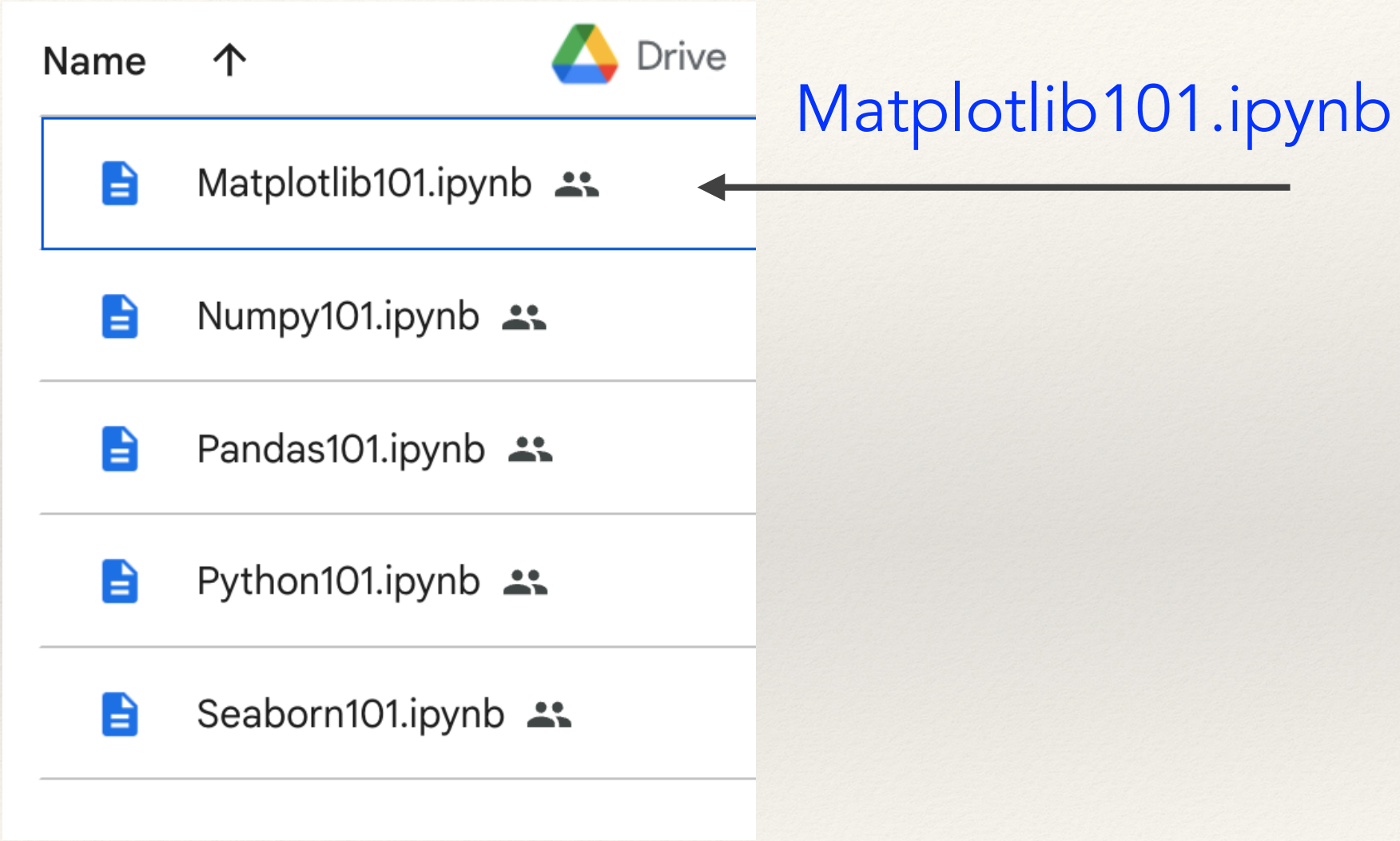
Scatter plot

In [3]:
```
x = numpy.array([1, 2, 3])
y = numpy.array([2, 4, 6])
plt.scatter(x,y)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```



Hands-on:

Matplotlib


Name
Drive
Matplotlib101.ipynb
Numpy101.ipynb
Pandas101.ipynb
Python101.ipynb
Seaborn101.ipynb
Matplotlib101.ipynb

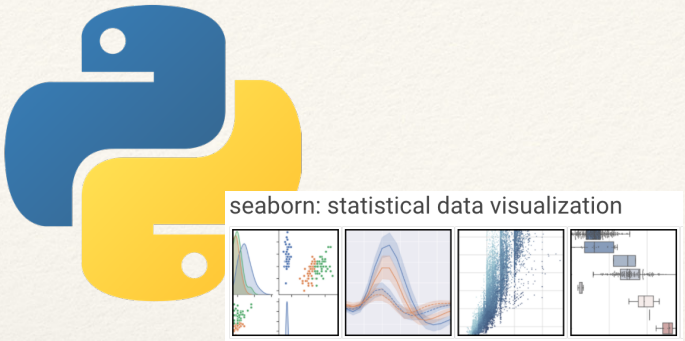
Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on: Seaborn

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

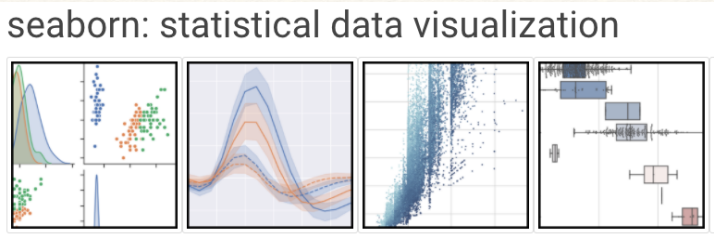


Python light overview:

some useful packages → **Seaborn**

DISCLAIMER: “This is not a programming course”, remember?

Seaborn

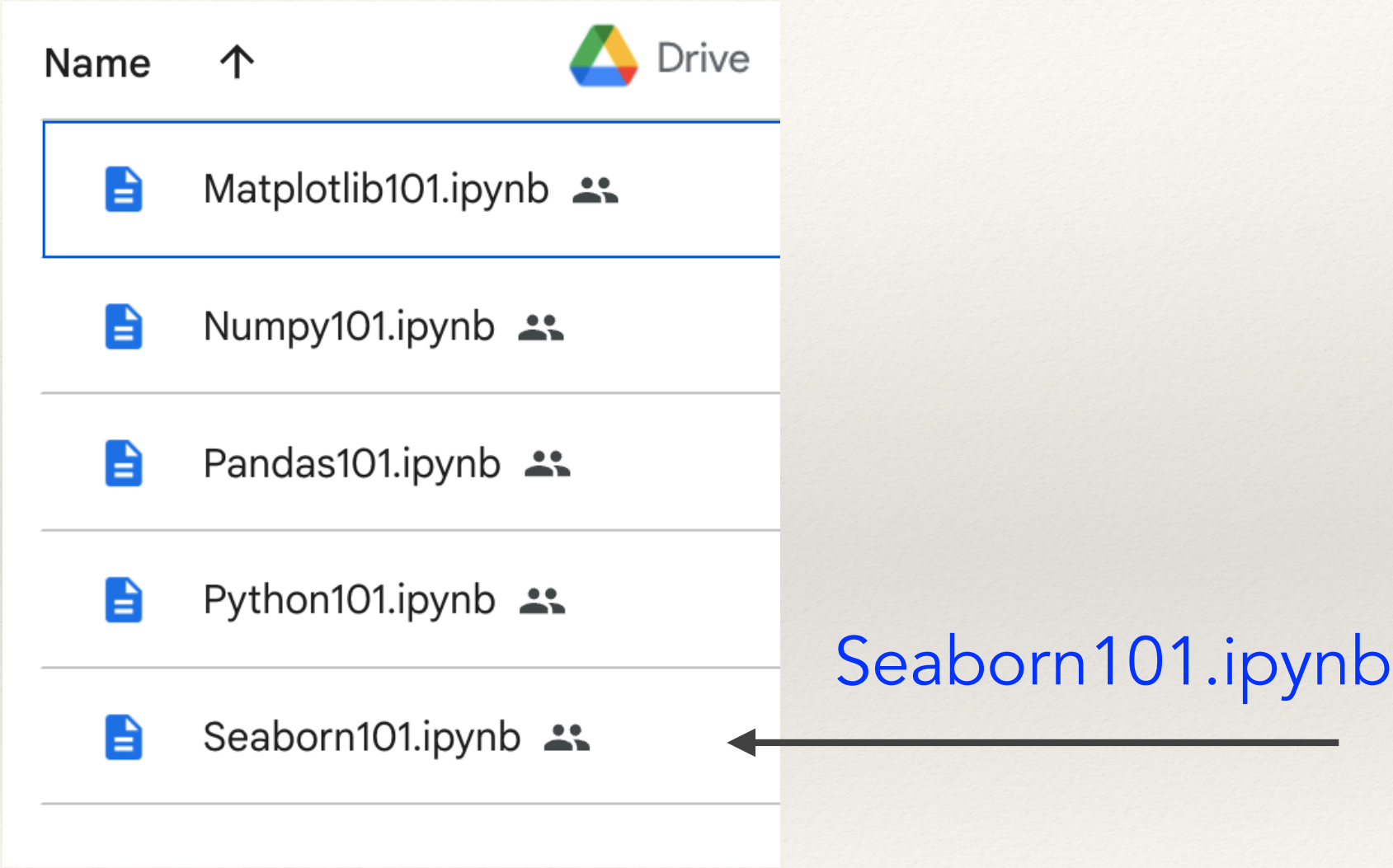


Seaborn is a Python data visualization library based on matplolib.

- It provides a high-level interface for drawing attractive and informative statistical graphics

Hands-on:

Seaborn


Name
Drive
Matplotlib101.ipynb
Numpy101.ipynb
Pandas101.ipynb
Python101.ipynb
Seaborn101.ipynb
Seaborn101.ipynb

SciPy

A final remark.

Note that most of it is "inside" SciPy, a Python-based ecosystem of open-source software for mathematics, science, and engineering

- In particular, these are some of the core packages - and you see some we covered a bit already..