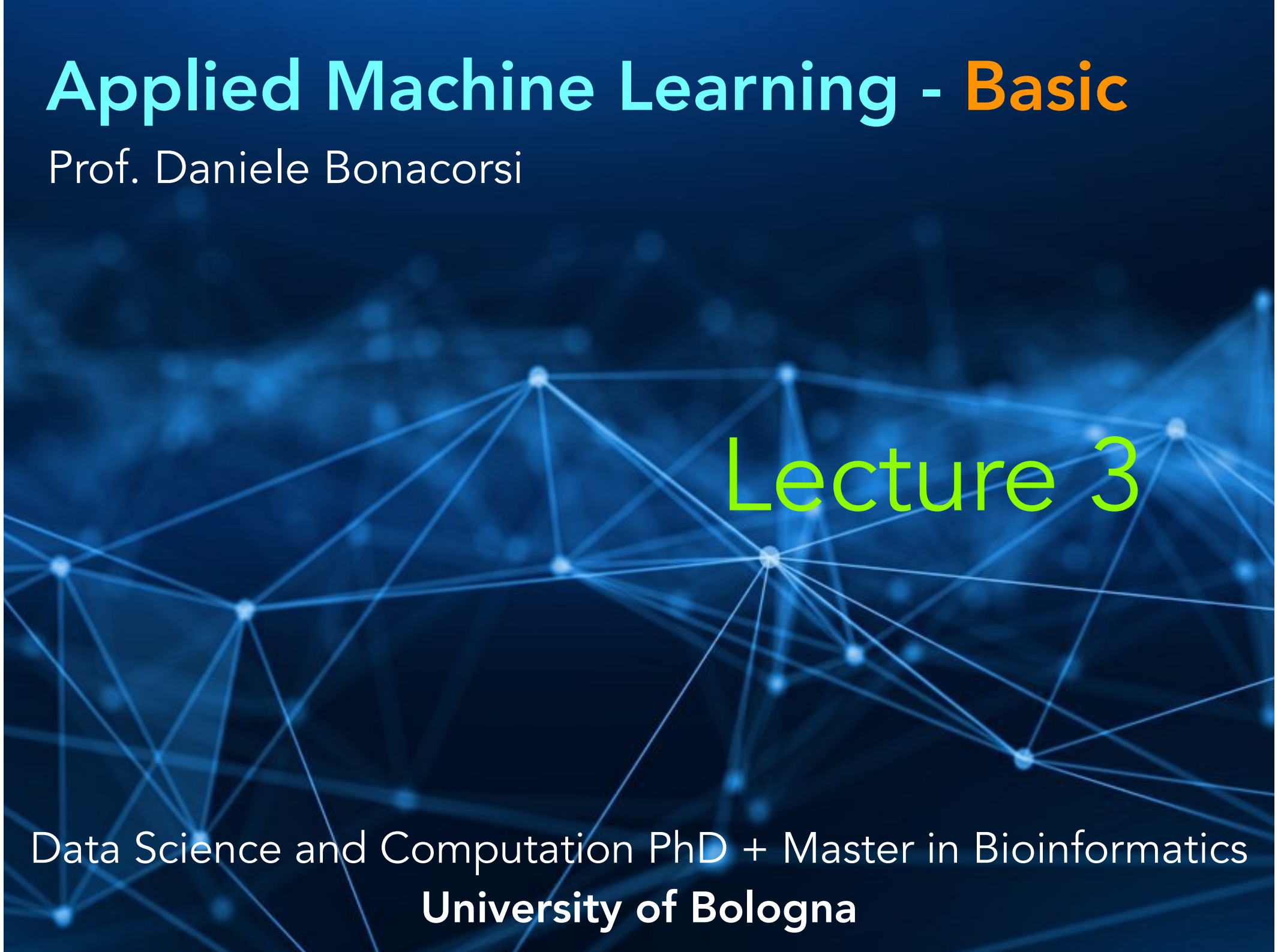
Applied Machine Learning - Basic

Prof. Daniele Bonacorsi



Lecture 3

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Communications

Miscellanea

I am protecting today's lecture and Friday's lecture

- Be ready for possible announcements, though → details at the lecture

Recap from **Lecture2**

Model representation

(…)

Univariate linear regression - Gradient Descent

Multivariate linear regression

Gradient Descent

Feature scaling and standardisation/normalisation

Polynomial regression

Analytical solution: Normal equation

NEXT: material on Py/Jupyter/Colab + Logistic Regression

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on: Python

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Python introduction (part 0)

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

Programming languages: the landscape

Let's see some examples

- **high-level** languages
  - e.g. C, C++, Fortran, Java, ..

Programming languages: the landscape

Let's see some examples

- **high-level** languages
  - e.g. C, C++, Fortran, Java, ..
- **very high-level interactive** computational environments (general-purpose)
  - e.g. Matlab/Octave, Mathematica, Python, ..

Programming languages: the landscape

Let's see some examples

- **high-level** languages
  - e.g. C, C++, Fortran, Java, ..
- **very high-level interactive** computational environments (general-purpose)
  - e.g. Matlab/Octave, Mathematica, Python, ..
- **text processing** utilities and **scripting** languages
  - e.g. Unix shell scripting, sed, awk, Perl, ..

Programming languages: the landscape

Let's see some examples

- **high-level** languages
  - e.g. C, C++, Fortran, Java, ..
- **very high-level interactive** computational environments (general-purpose)
  - e.g. Matlab/Octave, Mathematica, Python, ..
- **text processing** utilities and **scripting** languages
  - e.g. Unix shell scripting, sed, awk, Perl, ..
- **special-purpose** languages
  - e.g. R for stats, LabVIEW for lab activities, SQL for databases, ..

Very-high level languages: **Python**

General purpose, interpreted programming language

Compact and readable - like pseudo-code

- looks like an English language description of what needs to be done

Widely used for scientific programming (→ also data science)

Versatile to use, and very forgiving

- thus enabling quick coding times

Recognisable (e.g. indentation)

- e.g. Python uses whitespace indentation to delimit code blocks (instead of curly braces, as in C, or keywords as 'end ' in Fortran)

Chinese Python

[周蟒 → “Zhōu mǎng”]

http://zh.wikipedia.org/wiki/ZhPy → zh.wikipedia.org/wiki/周蟒

維基百科
自由的百科全書

没有登录 讨论 贡献 创建账户 登录

条目 讨论 不转换 汉 漢 阅读 编辑 查看历史 搜索维基百科

首页
分类索引
特色内容
新闻动态
最近更改
随机条目

帮助
帮助
维基社群
方针与指引
互助客栈
知识问答
字词转换
IRC即时聊天
联络我们
关于维基百科
资助维基百科

打印/导出
下载为PDF
打印页面

工具
链入页面
相关更改
上传文件
特殊页面
固定链接
页面信息
维基数据项
引用本页
左侧跳顶连接

语言
添加链接

中文维基百科条目协作计划专页已建立，欢迎报名参与！ [关闭]

周蟒 [编辑]

维基百科，自由的百科全书
（重定向自ZhPy）

本條目需要补充更多来源。(2016年11月30日)
请协助添加多方面可靠来源以改善这篇条目，无法查证的内容可能會因為异议提出而移除。

周蟒，又名**zhpy**，是一個輕量的，與Python語言互相兼容的中文Python語言。讓使用者可以使用純中文語句（繁體或簡體）來編寫程式。目前主要適用於教學上。

周蟒中文程式語言目前已不再更新。提供電子書、API、完整測試用例的開放原始碼中文程式語言。

周蟒中文程式語言的目標是協助使用者透過中文程式語言學習程式語言，進而接觸世界上大部份的程式語言，而不是脫離現實世界。 周蟒作者gasolin提出了周蟒編程風格[1]與 周蟒中文程式語言的四不一沒有，透過程式語言的約定，使用周蟒寫出的中文程式保有了易於閱讀的特性，並可完全轉換成英文Python語言程式。

周蟒語言擁有Python語言的所有特性，如高效率的高階資料結構、簡單而有效的物件導向程式設計方式等等。 由於周蟒語言完全相容Python程式語言，所以可以取用所有Python程式語言資源。

周蟒語言的長處是在於發揮"完全相容Python程式語言"的中文程式語言的優點，所有語法，關鍵詞都依照Python語言的風格。學習周蟒語言後要橋接到Python語言相當容易。

周蟒也同時提供咬一口周蟒中文程式語言[2]電子書，所有範例都一併提供與python語言程式碼的對照。

周蟒是開放原始碼的，可自由下載使用。

周蟒提供編譯器與互動式直譯器，也支援**中文腳本**執行。

目录 [隐藏]
1 周蟒的Hello World程序
2 範例
2.1 輸出結果
3 物件導向
4 参考文献
5 參見
6 外部連結

周蟒

编程范型	multi-paradigm
實作者	Fred Lin（gasolin）
发行时间	2007年
穩定版本	3.0.0a1 （ 2011年6月25日 ）
型態系統	Strong, dynamic（"duck"）
作業系統	跨平台
許可證	MIT許可證
網站	GitHub
主要實作產品	
zhpy（即周蟒）	
啟發語言	
Python, 中蟒	

周蟒的Hello World程序 [编辑]

下面是一個在標準輸出設備上輸出Hello World的簡單程式，這種程式通常作為開始學習程式語言時的第一個程式：

```
#!/usr/bin/env zhpy
印出 "哈嘍，世界"
```

Chinese Python

範例 [编辑]

以下是程式透過編譯器執行的範例：

```
#!/usr/bin/env zhpy
# 檔名: while.py
數字 = 23
運行 = 真
當 運行:
    猜測 = 整數(輸入('輸入一個數字: '))

    如果 猜測 == 數字:
        印出 '恭喜, 你猜對了.'
        運行 = 假 # 這會讓循環語句結束
    假使 猜測 < 數字:
        印出 '錯了, 數字再大一點.'
    否則:
        印出 '錯了, 數字再小一點.'
否則:
    印出 '循環語句結束'
印出 '結束'
```

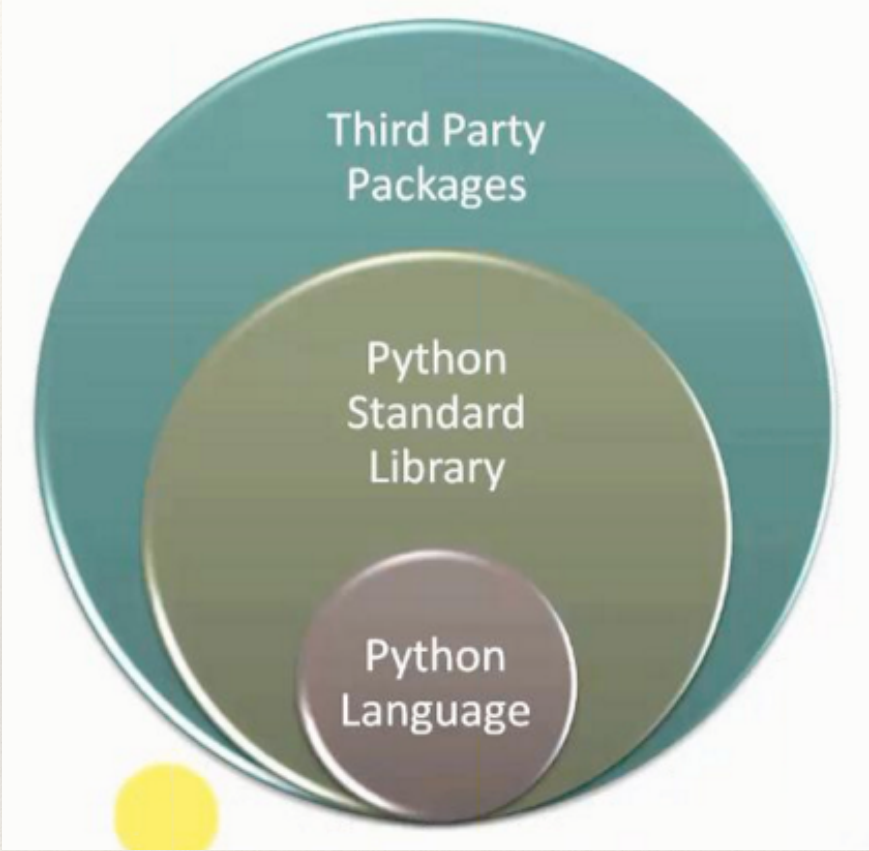
Python 版：

```
#!/usr/bin/env python
# File name: while.twpy
number = 23
running = True
while running:
    guess = int(raw_input('Enter an integer : '))

    if guess == number:
        print 'Congratulations, you guessed it.'
        running = False # this causes the while loop to stop
    elif guess < number:
        print 'No, it is higher than that.'
    else:
        print 'No, it is lower than that.'
else:
    print 'The while loop is over'
print 'Done'
```

e.g. **印出** [Yìn chū] indeed means “print out”

The **Python** env



a huge collection of
third party packages

very extensive standard library built around it - every normal capability you expect from a programming language can probably be found in this standard library

relatively small and
compact language:
www.python.org

More technically, on **Python**

Python supports **dynamic typing**

Object-oriented language from the ground up

- every function and class is actually an object

Includes **functional programming** features

Supports **meta-programming**

- you can use the Python language to extend Python itself

Extensible in C/C++

- Python can be extended by writing new modules in C/C++
- A number of the third party packages that have been contributed to the Python language are actually written in C/C++

Which **Python** version?

Python 2.x is the past

- end of support: January 2020

Python 3.x is the present (and the expected future for a while, now)

Unfortunately they are not compatible with each other.

If you start using Python today, you should start with Python 3 **but** you will soon bump into legacy packages written in Python 2

- e.g. machine learning?
- Big Science experiments code base? E.g. many (most) large physics experiments are still at some subversion of Python 2 for stability reasons..

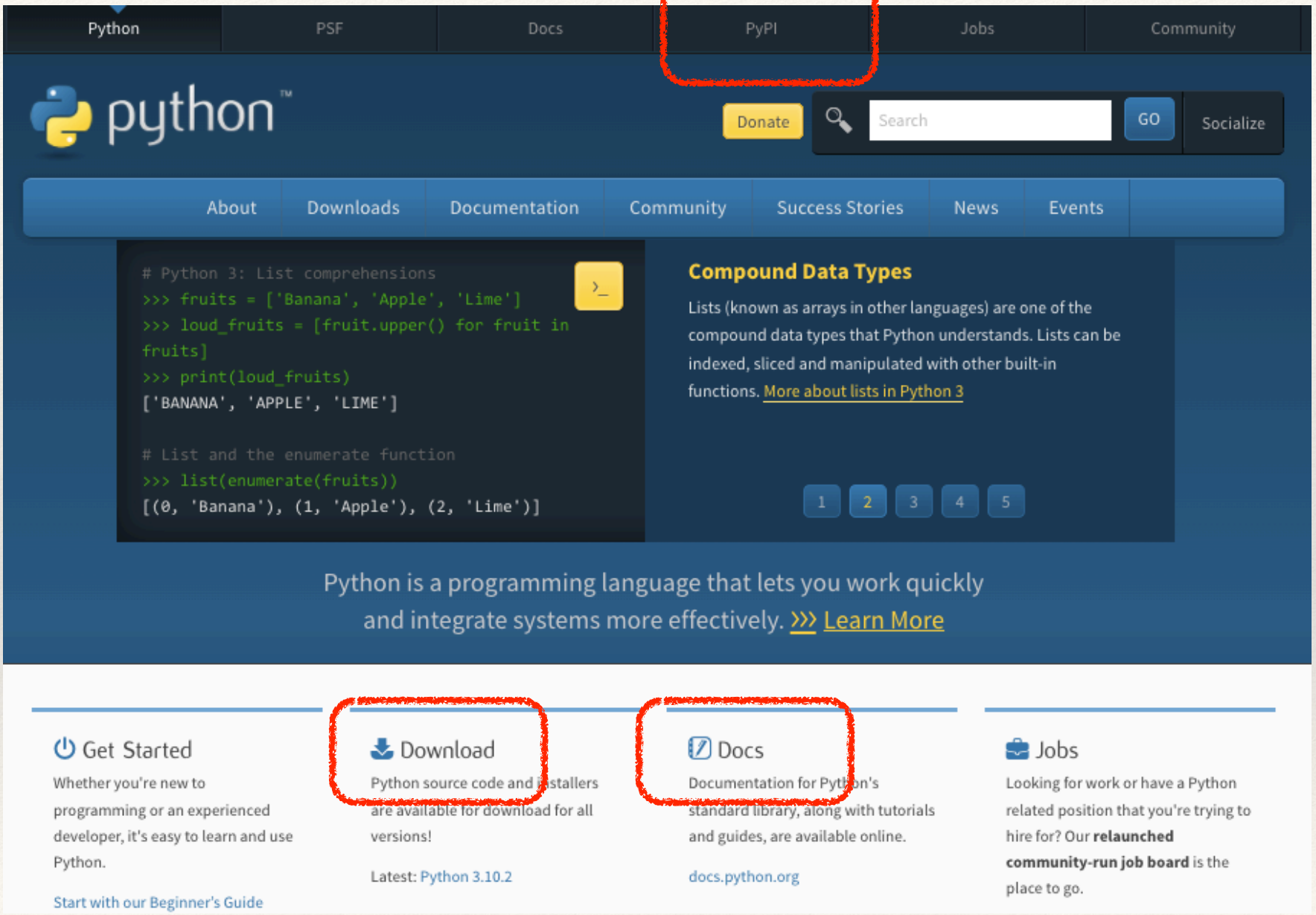
Unfortunately, you might need to be familiar with both 2 and 3, and the differences.. not a big deal, though!

Python light overview (part 1):

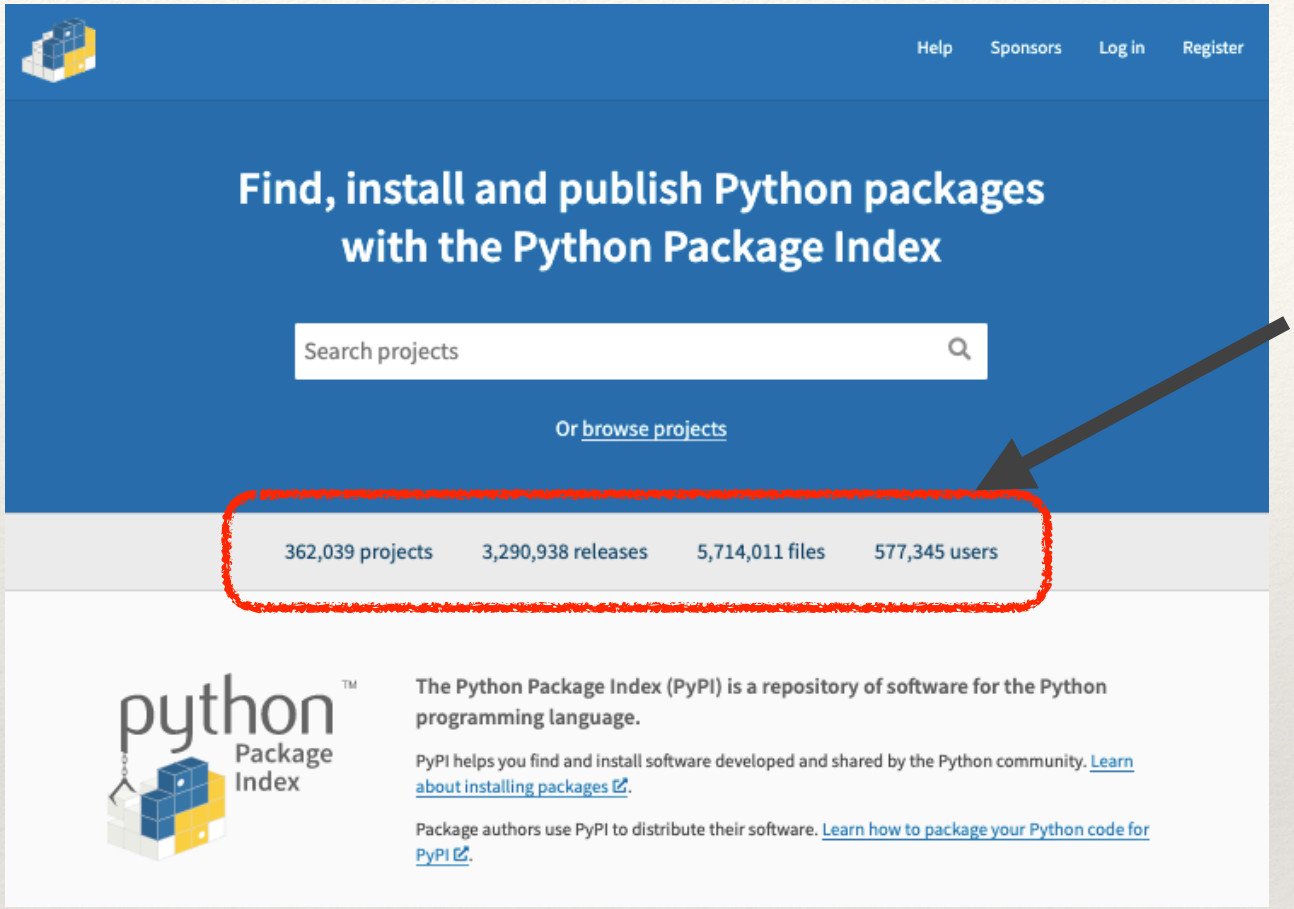
more on the language environment

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

www.python.org



Third party packages - pypi.python.org



[snapshot as of March 2022 - numbers may vary]

[last yr, same period of now, it was 3'995k releases…]

You can find them all in the **Python Package Index (PyPI)**, a repository of software for the Python programming language.

- any feature you think you might need in Python, it is probably in the **Standard Library**
- any **application or general purpose utility** you think you might need, it is probably in **PyPI**

pip as a package manager

You will most probably use **pip** to install packages

- google it..
- .. and get familiar with hands-on installations..

Python implementations

CPython

- the Python reference implementation (from python.org)

Many alternatives… (listed also at python.org itself)

Also a large number of integrated development environment for Python available

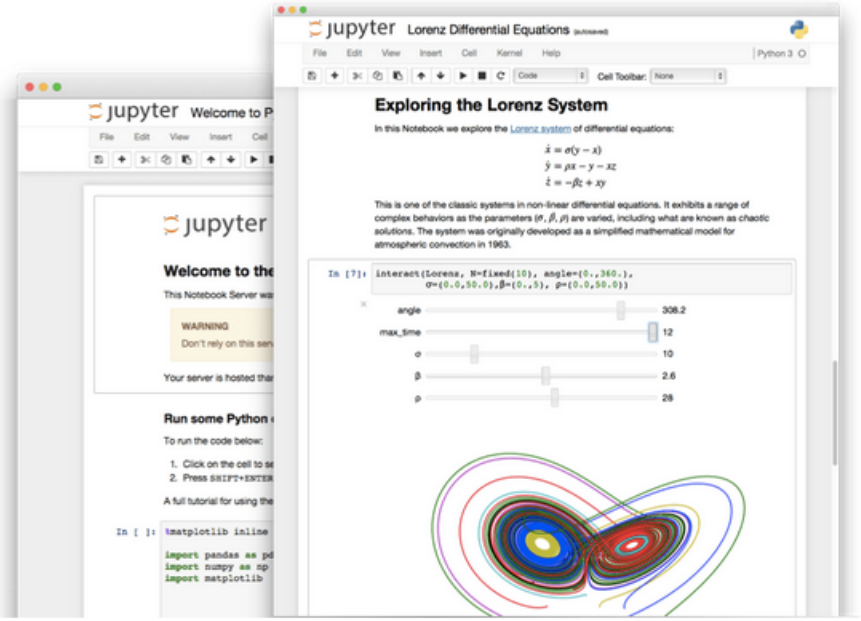
- both free and paid for
- same for Python editors

Very popular implementations and/or distributions:

- IPython -> **Jupyter Notebook** (also supports other languages)
- the **Anaconda** distribution: a single distribution with Python, Jupyter and also **Numpy**, **Scipy**, and many more packages popular for data science applications

Jupyter Notebook

http://jupyter.org/



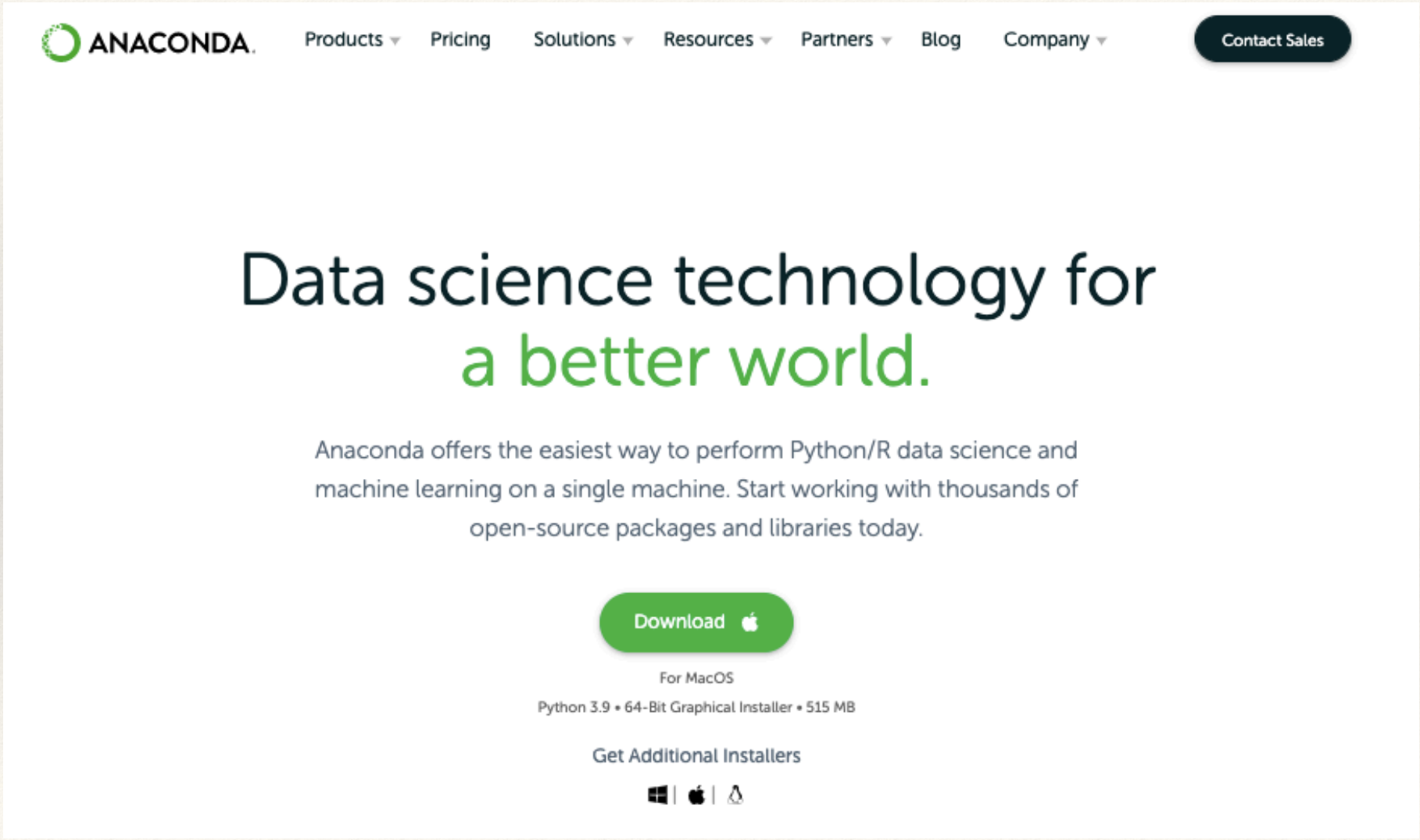
The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

More later..

Anaconda

https://www.anaconda.com/



More on Python implementations

https://www.python.org/download/alternatives/

https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

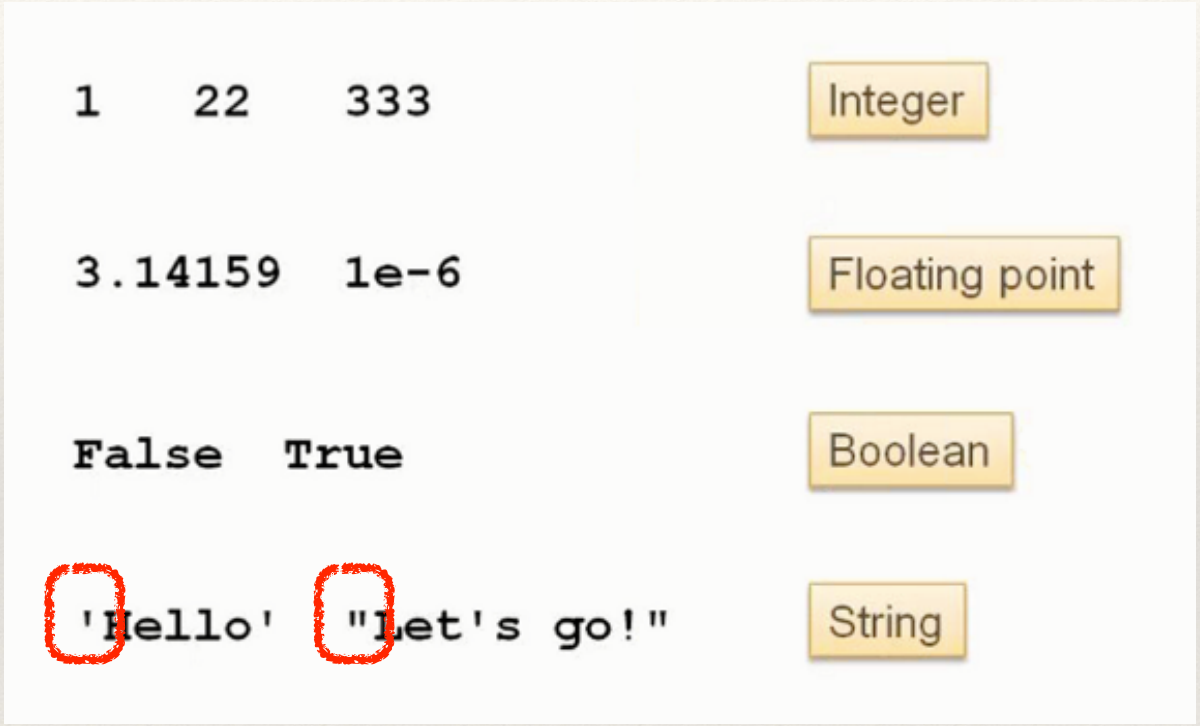
https://wiki.python.org/moin/PythonEditors

Python light overview (part 2):

language syntax

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

Variable types



Comments

```
# This is a regular Python comment

a = 1    # End-of-line comment


'''A comment can also take the form of a string'''


"""
A multi-line comment
"""

'''
Another
multi-line
comment
'''
```

Doc strings

There is a number of third-party packages for interpreting these multiline comments as documentation!

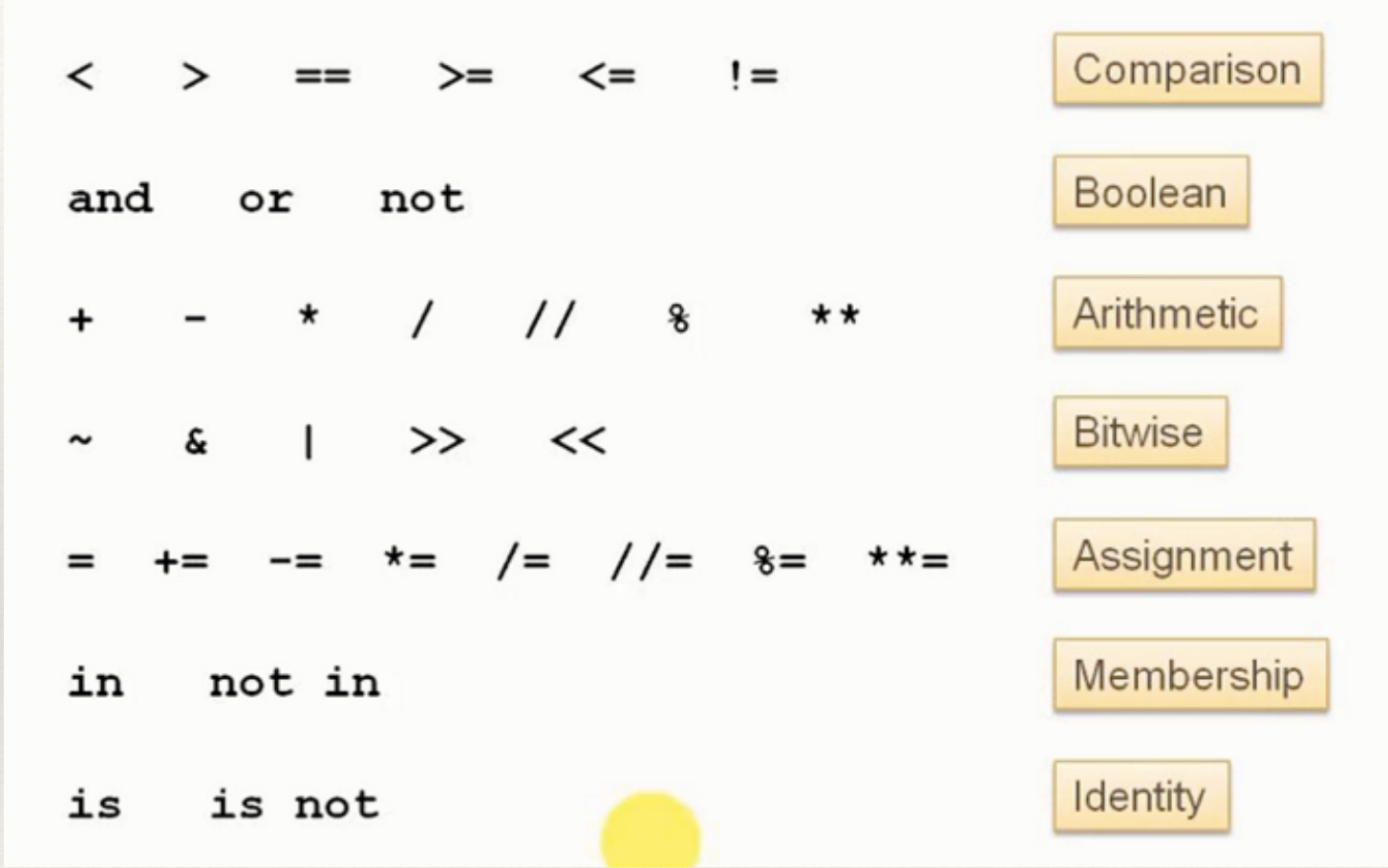
Control statements

```
if a == 0:
    b = 1
    if c < 0 and d > 0:
        b = 2
elif a > 0:
    b = 3
else:
    for i in range(4):
        b = 4
        while b > 0:
            print(b)
            b = b - 1
        if i == j: break
```

As any other language

Note:
INDENTATION is a vital part of the Python syntax as control structure. No "end" keyword. It looks quirky at the beginning, but you will get used to (and will love) it as a neat choice.

Operators



Familiar, if you know e.g. C..

Range

```
for i in range(4):
    print(i, end=' ')
```

```
0 1 2 3
```

Keyword argument

```
for i in range(2, 6):
    print(i, end=' ')
```

```
2 3 4 5
```

range(Start , Stop , Step)

Functions

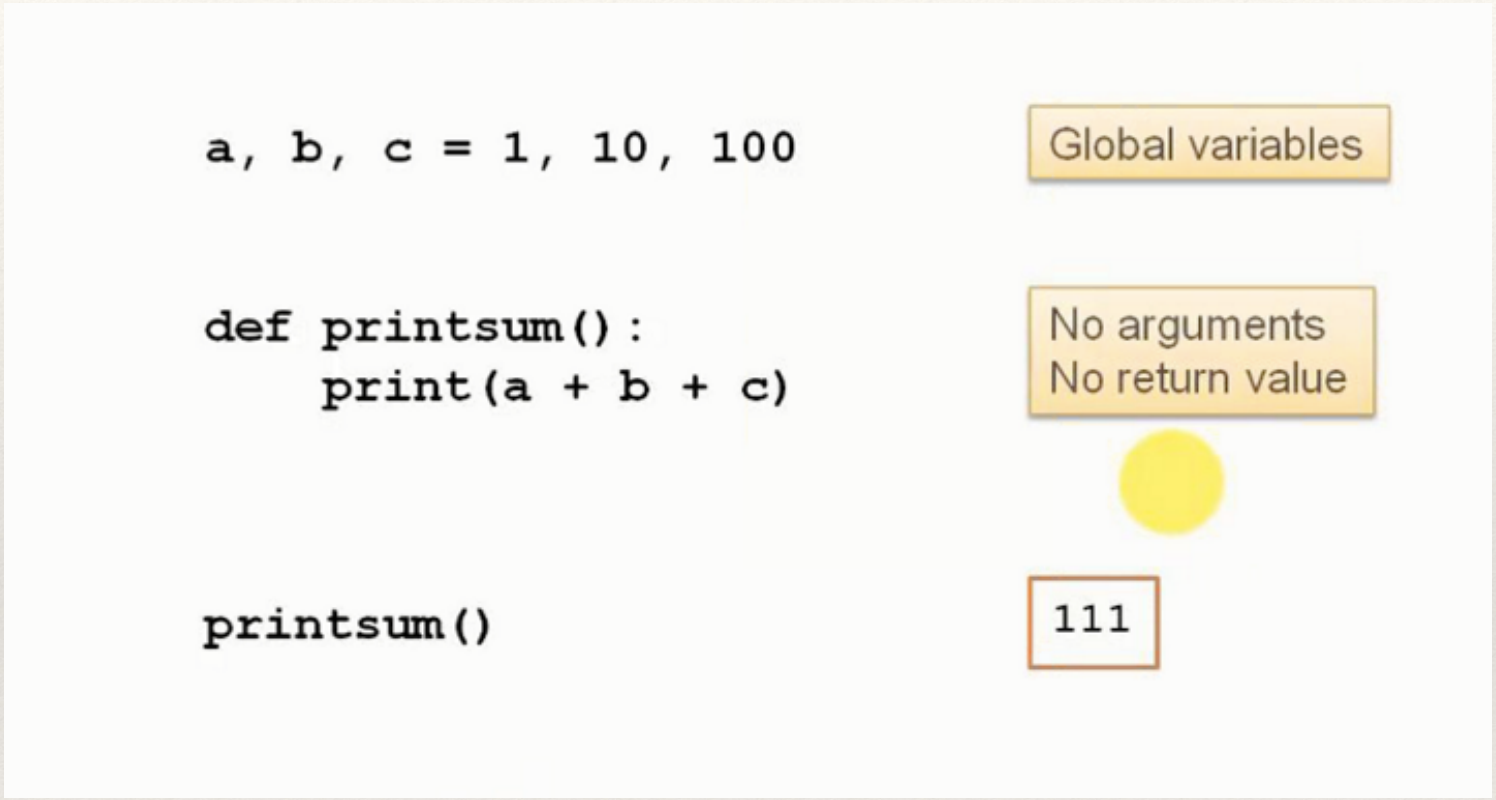
Just 2 keywords: **def** and **return**

```
def add(p, q, r):
    temp = p + q + r
    return temp

sum = add(1, 10, 100)
```

NOTE: variables are not declared explicitly, they are declared implicitly the first time they are assigned.

E.g. in this example, "temp" is bound to an object that results from evaluating a sum of objects that are defined as integers here. So the function returns an object that is an integer.

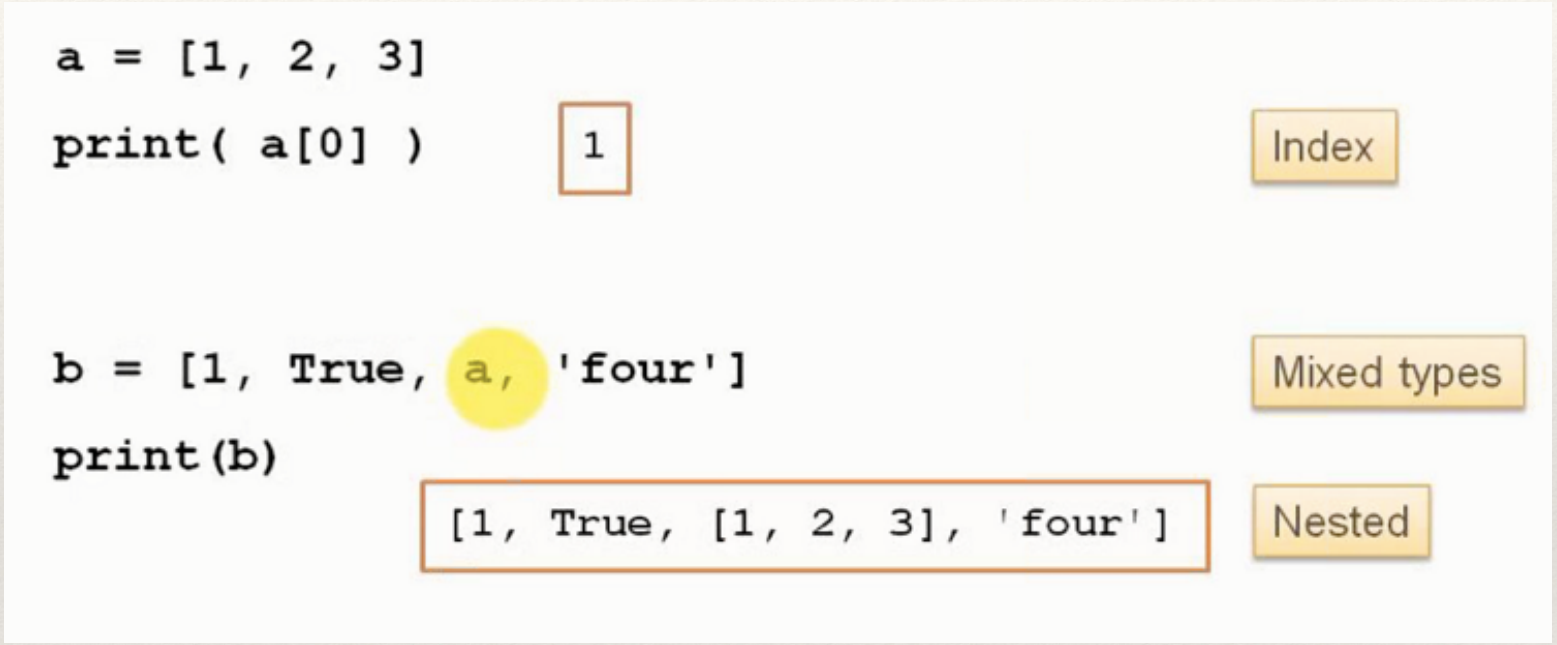


This function does not return any value (no **return** used in the **def**).

You can define variables outside of a function as **global variables**, then refer them inside the function. This is how outer and inner scope works in Python.

Lists

a is a variable (so an object), and this object is bound to a list of 3 integers



Slice of a list

```
a = [1, 2, 3]
print( a[0] )
```

1

Index

```
b = [1, True, a, 'four']
print(b)
```

Mixed types

[1, True, [1, 2, 3], 'four']

Nested

```
print( b[0:2] )
```

[1, True]

Slice

[Start : Stop : Step]

Function + List

A function that returns a list

```
def sum_and_diff(a, b):
    return [a + b, a - b]

x = sum_and_diff(7, 4)

print(type(x))
print(x)
print(x[0])
```

That list is an object

```
<class 'list'>
[11, 3]
11
```

Dictionary

```
ages = {'Tom': 32, 'Dick': 58, 'Harry': 28}

for k in ages.keys():
    print(k, ages[k])
```

```
Tom 32
Dick 58
Harry 28
```

```
del ages['Tom']

ages['Bert'] = 49

for k,v in ages.items():
    print(k, v)
```

```
Dick 58
Harry 28
Bert 49
```

Modules

Note: the 2 modules contains each a function which has the same name! But these are disambiguated by calling them with the module name, plus a "." and then the function name

spam.py

```
def fn():
    print('spam')
```

eggs.py

```
def fn():
    print('eggs')
```

```
import spam
import eggs

spam.fn()
eggs.fn()
```

```
spam
eggs
```

Modules

A couple of tricks..

```
import spam
import eggs

spam.fn()
eggs.fn()
```

```
spam
eggs
```

```
from spam import fn
from eggs import fn as fn2

fn()
fn2()
```

```
spam
eggs
```

Packages

Packages are simply fancy modules.

```
top/
    __init__.py
    mod1.py
    mod2.py
    part1/
        __init__.py
        mod3.py
        mod4.py
    part2/
        __init__.py
        mod5.py
        mod6.py
```

```
import top.mod1
import top.mod2
import top.part1.mod3
import top.part1.mod4
import top.part2.mod5
import top.part2.mod6

top.part2.mod6.fn()
```

If you want to avoid all this, just do:

```
from top.part2.mod6 import fn

fn()
```

Another example

Do you understand - or at least “feel” - what these few lines may do?

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()

model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32)

print(model.evaluate(x_test, y_test, batch_size=128)[1])
```

Classes

List

(this is real - despite basic - code that aims at creating a machine learning model..)

Context manager

The **with** construct. The context manager does not do anything :) it just sets up the following code..

```
with open('spam.txt') as file:
    for line in file:
        print(line, end='')
```

That's all for now on Python. What next?

About Python, **in general**, if you are a beginner:

- install python on your (local) resource (e.g. PC/laptop)
- browse online and pick up a good tutorial
- familiarise with the basic concepts and the syntax
- **exercise, exercise, exercise!** This will be most probably (one of) your "language(s)" at work, so get used to it, and constantly improve!

About Python, **in the context this course**:

- no installation, focus on understanding the syntax, as we will (sort of) give it for granted
- if you are unsure on what some portion of python code does, set aside, check/ study it carefully, then come back to main stream of the course..
  - you are students at different levels, so this part is left to you - ask if you need assistance!

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on: Jupyter/Colab

Data Science and Computation PhD + Master in Bioinformatics
University of Bologna

Hands-on:

Get familiar with Jupyter

Jupyter notebooks

https://jupyter.org/

An open-source web application that allows to create and share code, plots, documents.

It offers **one single environment** for:

- code
- comments on the code
- data analysis + data visualisation
- any additional context (e.g. text, formulas, even media files..)

Perfect for **streamlining an entire workflow.**

And excellent **in the prototyping phase**.

Jupyter notebooks

Other features:

- **pedagogical tasks**: produce a report, a course, a book, anything
- **sharing tasks**: individual analysis but also good for sharing with co-workers

In a nutshell, a powerful, versatile, shareable tool

- not a surprise that is **very popular among data scientists**

User interface

Any context is organised in independent **cells**.

A cell is executed individually

- a user can test a portion of her/his code without needing to run all code
- naturally fits interpreted languages

Getting started (local resource)

Install

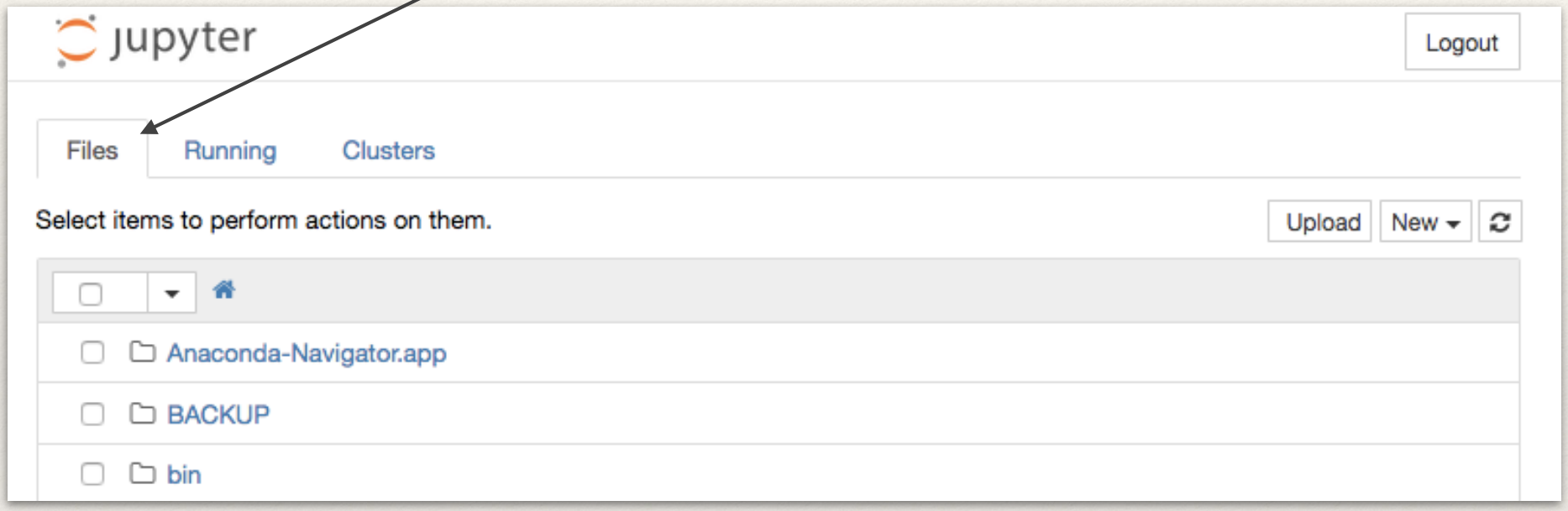
- e.g. anaconda gives you python + jupyter (and plenty more..) in one shot
- pip install jupyter

Launch

- $ jupyter notebook &
- Jupyter notebook opens up in your default web browser with the URL
  - http://localhost:8888/tree

Your console

List all the files in the dir
you launched "jupyter notebook" from



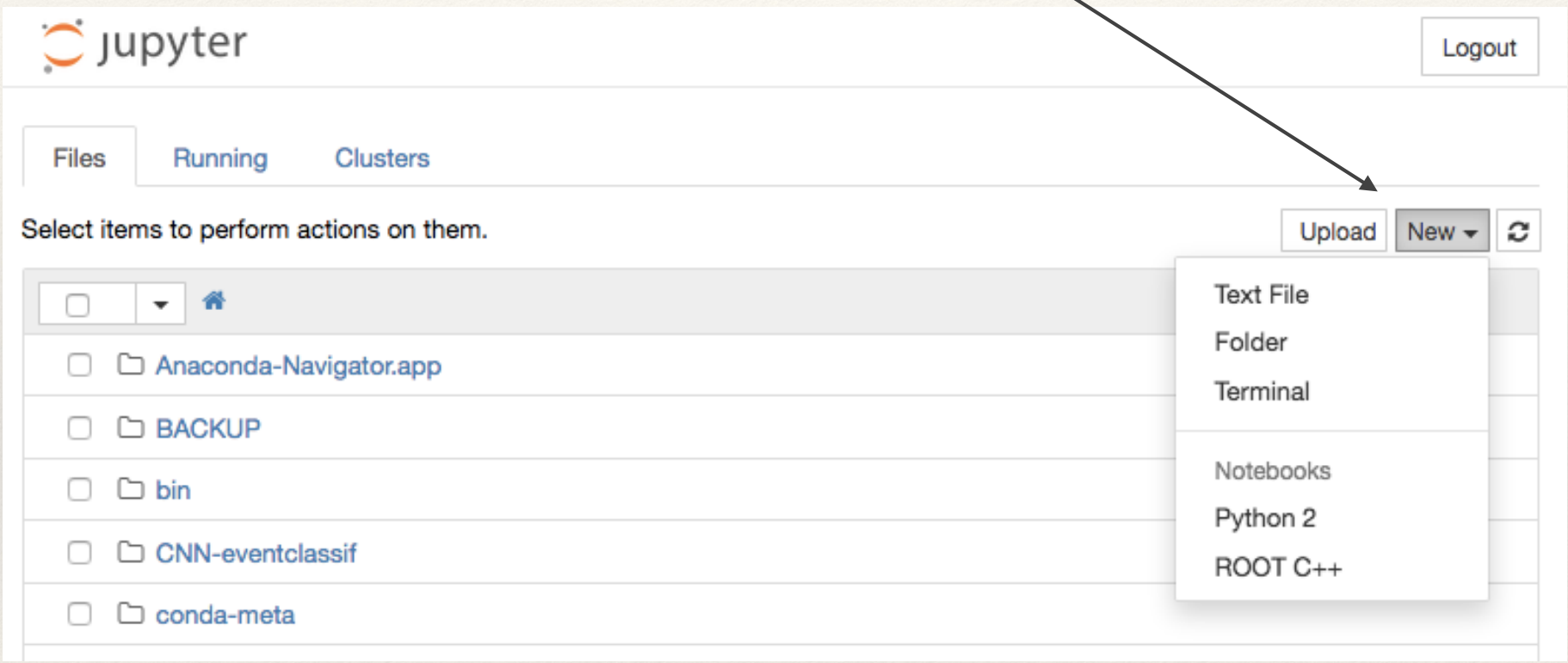
Your console

List all running processes, i.e. terminals and notebooks you currently have open

Provided by iPython parallel

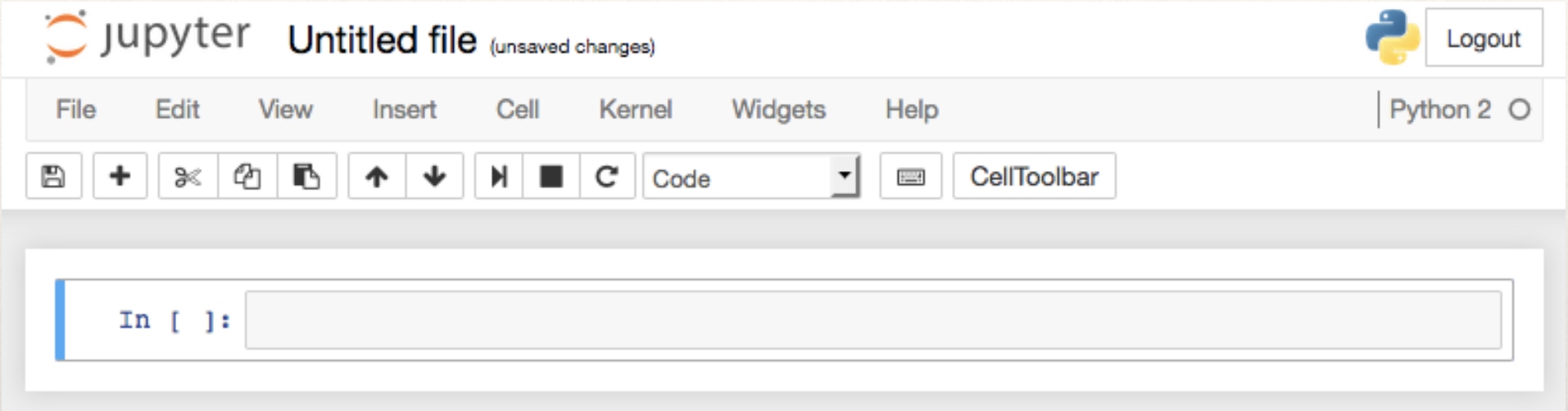
Start using a notebook

Open a new file: Text file, Folder, Terminal, plus **notebooks** in various languages



E.g. in this example, you could select "Python 2" and open a notebook where code cells are interpreted if you write code in Python 2 syntax into them

As simple as this



We will take some time later on, and you can play around it yourself!

Basic options

Explore it yourself. Documentation is good. Focus on two things.

Dropdown menu

- **Code**: self-explanatory; it is where you type your code
- **Markdown**: where you type your text, e.g. comments, notes, ..
- **Raw NBConvert**: a command line tool to convert your notebook into another format (like HTML)
- **Header**: you can use text and do the same.. in most recent versions this will be disappearing and shading into Markdown itself

Command palette

- shortcut commands: learning a few will suffice.

How many keyboard shortcuts do I need?

Few people know many of them.. most people knew just few

One can survive with B/A, DD, <ESC>+M/Y:

- **B**/**A**: position yourself and add a cell below/above the current one (B or A respectively)
- **DD:** delete a cell
- **<ESC>+M/Y**: change mode from code (Y) to markdown (M), and viceversa

plus a minimal adoption of easy and self-explanatory GUI items

- e.g. File → Open → ..

Not only Python

We showed an example interface allowing for Python (2).

Jupyter is **not** limited to Python. You can use other languages in Jupyter notebooks

- e.g. R, Julia, JavaScript, etc
  - suppose you like the 'ggplot2' package in R: you can use it within a Jupyter notebook!

You need to enable the relevant Jupyter "**kernel**"

- e.g. to enable R in Jupyter, you need the 'IRKernel' (a dedicated kernel for R) which is available on github.
- e.g. for Javascript, you need the 'IJavascript' kernel
- same for other languages..

More [1/2]: Magic functions

Many exist, do explore..

- they can be used line-wise and cell-wise

```
In [3]: %lsmagic
Out[3]: Available line magics:
        %alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cat
        %cd  %clear  %colors  %config  %connect_info  %cp  %debug  %dhist  %dirs
        %doctest_mode  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts  %
        ldir  %less  %lf  %lk  %ll  %load  %load_ext  %loadpy  %logoff  %logon  %
        logstart  %logstate  %logstop  %ls  %lsmagic  %lx  %macro  %magic  %man
        %matplotlib  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb  %pdef
        %pdoc  %pfile  %pinfo  %pinfo2  %popd  %pprint  %precision  %profile  %pr
        un  %psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickr
        ef  %recall  %rehashx  %reload_ext  %rep  %rerun  %reset  %reset_selectiv
        e  %rm  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %t
        ime  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

        Available cell magics:
        %%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascr
        ipt  %%js  %%latex  %%perl  %%prun  %%pypy  %%python  %%python2  %%python
        3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%wri
        tefile

        Automagic is ON, % prefix IS NOT needed for line magics.
```

- play around e.g. with %time and %%time

```
In [16]: %time sum(range(100))
```

```
In [19]: %%time
         sum(range(100))
         sum(range(100000))
```

More [2/2]: Interactivity via ipywidgets

Add more interactivity - sometimes an added value:

```
from ipywidgets import widgets
```

Many examples online - explore!

Hands-on:

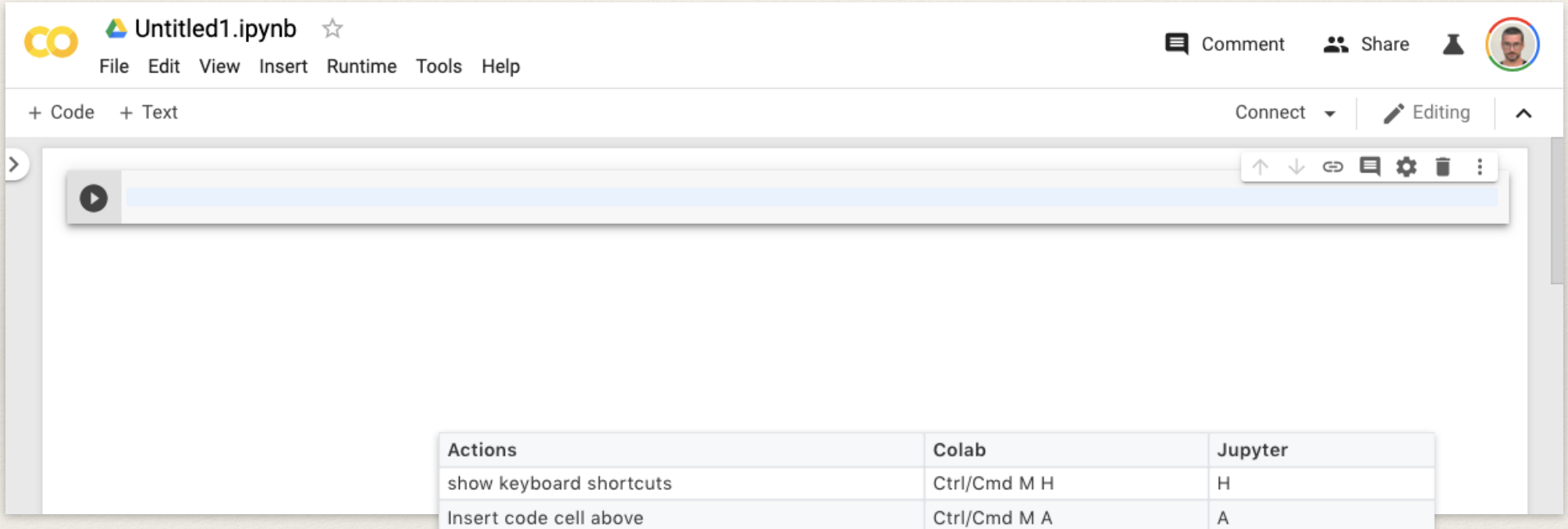
Get familiar with Google Colab

Google Colab(oratory)

https://colab.research.google.com/

(allow me oversimplifications here..)

In a nutshell: **Jupyter notebooks on the cloud**.



Keyboard shortcuts:
Google Colab vs Jupyter

Actions	Colab	Jupyter
show keyboard shortcuts	Ctrl/Cmd M H	H
Insert code cell above	Ctrl/Cmd M A	A
Insert code cell below	Ctrl/Cmd M B	B
Delete cell/selection	Ctrl/Cmd M D	DD
Interrupt execution	Ctrl/Cmd M I	II
Convert to code cell	Ctrl/Cmd M Y	Y
Convert to text cell	Ctrl/Cmd M M	M
Split at cursor	Ctrl/Cmd M -	Ctrl Shift -

How to get started on Colab

Quickest path to get started:

- **create** a new Python notebook, no installation, just **code**
  - import modules, code what you want, execute cells, ..
  - IMPORTANT: most used data sciences stuff is already installed ("**no pip, just import**")

Proper way to get started: **remember you are on the cloud!**

- **create** a new Python notebook
  - or **open an existing notebook** from gdrive, github, etc.. or upload your ipynb from your laptop
- **Save** a copy on your own gdrive, and **code** from that one
  - all future changes will be saved

Best of "Jupyter + Google drive"?

Google Colab's GUI is (intentionally) very similar to Jupyter's, plus features typical of Google Drive:

- you can share and use the notebooks like regular Google Docs
- you can create handy widgets using special comments in your code, and more

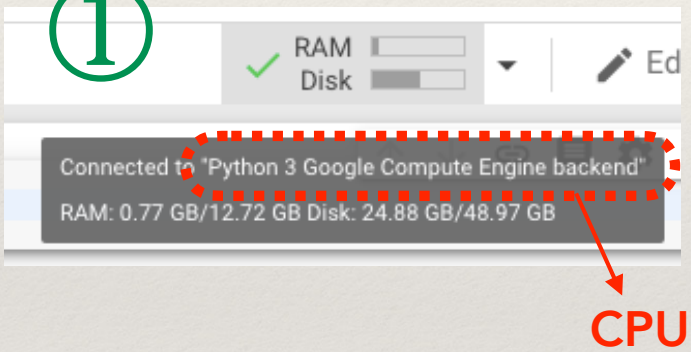
Colab: select your Runtime

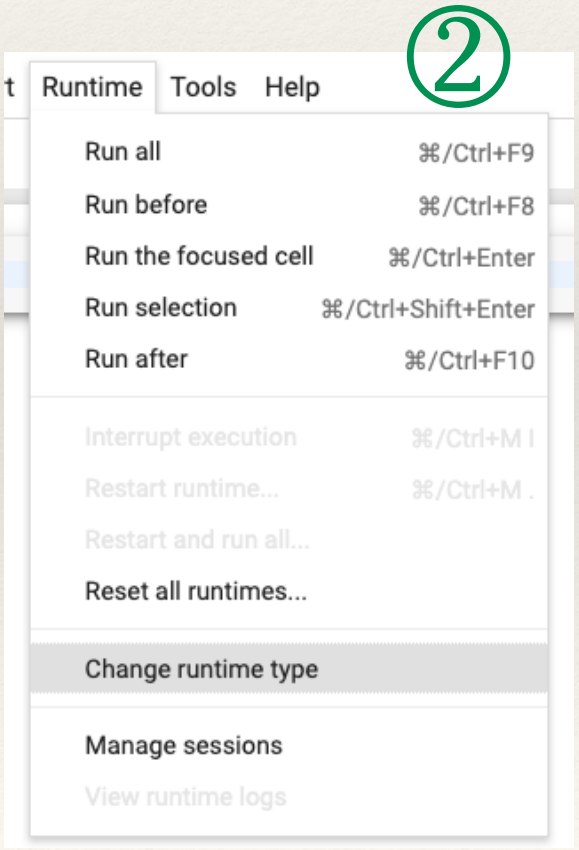
Once you open a Colab notebook, you get a Colab **Runtime**

- this is the free Google VM dedicated to you for a while

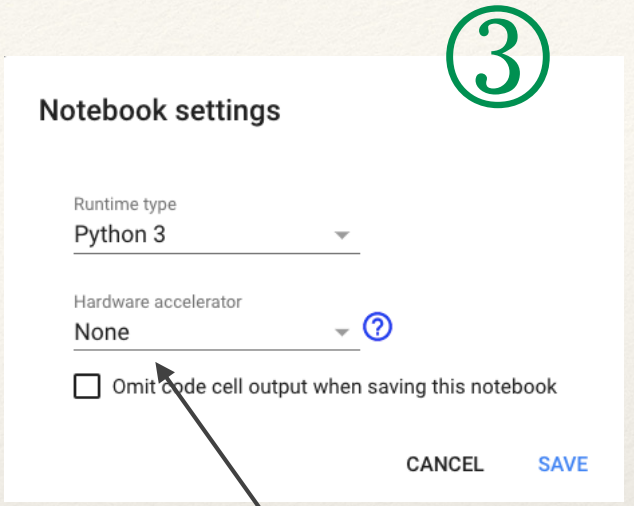
By default, the initial Runtime is CPU-only

- but you can change this:

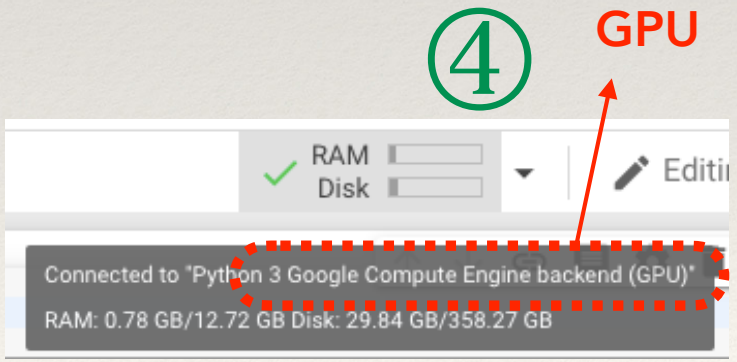




colab VM states are: **Connecting**, **Allocating**, **Initialising**, and its actual RAM/Disk monitoring widget

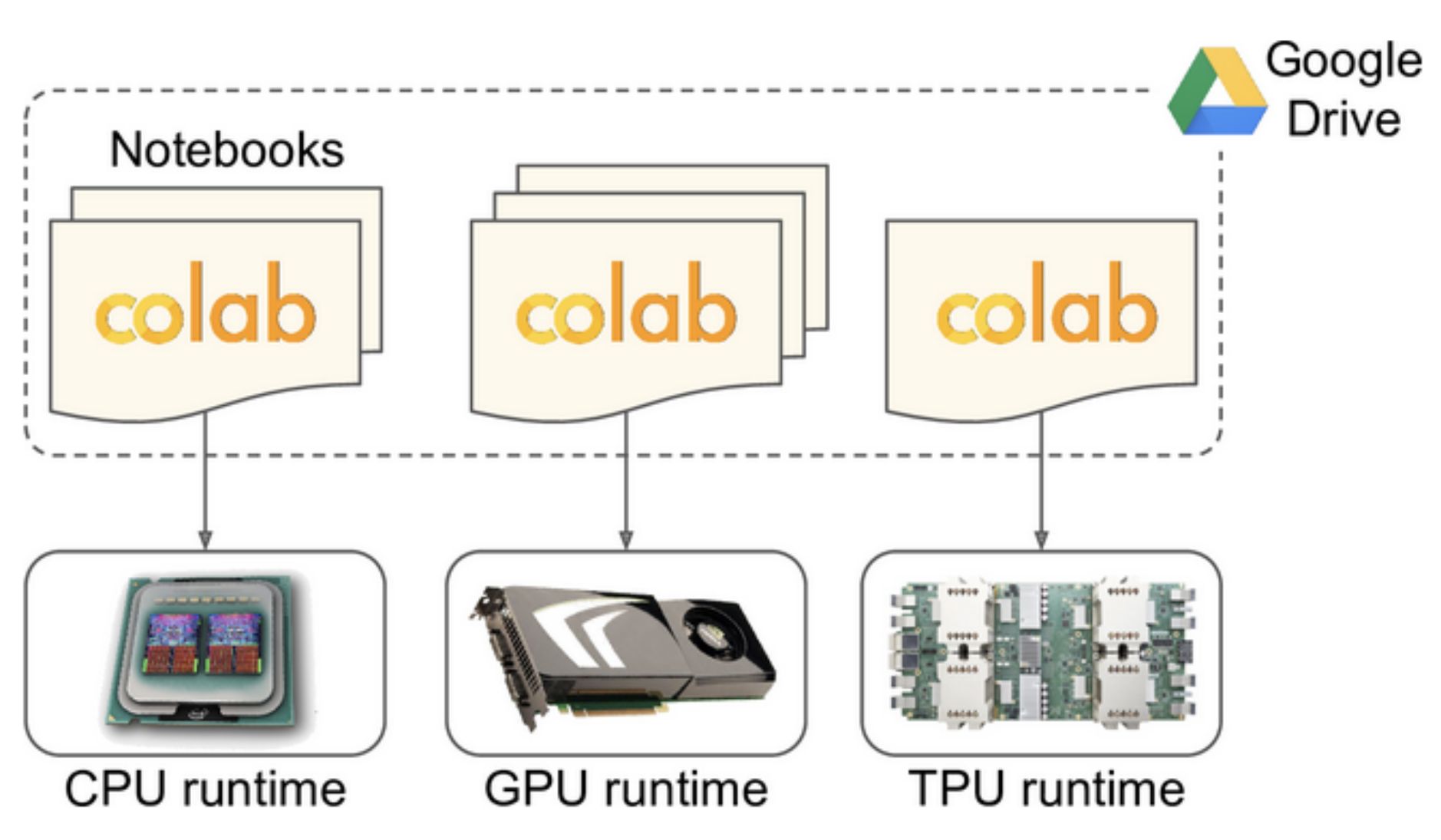


select GPU here



(NOTE: yes, also TPUs are available..)

Colab Runtimes



Lifetime of a Colab Runtime

As from the Colab FAQs:

- "Colaboratory is intended for interactive use. Long-running background computations, particularly on GPUs, may be stopped. Please do not use Colaboratory for cryptocurrency mining."

The web interface will automatically disconnect you from the Colab Runtime after **~30 minutes** of unattended connection

- you can have it again, but when you reconnect to the Colab Runtime, it may have been reset (so make sure you download what you needed before moving to something else)

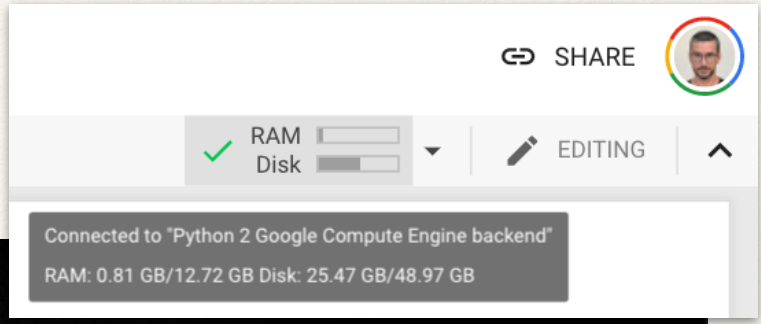
Even if you stay connected and actually code, the Colab Runtime will automatically shut down after **12 hours**

- this is meant to prevent long-running computations

Still a neat tool for experimenting on GPUs (and for free)..

Which GPU am I getting in Colab?

Tesla K80, 12 GB RAM,
up to 12 hrs in a row.



[DISCLAIMER: this information might be outdated in the future, pretty soon..]

Which GPU on Colab

Actually:

- the free version of **Colab** mostly provides Tesla K80 GPU
- **Colab pro**, the paid version ($9.99/month (*)), provides access to Tesla T4 or P100 GPUs and also other benefits like longer runtimes and priority access to TPUs

(*) as of March 2023

Additional material

Not only GPU..
the Google TPU

Specialised hw implementations for Deep Learning

— Google's (proprietary) **TPU** —

2006: Google engineers started working on deploying GPUs, FPGAs, or custom ASICs in their Data Centres (only few applications could effectively run on them, though)

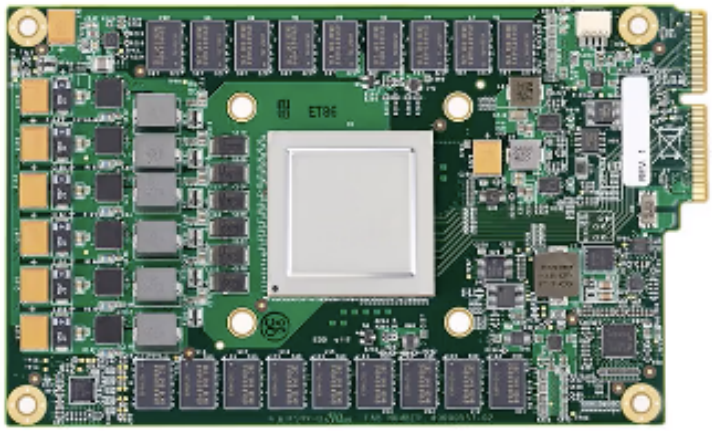
2013: projection that if 100% people used voice search for 3 mins/day using speech recognition DNNs, it would have required Google's data centres to *double in size* in order to meeting computation demands

Google started a project whose mandate was improve cost-performance by 10x over GPUs

- quickly produce a custom ASIC for inference
- bought off-the-shelf GPUs for training

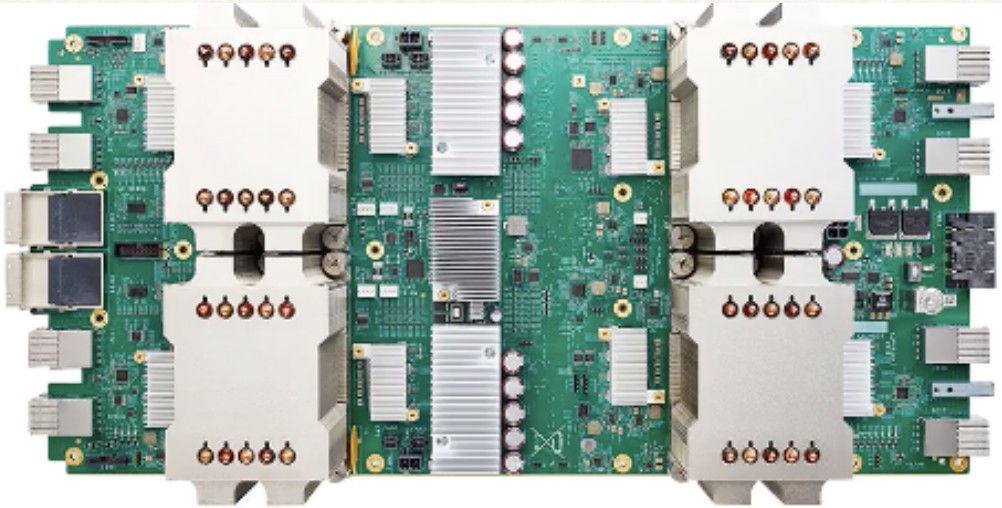
The **Tensor Processing Unit (TPU)** - Google's first custom ASIC to be used in the domain of fast inference of DNNs, programmed using the Tensorflow framework - was designed, verified, built, and deployed (first in 2015) in just 15 months (!)

Google's TPU v1 vs v2



TPU v1

Launched in 2015
Inference only



TPU v2

Launched in 2017
Inference and training

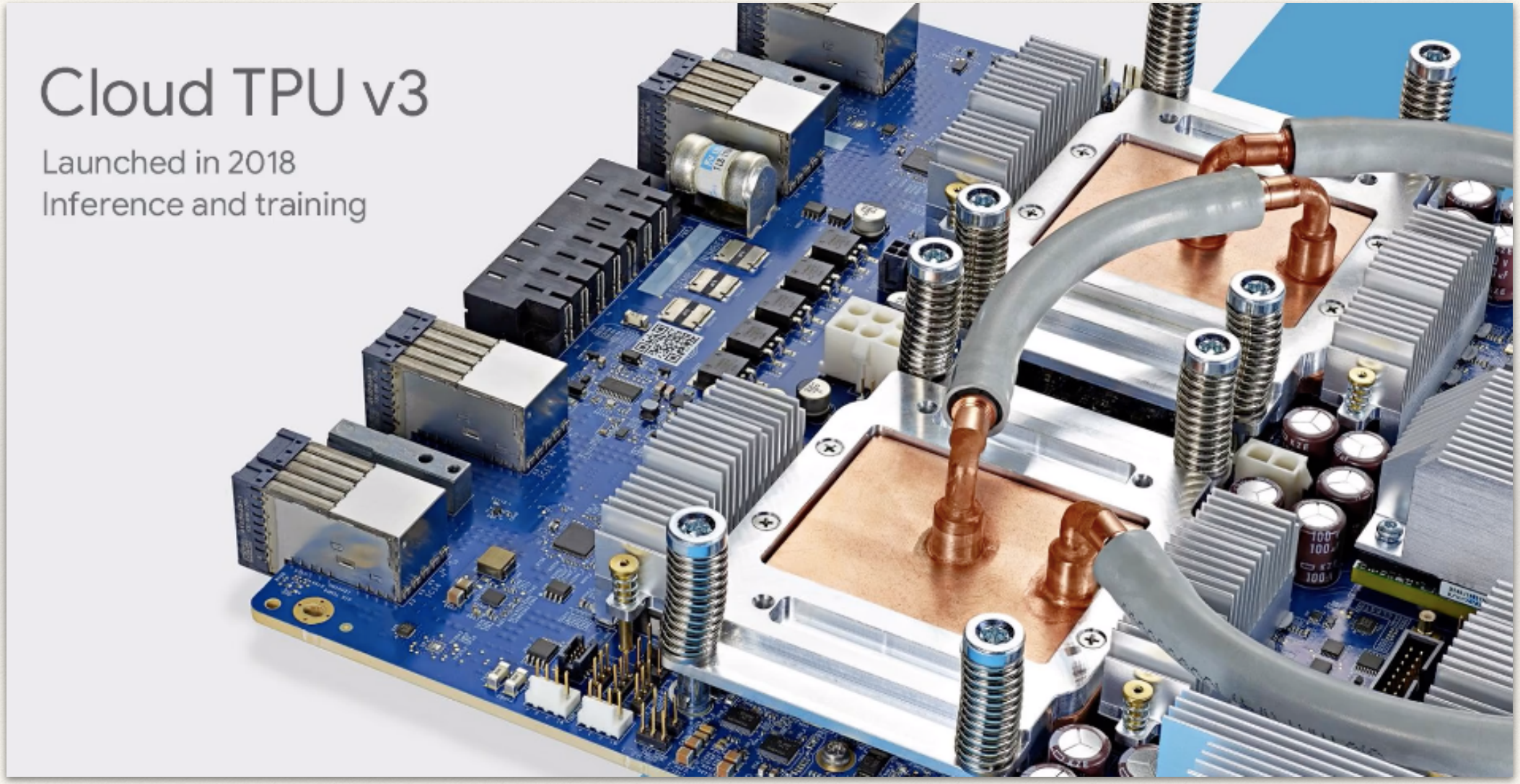
Search
Search ranking
Speech recognition

Translate
Text, graphic and speech translation

Photos
Photos search

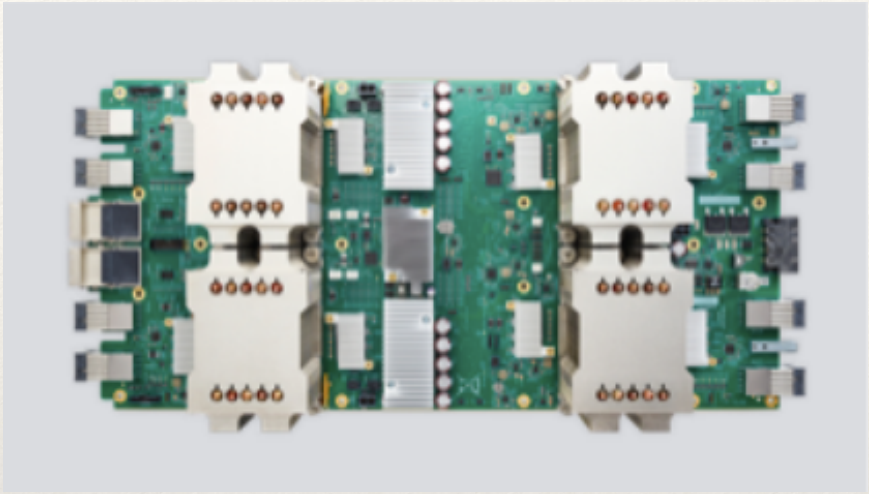
v2: both **internally** and on **Cloud**

Google's Cloud TPU v3



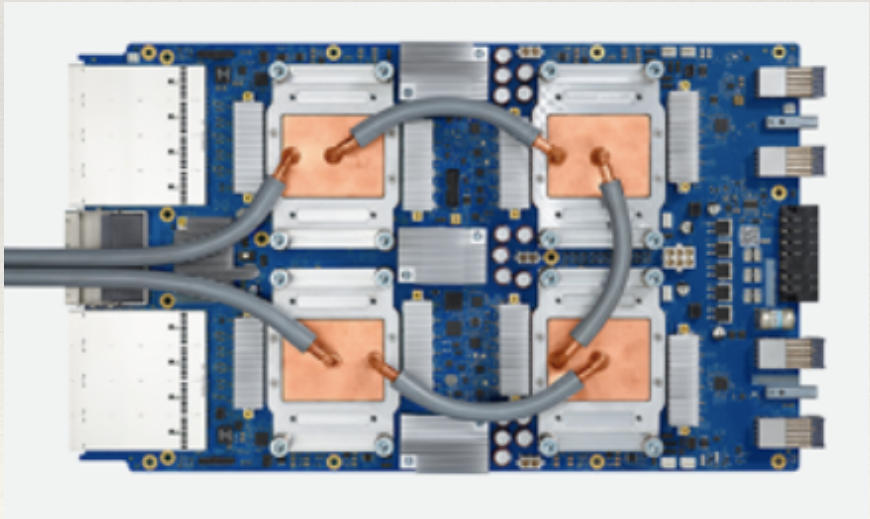
Google's TPU v2 vs v3

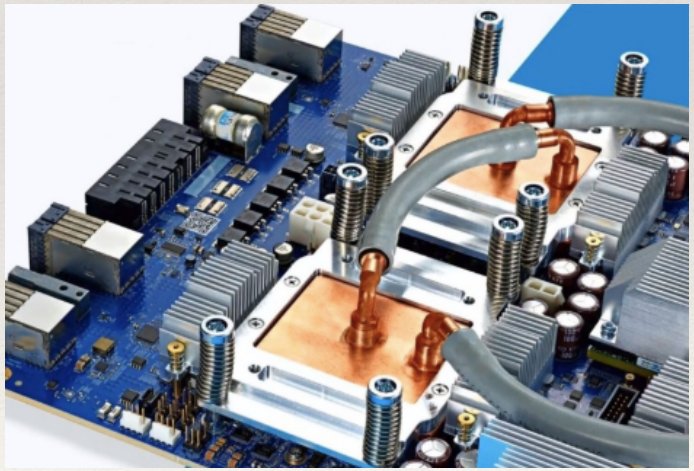
Cloud TPU v2
180 teraFLOPS
64 GB High Bandwidth Memory



>2x teraFLOPS

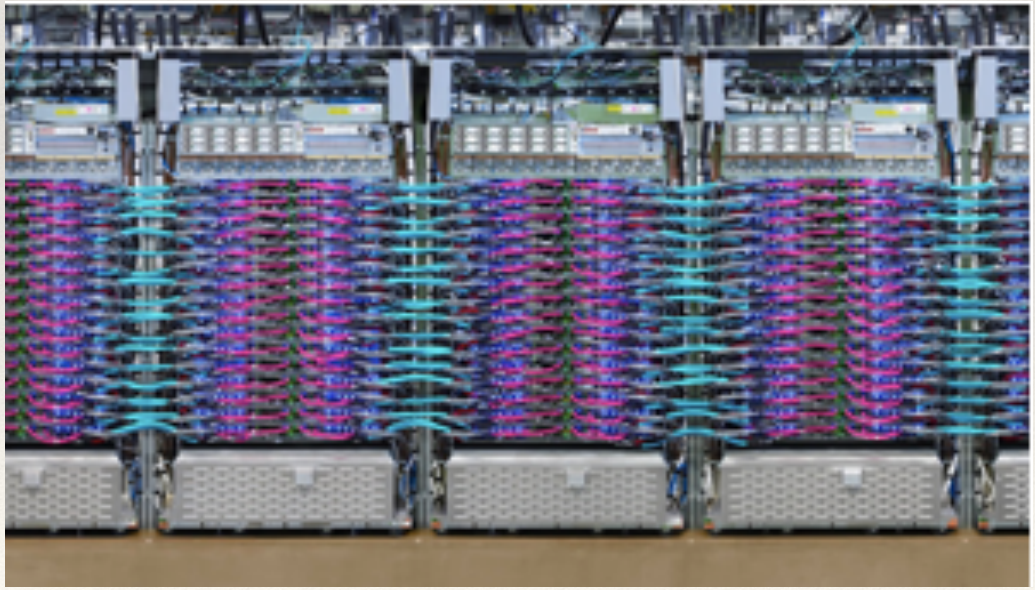
Cloud TPU v3
420 teraFLOPS
128 GB HBM





Interconnect TPUs…

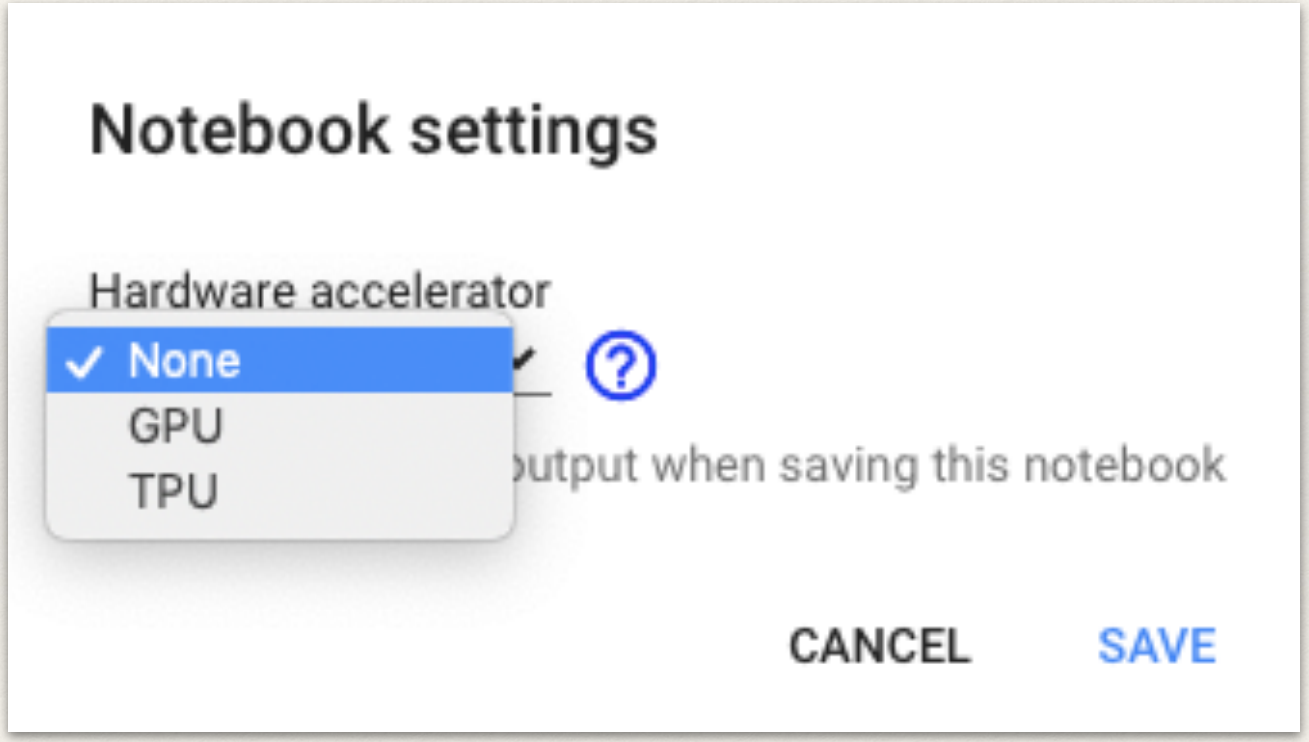
Cloud TPU v2 Pod (beta)
11.5 petaflops
4 TB HBM
2-D toroidal mesh network



Cloud TPU v3 Pod (beta)
100+ petaflops
32 TB HBM
2-D toroidal mesh network

You can actually choose:

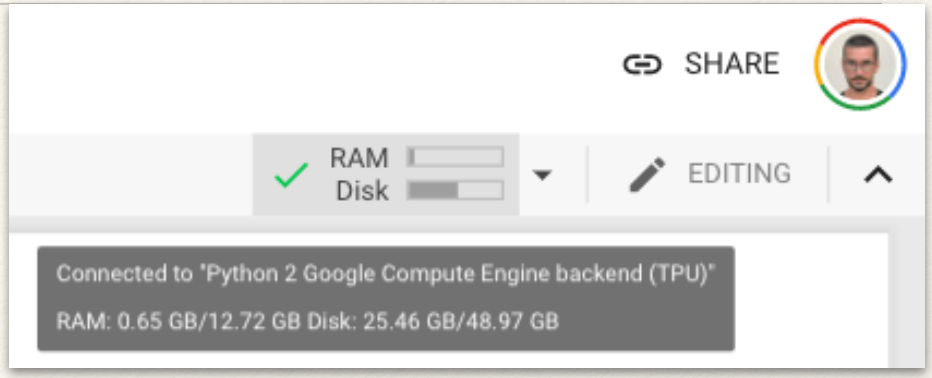
- no accelerator
- GPU
- TPU



"Why am I seeing TPU slower than GPU?"

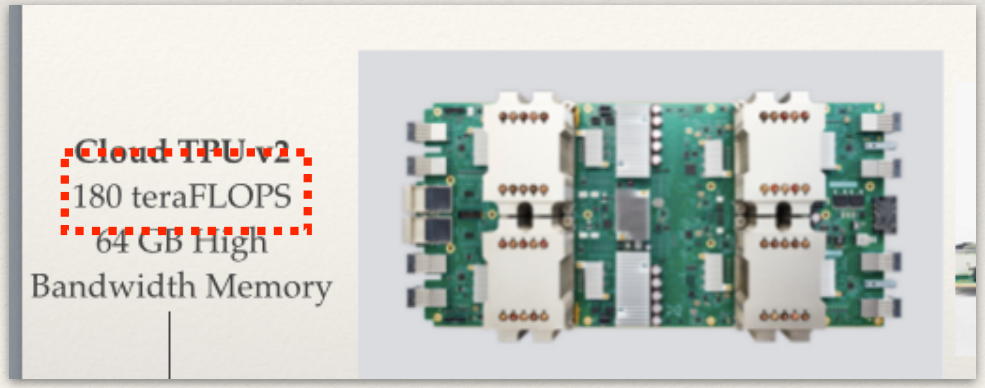
Testing != Benchmarking/Profiling ..

be careful about your conclusions!



Try some floating point computations that give you the (T)FLOPS

- e.g. https://colab.research.google.com/notebooks/tpu.ipynb




```
[3]  N = 4096
     COUNT = 100
     import time

     def flops():
       x = tf.random_uniform([N, N])
       y = tf.random_uniform([N, N])
       def _matmul(x, y):
         return tf.tensordot(x, y, axes=[[1], [0]]), y

       return tf.reduce_sum(
         tf.contrib.tpu.repeat(COUNT, _matmul, [x, y])
       )

     tpu_ops = tf.contrib.tpu.batch_parallel(flops, [], num_shards=8)

     session = tf.Session(tpu_address)
     try:
       print('Warming up...')
       session.run(tf.contrib.tpu.initialize_system())
       session.run(tpu_ops)
       print('Profiling')
       start = time.time()
       session.run(tpu_ops)
       end = time.time()
       elapsed = end - start
       print(elapsed, 'TFlops: {:.2f}'.format(1e-12 * 8 * COUNT * 2*N*N*N / elapsed))
     finally:
       session.run(tf.contrib.tpu.shutdown_system())
       session.close()

Warming up...
Profiling
0.6749346256256104 TFlops: 162.91
```

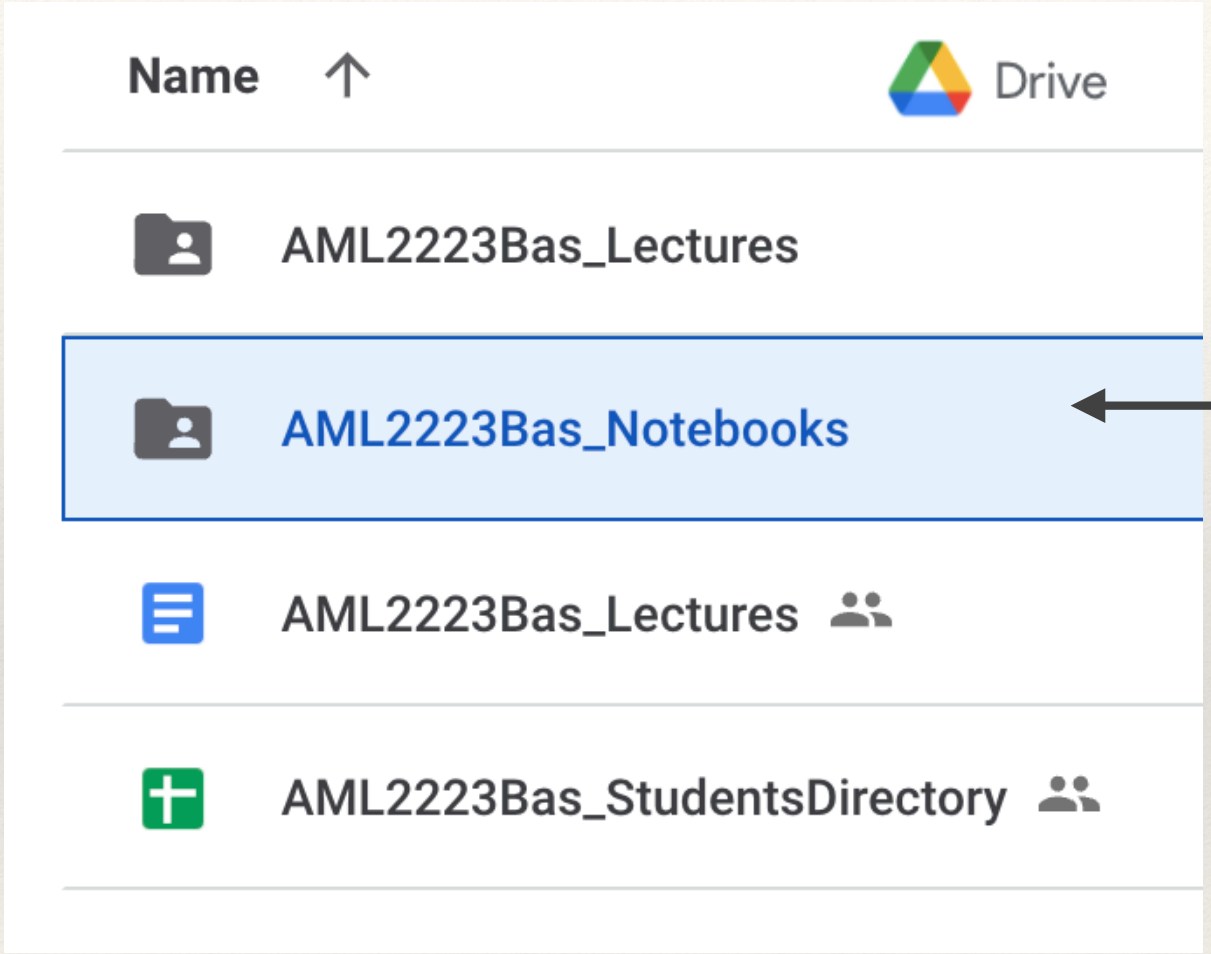
If you want to know more about a TPU

Google TPU demo

[URL: https://storage.googleapis.com/nexttpu/index.html]

Hands-on:

Python 101 notebook on Jupyter



Python101.ipynb

Exercise, and keep this as a bonus for any future reference!


Logistic Regression
Classification and representation
Classification

In **classification** problems, the variable **y** that you want to predict tells you if your example belongs to a "class" or not.

We will discuss an algorithm called **logistic regression**

- one of the most popular and most widely used learning algorithms today

Note: more on the its name later..

Classification

Examples:

1. email spam classification: spam / not spam?
    - Example: mail spam classifier, x(i) may be some features of a piece of email, labels are y∈{0,1}, i.e. y may be 1 if it is spam mail (positive class), and 0 otherwise (negative class)
2. tumors: malignant/benign?
3. classifying online transactions: fraudulent y/n?
4. .. many more! ..

What is common among all these problems?

- the variable that we are trying to predict is a **y** that can take on two values

Basically, **0** or **1**. Two **classes** (i.e. "classification"). A **negative class** and a **positive class**.

Classification

BINARY CLASSIFICATION

$y \in \{0, 1\}$

NEGATIVE class

POSITIVE class

MULTICLASS CLASSIFICATION

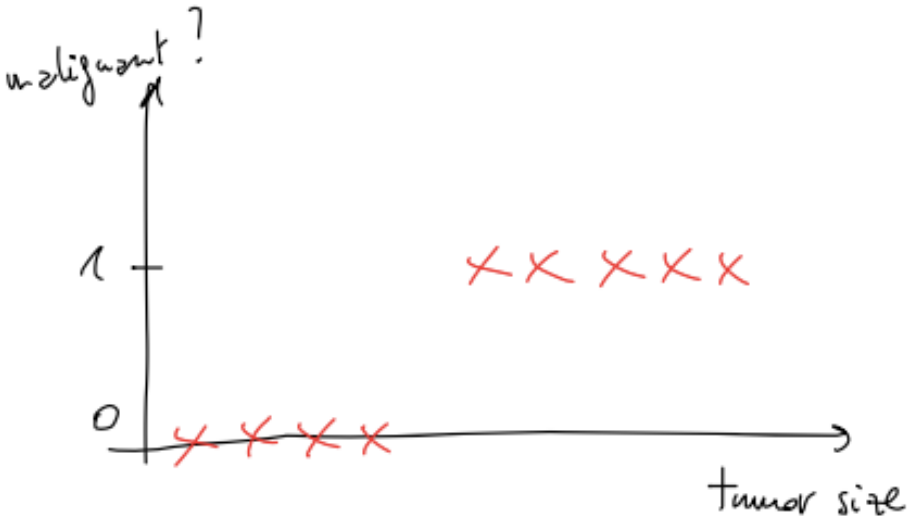
$y \in \{0, 1, \ldots, n\}$

Classification examples

NOTE: The assignment of the 2 classes - **negative** and **positive** - to **0** and **1** is somewhat arbitrary

- often there is an intuition that a negative class is conveying the absence of something..
  - (e.g. the absence of a malignant tumour)
- .. whereas the positive class is conveying the presence of something
  - (e.g. the presence of a malignant tumour - i.e. like in medical exams in general, actually, where you are positive if you indeed have something
- we will follow some intuition in this, but to be clear: it doesn't matter

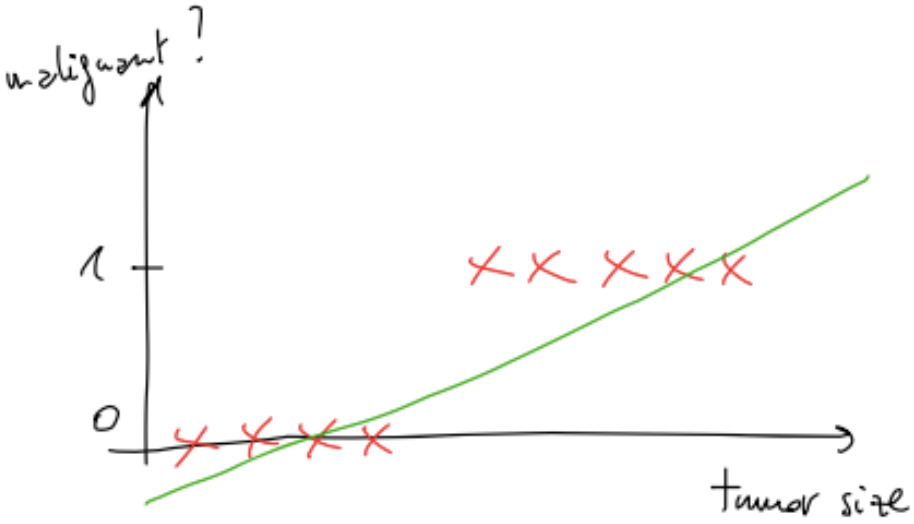
Applying linear regression to a classification problem



NOTE: not using different symbols, as their "true" value is known by where the data point is on the plot

Let's try to apply linear regression to this problem.
Do you think it would work?

Applying linear regression to a classification problem



assume you run a linear regr.

$h_\theta(x) = \theta^T x$

How do you predict ?

Applying linear regression to a classification problem

Assume you run a linear regr.

$h_\theta(x) = \theta^T x$

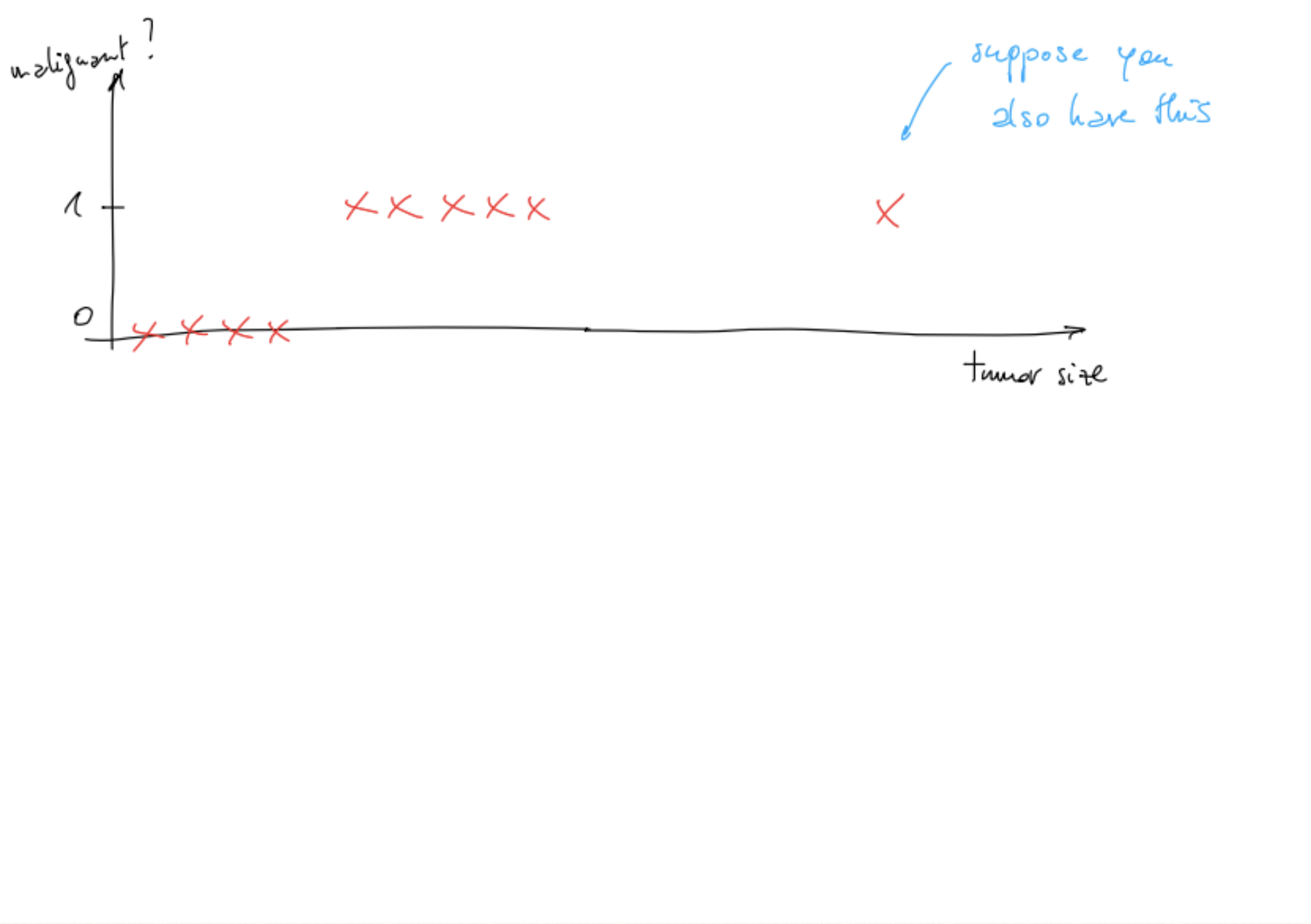
How do you predict?

Threshold classifier output $h_\theta(x)$ at 0.5:

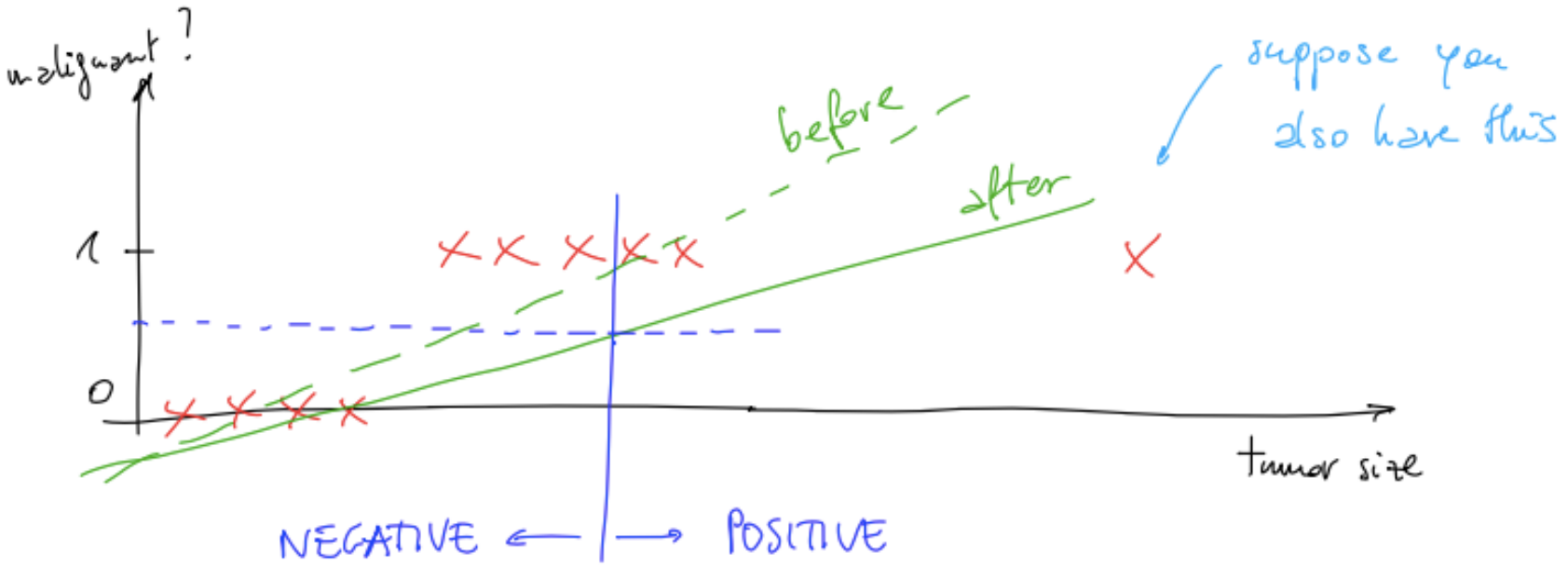
$$h_\theta(x) \begin{cases} \geq 0.5 \Rightarrow \text{predict } "y = 1" \\ < 0.5 \Rightarrow \text{predict } "y = 0" \end{cases}$$

Is this good enough? And what if I have more data?

Applying linear regression to a classification problem

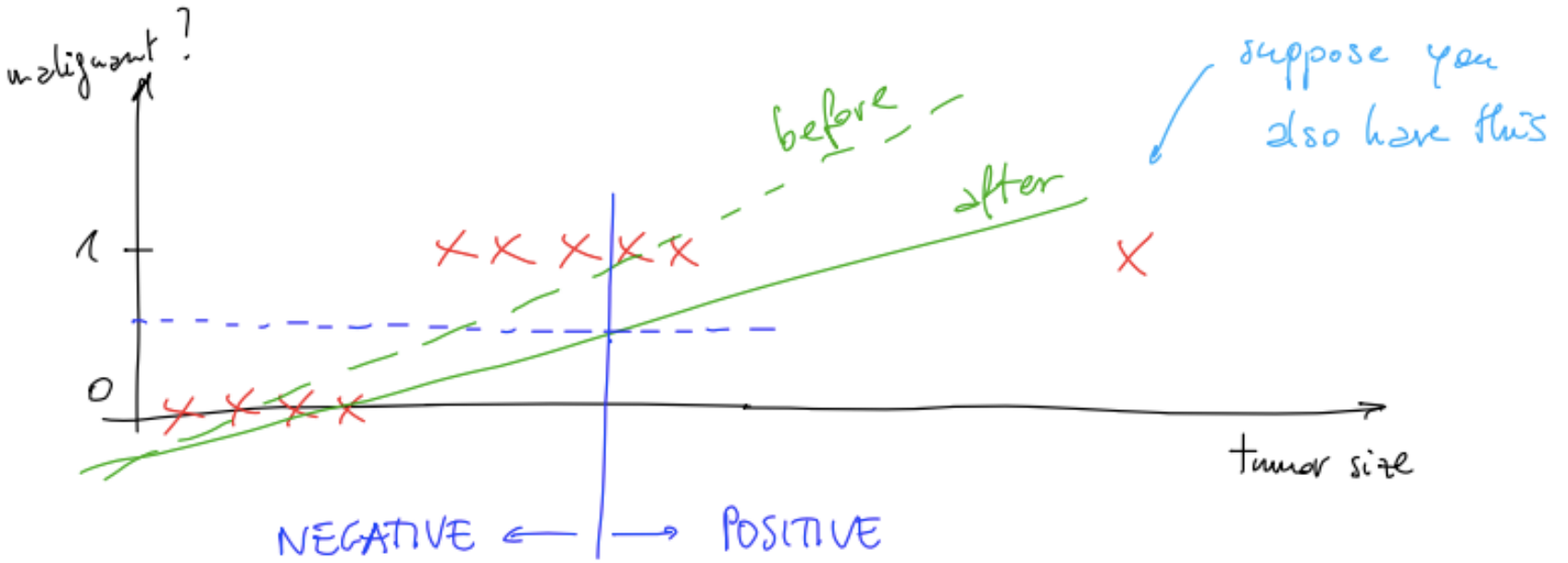


Applying linear regression to a classification problem



One example (that was clear) caused a mess !

Applying linear regression to a classification problem



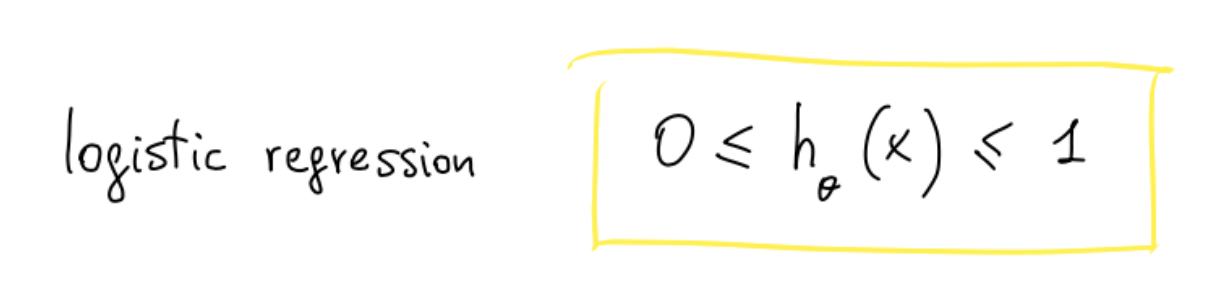
One example (that was clear) caused a mess !

Applying linear regression to a pure classification problem *usually* is **not** a promising approach

- additionally, your hypothesis can also yield values <0 and >1, weird for a binary classification problem (should boil down to either 0 or 1, nothing in-between, and for sure nothing outside the interval [0,1] !)

Logistic regression

We will introduce a simple algorithm called



And **logistic regression** is - and we will use it as - a **classification** algorithm.

NOTE: It is maybe sometimes confusing that the term "regression" appears in this name even though logistic regression is actually a "classification" algorithm.

- But that's just a name it was given for historical reasons (you'll see). So don't be confused..
- See it like this: a classification problem is just like a regression problem, except that the values you want to predict take on only a small number of discrete values
  - e.g. predict the probably of raining (regression) but then take the umbrella only in prob >0.8 (i.e. use the result of regression as a classification for your purposes)

Logistic regression is a classification algorithm that we apply to settings where the label y has discrete values, e.g. when it is either 0 or 1.

Summary

So far, we have a binary classification problem in which **y** could take on only two values, 0 and 1

- most of what we said will also generalise to the multiple-class case

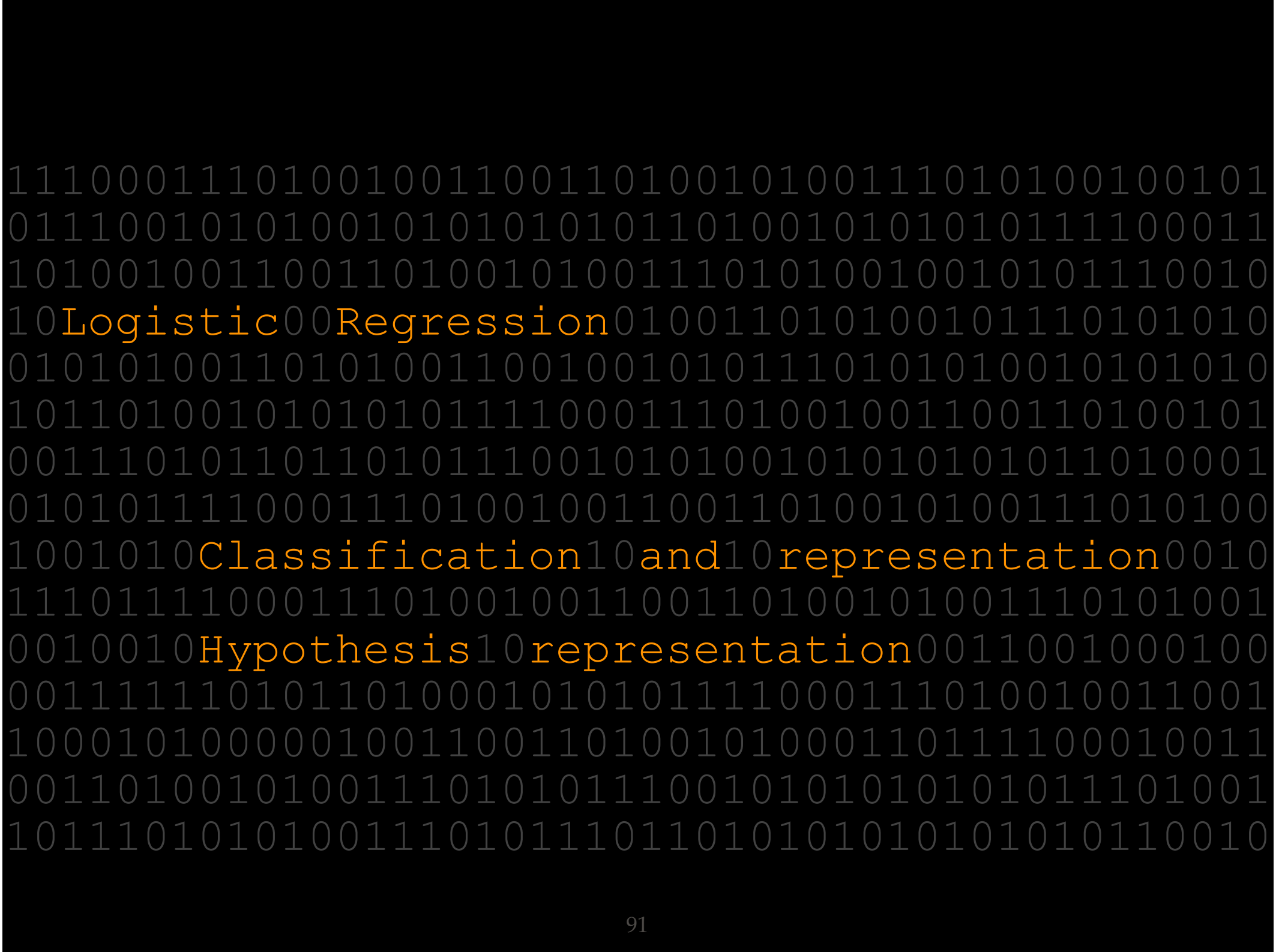
A first attempt at classification was towards trying linear regression

- i.e. mapping predictions greater than 0.5 as a 1 and smaller than 0.5 as a 0.

However, this method did not work well: classification is actually not dealt with well by a linear function

- As we tried to approach the classification problem just ignoring the fact that **y** is discrete-valued and using a linear regression algorithm to try to predict **y** given **x** as if it was a regression problem, we soon realised how easy it is to construct examples where this approach performs very poorly.
- In addition, we would have been using an approach that allow **h** to be larger than 1 or smaller than 0, when we know that it could be $y \in \{0, 1\}$

It seems we may benefit indeed from something like **logistic regression** to attack classification problem.


Logistic Regression
Classification and representation
Hypothesis representation

Logistic regression: hypothesis representation

I.e. what is the function **h** we can use to represent our hypothesis in the case of a logistic regression problem?

In other words, we need to move beyond this:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

For logistic regression, we will be trying a function **g** of **h** above..

Logistic regression: sigmoid

LOGISTIC REGRESSION MODEL

I want $0 \leq h_\theta(x) \leq 1$

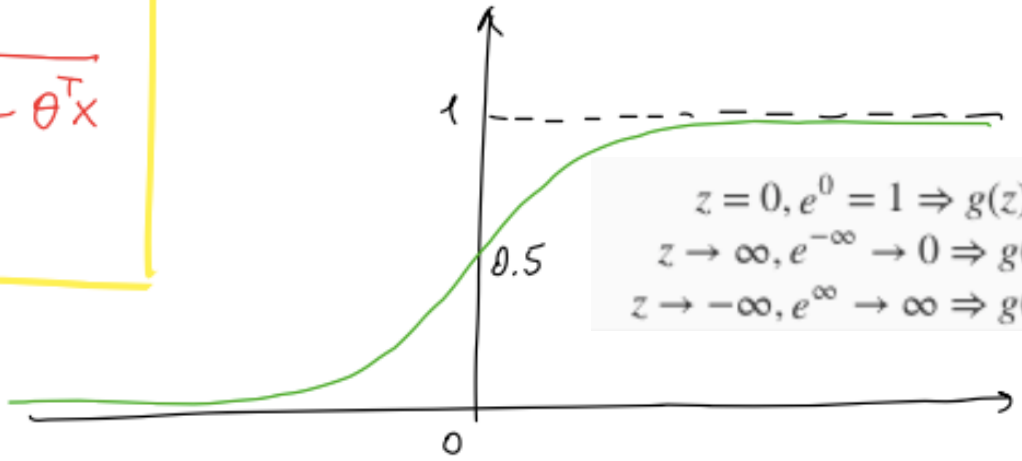
$$h_\theta(x) = g(\theta^T x)$$

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

SIGMOID FUNCTION (or LOGISTIC)

origin of the name...

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



$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$

$z \to \infty, e^{-\infty} \to 0 \Rightarrow g(z) \to 1$

$z \to -\infty, e^{\infty} \to \infty \Rightarrow g(z) \to 0$

Logistic regression: **g**(**h**) is the new **h**

Moving from the "old" **h** to "new" **g**(**h**) in classification tasks requires some thinking to understand:

How do we interpret now the **h**?

- before, **h** was my guess for y values.. and now?

How to re-interpret the **h** output for the classification task

$h_\theta(x)$ = estimated probability to get $y=1$ on a new input example x

example: tumors, $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$ feature vector for a patient

$h_\theta(x) = 0.7$ means that patient has 70% prob. that tumor is malignant

You can formalize this as:

$$h_\theta(x) = P(y=1 \mid x; \theta)$$

"my hypothesis can be interpreted as probability that $y=1$, given features x, parametrized by θ"

Deal with both positive/negative classes

We need to state something for 0 too, as:

$$P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$$

$$\Rightarrow \quad P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta)$$

Let's digest this..

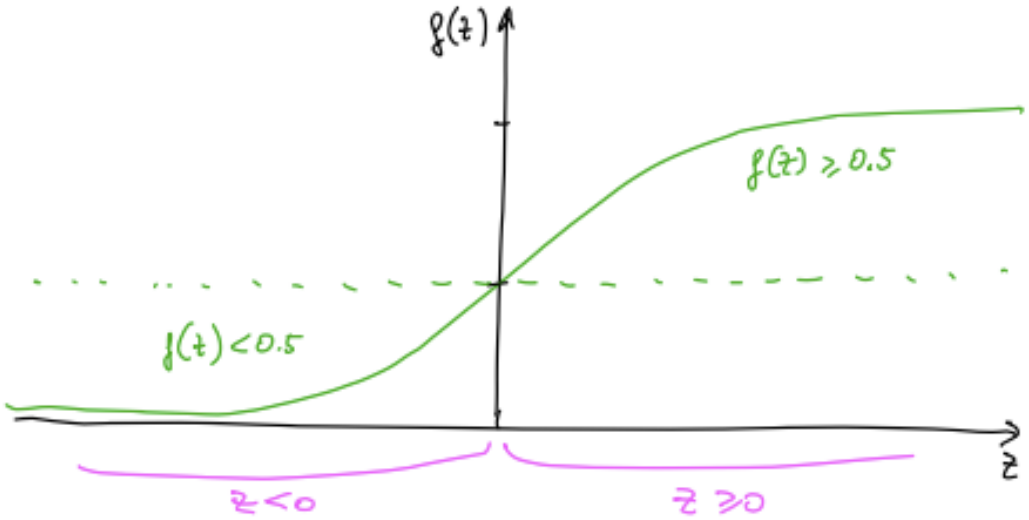
$h_\theta(x) = g(\theta^T x)$ ⟶

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

- predict "$y=1$" if $h_\theta(x) \geq 0.5$

quite random ...

- predict "$y=0$" if $h_\theta(x) < 0.5$



$\theta^T x \geq 0$

$\theta^T x < 0$

From $h_{(old)}$ to $h_{(new)}=g(h_{(old)})$

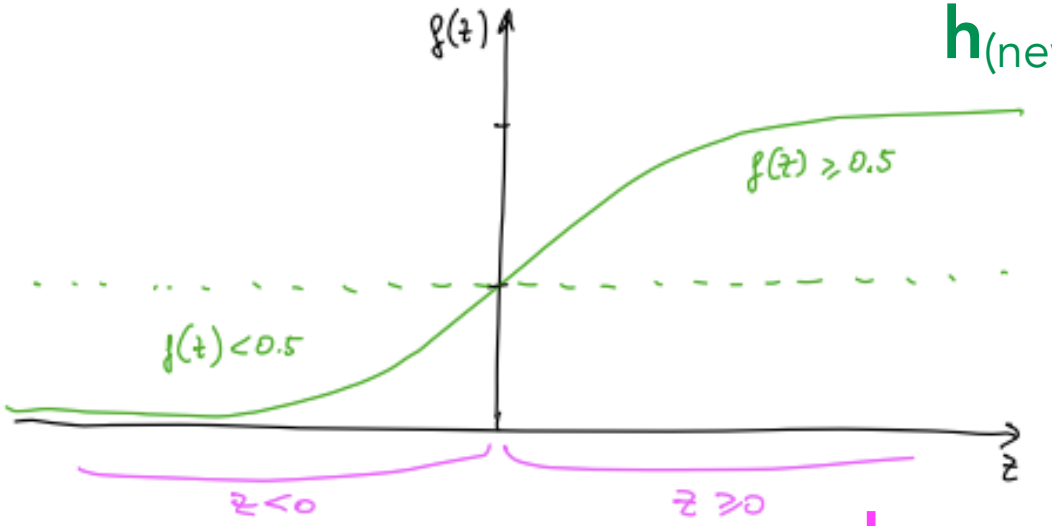
$h_\theta(x) = g(\theta^T x)$ ⟶

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

- predict "$y=1$" if $h_\theta(x) \geq 0.5$
- predict "$y=0$" if $h_\theta(x) < 0.5$

$h_{(new)}$

quite random ...



$h_{(new)}$

$h_{(old)}$

$\theta^T x \geq 0$

$h_{(old)}$

$\theta^T x < 0$

Summary

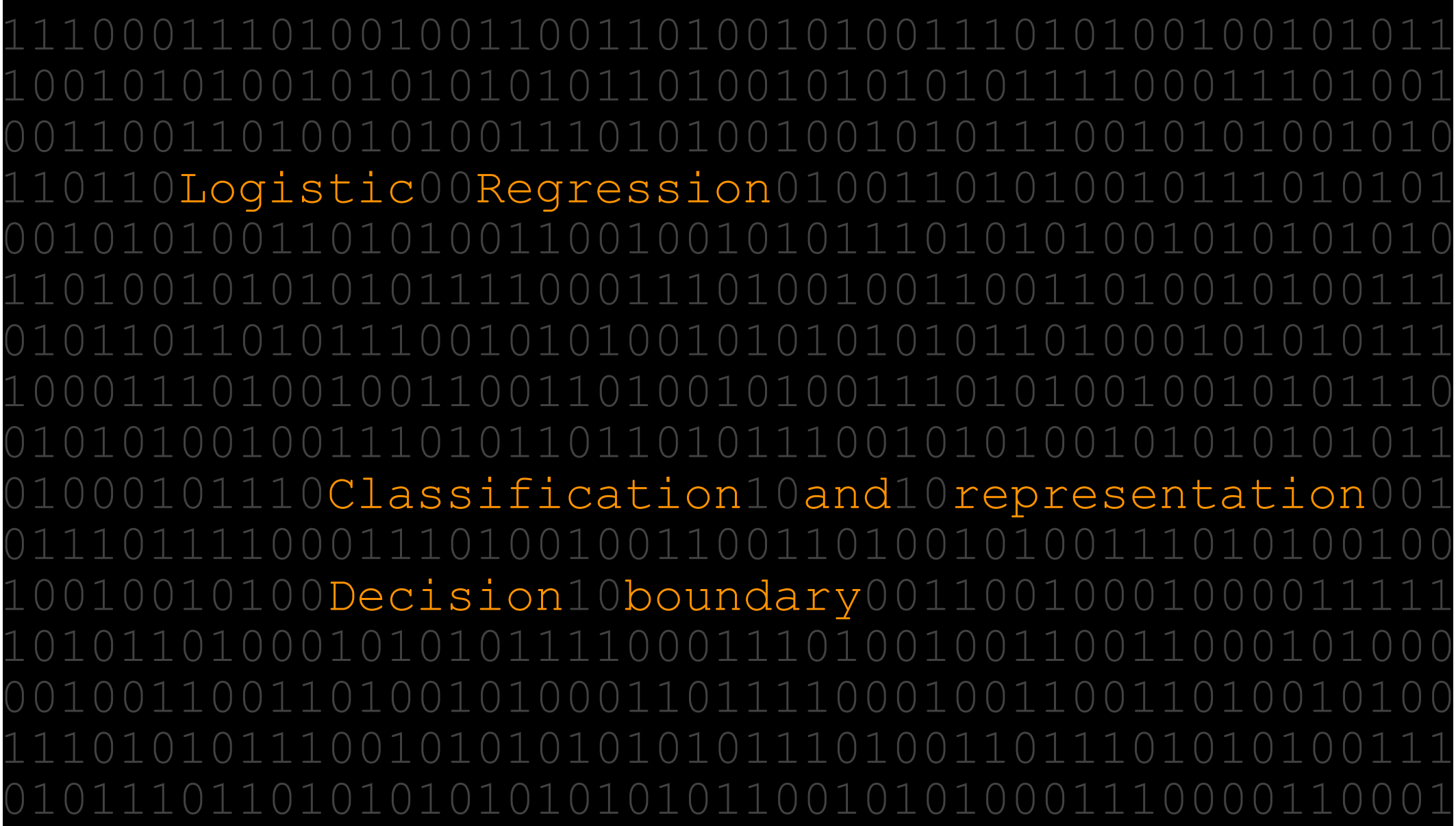
We changed the form for our hypotheses **h** in a way it is more solid against additional data points, and by the way it also resumes the fact it belongs to the [0, 1] range

We did so by plugging the "old" **h** itself into the **logistic function** that becomes the "new" **h** (which we called **g(z)**).

The **"new" h** maps any real number into the [0, 1] interval

- hence making it easy for transforming an arbitrary-valued function into a classification-friendly function

We should interpret **h** as the **probability** that our output is 1, with the prediction it is 0 being its complement.


Logistic Regression
Classification and representation
Decision boundary

Decision boundary

To get a better sense of what the logistic regressions hypothesis function is actually computing, we need to introduce the concept of **decision boundary**

- we briefly introduce its concept, now we see it in some more details

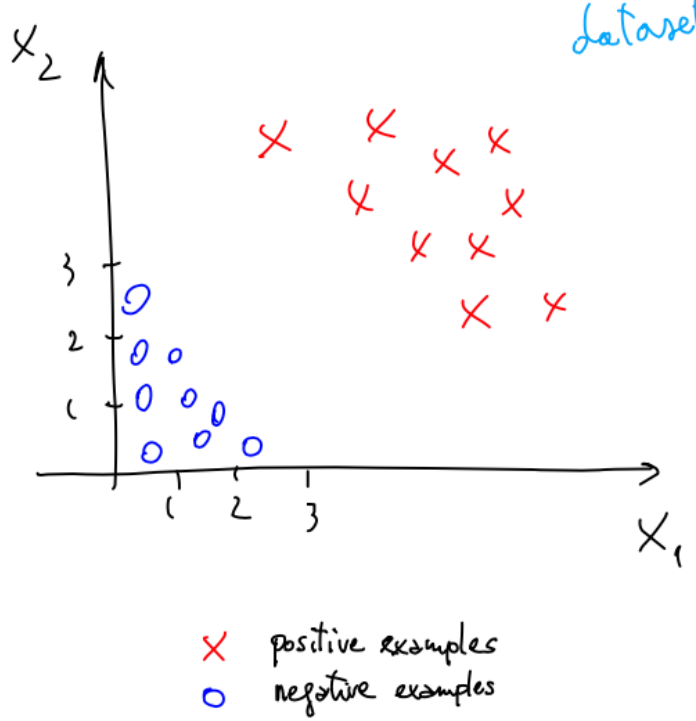
We discuss:

- linear decision boundary
- non-linear decision boundary

How logistic regression makes its predictions

Example:

Suppose this dataset:



Suppose this hypothesis:

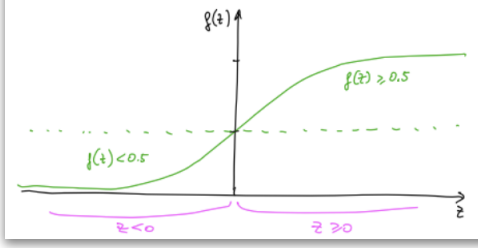
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Suppose we already fit the values of θs:

-3 $+1$ $+1$ $\Rightarrow \theta = \begin{pmatrix} -3 \\ +1 \\ +1 \end{pmatrix}$

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

- predict "$y=1$" if $h_\theta(x) \geq 0.5$ (quite random...)
- predict "$y=0$" if $h_\theta(x) < 0.5$

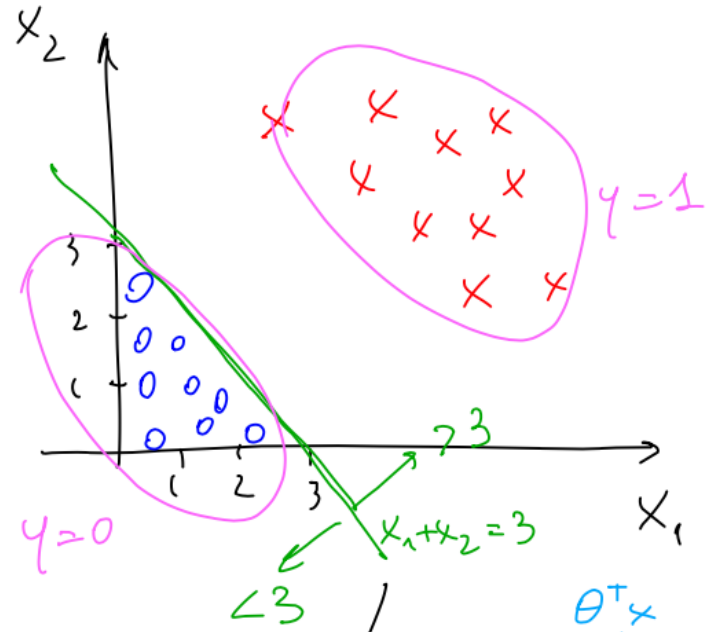


$\theta^T x \geq 0$

$\theta^T x < 0$

How logistic regression makes its predictions

Example:



Suppose this hypothesis:

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

Suppose we already fit the values of θs:

-3 $+1$ $+1$ $\Rightarrow \theta = \begin{pmatrix} -3 \\ +1 \\ +1 \end{pmatrix}$

predict "$y=1$" if $\underbrace{-3+x_1+x_2}_{\theta^T x} \geq 0$ ie. $x_1+x_2 \geq 3$

< 0 < 3

→ called DECISION BOUNDARY: line that separates region that predicts $y=0$ from " " $y=1$

Decision boundary: a property of the h

So, the decision boundary is, in the feature space, the "border" between the regions where we predict y =1 vs the regions where we predict y = 0

Note: **the decision boundary is a property of the hypothesis**, including the parameters θ. It is *not a property of the training set*.

- i.e. it is still of the same type (e.g. a line) even if e.g. we would change all data points in the training set
  - Later on we will talk about how to fit the θs and there we'll end up using the training set, i.e. using our data: the data drives the training, not the type of decision boundary.

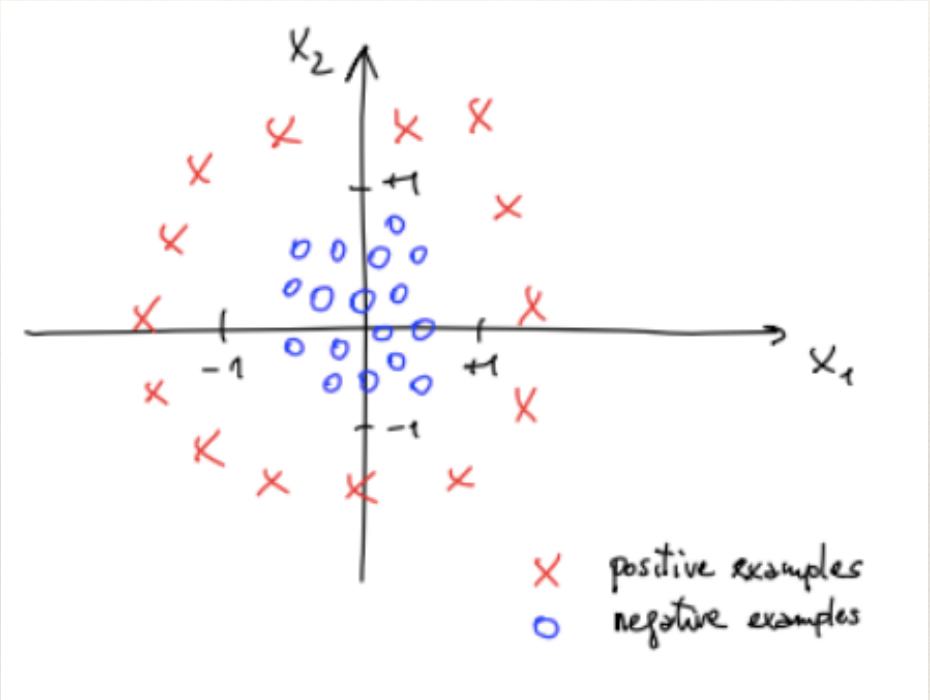
Recap:

- **the training set drives the training and it is used to fit the θ parameters…**
- …but once you have the θs for your hypothesis, it is **the hypothesis** that **defines the type of decision boundary**
  - in other words, you don't actually need to plot any training set in order to figure out the decision boundary - just study your hypothesis! In this way, a decision boundary linked to an hypothesis applies to other datasets too

Non-linear decision boundary

Let's now look at a more complex example.

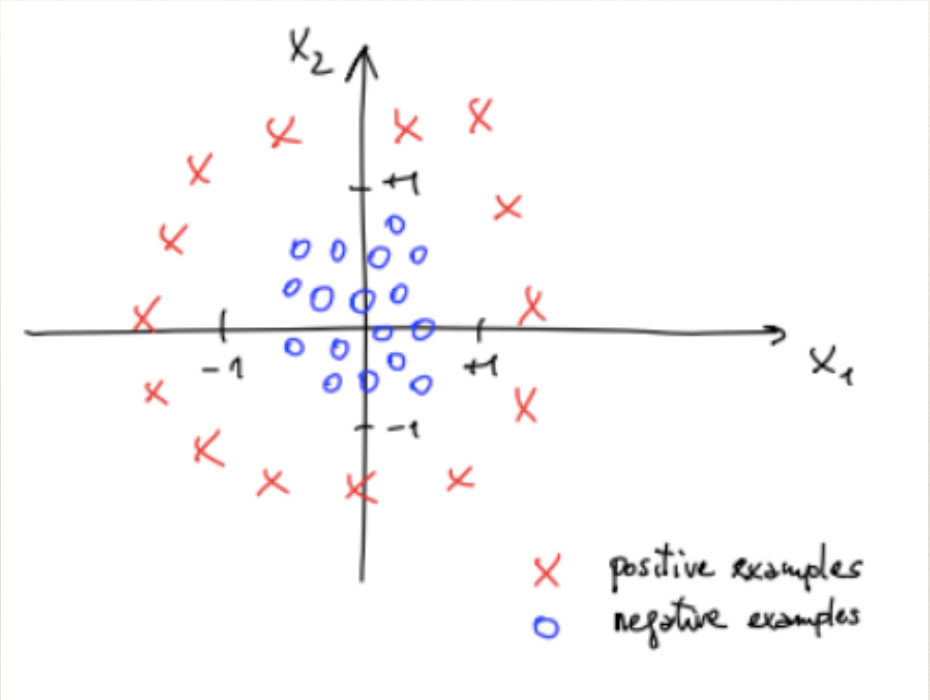
Given a training set like the one on the right, how can I get logistic regression to fit this data?



Non-linear decision boundary

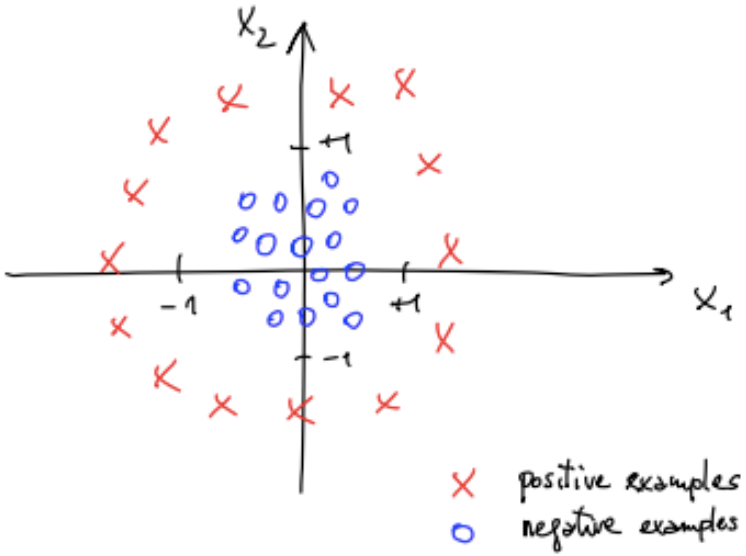
Let's now look at a more complex example.

Given a training set like the one on the right, how can I get logistic regression to fit this data?



Idea: in *regression*, we discussed about polynomial regression, i.e. adding extra higher order polynomial terms to the features vector. Why not doing the same for *logistic regression*?

Non-linear decision boundary



Suppose this hypothesis:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4)$$

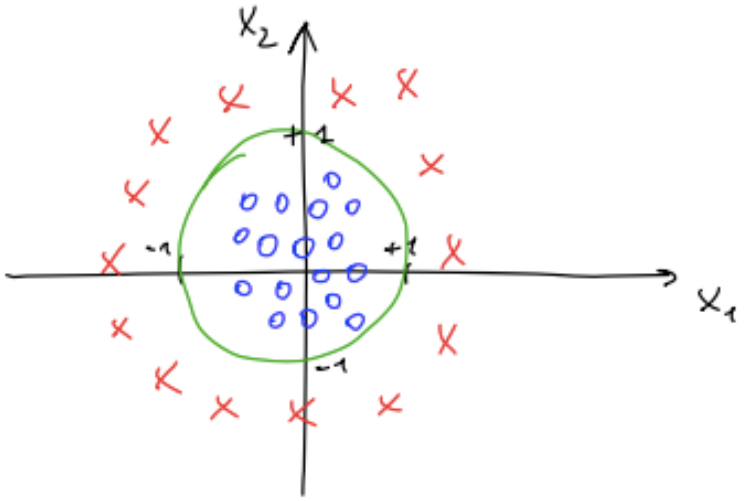
$x_3 = x_1^2$, $x_4 = x_2^2$

Suppose we already fit the values of θs:

-1, 0, 0, 1, 1

$$\theta = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Non-linear decision boundary



Suppose this hypothesis:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4)$$

$x_3 = x_1^2$, $x_4 = x_2^2$

Suppose we already fit the values of θs:

$\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

$$\theta = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

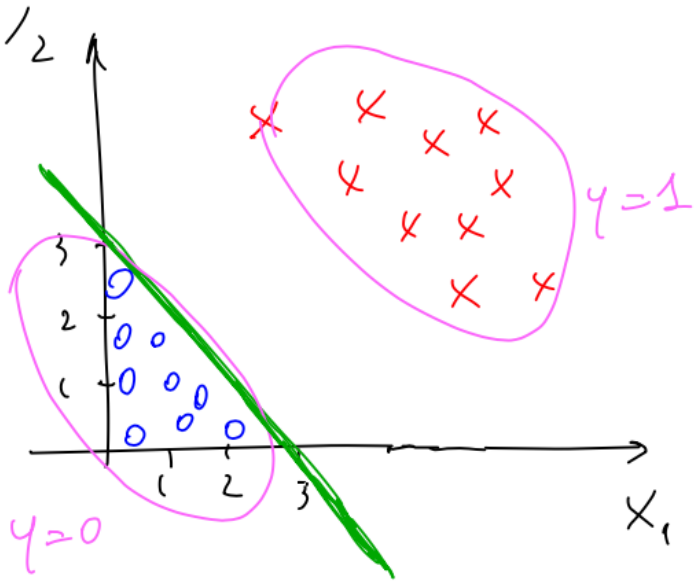
predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

predict "$y = 0$" if $-1 + x_1^2 + x_2^2 < 0$

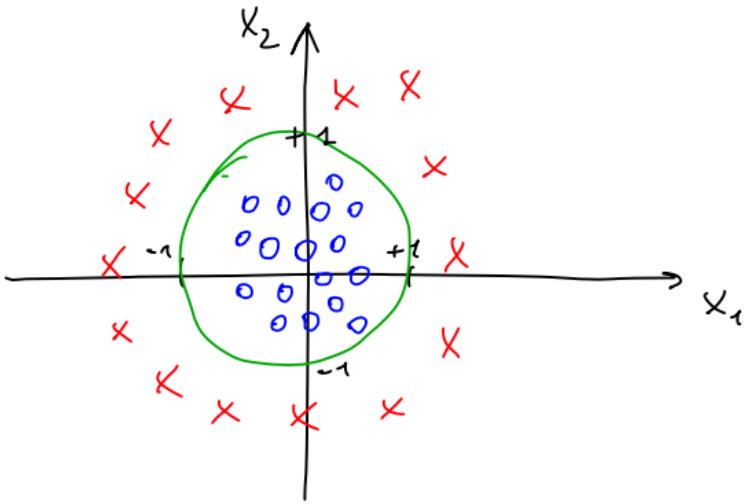
$\Downarrow$

$x_1^2 + x_2^2 = 1$ is the decision boundary

Decision boundaries: property of.. ?



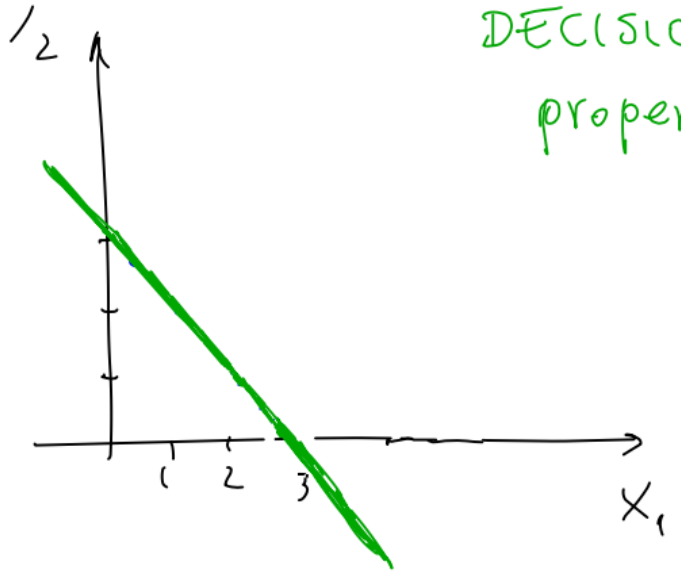
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4)$$

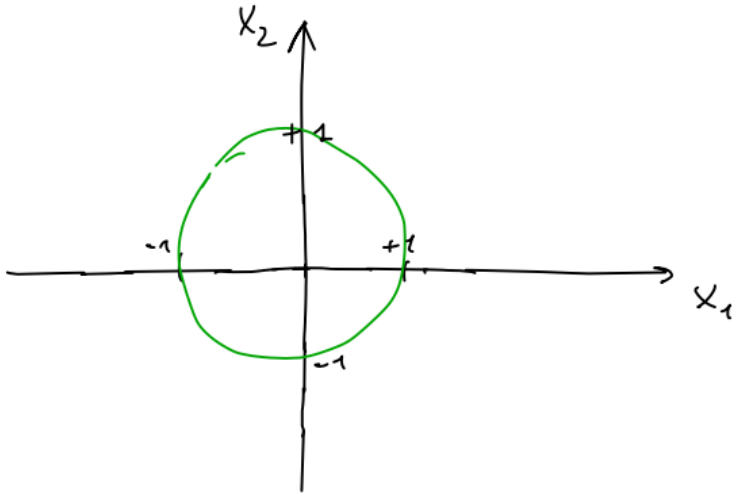
$x_3 = x_1^2$ $\quad$ $x_4 = x_2^2$

Decision boundary: a property of the h



DECISION BOUNDARIES as properties of the hypothesis !

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



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4)$$

$x_3 = x_1^2$ $\quad$ $x_4 = x_2^2$

Summary

In order to get our discrete 0 or 1 classification, we translated the output of the hypothesis **h** function into a logistic function **g**.

- When its input is greater than or equal to zero, its output is greater than or equal to 0.5.

This turns out to be as follows:

$\theta^T x \geq 0 \Rightarrow y=1$

$\theta^T x < 0 \Rightarrow y=0$

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. **It is created by our hypothesis function**.

The input to the sigmoid function g(z) (e.g. $\theta^T x$) doesn't need to be linear

- it could be a function that describes e.g. a circle (e.g. $z=\theta_0+\theta_1 x_1^2+\theta_2 x_2^2$) or any shape to fit your data


Logistic Regression
Logistic regression model
Cost function

Cost function for logistic regression

Let's discuss how to fit the parameters θ for logistic regression.

In particular, let's define the optimisation objective, i.e. the **cost function** that we will use to fit the parameters.

Fitting a logistic regression model

training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ m examples

vectors

m examples $x \in \begin{pmatrix} x_0 = 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$ $\mathbb{R}^{n+1}$

$y \in \{0, 1\}$

classification

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

hypothesis

parameters

How to choose parameters θ ?

Building a cost function for logistic regression

Let's rewrite the cost function for LINEAR REGRESSION:

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

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

$$\text{cost}\left(h_\theta(x^{(i)}), y^{(i)} \right)$$

sum over our training set

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

For linear regression this works just fine. But…

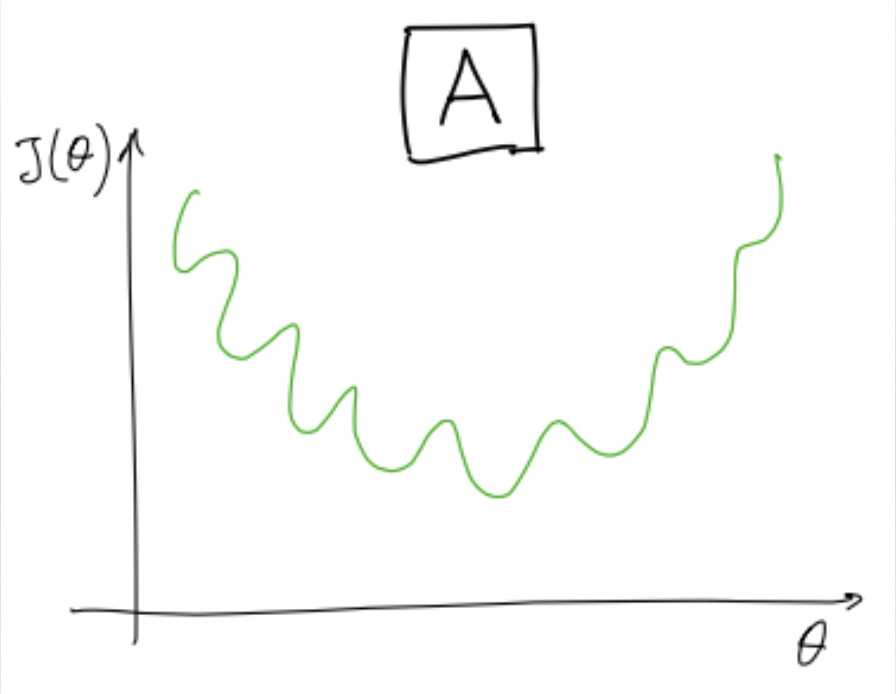
Building a cost function for logistic regression

..what if we try exactly this for logistic regression?

- in principle, if we could minimise this "cost()" part that is plugged into J, that would still work OK also for logistic regression.

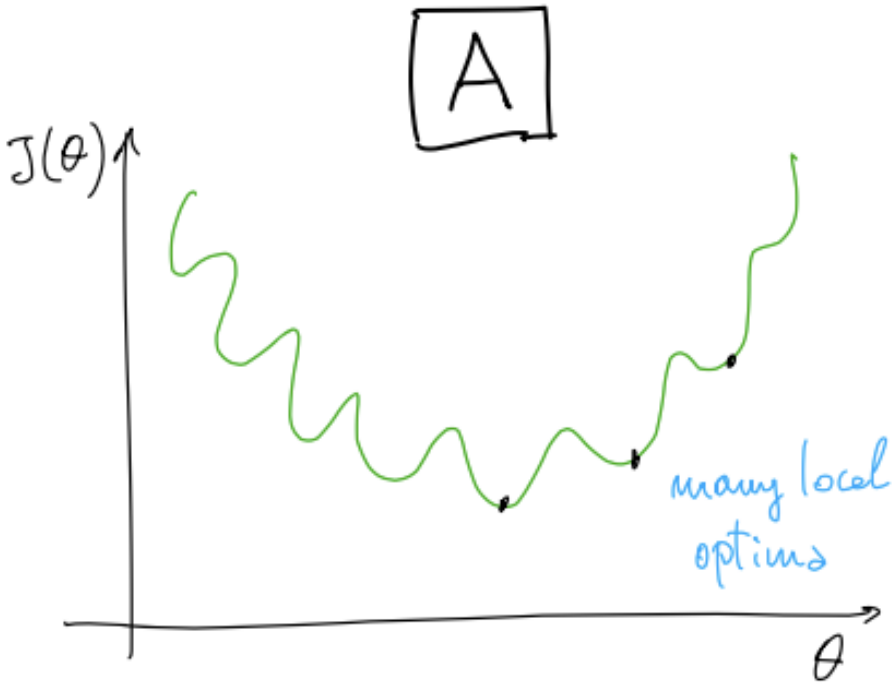
But it turns out that - for logistic regression - we can't.

- If we use this particular cost function, this would be a non-convex function of the parameters.
  - this, as a candidate cost function $J(\theta)$ for logistic regression too, is built on a hypothesis h that has a strong nonlinearity (it is the sigmoid function!). So if you take that function, plug it in the "cost()" part and plug that into the $J(\theta)$, and then plot $J(\theta)$, it may look like plot A on the right



- if you were to run GD on this sort of function it is not guaranteed *at all* to converge to the global minimum!

We are stuck with this..



$h_\theta(x)$ non linear

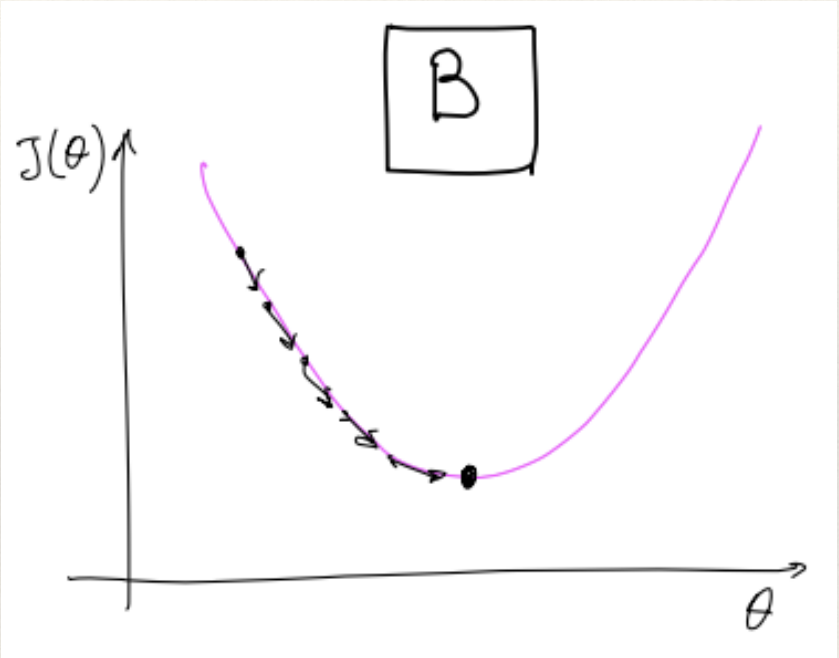
$\Gamma 1$

$J(\theta)$ non convex

Building a cost function for logistic regression

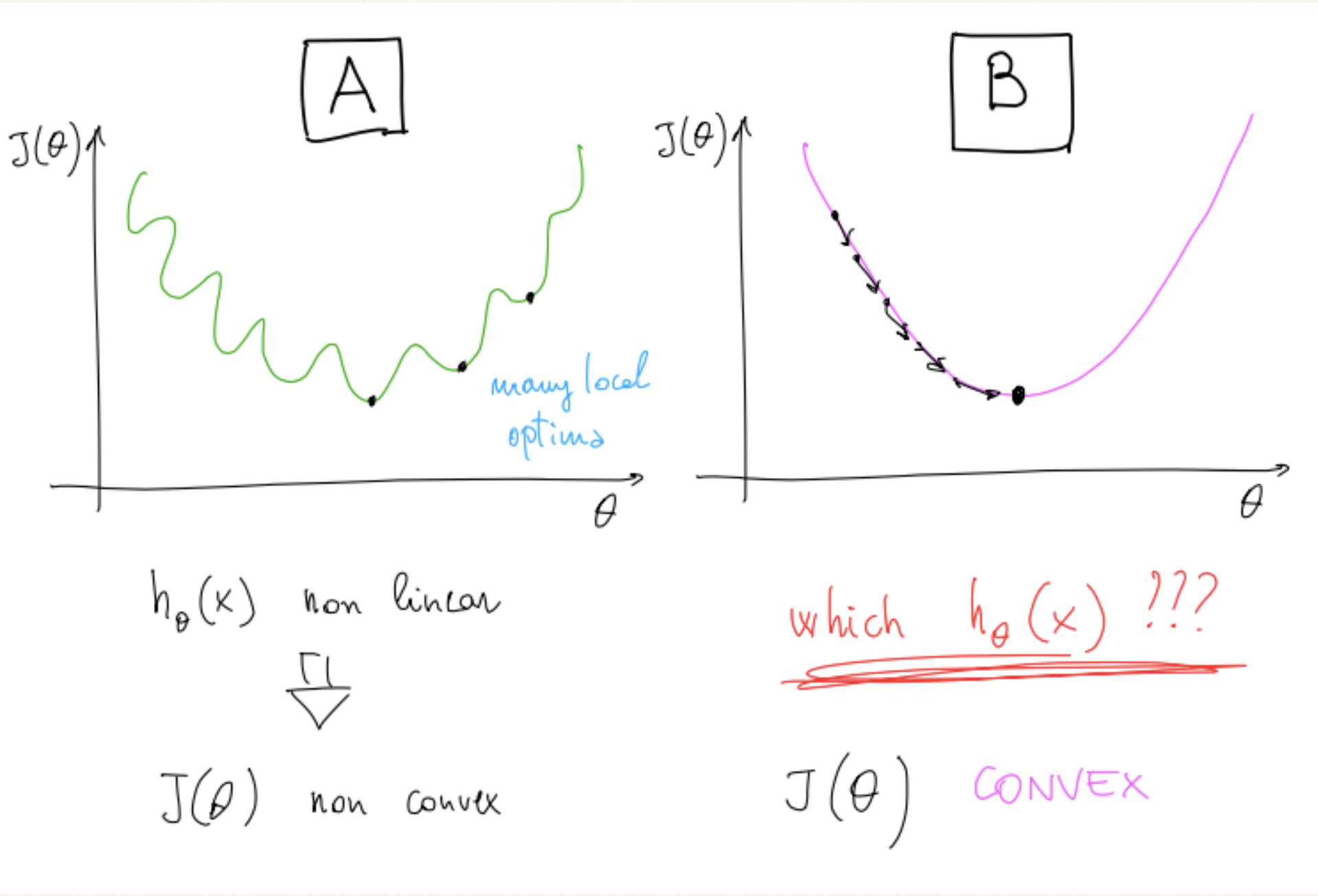
Instead, what we would like is to have a cost function J(θ) that is convex ("bowl-shaped") - see B - so that we would be guaranteed that GD would converge to a (possibly global) minimum.

Well, nobody said - given the unpracticality of having a very nonlinear sigmoid function that forces J(θ) to be non-convex - that we need to define a cost function indeed as a squared cost function!



So: let's come up with something that allows J(θ) to be convex, so we can apply GD and we are guaranteed to find a good (global) minimum..

Pictorial recap on building a cost function for logistic regression



Logistic regression cost function

[NOTE: we drop "(i)"]

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

NOTE: Intuition goes first. The topic of convexity analysis is beyond the scope of this course, but it is possible to show that with this particular choice of cost function, this will give a convex optimisation problem, i.e. J(θ) will be convex and local optima free.

Remember:

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

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

$$\boxed{\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2} = \text{cost}\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

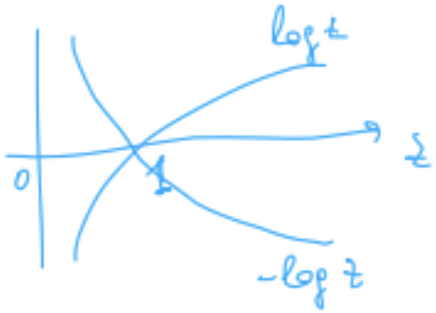
Sum over my training set

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

Logistic regression cost function

[NOTE: we drop "(i)"]

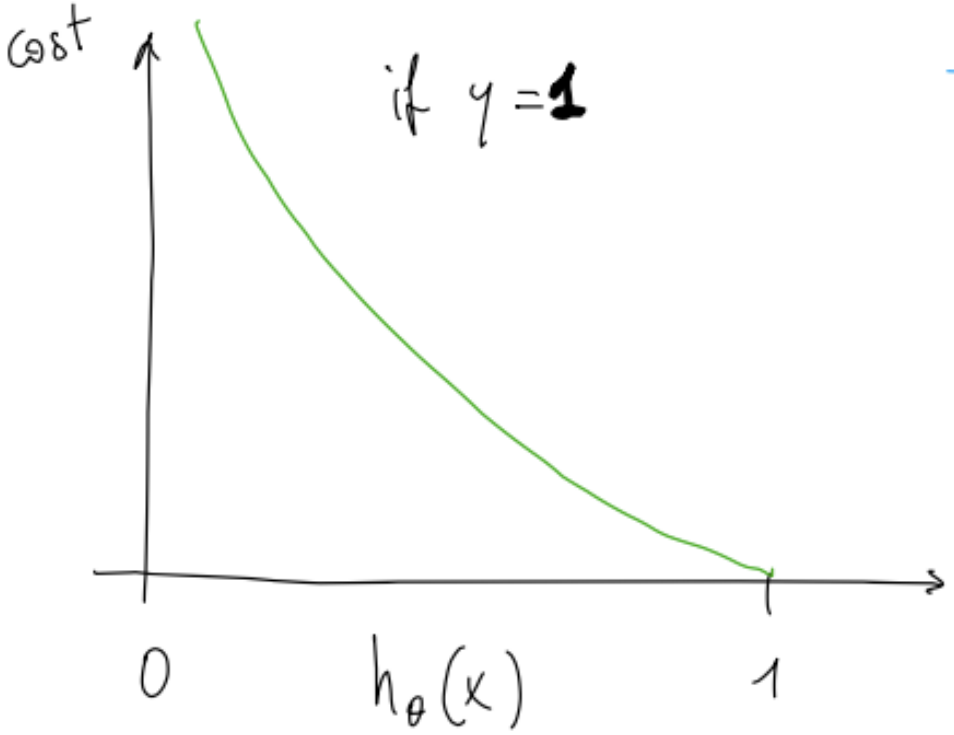
$$\text{cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y=1 \\ -\log\left(1-h_\theta(x)\right) & \text{if } y=0 \end{cases}$$

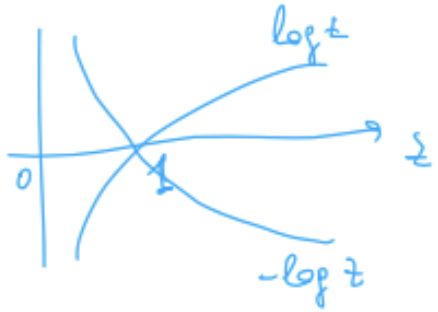


Logistic regression cost function (y=1)

[NOTE: we drop "(i)"]

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

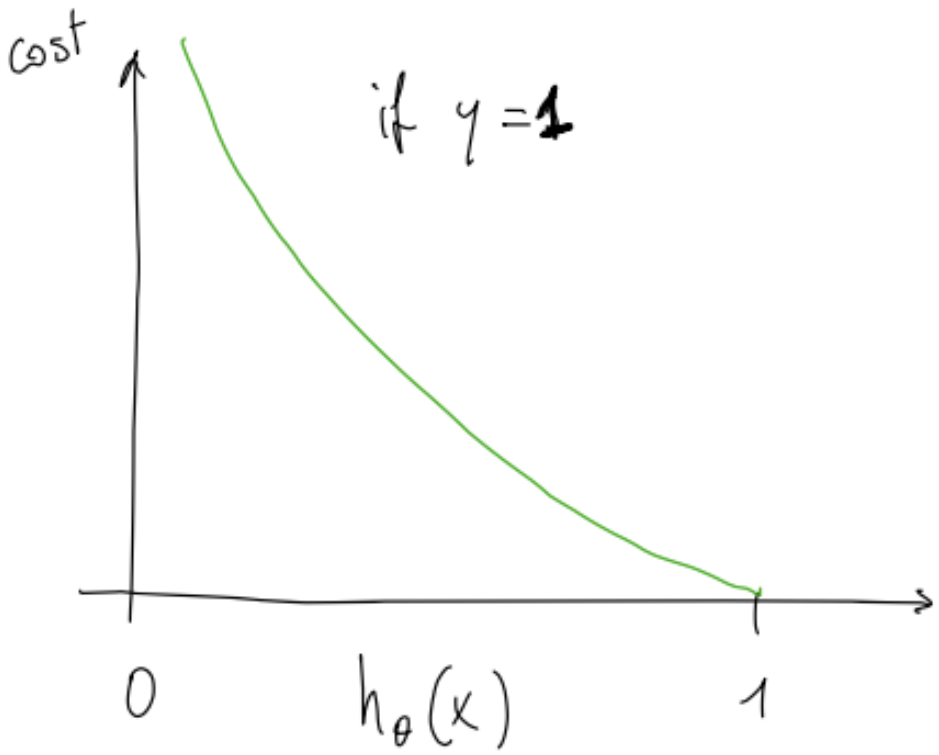




Logistic regression cost function (y=1)

[NOTE: we drop "(i)"]

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



Nice! Because:

perfect prediction!

$h_\theta(x)=1$, $y=1$ $\Rightarrow$ cost $=0$

as $h_\theta(x)\to 0$, $y=1$ $\Rightarrow$ cost $\to \infty$

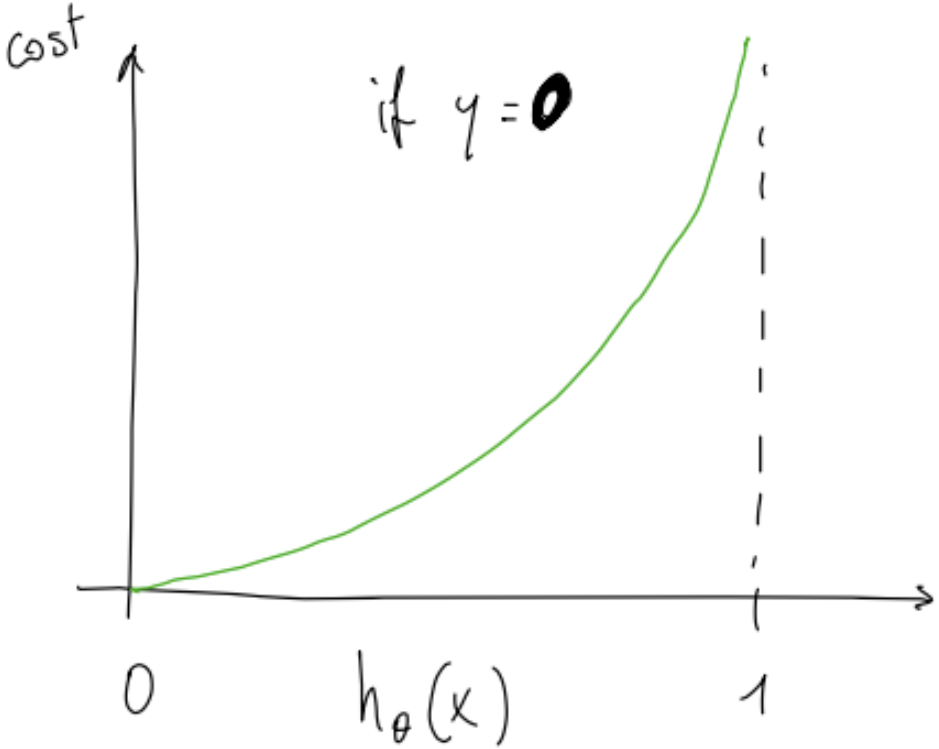
Able to capture intuition that if $h_\theta(x)=0$ (predict $P(y=1|x;\theta)=0$), but $y=1$ $\Rightarrow$ it penalizes learning algo by a very large cost!

think of tumors!

Logistic regression cost function (y=0)

[NOTE: we drop "(i)"]

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

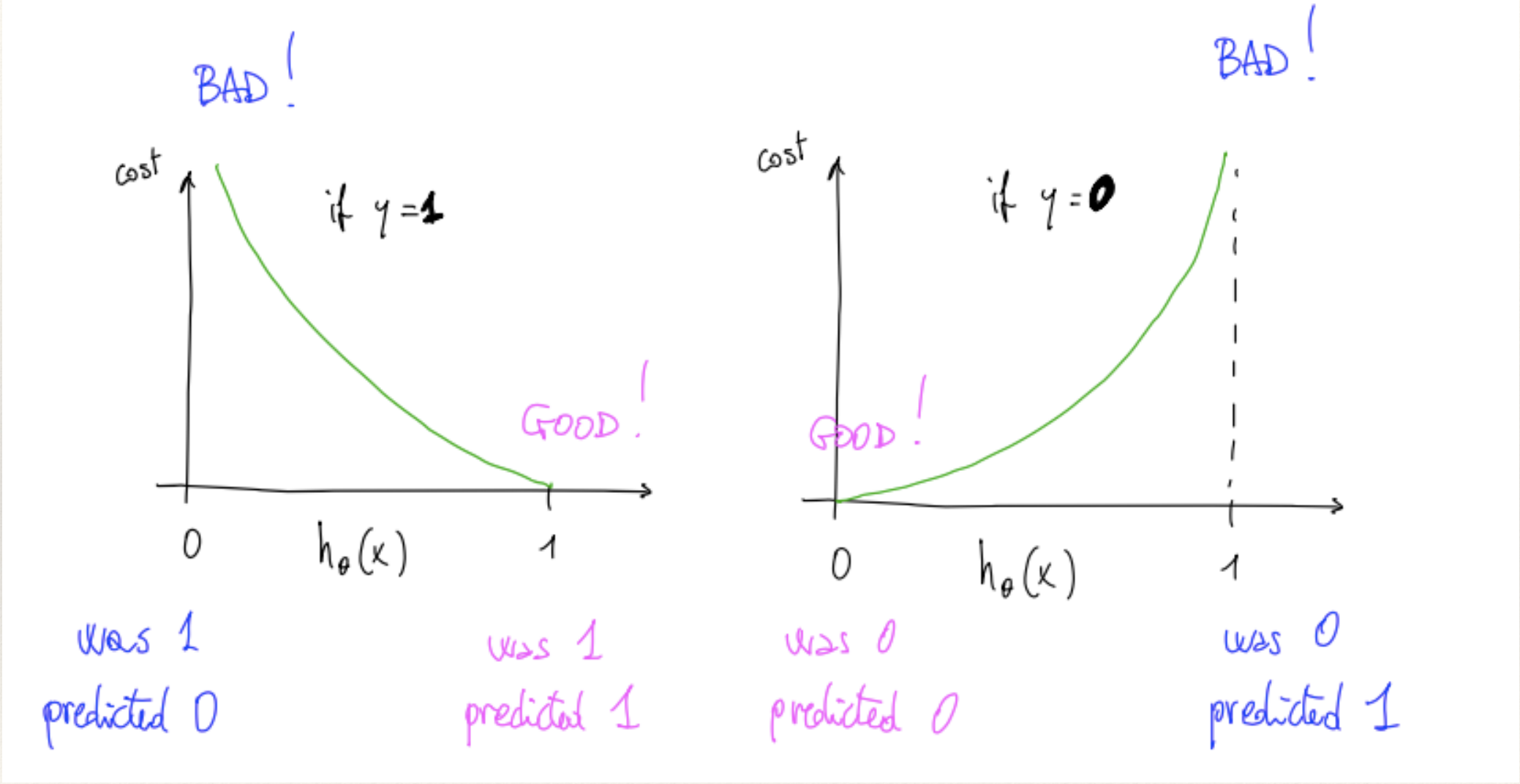


good!

$h_\theta(x) = 0$, $y = 0$ $\Rightarrow$ cost 0

$h_\theta(x) = 1$, $y = 0$ $\Rightarrow$ cost $\to \infty$

Recap on logistic regression cost function



Summary

We cannot use the same cost function used for linear regression because the logistic function will cause the output to be wavy, causing many local optima.

We then managed for logistic regression to define a cost function that is convex.

[NOTE : we drop "(i)"]

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

But, we are not yet there..

So far, we have defined the cost function for a single training example. Now we are going to develop this further, and define the cost function for the entire training set. This will bring us to a simpler way to write it, and then we will work out GD for logistic regression.