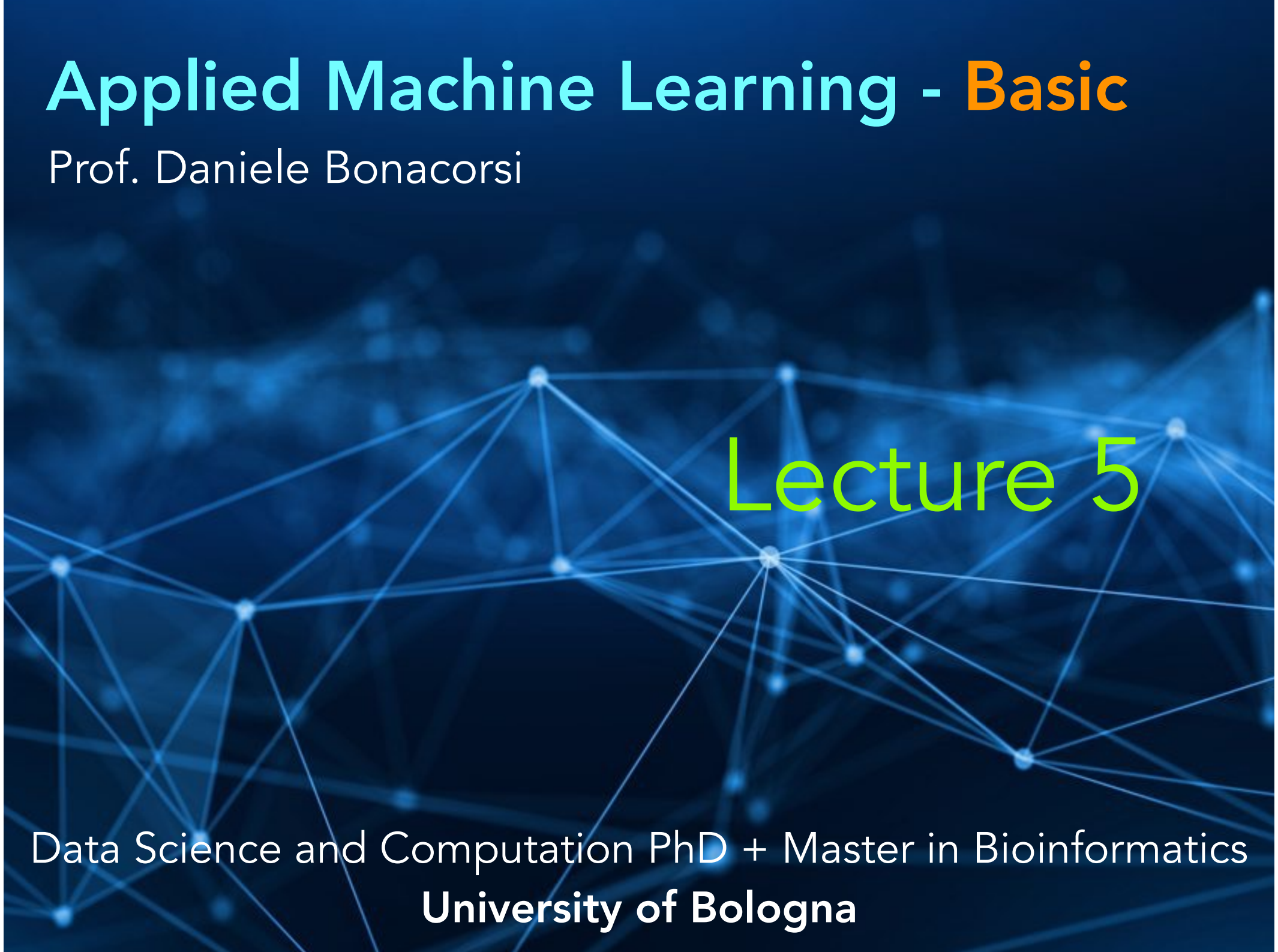
Applied Machine Learning - Basic

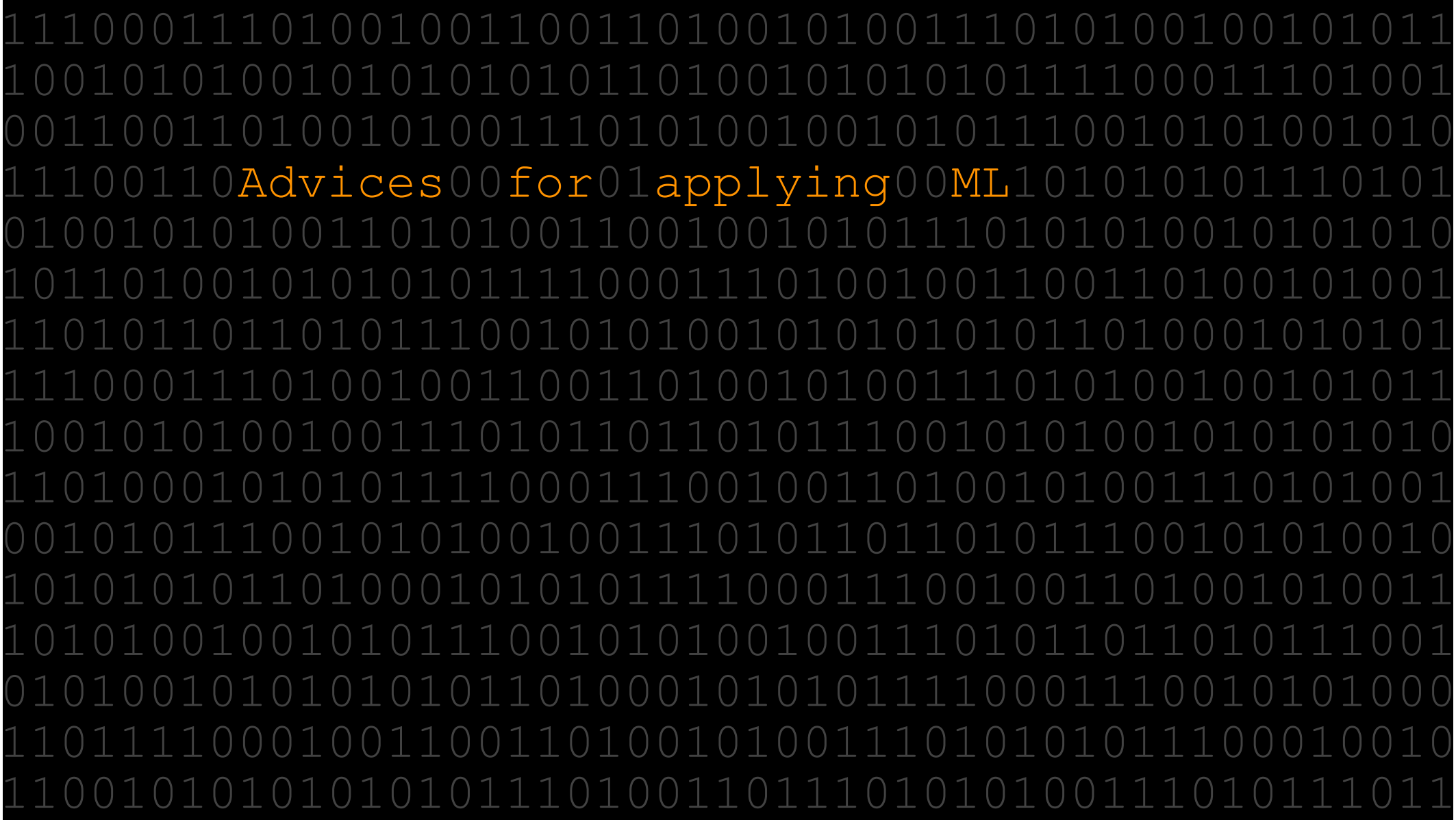
Prof. Daniele Bonacorsi

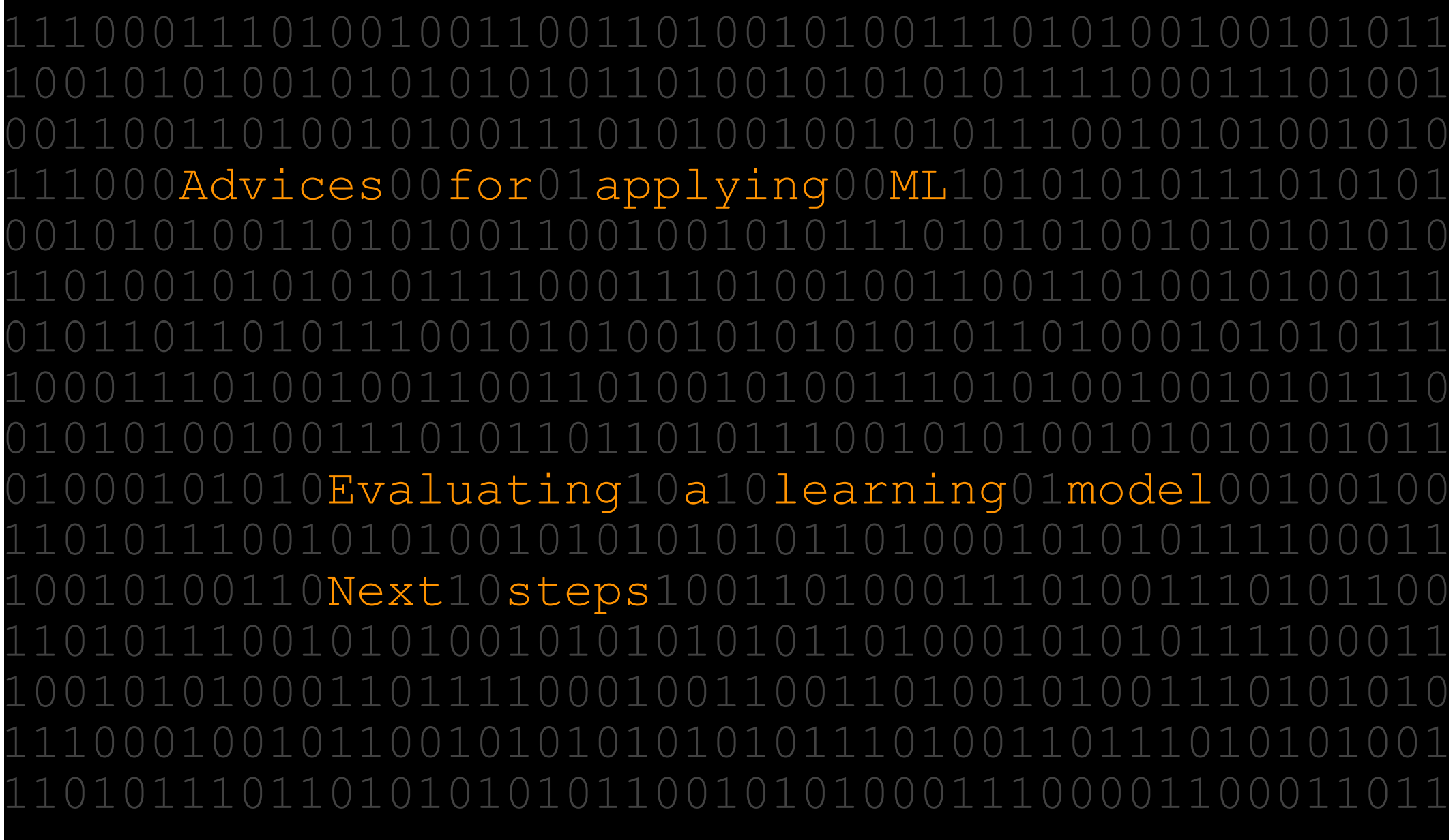


Lecture 5

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna


Advices for applying ML


Advices for applying ML
Evaluating a learning model
Next steps

Knowing learning algos is not enough.

Application of ML on real-world datasets offer **no "gold recipes"**

- i.e. precise steps to follow that would solve your problem, whatever it is
- all efforts are (potentially) painful and time-consuming, and there is no obvious direction to follow..

Any ML practitioner (less or more expert) need to be able to realise quickly in which direction next efforts had better be invested.

What follows aims at giving a number of practical suggestions, advice, guidelines on how to improve in that.

Key question in this section is: **in developing a ML system and/or trying to improve the performances of a developed one, how do I decide what are the best next steps?**

Example: housing prices, again.

You have implemented and regularised linear regression, minimised the cost function J, tuned a bit your hyper-parameters.

But if you test your hypothesis on new set of houses, suppose you find that this is making huge errors in the prediction of the housing prices.

What should you then try in order to improve the learning algo?

Improving performances of a learning algo

One thing you could try, is to **get more training examples**.

- really, stop trying to make the model work. Assess the amount of data you have, and decide. Think about how to get more.
  - "oh, if I had twice as many data…"

Sometimes getting more training data doesn't actually help. We will see.

Another things you might try is to **try a smaller set of features**.

- carefully spend some time to select a subset only, just to prevent overfitting with brute force.

Or maybe you need to **get additional (or different) features**

- maybe the current set of features aren't informative enough
- maybe you want to collect more data also in order to have a chance to extract more (or better) features

Sometimes a change in these directions requires a lot of work to get more data (e.g. surveys, research, etc..) so huge projects, and so once again it would be nice to know in advance if this is ultimately going to help **before** spending a lot of time pursuing this..

You can also try **adding polynomial features**

Try **decreasing/increasing lambda**

So, what?!

Unfortunately, the most common method that people use to pick one of these is as simple as: **go by experience and gut feeling**.

Chances that all this turns out to be **very ineffective** are high..

ML diagnostics

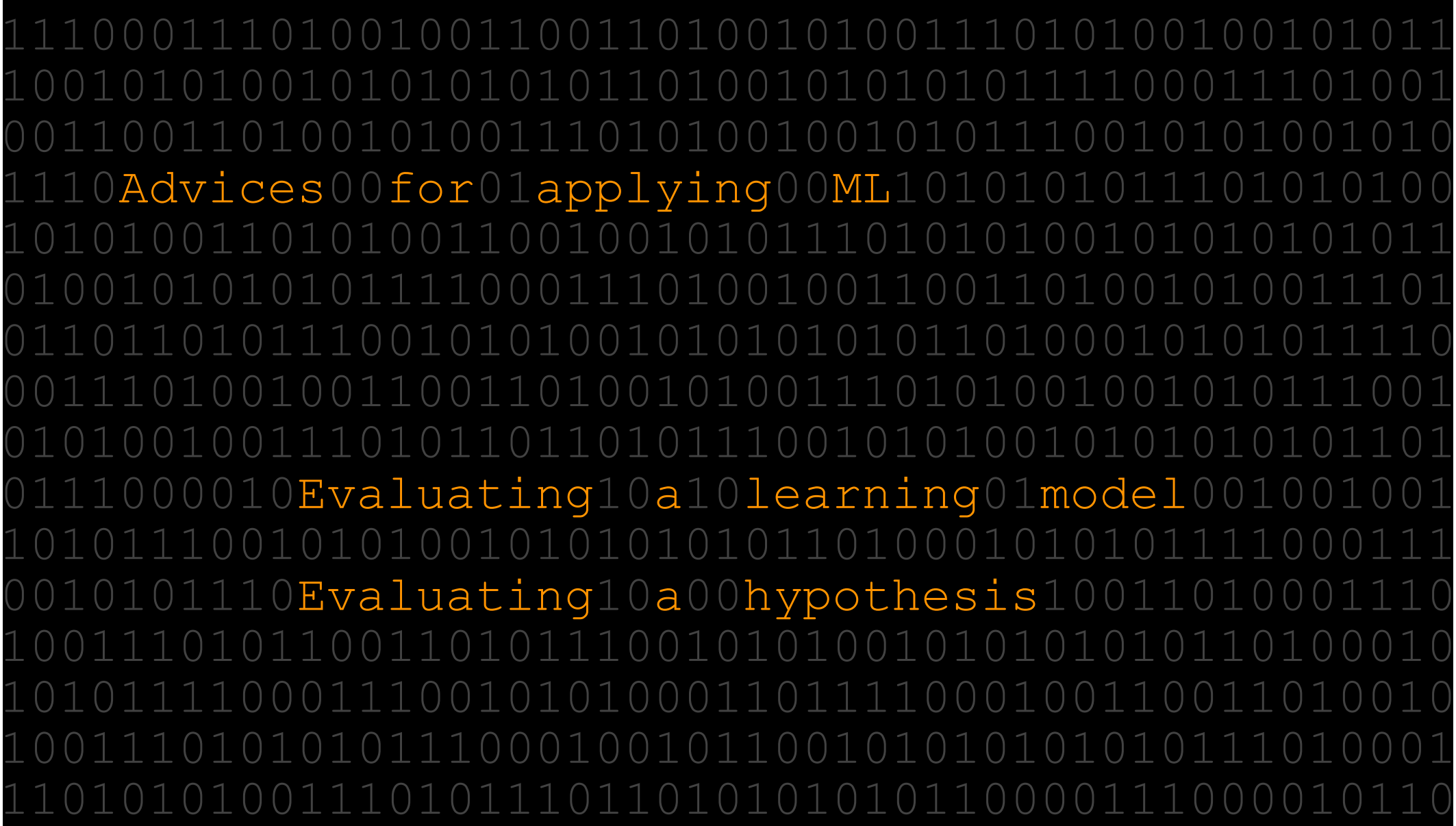
Be pragmatic.

Isn't there a pretty simple technique that can let you very quickly select a fraction of the actions on the aforementioned list as being potentially promising actions to implement, and rule out all the rest?

Fortunately yes!

We will talk about **machine learning diagnostics**, i.e. **how to evaluate learning algorithms**

- tests you can run, to get insight into what is or isn't working with an algorithm, which will often give you insight as of what are promising things to try to improve a learning algorithm's performance
- *NOTE: diagnostics efforts can take time to implement and understand, but it is ultimately a good use of your time. Some ML, by the way, are more friendly than others to do so..*


Advices
for
applying
ML
Evaluating
a
learning
model
Evaluating
a
hypothesis

We have made a hypothesis.

This “has been learned” by my system (i.e. I have θs)

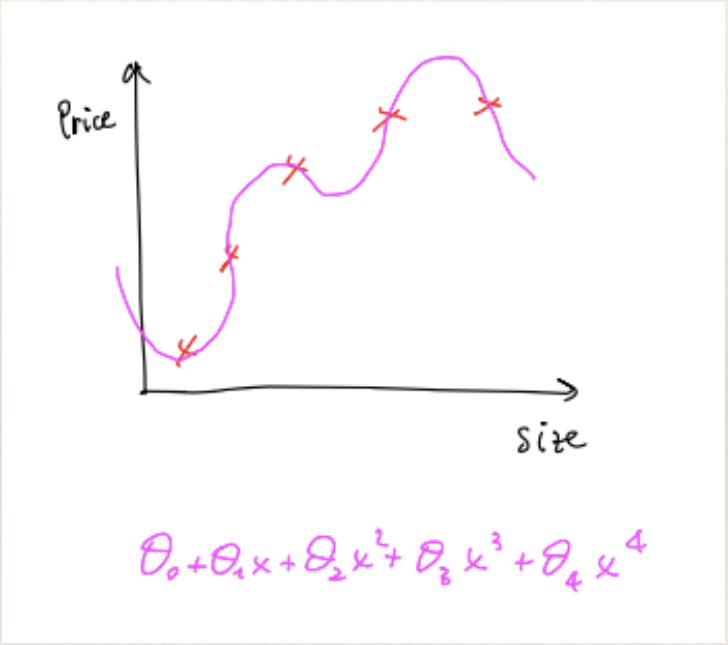
How to evaluate it?

Check for overfitting

We fitted the parameters of our learning algorithm, and we extracted the parameters that minimise the training error.

Getting a really low value of training error for a hypothesis might be a good thing, but we have already seen that that doesn't mean it is a good hypothesis.

- E.g. overfitting -> it fails to generalise to new examples not in the training set



How do you tell if your hypothesis is suffering from overfitting?

- of course, plotting does not help in higher dimensions…

size (mq)	price (k€)
150	300
100	200
80	200
75	250
160	400
210	410
90	200
95	280
105	350
120	370

NOTE : you can have THOUSANDS ...

size (mq)	price (k€)
150	300
100	200
80	200
75	250
160	400
210	410
90	200
95	280
105	350
120	370

70%

TRAINING SET

$(x^{(1)}, y^{(1)})$

$\vdots$

$(x^{(m)}, y^{(m)})$

m = ## training examples

TEST SET

30%

$(x_{test}^{(1)}, y_{test}^{(1)})$

$\vdots$

$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = ## test examples

NOTE: if there is some ordering into the data, shuffle and after do the 70-30 split.

Training/testing procedure for **linear** regression

Procedure:

- learn parameters θ from the training dataset (i.e. the 70% of the total)…
- … and minimise the training error $J(\theta)$ (or $J_{train}(\theta)$ now, if you wish)
- then, compute the **test set error**

linear regression (w/o regularization)

$$J_{test}(\theta) = \frac{1}{2\, m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta\left(x_{test}^{(i)}\right) - y_{test}^{(i)} \right)^2$$

Training/testing procedure for **logistic** regression

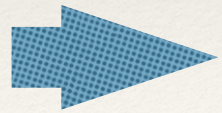
Procedure **is the same as before**:

- learn parameters θ from training dataset (i.e. the 70% of the total)…
- … and minimise the training error $J(\theta)$ (or $J_{train}(\theta)$ now, if you wish)
- then, compute the **test set error**

logistic regression (w/o regularization)

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta\left(x_{test}^{(i)}\right) + \left(1 - y_{test}^{(i)}\right) \log h_\theta\left(x_{test}^{(i)}\right)$$

.. OK, but not the best one ..



0/1 misclassification error

While this definition of the test set error J_{test} is perfectly reasonable, sometimes there is an alternative test set metric that might be easier to interpret:

the **0/1 misclassification error**, with the 0 and 1 labels denoting that you either get an example right or you get an example wrong.

0/1 misclassification error

a classification error metrics

$$\underbrace{err}_{\text{error of a prediction}}(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \; y = 0 \\ & \text{or} \\ & \text{if } h_\theta(x) < 0.5, \; y = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{error}$$

$$\text{test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err\left(h_\theta(x_{test}^{(i)}), y^{(i)}\right)$$

this is exactly the fraction of the examples in my test set that my hypothesis has mislabeled

Summary

Once an hypothesis is done and the parameters that minimise the training error have been extracted:

- firstly, trouble shoots for errors in your predictions by getting more training examples, trying smaller sets of features, etc..
- secondly, move on to evaluating your hypothesis

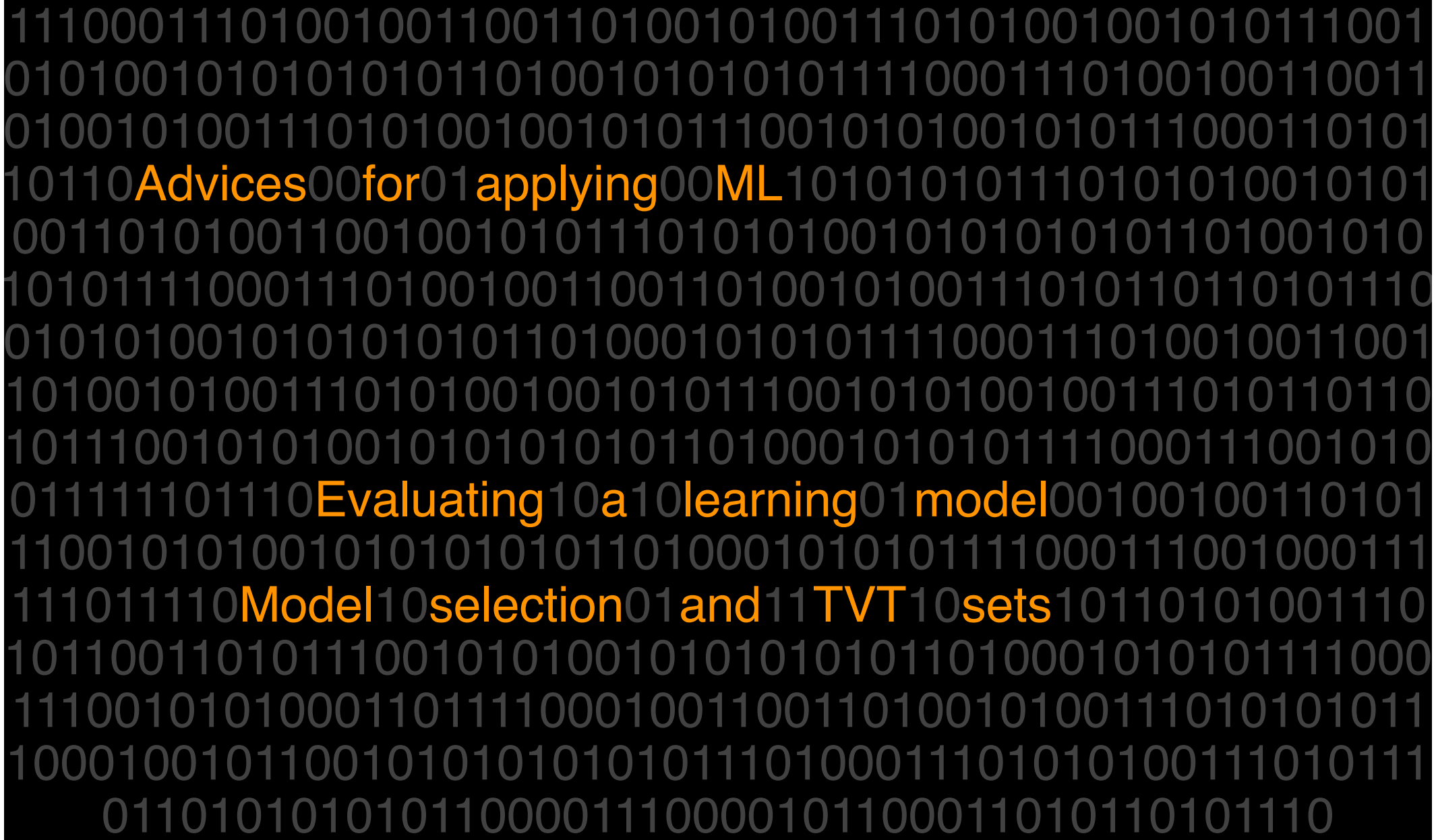
Still, a hypothesis might have a low error for the training examples but still be inaccurate - because of overfitting.

Suggestion (simplest!) is to split the dataset into a **training set** and a **test set** - with a typical 70/30 ratio.

The new procedure is:

- Learn Θ and minimise $J_{train}(\Theta)$ - that we called just $J(\Theta)$ in the previous slides
- compute the test set error $J_{test}(\Theta)$
  - for linear regression: see formula
  - for classification: see the 0/1 misclassification error (proportion of the test data that has been badly misclassified)

The **test error** for the test set gives you a score, "how good I was", in predicting a value or in performing a classification


Advices for applying ML
Evaluating a learning model
Model selection and TVT sets

We have discussed the perils of overfitting, in which just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis.

More generally, this is why the training set's error is not a good predictor for how well the hypothesis will do on new examples. And, similarly speaking, $J_{train}(\Theta)$ is unlikely to be a good estimate of your actual generalisation error: it is likely to be lower than the actual generalisation error.

And, for this purpose, we introduced a **test** set.

Key question now: **is it enough?**

model selection

(1) take $d=1$, minimize training error
=> you get some parameter vector $\theta^{(1)}$

(2) same with $d=2$ => $\theta^{(2)}$

(3) => $\theta^{(3)}$

$d=1$ $\quad h_\theta(x) = \theta_0 + \theta_1 x$

$d=2$ $\quad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

$d=3$ $\quad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

...

$d=10$ $\quad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_{10} x^{10}$ (10) => $\theta^{(10)}$

Notation : $\theta^{(i)}$ = parameters vector I get by fitting polynomial of degree i to the training data

model selection

$d=1$ $\quad \theta^{(1)} \xrightarrow{\text{compute the test set error}} J_{test}(\theta^{(1)})$

$d=2$ $\quad \theta^{(2)} \longrightarrow J_{test}(\theta^{(2)})$

$d=3$ $\quad \theta^{(3)} \longrightarrow J_{test}(\theta^{(3)})$

...

$d=10$ $\quad \theta^{(10)} \longrightarrow J_{test}(\theta^{(10)})$

Notation : $\theta^{(i)}$ = parameters vector I get by fitting polynomial of degree i to the training data

$J_{test}(\theta^{(i)})$ = test set error — a measure of the performance on the test set

model selection

$d=1$ $\quad \Theta^{(1)} \longrightarrow J_{test}\left(\Theta^{(1)}\right)$

$d=2$ $\quad \Theta^{(2)} \longrightarrow J_{test}\left(\Theta^{(2)}\right)$

$d=3$ $\quad \Theta^{(3)} \longrightarrow J_{test}\left(\Theta^{(3)}\right)$

...

$d=10$ $\quad \Theta^{(10)} \longrightarrow J_{test}\left(\Theta^{(10)}\right)$

check which model has the lower test set error

$\downarrow$

j t.c. $J_{test}\left(\Theta^{(j)}\right)$ min

model selection

$d=1$ $\quad \theta^{(1)} \longrightarrow J_{test}\left(\theta^{(1)}\right)$

$d=2$ $\quad \theta^{(2)} \longrightarrow J_{test}\left(\theta^{(2)}\right)$

$d=3$ $\quad \theta^{(3)} \longrightarrow J_{test}\left(\theta^{(3)}\right)$

...

$d=10$ $\quad \theta^{(10)} \longrightarrow J_{test}\left(\theta^{(10)}\right)$

Suppose $j=5$ is j t.c. $J_{test}(\theta^{(j)})$ minimal

$$\Rightarrow \quad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

How well does this model generalize?

model selection

$d=1$ $\quad \theta^{(1)} \longrightarrow J_{test}(\theta^{(1)})$

$d=2$ $\quad \theta^{(2)} \longrightarrow J_{test}(\theta^{(2)})$

$d=3$ $\quad \theta^{(3)} \longrightarrow J_{test}(\theta^{(3)})$

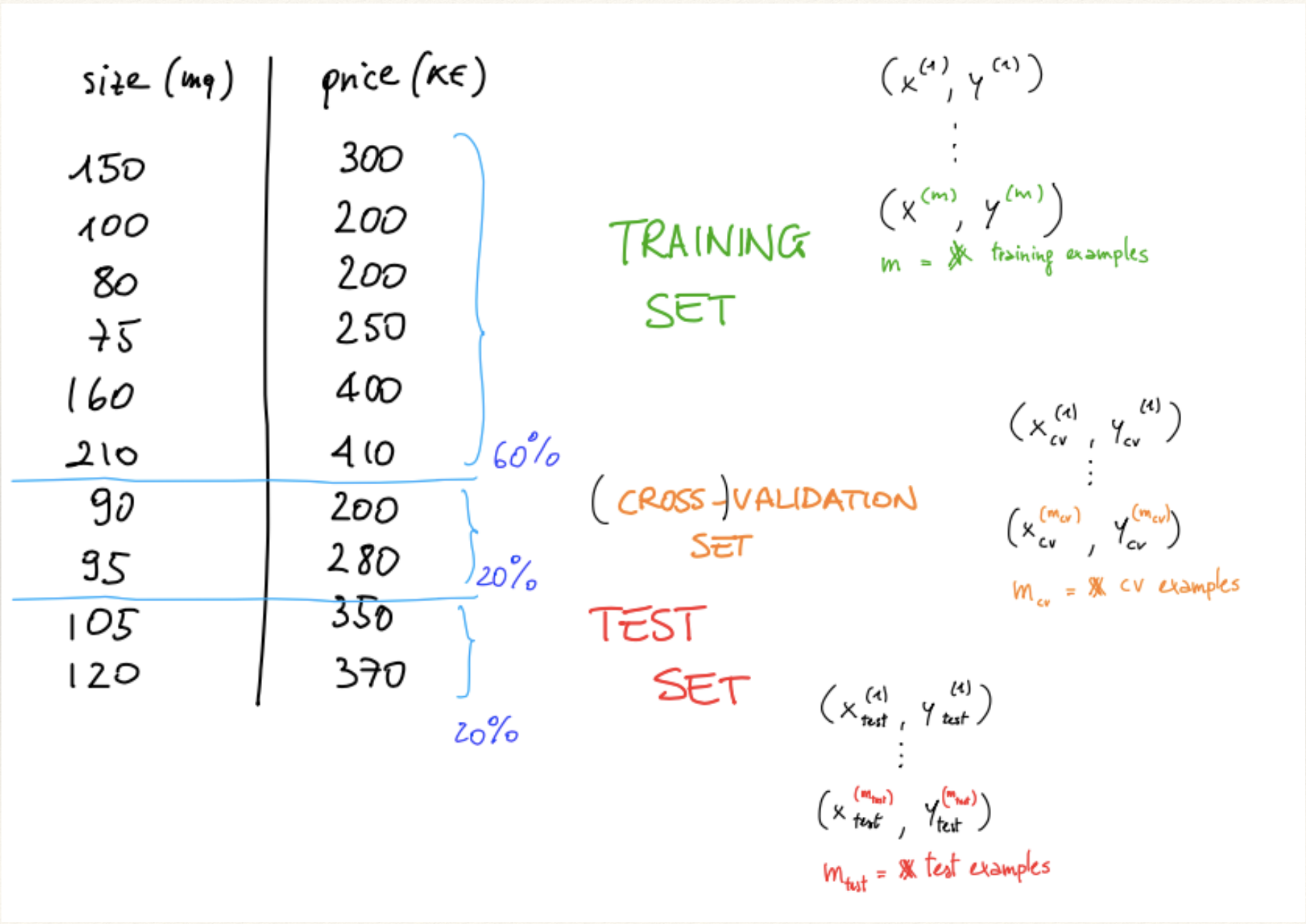
...

$d=10$ $\quad \theta^{(10)} \longrightarrow J_{test}(\theta^{(10)})$

How well does this model generalize?

Look at how well my 5th order polynomial hypothesis has done on my test set, i.e. $J_{test}(\theta^{(5)})$

This will not be a fair estimate of how well my hypothesis generalizes: we chose $d=5$ because it gave best J_{test}; it is likely to be an optimistic estimate of the generalization error (namely, I have chosen $d=5$ based on the test set!) so my hypothesis is likely to do better on the test set...


size (mq)
price (k€)
150
100
80
75
160
210
90
95
105
120
300
200
200
250
400
410
200
280
350
370
60%
20%
20%
TRAINING SET
(CROSS-)VALIDATION SET
TEST SET
$(x^{(1)}, y^{(1)})$
$(x^{(m)}, y^{(m)})$
m = % training examples
$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
m_{cv} = % cv examples
$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$
m_{test} = % test examples

So, now that we've defined the **training set**, (cross) **validation set** and **test set**..

.. we can also define the **training error**, (cross) **validation error**, and **test error**.

TRAINING ERROR

same as my previous $J(\theta)$

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

CROSS-VALIDATION ERROR

$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

TEST ERROR

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta\left(x_{test}^{(i)}\right) - y_{test}^{(i)} \right)^2$$

Similar formulas, but note there is **absolutely no overlap** among the m_{train}, m_{cv} and m_{test} examples

The procedure with training/validation/test (TVT)

model selection (using training / test / CV sets)

I test h on the CV set

1. $h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(1)} \longrightarrow J_{CV}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \longrightarrow \theta^{(2)} \longrightarrow J_{CV}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \longrightarrow \theta^{(3)} \longrightarrow J_{CV}(\theta^{(3)})$

...

10. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_{10} x^{10} \longrightarrow \theta^{(10)} \longrightarrow J_{CV}(\theta^{(10)})$

Basically, we do <u>not</u> use the <u>test set</u> to select the model, but the <u>CV set</u>.

→ $J_{CV}(\theta^{(i)})$ tells me how well each h do on my CV set,

→ I pick the h with the lowest CV error

model selection (using training / test / CV sets)

1. $h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow \theta^{(1)} \longrightarrow J_{CV}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \longrightarrow \theta^{(2)} \longrightarrow J_{CV}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \longrightarrow \theta^{(3)} \longrightarrow J_{CV}(\theta^{(3)})$

...

10. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_{10} x^{10} \longrightarrow \theta^{(10)} \longrightarrow J_{CV}(\theta^{(10)})$

Suppose lowest is $J_{CV}(\theta^{(4)})$:

$\Rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ good! selected!

Now :

Use the test set to estimate the generalization error of the model selected using the CV set. → i.e. compute $J_{test}(\theta^{(4)})$

Take-away message

Take your dataset, split into TVT, isolate the **test set** (→ "Put it in the closet, hide the key!"). Then, perform your ML modelling with the rest of the datasets.

NEVER, EVER LOOK AT THE TEST SET in this early phase.

Bring back the key, get the test set back on the table, only when you are done with model creation, training, tuning.. and you use it ONLY to assess the generalisation error.

Summary

A learning algo that fits a training set well is not necessarily a good hypothesis. It could overfit and, as a result, you would be entering the real application of your ML model being unmotivatedly optimistic, and your final predictions would eventually be disappointingly poor.

- the error of your h as measured on the training set will be lower than the error on any other data set

If you e.g. need to decide among many models, each with a different polynomial degree, in order to identify the "best" one you can compute the error result for each.

Break your dataset into e.g. Train 60%, CV 20%, Test 20%

Then, follow this procedure:

- optimise the θs using the **training** set (polynomial h? then, try each polynomial degree d)
- find the best model (polynomial? find the best d), i.e. smallest error, using the **cv** set
- estimate the generalisation error using the **test** set (polynomial? with $J_{test}(\Theta^{(d)})$)

In this way (polynomial example), d has not been trained using the test set.

A final remark (personal opinion)

In the code, all this will be very simple and quick, and often almost automatic to execute in your ML projects implementation, and you will be greatly helped by libraries to do all this.

But:

That does not mean this part is unnecessary. Quite the opposite. It gives solidity to the steps you would do quickly afterwards. It is important, IMO, in a basic applied ML course, that you spend some time to think about the **diligence needed in dealing with your dataset and the TVT splitting**: it will increase your awareness on what you do, your ability to spot logical mistakes, your speed in debugging code, your talent in spotting data leakage evidences, etc.

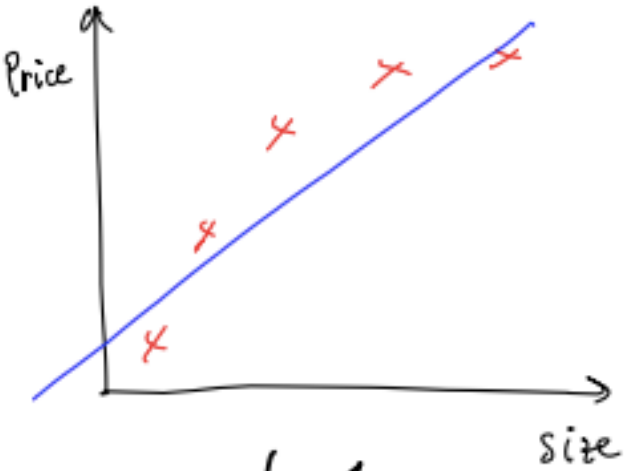

Advices for applying ML
Bias and Variance
Diagnosis bias vs variance

If we run a learning algorithm to make predictions and it does not do as well as we hoped, almost always it is because we have either a **high bias problem** or a **high variance problem**

- in other words, either an **underfitting** problem or an **overfitting** problem

Let's dive more into this.

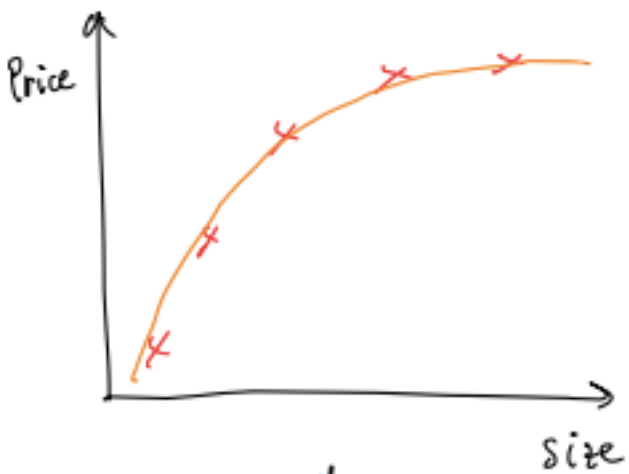
Restart from this :



$d = 1$

$\theta_0 + \theta_1 x$

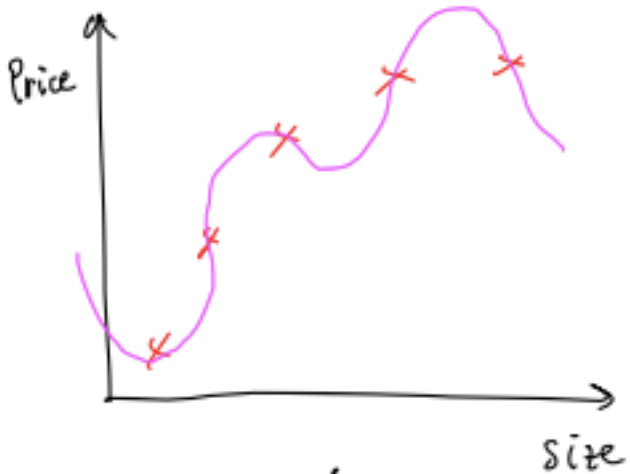
"UNDERFITTING" or "HIGH-BIAS"



$d = 2$

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

"JUST RIGHT"



$d = 4$

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

"OVERFITTING" or "HIGH-VARIANCE"

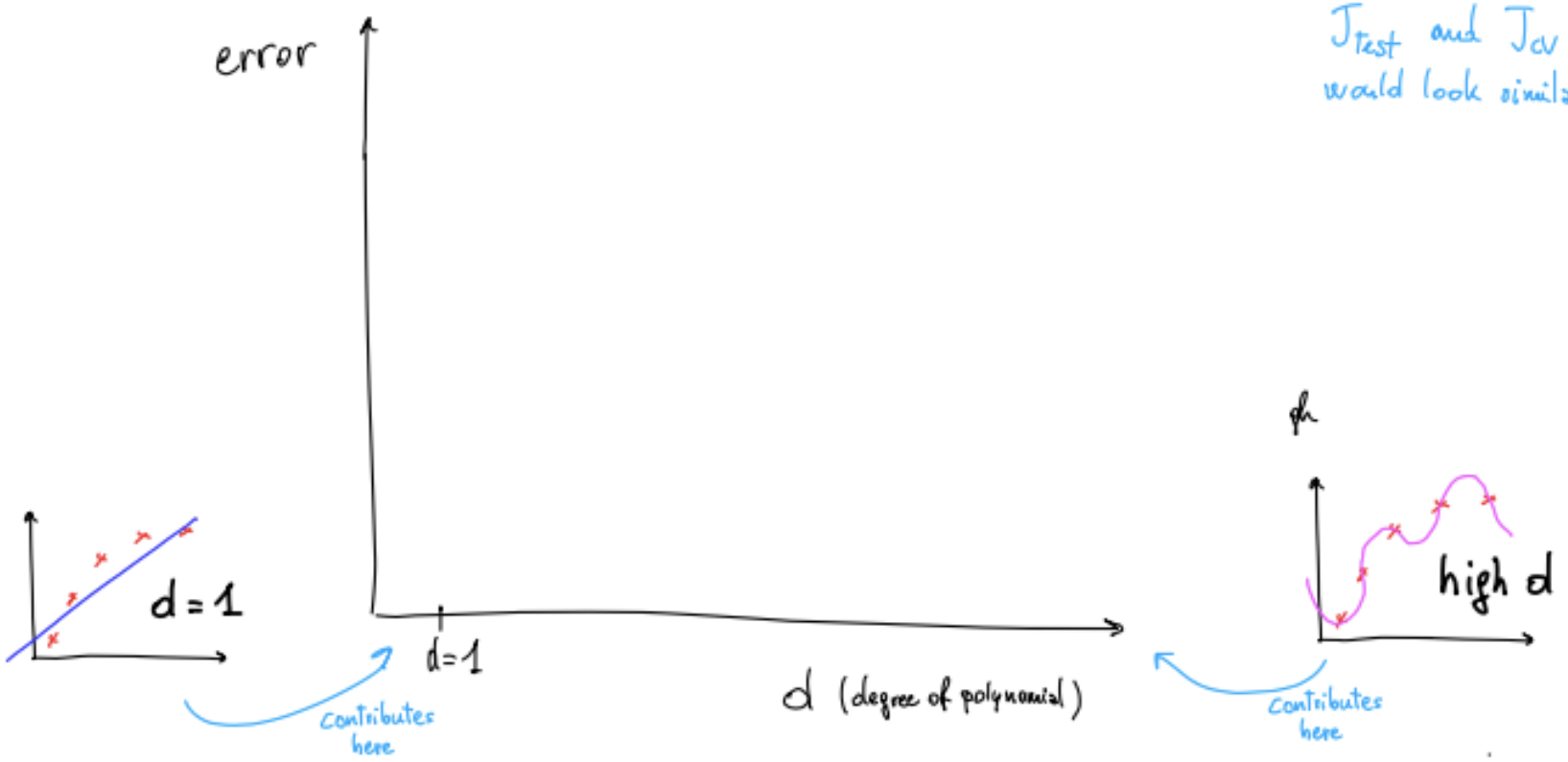
TRAINING ERROR

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

CROSS-VALIDATION ERROR

$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

NOTE:
J_{test} and J_{CV} would look similar

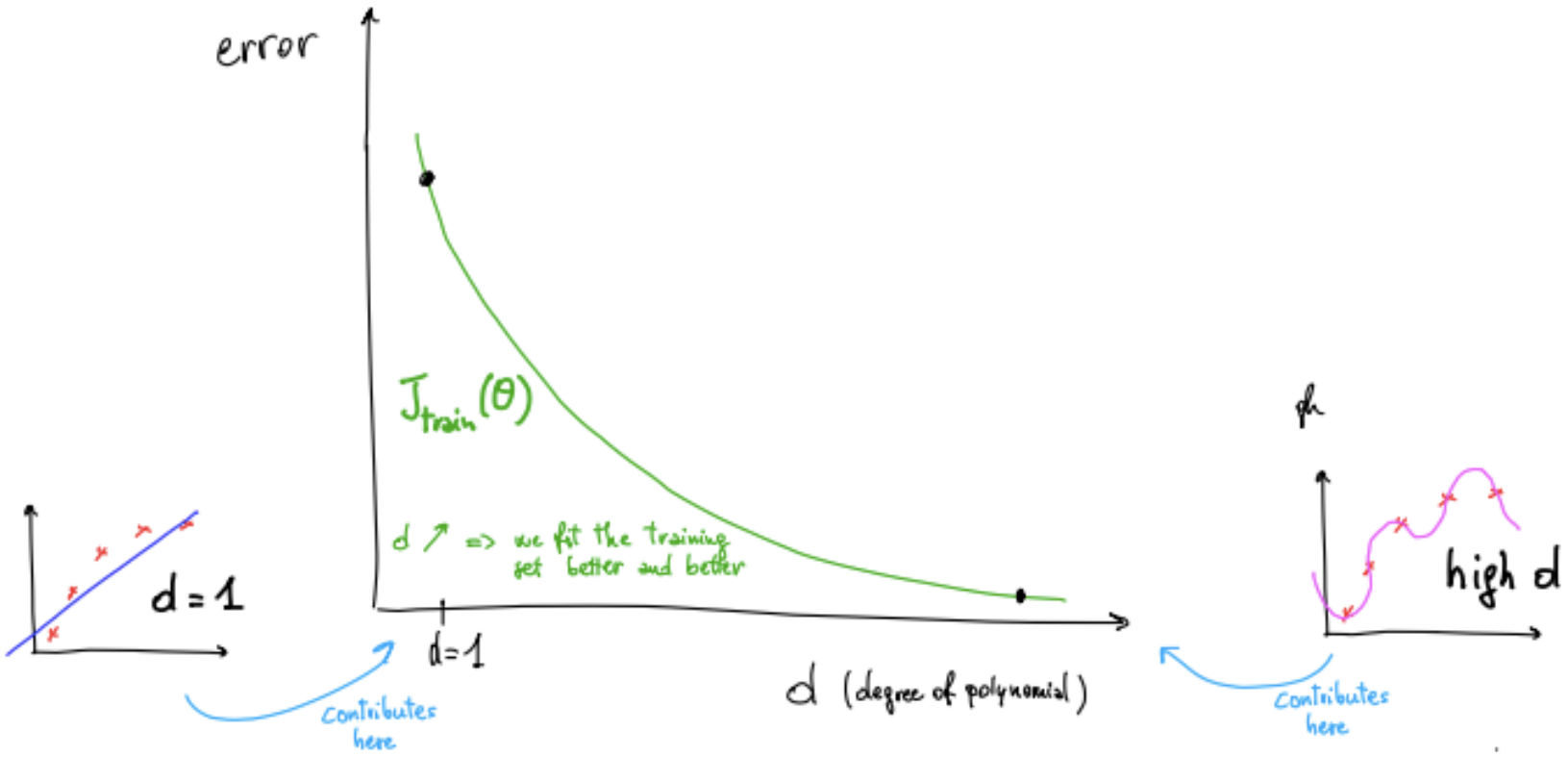


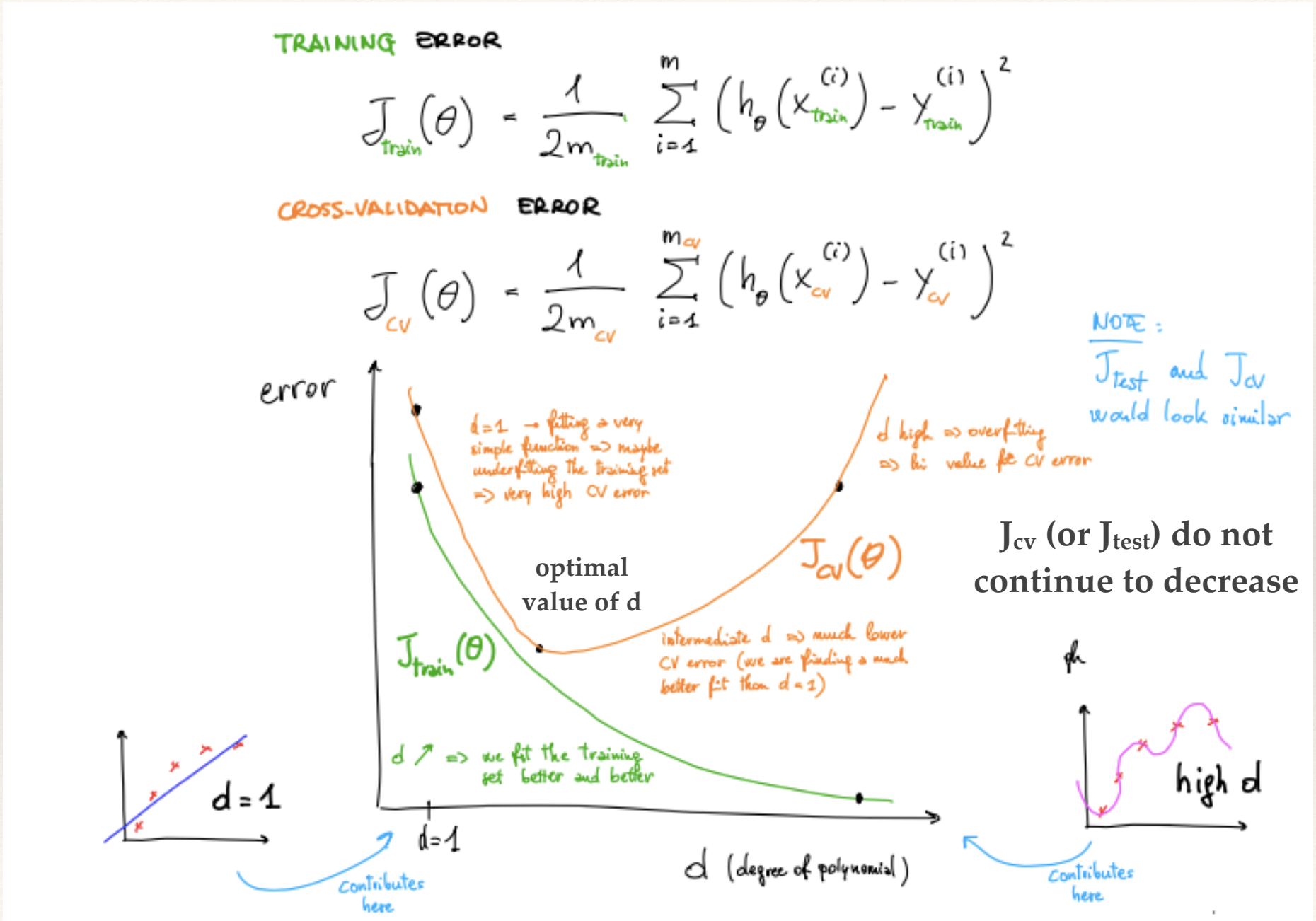
TRAINING ERROR

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

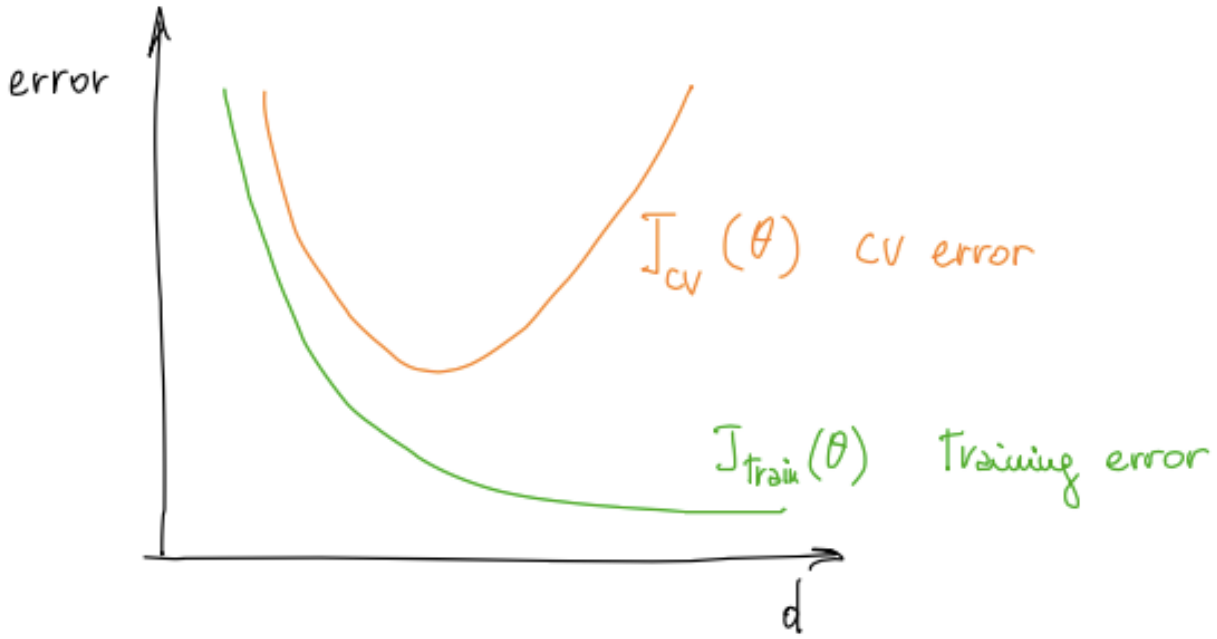
CROSS-VALIDATION ERROR

$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$




TRAINING ERROR
$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)} \right)^2$
CROSS-VALIDATION ERROR
$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta(x_{CV}^{(i)}) - y_{CV}^{(i)} \right)^2$
NOTE:
J_{test} and J_{CV} would look similar
error
d = 1 → fitting a very simple function => maybe underfitting the training set => very high CV error
d high => overfitting => hi value for CV error
optimal value of d
$J_{CV}(\theta)$
$J_{train}(\theta)$
intermediate d => much lower CV error (we are finding a much better fit than d = 1)
d ↗ => we fit the training set better and better
d = 1
d=1
d (degree of polynomial)
contributes here
high d
contributes here
J_{cv} (or J_{test}) do not continue to decrease

Suppose your learning algo is performing less well than hoped.
Is it a **bias** problem or a **variance** problem ?

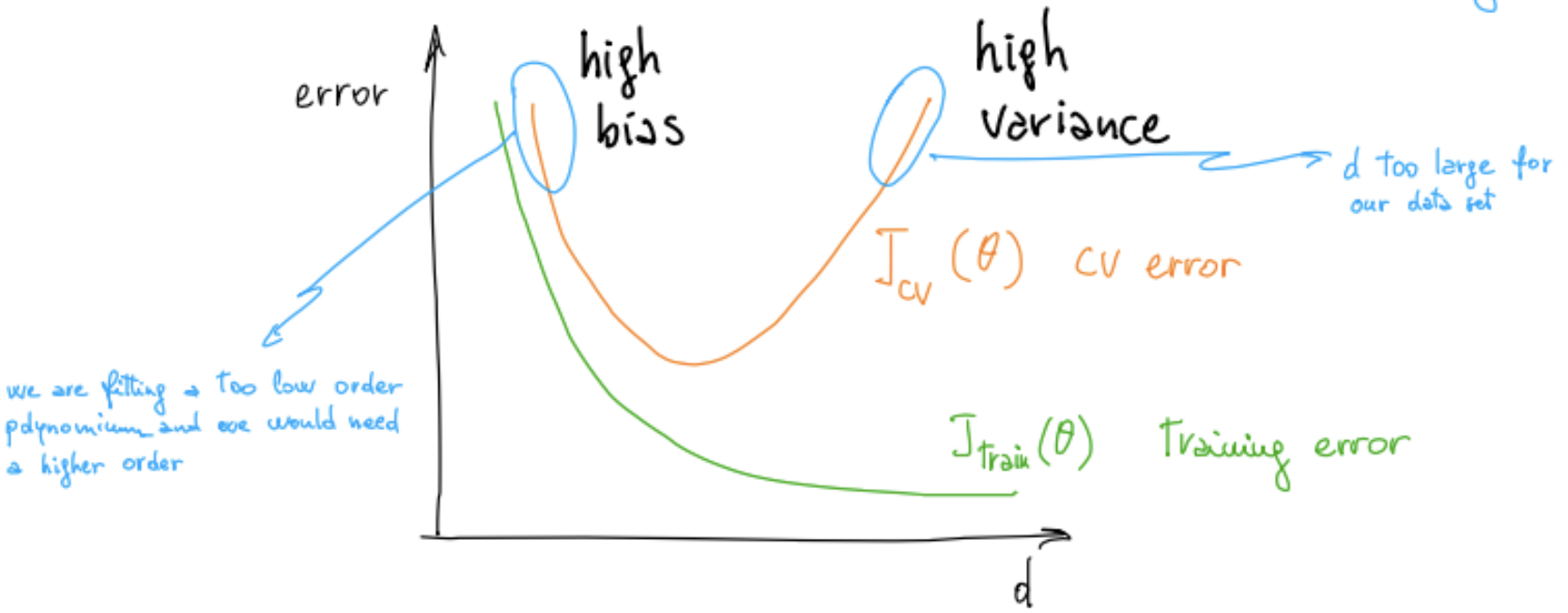


Suppose your learning algo is performing less well than hoped.

Is it a **bias** problem or a **variance** problem ?

(high bias) (high variance)

$J_{CV}(\theta)$ (or $J_{test}(\theta)$) too high

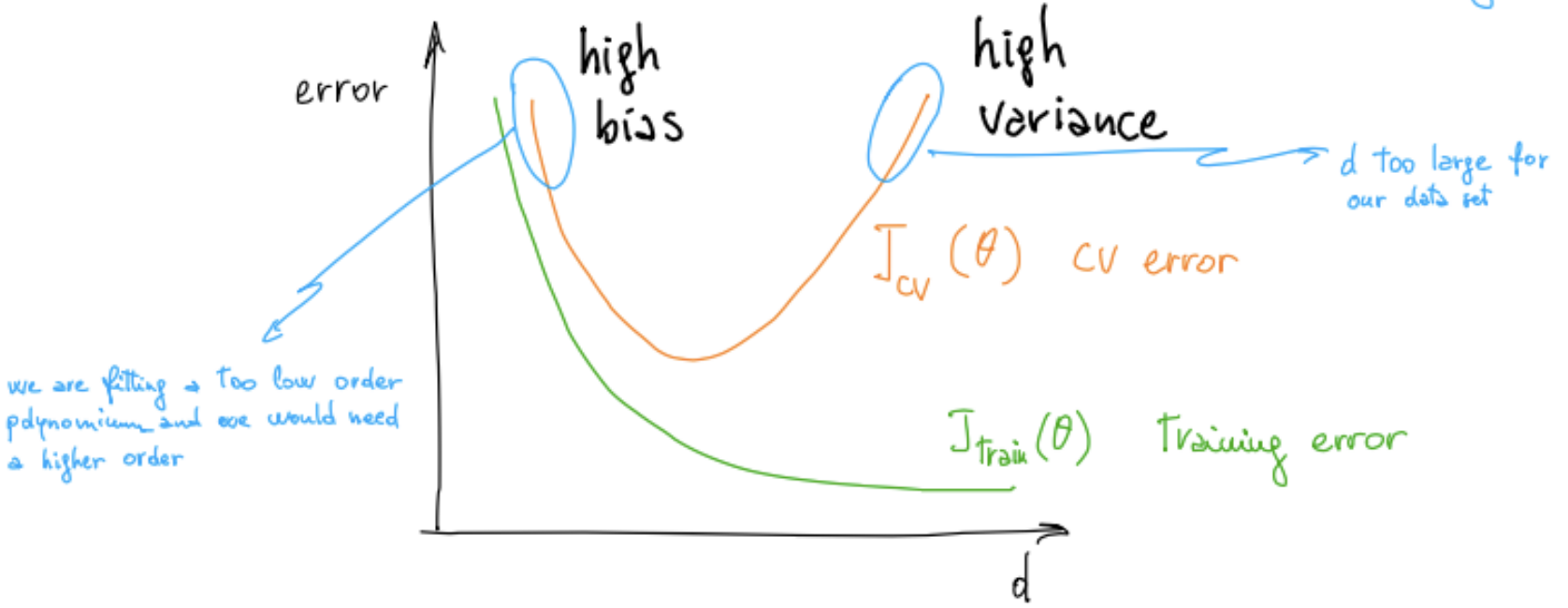


Suppose your learning algo is performing less well than hoped.

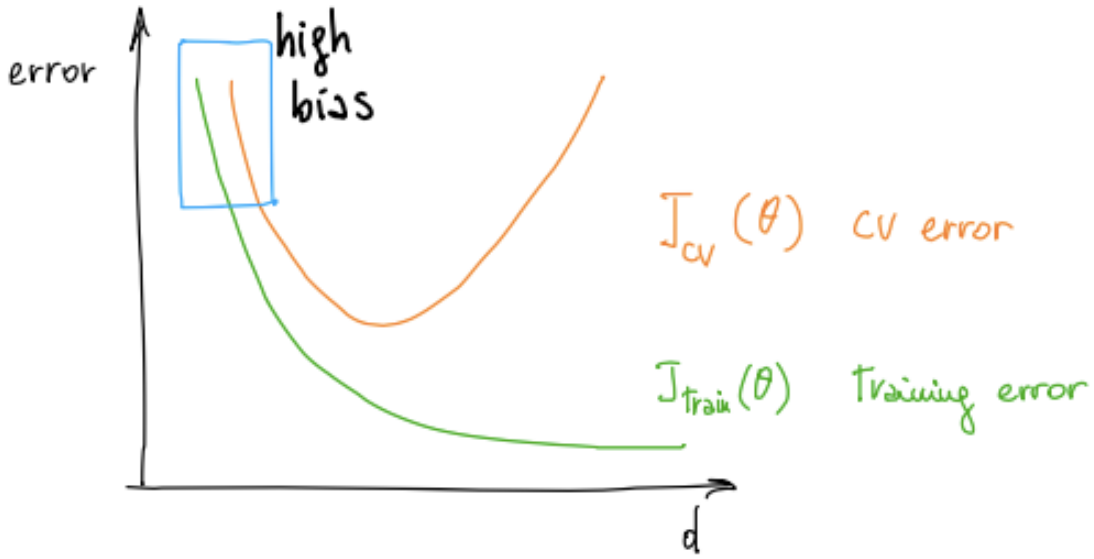
Is it a **bias** problem or a **variance** problem ?

(high bias) (high variance)

$J_{CV}(\theta)$ (or $J_{test}(\theta)$) too high



This gives us a way to disentangle **high bias** vs **high variance** !

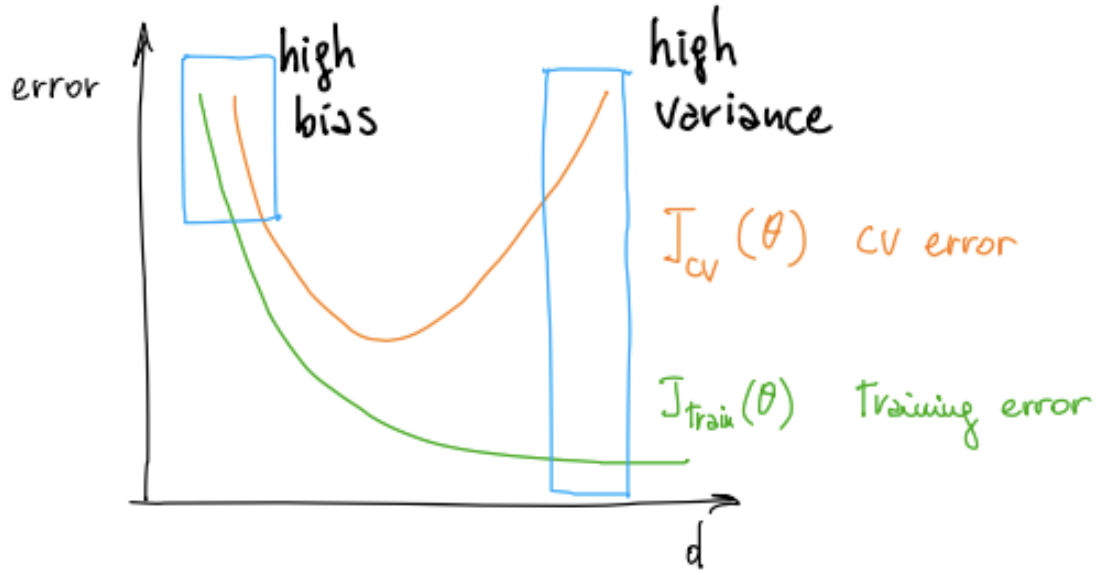


$J_{cv}(\theta)$ CV error

$J_{train}(\theta)$ Training error

BIAS (underfit) → $J_{train}(\theta)$ high (h not fitting training set well)

$J_{cv}(\theta)$ high ($\approx J_{train}$, maybe higher)



BIAS (underfit) → $J_{train}(\theta)$ high (h not fitting training set well)

$J_{cv}(\theta)$ high ($\approx J_{train}$, maybe higher)

VARIANCE (overfit) → $J_{train}(\theta)$ low (ie. fitting training set well)

$J_{cv}(\theta)$ high ($>> J_{train}$)

Summary

We discussed the **diagnosis of bias vs variance** in case of bad predictions

We examined, in our explanatory example, the relationship between the degree of the polynomial d and the underfitting vs overfitting of our hypothesis.

We associate **high bias** to underfitting, and **high variance** to overfitting. We need to distinguish whether bias or variance is the problem contributing to bad predictions. Ideally, we need to find a golden working point between these 2 points

The training error will tend to decrease as we increase the degree d of the polynomial. At the same time, the cv error will tend to decrease as we increase d up to a point, and then it will increase as d is increasing, forming a convex curve

High bias (underfitting):

- both $J_{train}(\Theta)$ and $J_{cv}(\Theta)$ will be high. Also, $J_{cv}(\Theta) \approx J_{train}(\Theta)$

High variance (overfitting):

- $J_{train}(\Theta)$ will be low and $J_{cv}(\Theta)$ will be $>> J_{train}(\Theta)$


Advices for applying ML
Bias and Variance
Regularisation and bias/variance

We have seen how regularisation can help prevent overfitting.

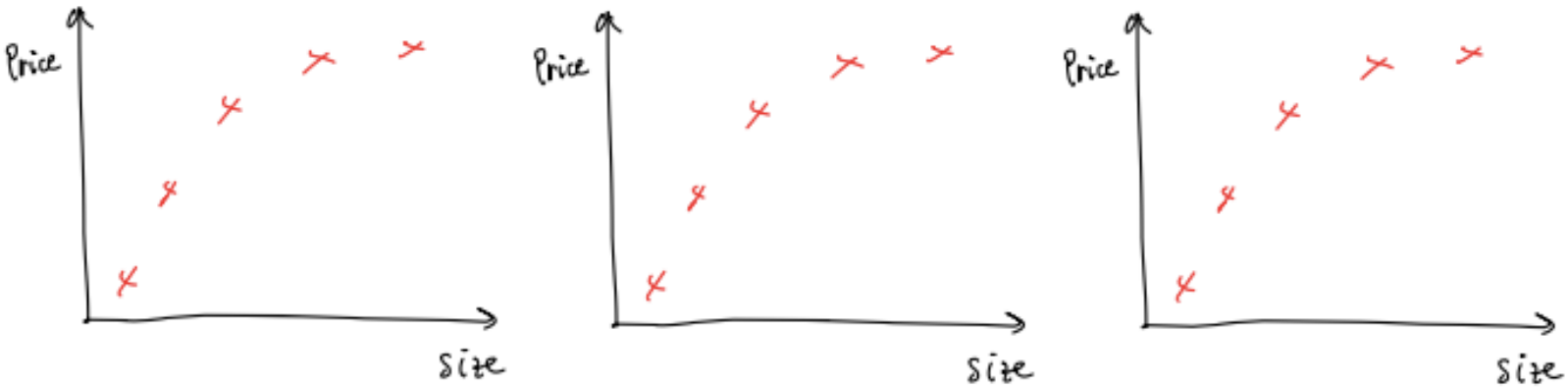
How does it affect (visibly?) the bias vs variances of a learning algorithm, that we discussed?

Linear regression (w regularization)

already applied to attack overfitting

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

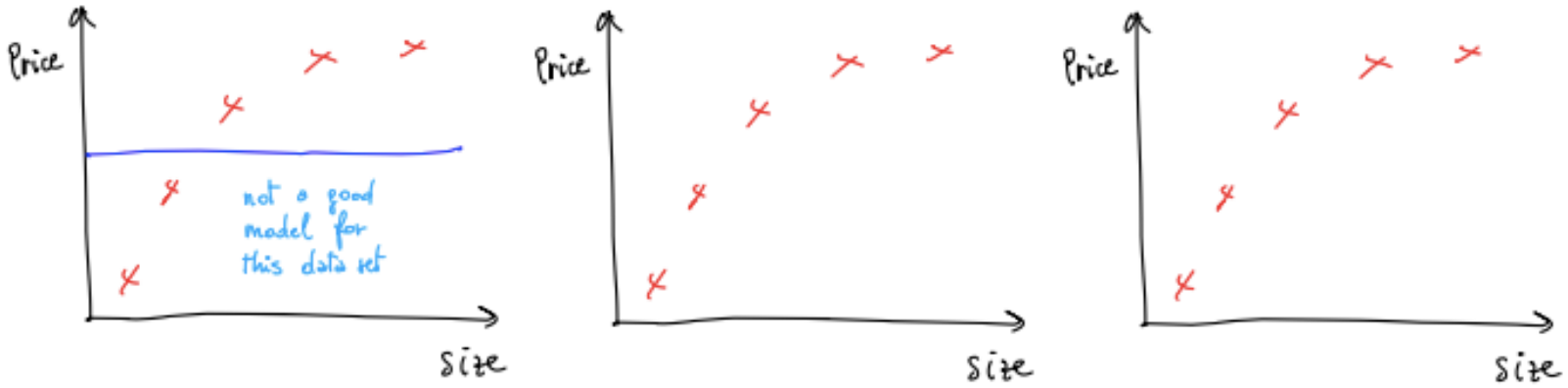


Linear regression (w regularization)

already applied to attack overfitting

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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



large λ (penalized)

$\lambda = 10'000$

$\theta_1, \dots \theta_n \approx 0$

$h_\theta(x) \approx \theta_0$

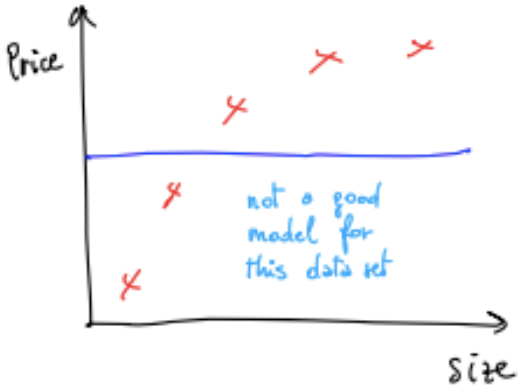
high bias (underfitting)

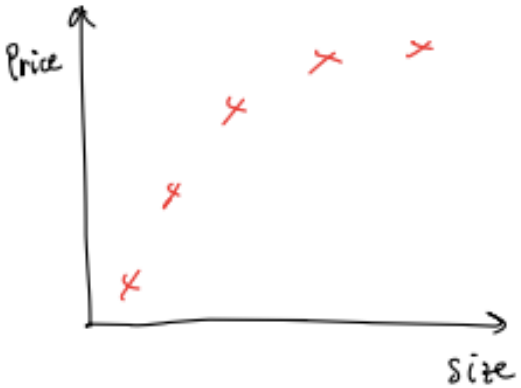
Linear regression (w regularization)

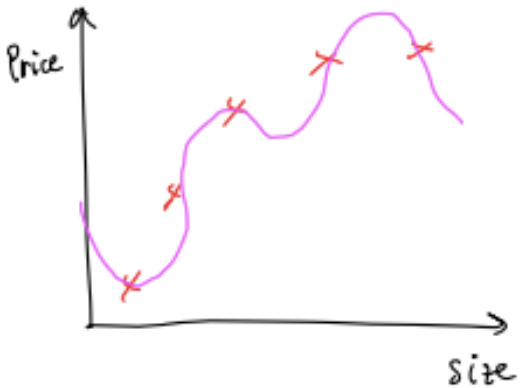
already applied to attack overfitting

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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







large λ

penalized

$\theta_1, \dots \theta_n \approx 0$

$\lambda = 10'000$

$h_\theta(x) \approx \theta_0$

high bias (underfitting)

small λ

$\lambda = 0$ $h_\theta(x)$ fitted

high variance (overfitting)

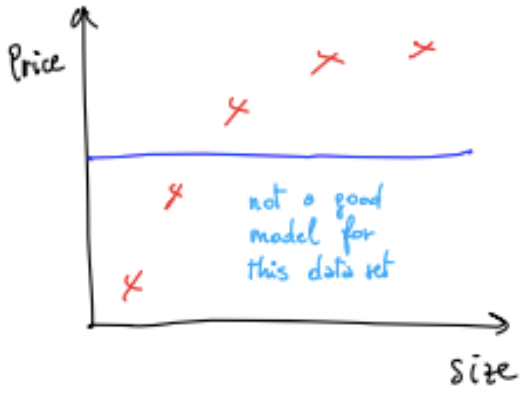
fitting w/o regularization

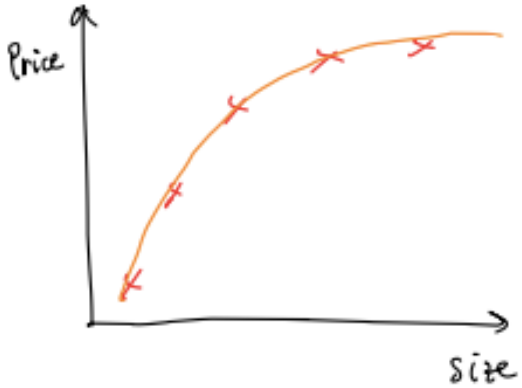
Linear regression (w regularization)

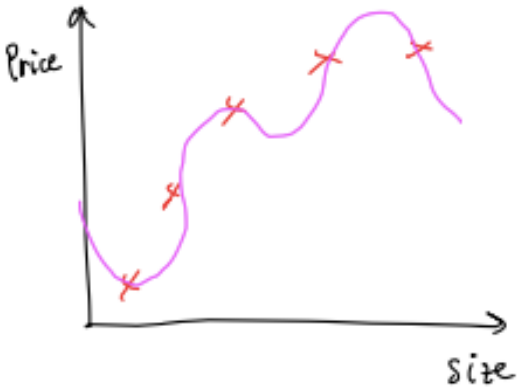
already applied to attack overfitting

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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







large λ

penalized

$\lambda = 10'000$

$\theta_1, \dots \theta_n \approx 0$

$h_\theta(x) \approx \theta_0$

high bias (underfitting)

intermediate λ

"just right"

small λ

$\lambda = 0$ $h_\theta(x)$ fitted

high variance (overfitting)

fitting w/o regularization

How do we automatically choose a good value for the regularisation parameter λ?

We want to apply the model selection procedure (seen before) to selecting the regularisation parameter λ.

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

Let's define J_{train}, J_{cv}, J_{test} to be the optimization objective but w/o the regular. term (AS BEFORE)

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)} \right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2$$

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

1. Try $\lambda = 0$ ← not using regul.
2. Try $\lambda = 0.01$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$

e.g. multiples of 2

...

12 Try $\lambda = 10$ (~ 10.24)

e.g.
12 different models,
each with a different λ

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

1. Try $\lambda = 0$ $\longrightarrow$ $\min_\theta J(\theta)$ $\longrightarrow$ $\theta^{(1)}$
2. Try $\lambda = 0.01$ $\longrightarrow$ $\min_\theta J(\theta)$ $\longrightarrow$ $\theta^{(2)}$
3. Try $\lambda = 0.02$ $\dashrightarrow$ $\longrightarrow$ $\theta^{(3)}$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$

...

12 Try $\lambda = 10$ $\dashrightarrow$ $\longrightarrow$ $\theta^{(12)}$

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

1. try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$
2. try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$
3. try $\lambda = 0.02$ → → $\theta^{(3)}$
4. try $\lambda = 0.04$
5. try $\lambda = 0.08$

...

12 try $\lambda = 10$ → → $\theta^{(12)}$

evaluate them with CV set

$J_{cv}(\theta^{(1)})$

$J_{cv}(\theta^{(2)})$

$J_{cv}(\theta^{(3)})$

...

$J_{cv}(\theta^{(12)})$

pick model j that gives smallest $J_{cv}(\theta^{(j)})$ => you get your λ

first step

second step

Choosing the regularization parameter λ (automatically!)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

1. Try $\lambda = 0$
2. Try $\lambda = 0.01$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$

...

12 Try $\lambda = 10$

(...) =>

lowest is e.g. $J_{cv}(\theta^{(5)})$

Report the test set error for the $\theta^{(5)}$ choice, i.e.:

$J_{test}(\theta^{(5)})$

So, this is model selection method applied also to **selection of the best regularization parameter λ**

- it is the logic behind the implementation of various hyper parameters scan method you will just import and use in your code!

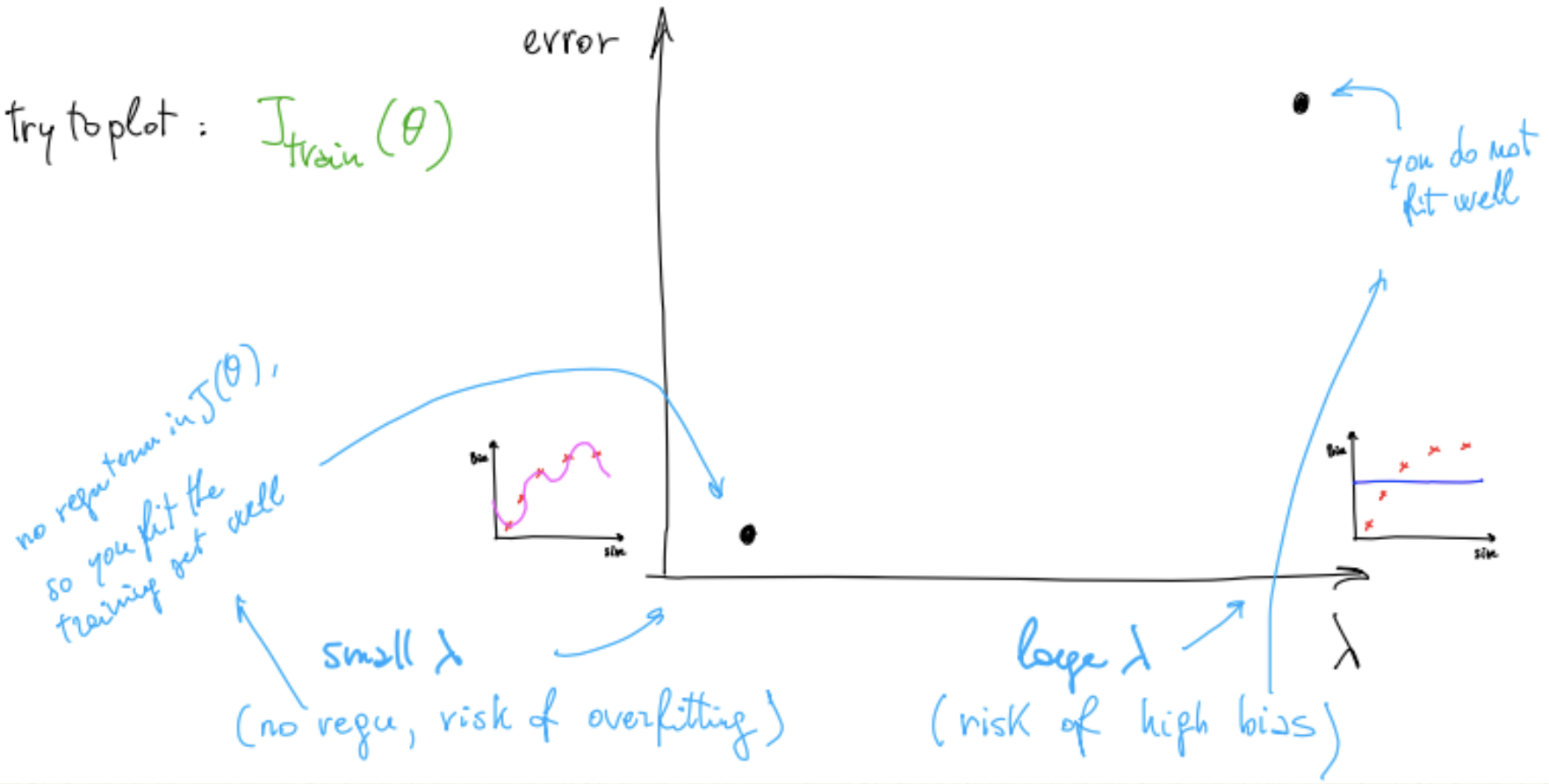
One last thing we should cover is to get a better understanding of how cross validation and training errors vary as we vary the regularisation parameter λ.

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

$$\rightarrow \quad J_{train}(\theta) = \frac{1}{2m_{train}}\sum_{i=1}^{m}\left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)}\right)^2$$

$$\rightarrow \quad J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2$$

try to plot : $J_{train}(\theta)$



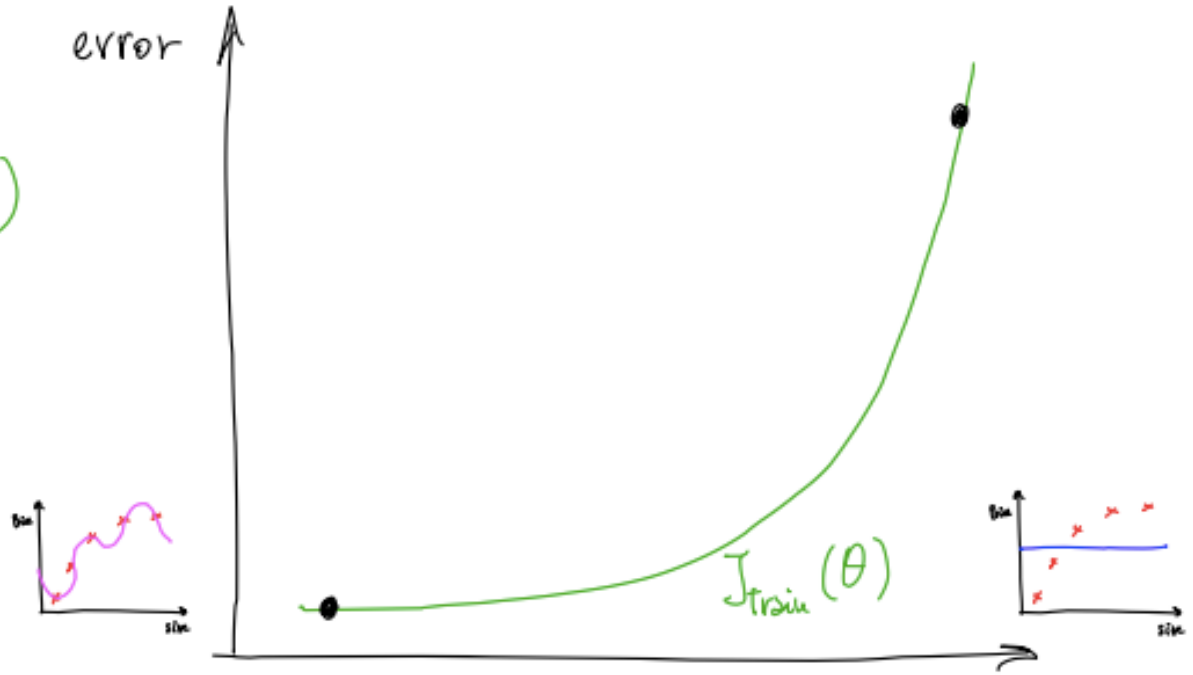
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$

$$\rightarrow \quad J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)} \right)^2$$

$$\rightarrow \quad J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$$

try to plot : $J_{train}(\theta)$

DONE !

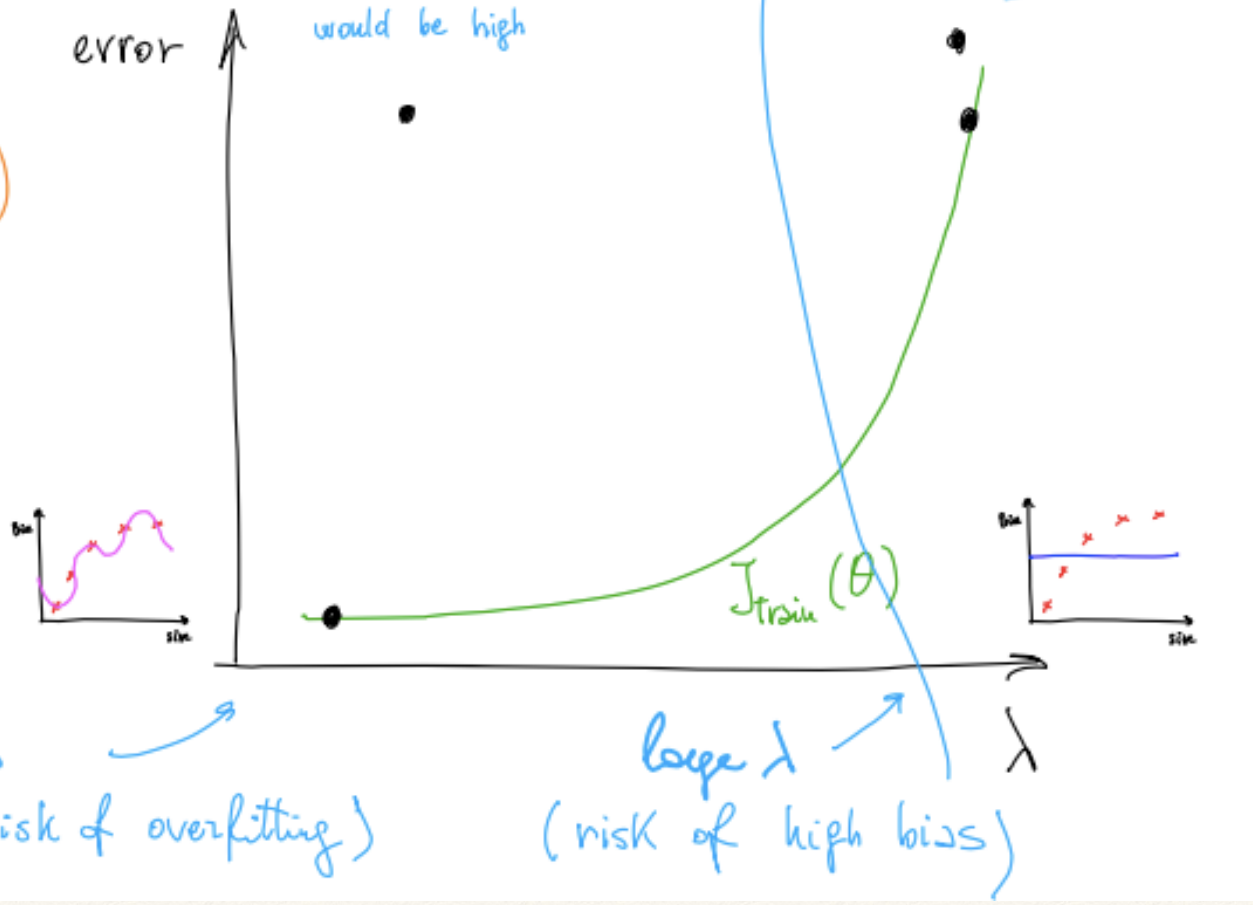


$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$

$$\rightarrow \quad J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}_{train}) - y^{(i)}_{train} \right)^2$$

$$\rightarrow \quad J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta(x^{(i)}_{CV}) - y^{(i)}_{CV} \right)^2$$

try to plot : $J_{CV}(\theta)$



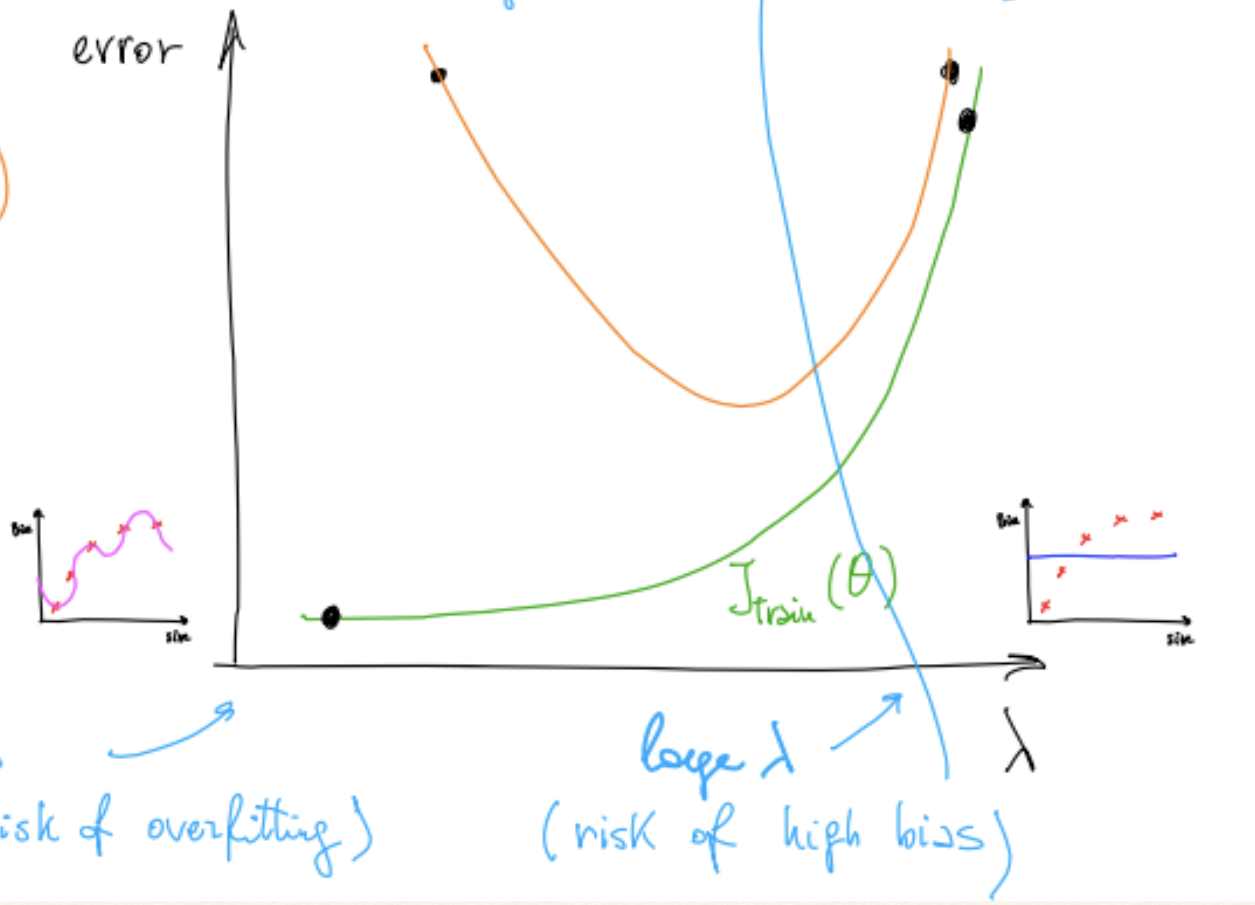
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$

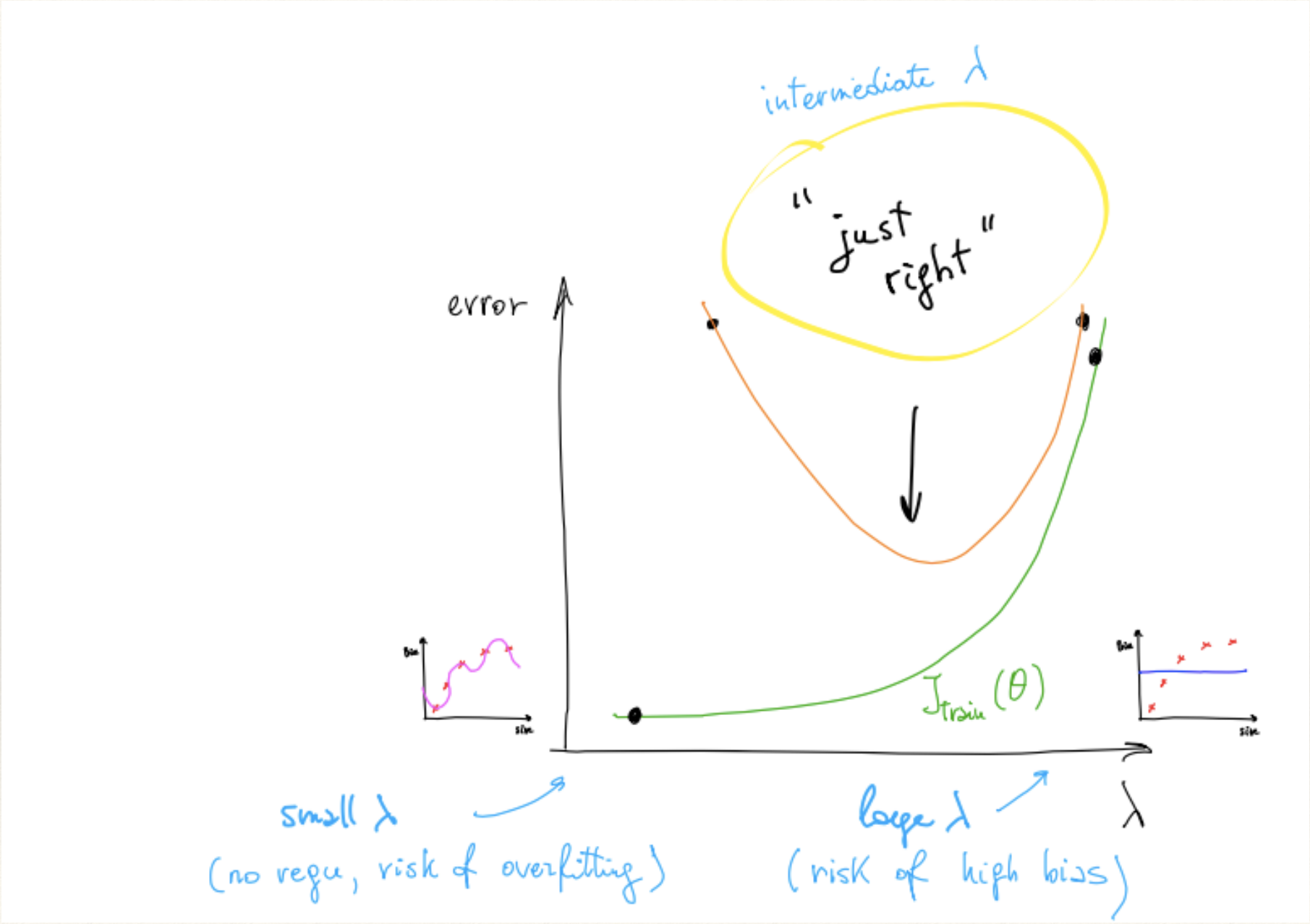
$$\rightarrow \quad J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}_{train}) - y^{(i)}_{train} \right)^2$$

$$\rightarrow \quad J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x^{(i)}_{cv}) - y^{(i)}_{cv} \right)^2$$

try to plot : $J_{cv}(\theta)$

DONE!




intermediate λ
"just right"
error
$J_{train}(\theta)$
small λ
(no regu, risk of overfitting)
large λ
(risk of high bias)
λ

Summary (1/2)

If we plot our "price to pay" versus λ we see that:

- as λ increases, our fit becomes more rigid
- on the other hand, as λ approaches 0, we tend to overfit the data

So, how do we choose the best λ to be "just right"?

In order to choose the model and the regularisation term λ we could, as a principle, to:

- create a list of λs (i.e. 0, 0.01, 0.02, …, 10)
- create a set of models with different degrees of any other variants
- iterate through the λs and for each λ go through all the models and learn Θs
- compute the cv error using the learned Θ (computed with a λ) on the $J_{cv}(\Theta)$ without regularisation (i.e. λ=0)
- select the best combination (Θ,λ) that produces the lowest error on the cv set
- use (Θ,λ), apply it to $J_{test}(\Theta)$ to see if it has a good generalisation of the problem

Summary (2/2)

Note: the drawn curves are somewhat cartoon-like; real curves are not so idealised, and more messy and noisy

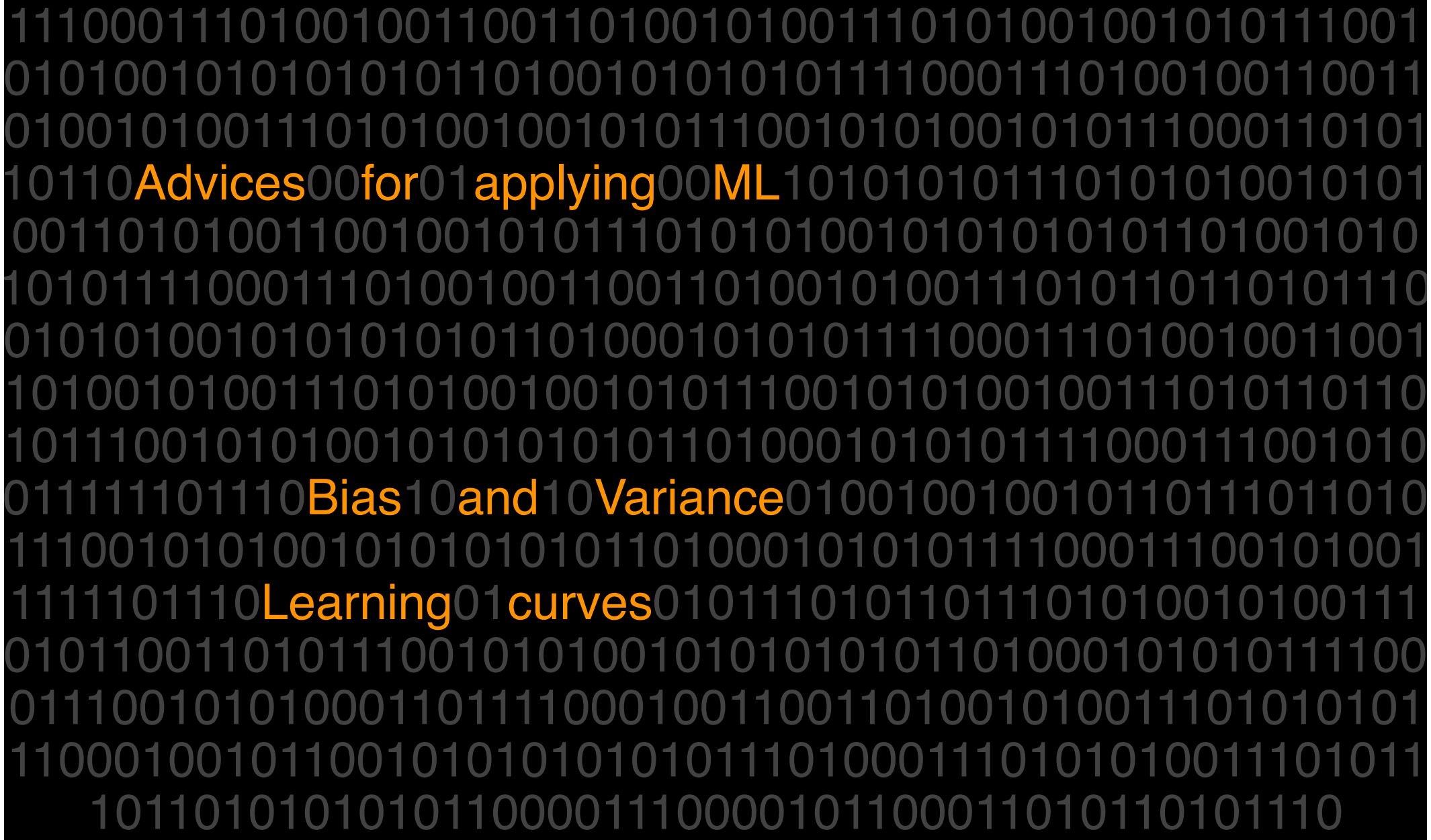
But:

Trends are meaningful. Sometimes on a dataset you spot these trends if you plot a cv error, and then you can (either manually or automatically) select a point that minimises the cv error and yields a value of λ **that** corresponds to a low cv error - i.e. presumably also a low generalisation error.

In other words, plots like these help to understand better what's going on, and help to verify that you are picking a good value for the regularisation parameter λ.

Next

We are ready to put together a diagnostic tool called **learning curves**.


Advices for applying ML
Bias and Variance
Learning curves

If either you wanted a sanity check that your algorithm is working correctly, or if you want to improve the performance of the algorithm, **learning curves** are often very useful to plot.

It is a tool often used to diagnose if a learning algorithm may be suffering from a bias problem or a variance problem, or a little bit of both.

Learning curves

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

start with 1; later

Plot now as a function of m (training set size)

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)} \right)^2$$

$$J_{CV}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$$

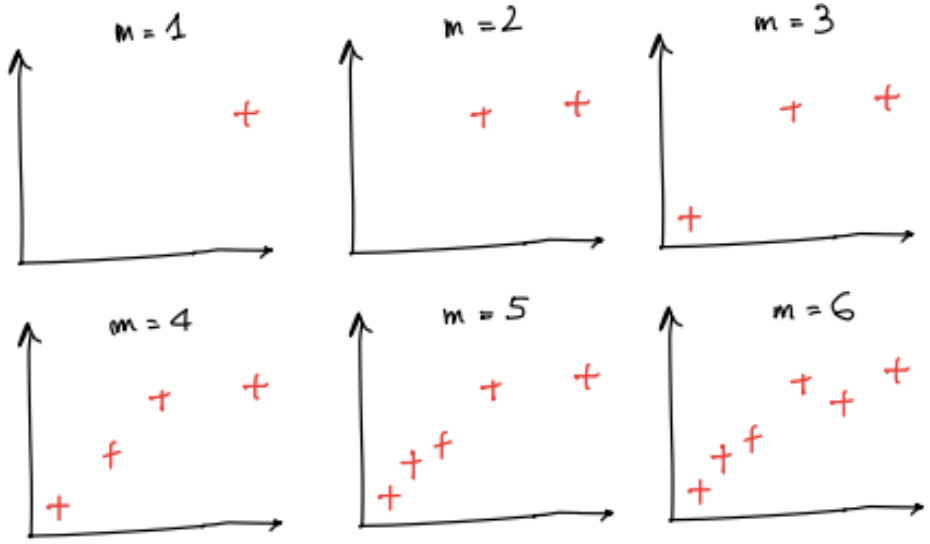
start with ; later

Plot now as a function of m (training set size)

To do so, start with $m = 1$ and add examples one by one ...

Suppose 6 ($m = 6$) examples, and you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

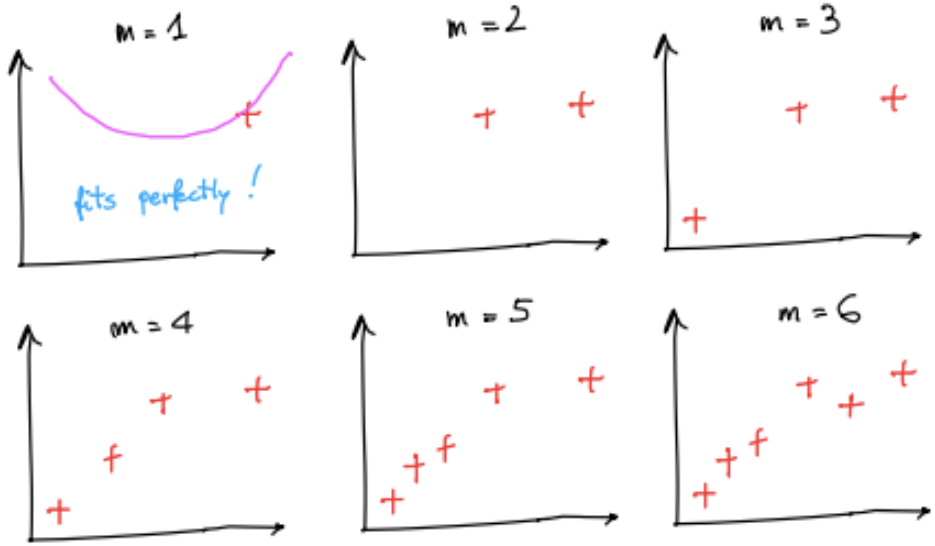
$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

Plot now as a function of m (training set size)

To do so, start with $m = 1$ and add examples one by one ...

Suppose 6 ($m = 6$) examples, and you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

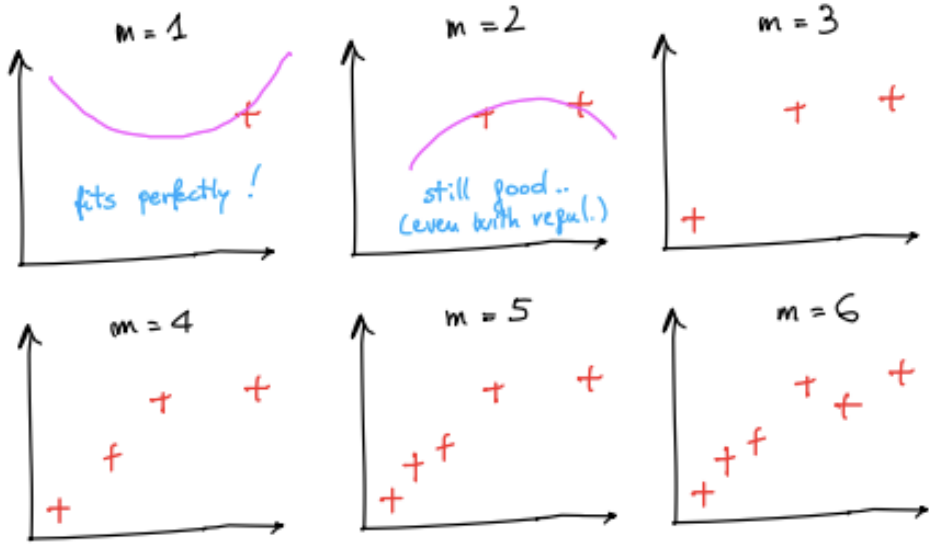
$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

Plot now as a function of m (training set size)

To do so, start with $m = 1$ and add examples one by one ...

Suppose 6 ($m = 6$) examples, and you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

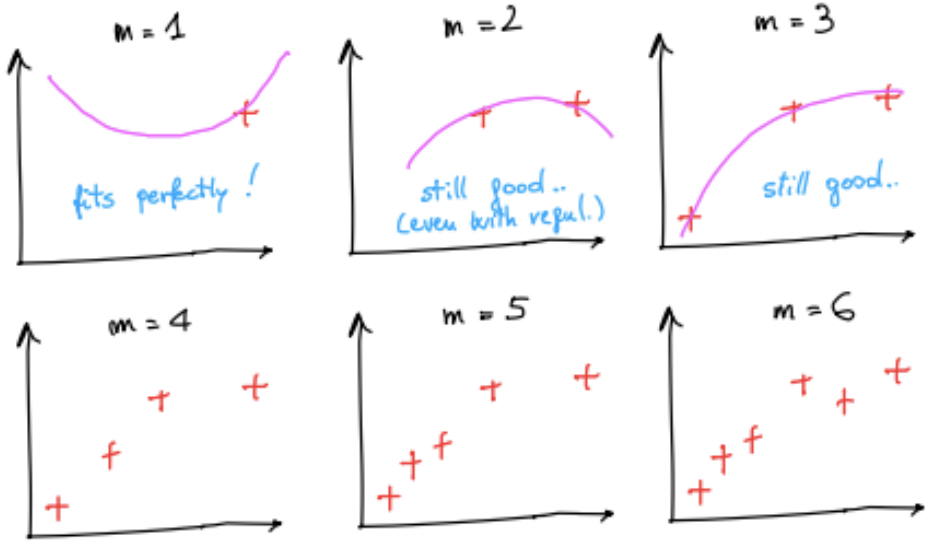
$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

Plot now as a function of m (training set size)

To do so, start with $m = 1$ and add examples one by one ...

Suppose 6 ($m = 6$) examples, and you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

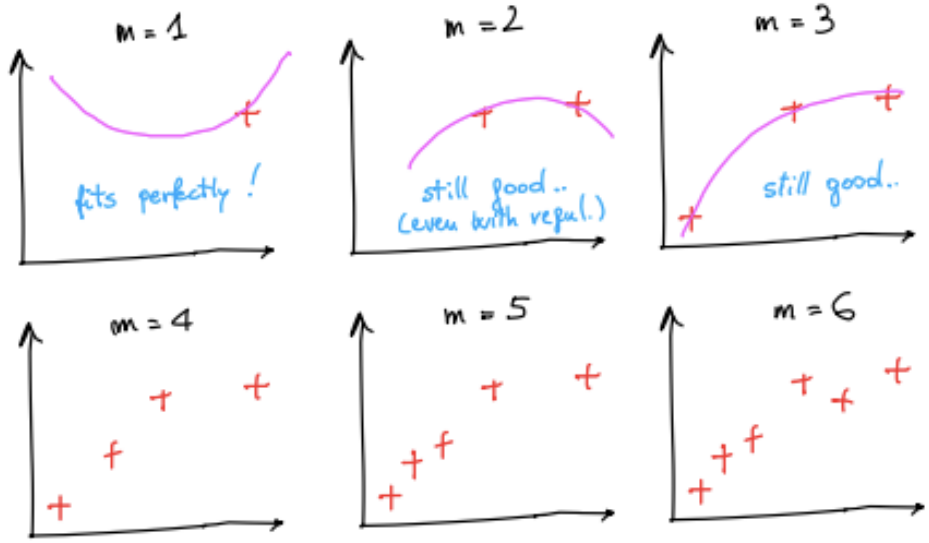
$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

Plot now as a function of (m) (training set size)

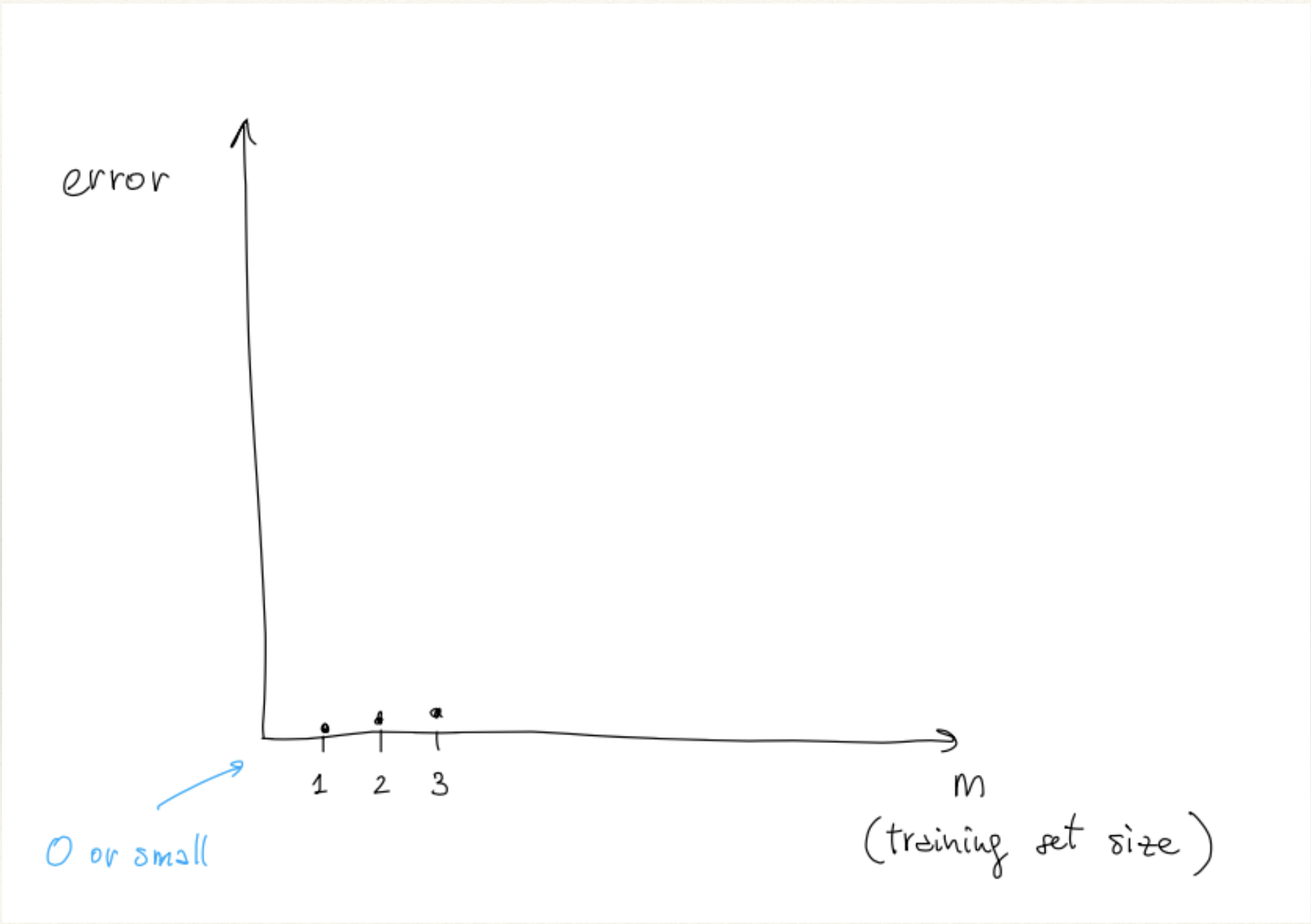
To do so, start with $m = 1$ and add examples one by one ...

Suppose (6) examples, and ($m = 6$) you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$m = 1, 2, 3 \Rightarrow$ training error w/o regul. is 0 (or slightly higher than - but close to - w regul.)


error
1
2
3
m
(training set size)
0 or small

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

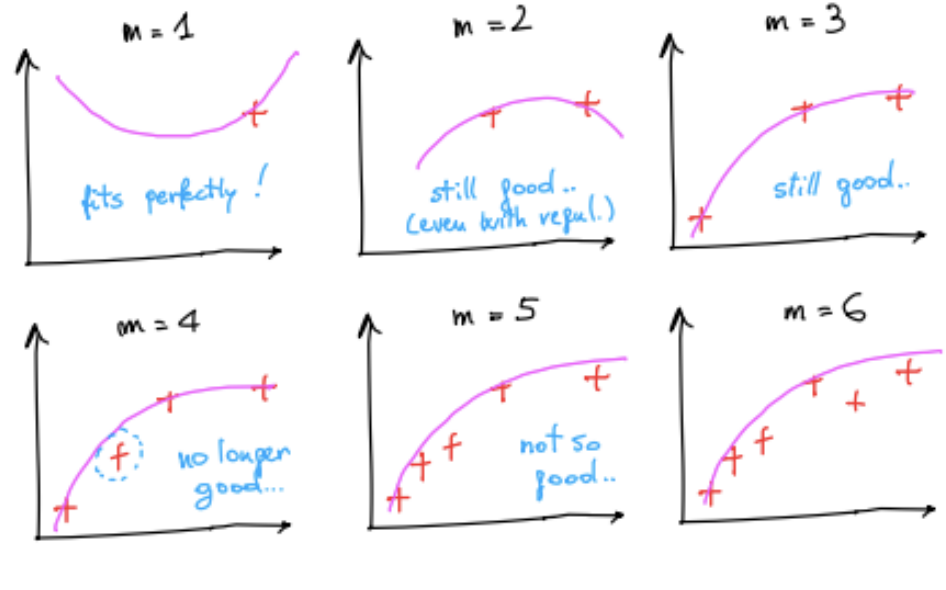
$$J_{CV}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)} \right)^2$$

Plot now as a function of m (training set size)

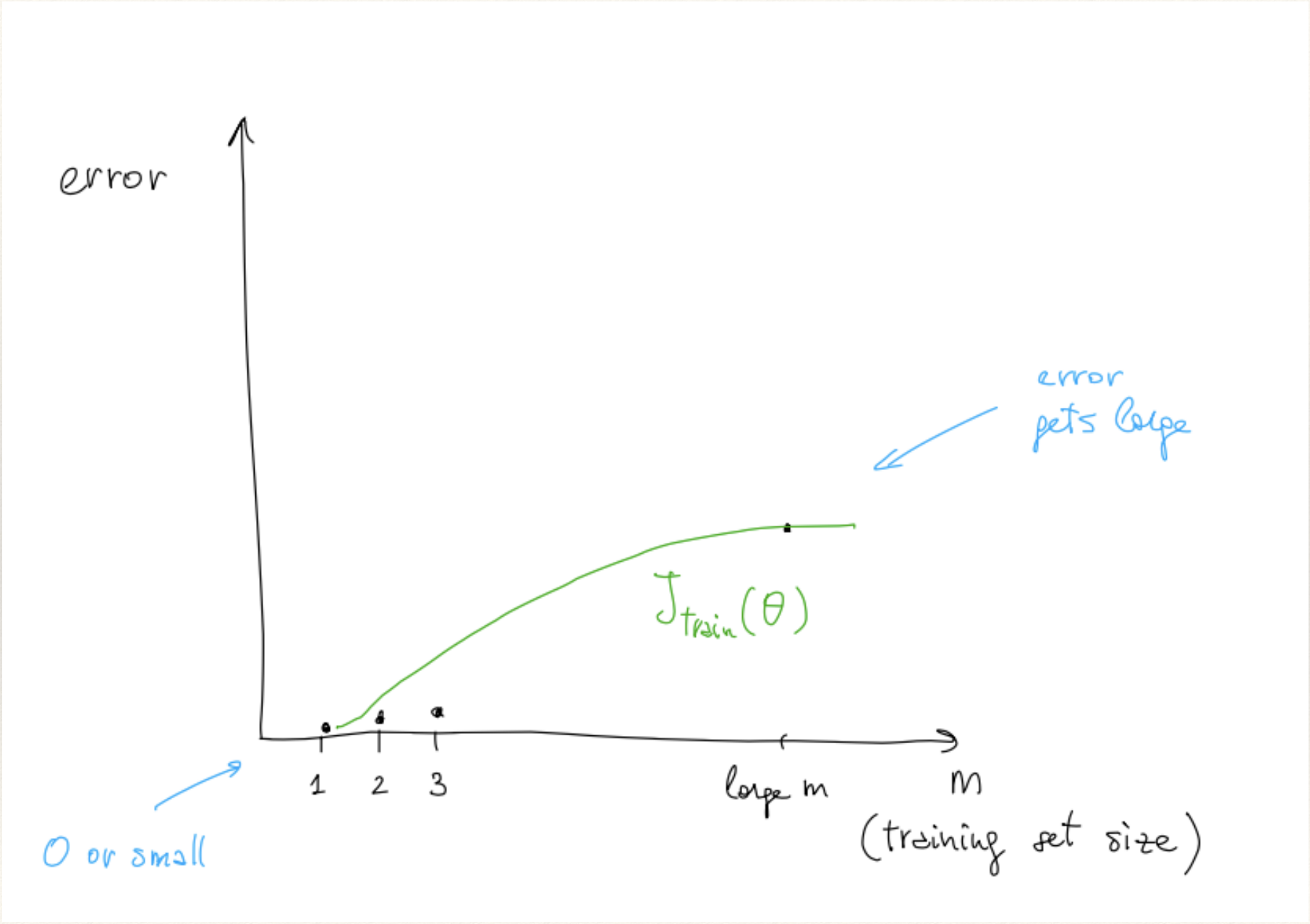
To do so, start with $m = 1$ and add examples one by one ...

Suppose 6 ($m = 6$) examples, and you try this:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



As m ↗, it becomes harder and harder to fit well all points. So the average training error ↗ as m ↗.

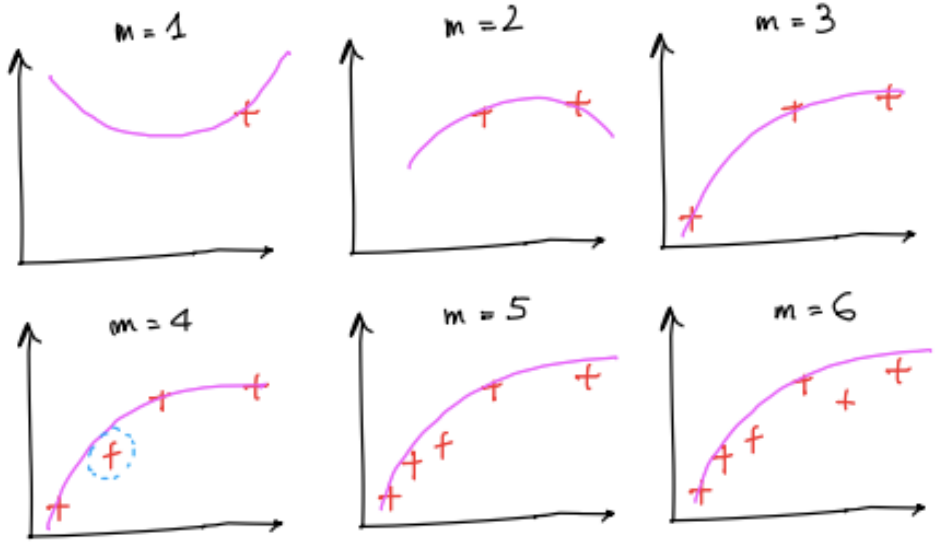

error
error gets large
$J_{train}(\theta)$
1
2
3
large m
m
(training set size)
0 or small

$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

Now:

Plot now as a function of m (training set size)



$$J_{train}(\theta) = \frac{1}{2m_{train}} \sum_{i=1}^{m} \left(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2$$

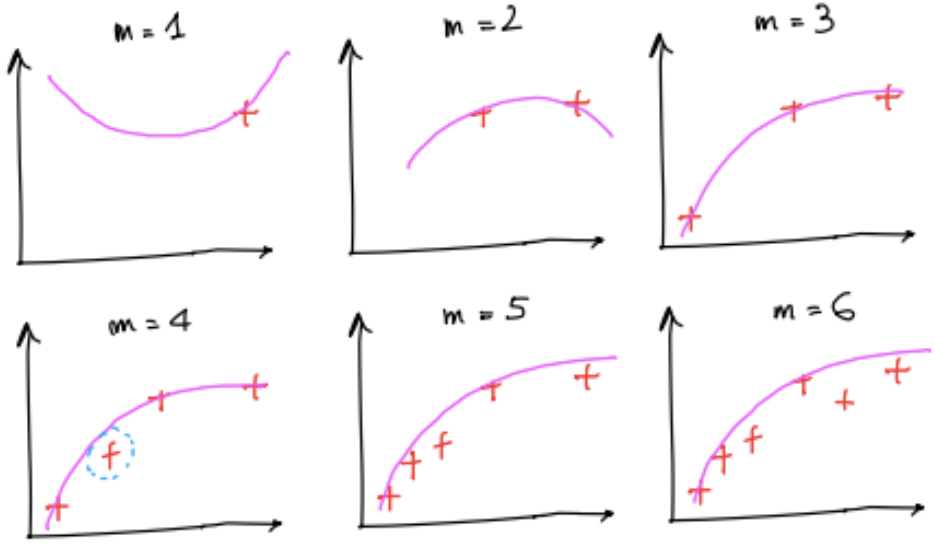
$$J_{CV}(\theta) = \frac{1}{2m_{CV}} \sum_{i=1}^{m_{CV}} \left(h_\theta\left(x_{CV}^{(i)}\right) - y_{CV}^{(i)} \right)^2$$

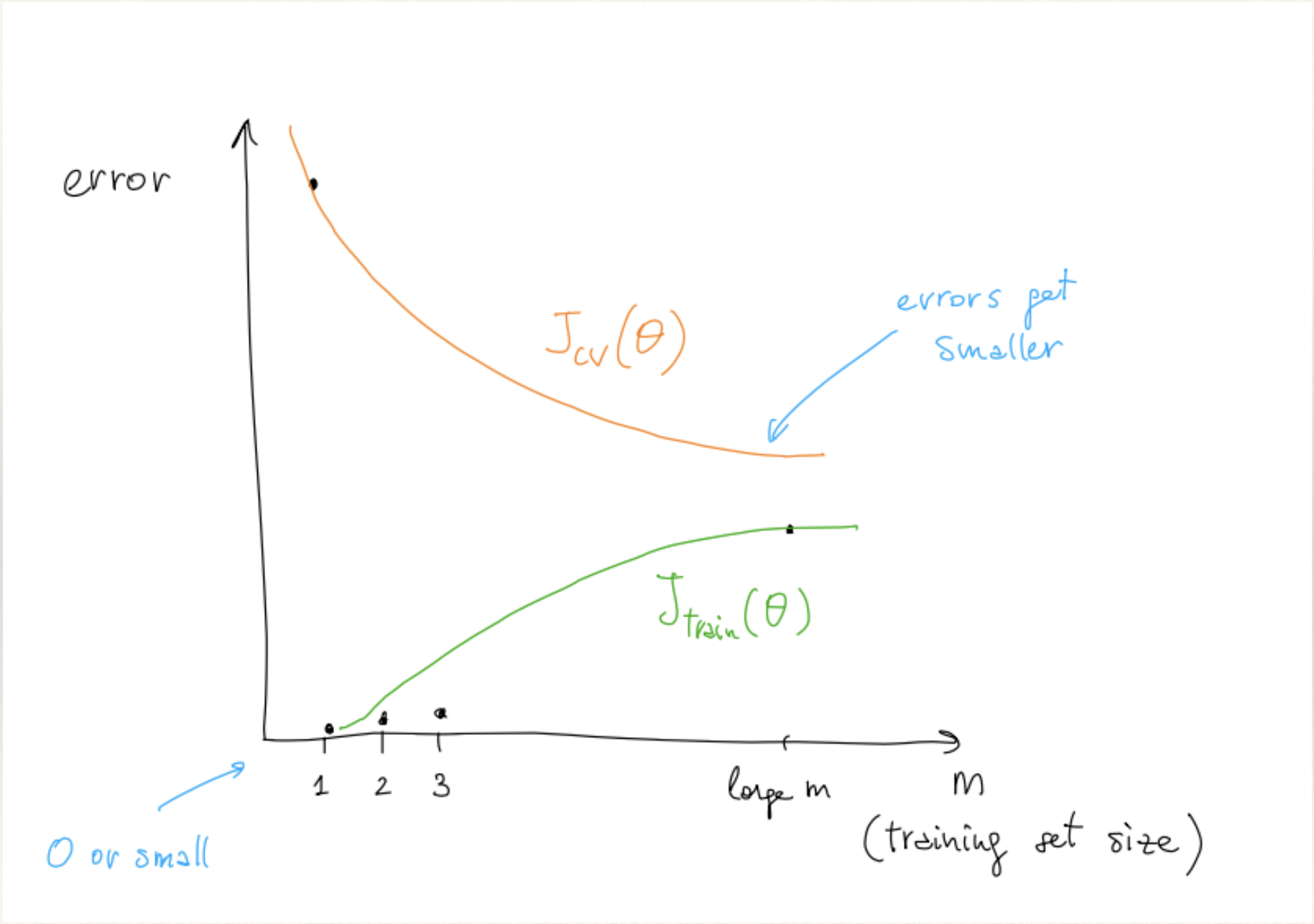
Now:

Plot now as a function of m (training set size)

$J_{CV}(\theta)$ tells me how well I can generalize to unseen data => with small m values, i.e. a small training set, I cannot generalize well => CV error would be large. When I get to larger m I start to see h fitting data better => CV error ↘

↑ the more data you have, the better you generalize to new data.



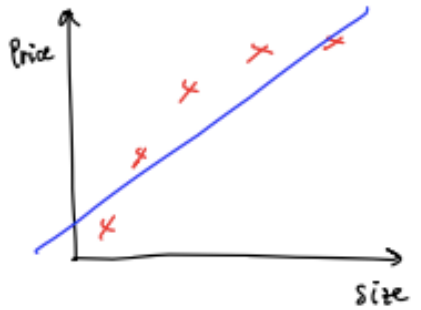

error
$J_{cv}(\theta)$
errors get smaller
$J_{train}(\theta)$
1
2
3
large m
m
(training set size)
0 or small

Now let's look at what the learning curves may look like if we have either high bias or high variance problems.

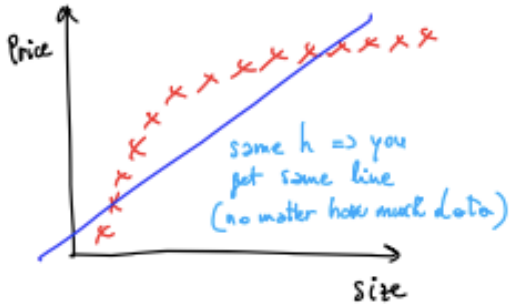
One at a time: **high-bias** first.

high - bias → aim at $J_{cv}(\theta)$ first →

Example: $h_\theta(x) = \theta_0 + \theta_1 x$

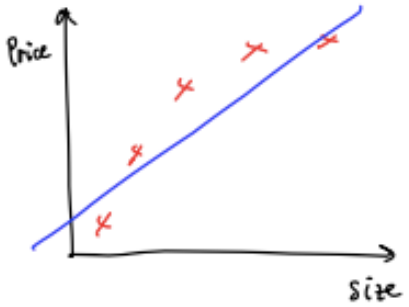


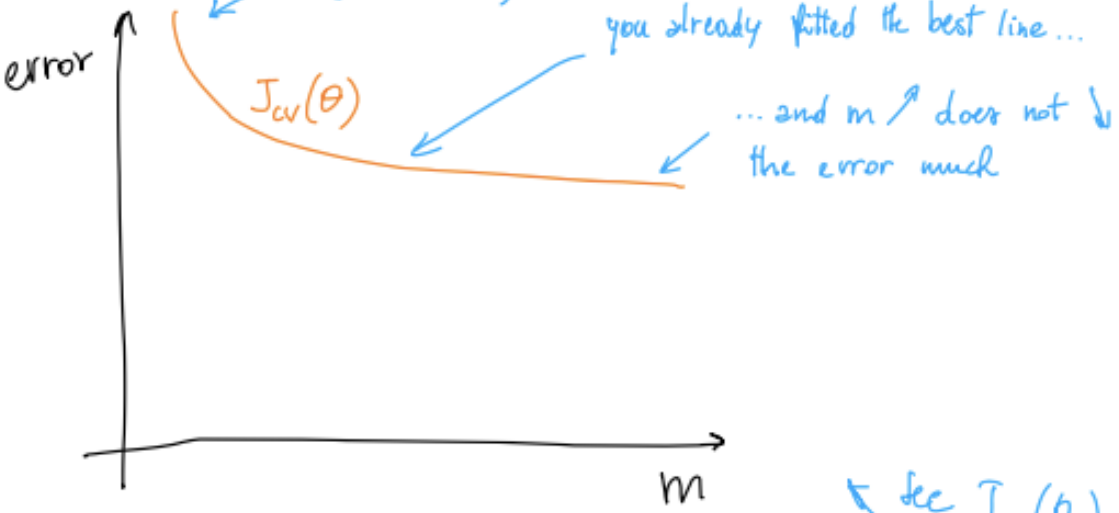
imagine we have many more examples



high - bias → aim at $J_{cv}(\theta)$ first →

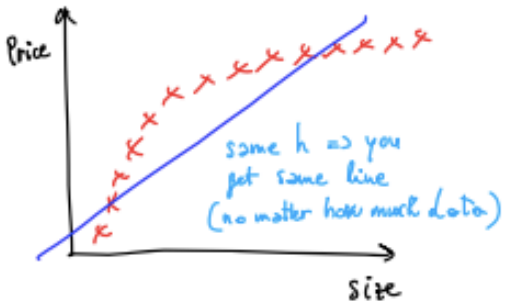
Example: $h_\theta(x) = \theta_0 + \theta_1 x$





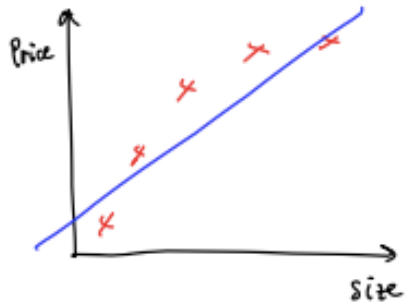
imagine we have many more examples

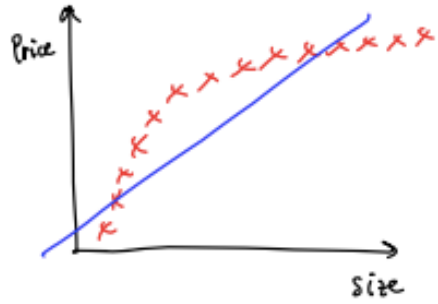
see $J_{cv}(\theta)$

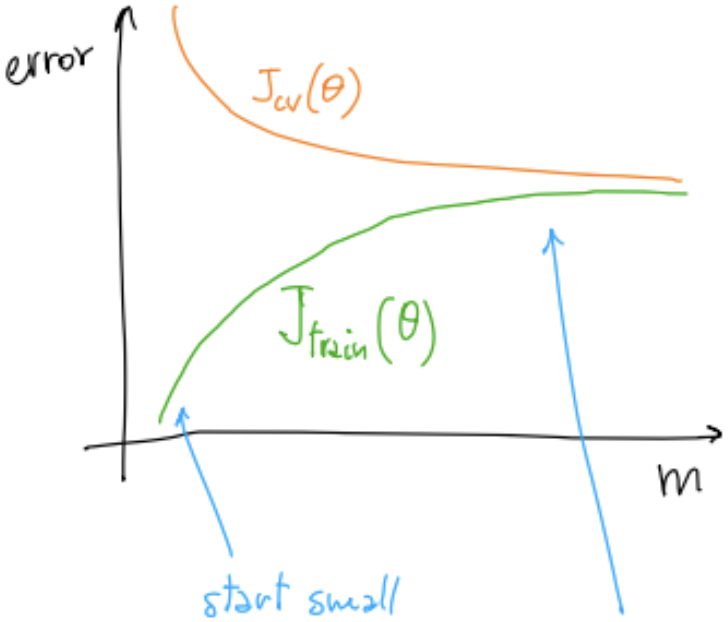


high-bias → aim at $J_{train}(\theta)$ now

Example: $h_\theta(x) = \theta_0 + \theta_1 x$



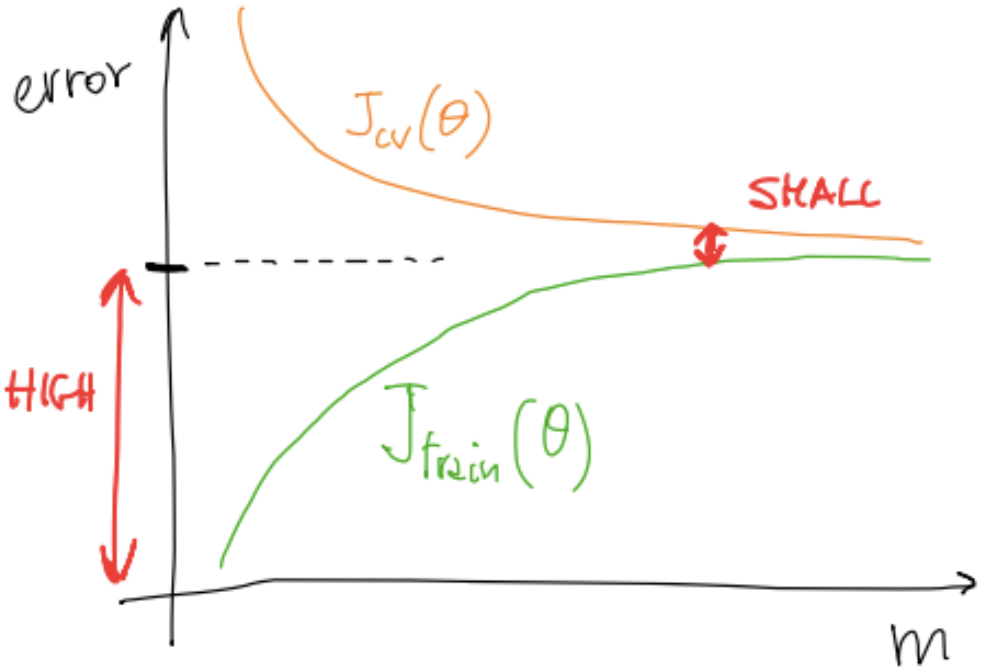




error ↗ and in high-bias it ends up close to CV error: so few params and so much data that the performance on training/CV set would be very similar

high - bias

The problem with high-bias is that both J_{train} and J_{cv} are high => ends up with a relatively high error



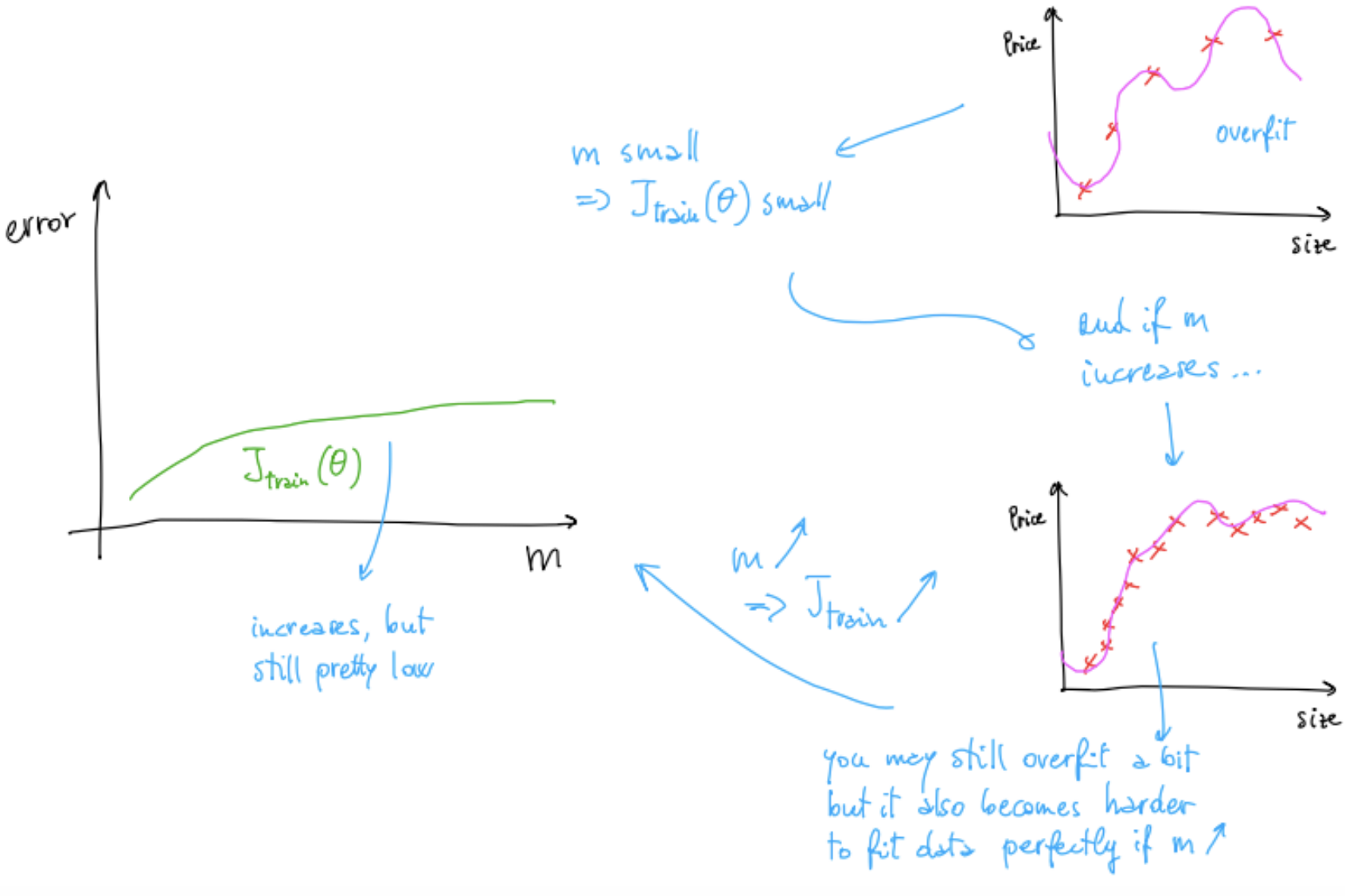
IMPORTANT : if a learning algo is suffering from high-bias, getting more training data (m ↑) will not (by itself) help much

does not help...

diagnostics info ← this can prevent you ← to waste time in collecting data for nothing...

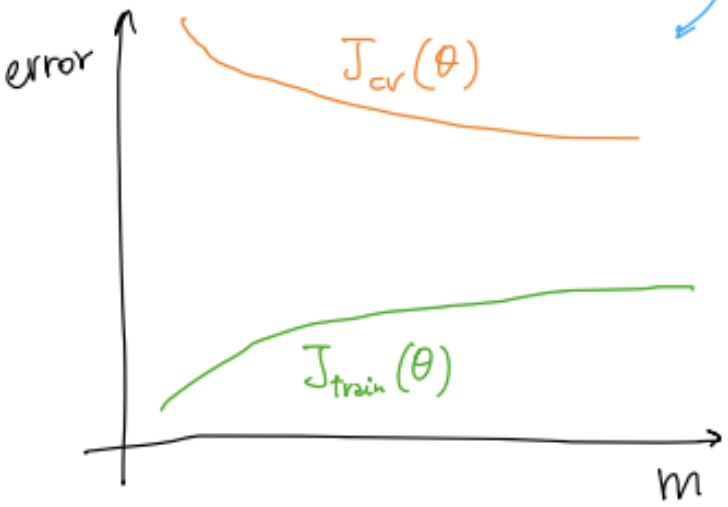
And now **high-variance**.

high-variance → aim at $J_{train}(\theta)$ first → Example: $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{100} x^{100}$ (and small λ)

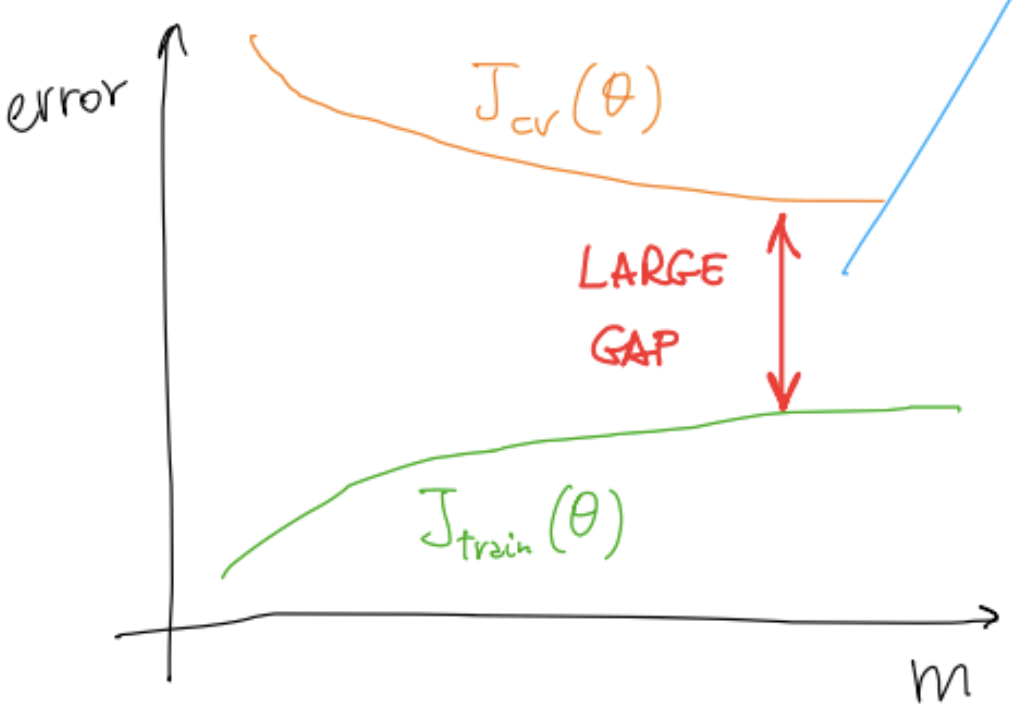


high-variance → aim at $J_{cv}(\theta)$ now

our h is overfitting ⇒ CV error would remain high even if we get more examples



high-variance



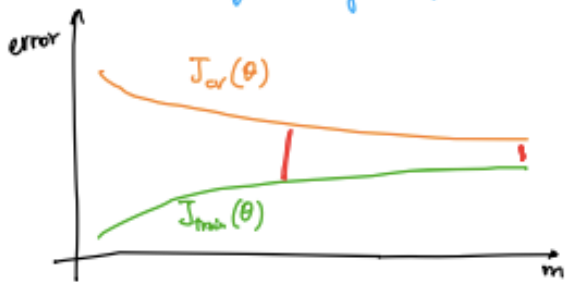
diagnostic info

this large gap is indicative (as diagnostic) of a high variance scenario.

↓

IMPORTANT: if a learning algo is suffering from high-variance, getting more training data (m ↑) will likely help!

↓ extrapolate to large m



Caveats (in all curves so far..)

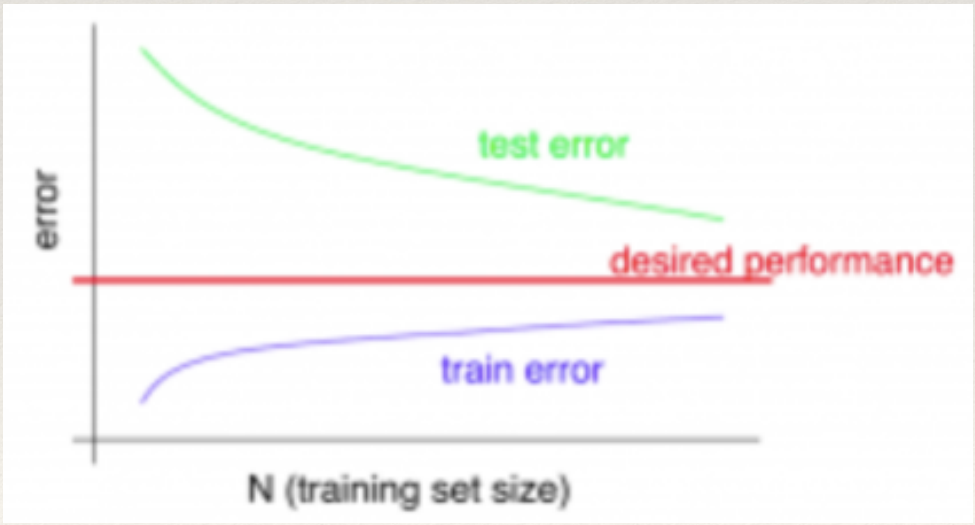
We have drawn quite clean and idealised curves.

- real life (as said already): more noisy curves, much messier..

Still, plotting learning curves like these can often help you figure out if your learning algorithm is suffering from bias or variance

high-bias

high-variance



Summary

If I train a learning algo on very few data points (e.g. 2,3) I will easily get error=0 as I can always find e.g. a quadratic function that touches exactly all these data points. So:

- as the training set gets larger, the error for a quadratic function increases
- the error value will plateau out after a certain m (= training set size)

Algo suffering from **high-bias**:

- <u>small m</u> (training set size): it causes $J_{train}(\Theta)$ to be low and $J_{cv}(\Theta)$ to be high
- <u>large m</u>: it causes both $J_{train}(\Theta)$ and $J_{cv}(\Theta)$ to be high, with $J_{train}(\Theta) \approx J_{cv}(\Theta)$

⇒ getting more training data will not (by itself!) help much

Algo suffering from **high-variance**:

- <u>small m</u>: it causes $J_{train}(\Theta)$ to be low and $J_{cv}(\Theta)$ to be high
- <u>large m</u>: $J_{train}(\Theta)$ increases with m, and $J_{cv}(\Theta)$ decreases with m, without levelling off. Also, $J_{train}(\Theta) < J_{cv}(\Theta)$ and the difference between them remains significant

⇒ getting more training data is likely to help!

Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

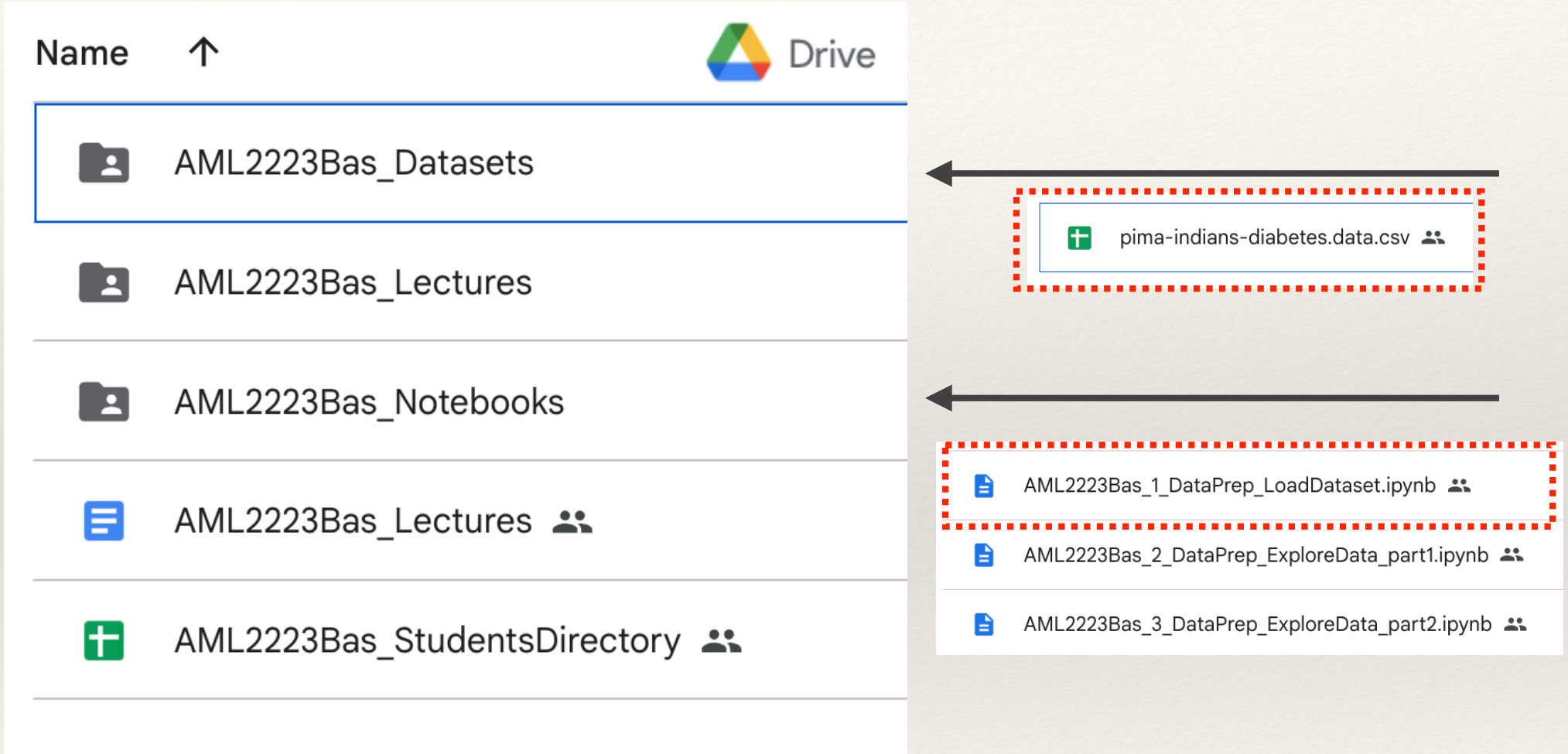
Hands-on:

1 - Data prep - load dataset

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Look into these:



Applied Machine Learning - Basic

Prof. Daniele Bonacorsi

Hands-on:

2 - Data prep - explore data (part 1)

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna

Look into these: