جامعة قطر
QATAR UNIVERSITY

CMPS 460 – Spring 2022

MACHINE LEARNING

Tamer Elsayed

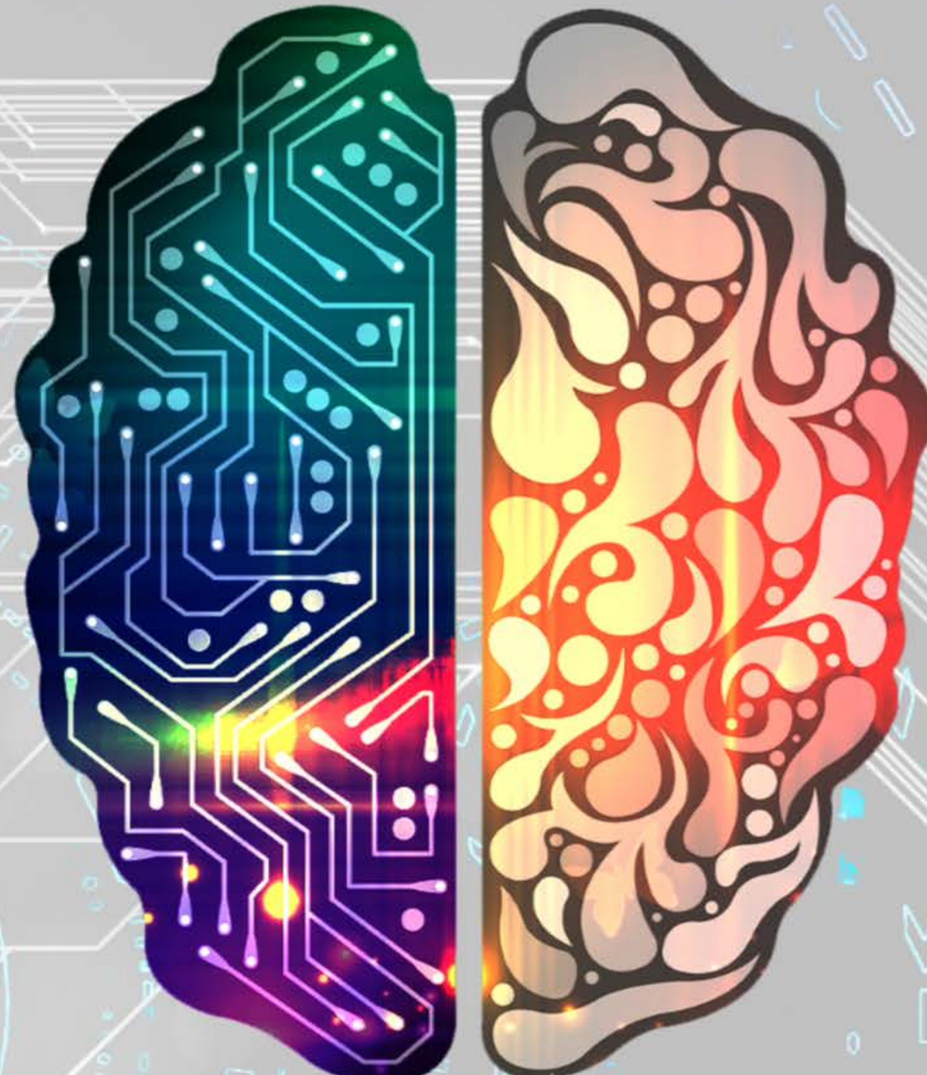


Image hosted by: WittySparks.com | Image source: Pixabay.com

2

Limits of Learning

Chapter 2

Machine Learning is <u>NOT</u> magic!

Machine Learning will <u>NOT</u> always work!

What we will learn in this session …

- Inductive Bias
- Why might ML fail?
- Overfitting/Underfitting
- How to test the model's generalizability

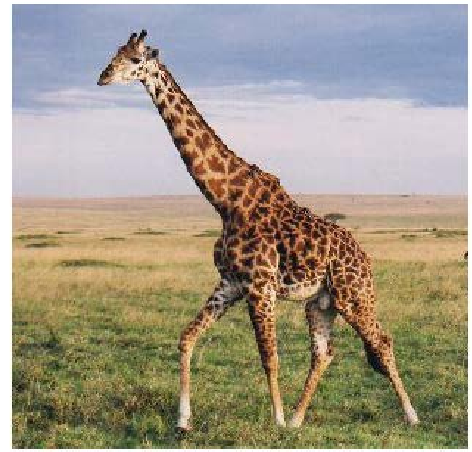
Inductive Bias

Examples of Class A

Class A

Examples of Class B

Class B



How about these?

Background in focus or not?

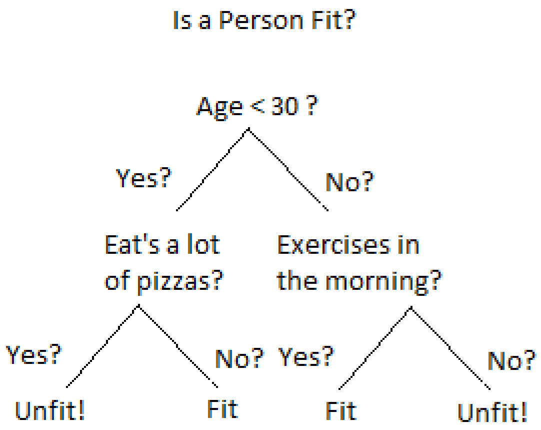
Inductive Bias

- In the absence of data that narrow down the relevant concept, what type of solutions are we more likely to prefer?
  - What we know before the data arrives!
- Many classifiers need assumptions about the nature of the relation between examples and classes.
- Some hypotheses are more probable than others.
  - e.g., nobody came up with the "background" classification.

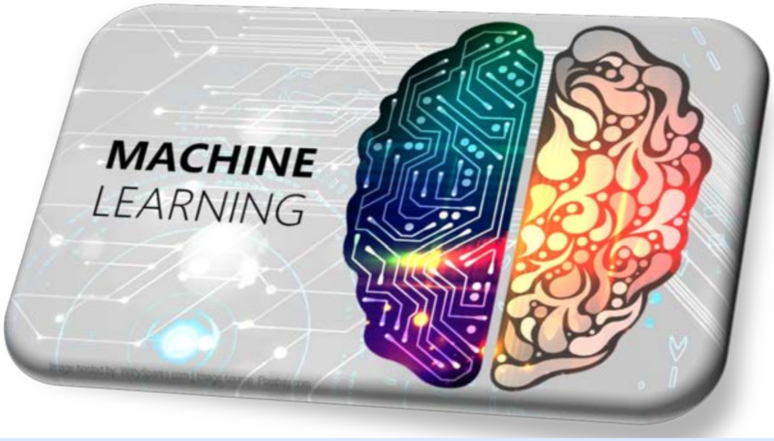
Approaches differ primarily
in the sort of inductive bias that they exhibit

Inductive Bias in DT learning

- Shallow DT: grow but with max depth ***d***.



What's the inductive bias here?



Not Everything is Learnable

Not everything is learnable

ML might fail on a task for many reasons:

- Very small training data
- Noisy training data
  - Noise could be in features, or in labels
- Features are not useful or insufficient
- Some examples might not have single correct answer
- Mismatch between inductive bias of learner and concept we aim to learn.

Sources of error!

Overfitting/Underfitting

Evaluating the learned hypothesis h

- Assume we've learned a tree h using the top-down induction algorithm.
- And it fits the training data ***perfectly.***

Are we done?

Can we guarantee we have found a good hypothesis?

Training error is not sufficient!

- Goal is **NOT** to build a model that gets 0% error on the training data.
  - this would be easy!
- A tree can classify training data perfectly, yet classify new examples incorrectly.

We care about generalization to new (unseen) examples

Overfitting …

- **Overfitting** is when you pay too much attention to idiosyncracies of the training data, and aren't able to generalize well.
  - Often this means that your model is fitting noise, rather than whatever it is supposed to fit.

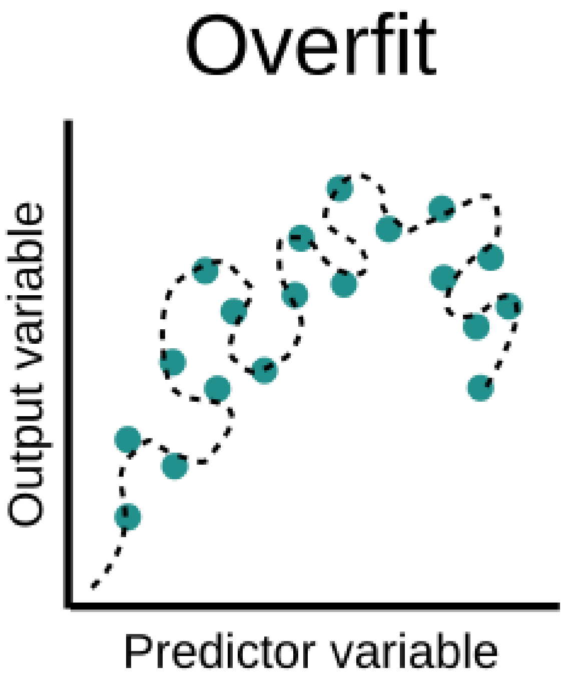
- Overfitting in DT?
- Overfitting in the student course understanding?

Underfitting

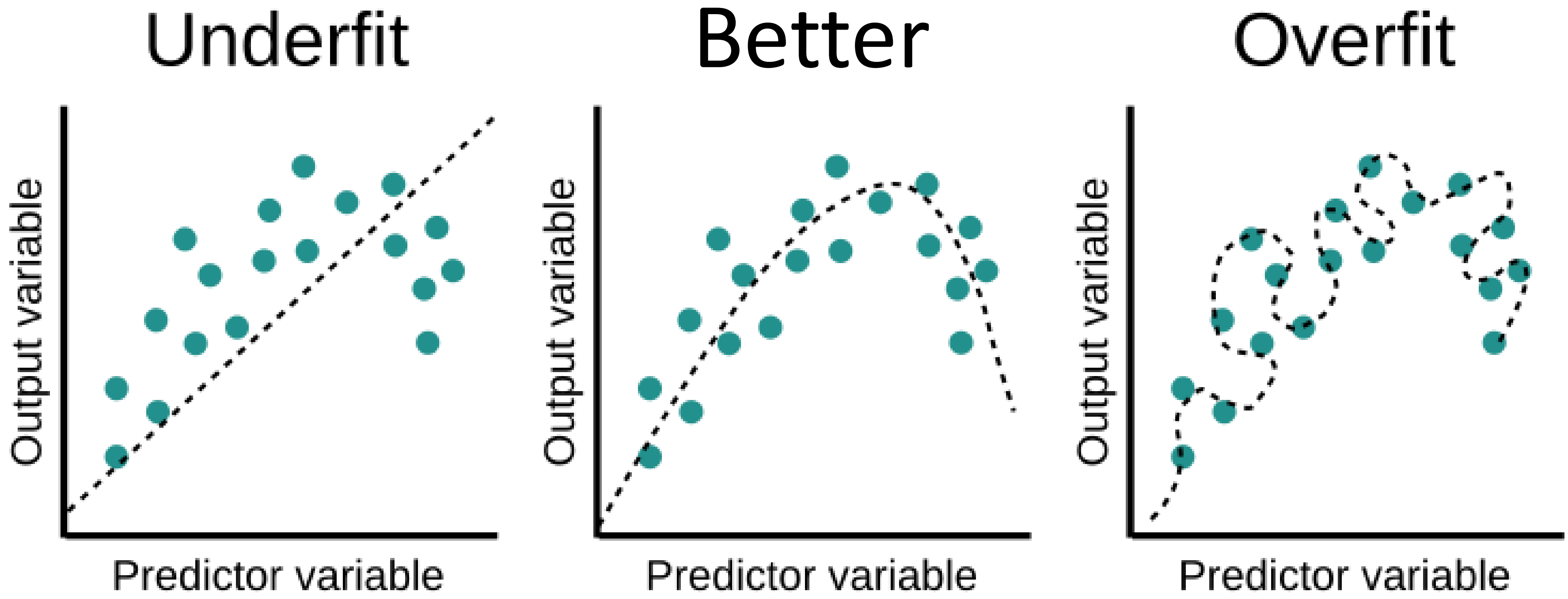
- How about an empty tree?

- Underfitting
  - Learning algorithm had the opportunity to learn more from training data, but didn't.
  - Or didn't have sufficient data to learn from.

Example



Example



Underfit
Better
Overfit
Output variable
Output variable
Output variable
Predictor variable
Predictor variable
Predictor variable

Proper Evaluation …

Recall: Formalizing Induction

- **Given**
  - a loss function l
  - a sample from some unknown data distribution D

- **Our task** is to compute a function f that has low ***expected*** error over D with respect to l.

$$\mathbb{E}_{(x,y)\sim D}\{l(y,f(x))\} = \sum_{(x,y)} D(x,y)l(y,f(x))$$

Overfitting

- Consider a hypothesis h and its:
  - Error rate over training data $error_{train}(h)$:

$$error_{train}(h) = \sum_{n=1}^{N} \frac{1}{N} l(y^{(n)}, h(x^{(n)}))$$

  - True error rate over all data $error_{true}(h)$:

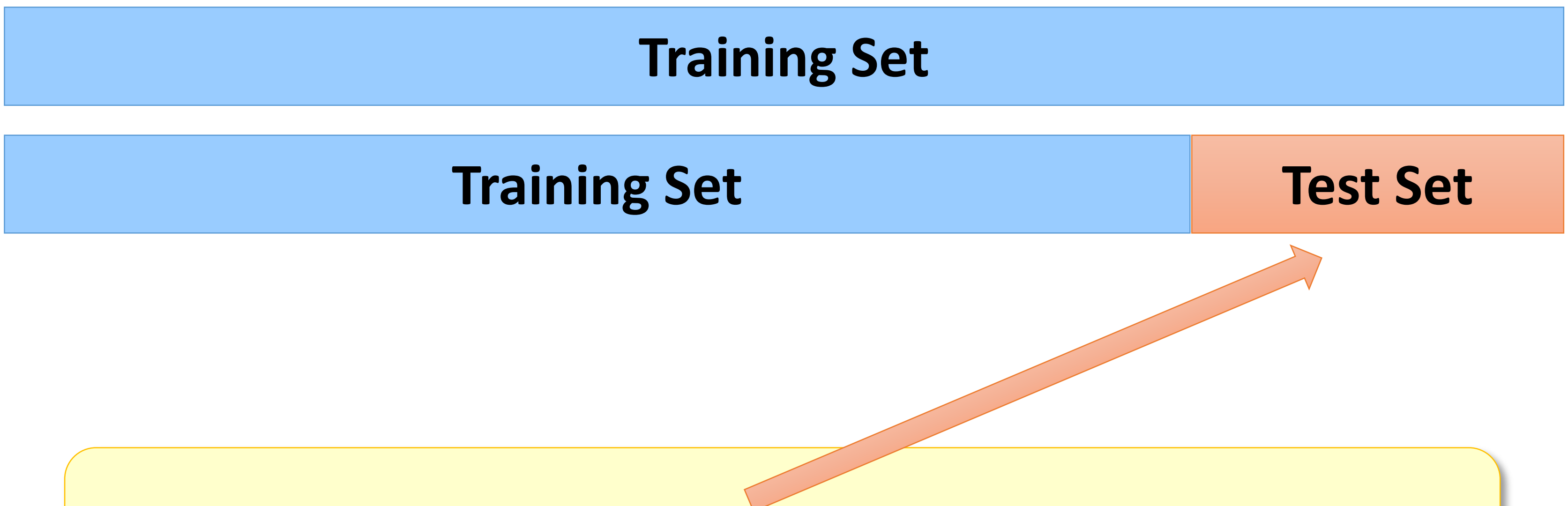
$$error_{true}(h) = \mathbb{E}_{(x,y)\sim D}\{l(y, h(x))\} = \sum_{(x,y)} D(x,y) l(y, h(x))$$

- We say h overfits the training data if
  - $error_{train}(h) < error_{true}(h)$
- Amount of overfitting
  = $error_{true}(h) - error_{train}(h)$

BUT
We don't know $\boldsymbol{error_{true}(h)}$!

Solution: Evaluate on Test Data

- Set aside a test set
  - some examples that will be used for evaluation



Don’t look at them during training!

Cardinal rule of machine learning

Never ever *touch* your test data!

Solution: Evaluate on Test Data

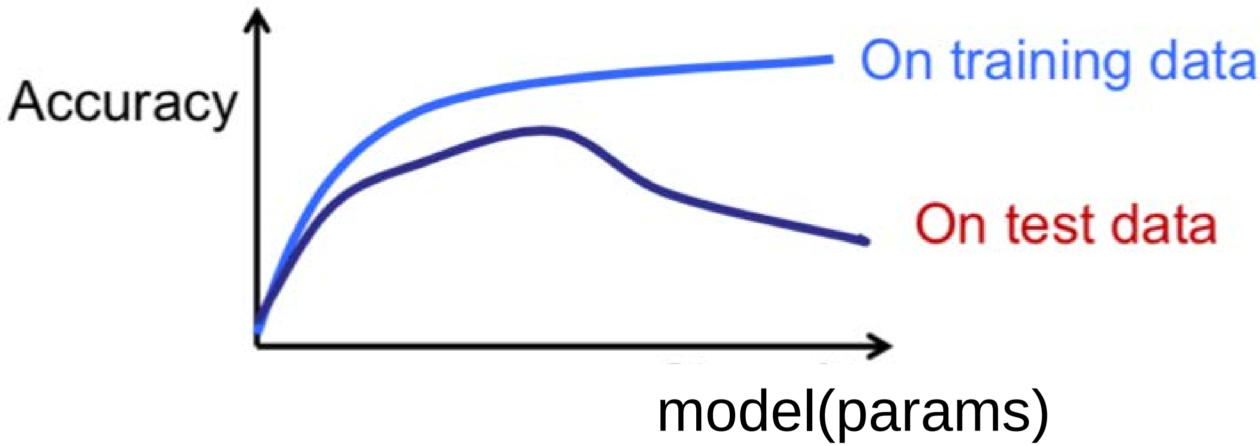
- Set aside a test set
  - some examples that will be used for evaluation

- Don’t look at them during training!

- After learning a DT, we calculate $error_{test}(h)$.

$$error_{test}(h) = \sum_{n=1}^{N} \frac{1}{N} l({y_{test}}^{(n)}, h({x_{test}}^{(n)}))$$

Overfitting



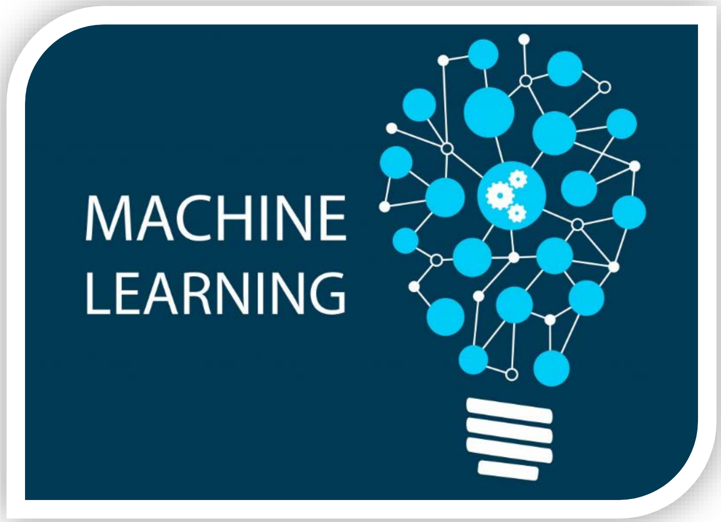
- Learned model overfits the training data when its accuracy on the training data goes up but its accuracy on unseen data goes down.
  - e.g., resulting tree doesn't generalize.
- Often this means that your model is fitting noise, rather than whatever it is supposed to fit.

Overfitting

Another way of putting it:

A hypothesis h is said to overfit the training data, if there is another hypothesis h', such that:

- h has a smaller error than h' on the training data
- but h has larger error on the test data than h'.



Model, Parameters, and Hyper-parameters

Models & Parameters

- The model tells us what sort of things we can learn, and also tells us what our inductive bias is.
- For most models, there will be associated parameters.
  - we use the data to decide on.

DT parameters?

- The job of a learning algorithm is to take data and figure out a good set of parameter values.

Hyper-Parameters

- Many learning algorithms will have additional knobs that you can adjust.
  - In most cases, these knobs amount to tuning the inductive bias of the algorithm.
- Called hyper-parameters.
  - parameters that control other parameters of the model.

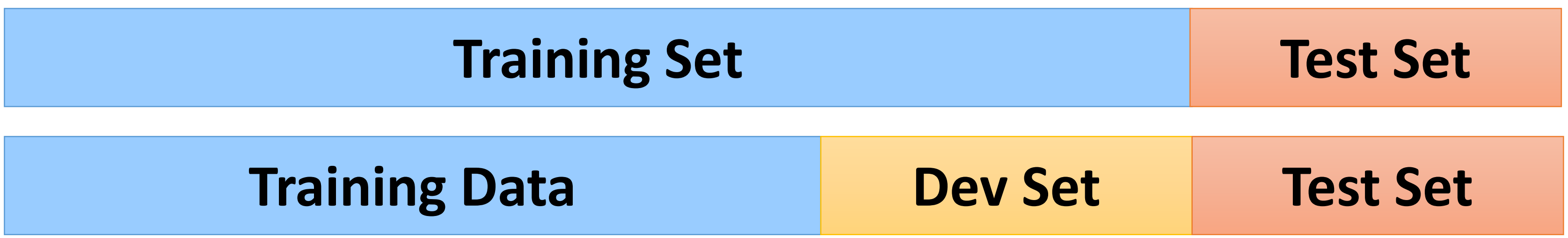
DT hyper-parameters?

Tuning Hyper-Parameters

- On training data?
- On testing data?

How?

Train/Dev/Test Sets



Train/Dev/Test Sets

In practice, we always split examples into 3 distinct sets:

- **Training set**
  - Used to learn the parameters of the ML model
    - e.g., nodes and branches of the decision tree
- **Development set**
  - aka *tuning set*, aka *validation set*, aka *held-out data*
  - Used to learn/tune hyper-parameters
    - e.g., max depth of decision tree
- **Test set**
  - Used to evaluate how well we're doing on new unseen examples

Example: DT

Training Data	Dev Set	Test Set

$\text{tree}_0 \longrightarrow \text{acc}_{0,dev}$

$\text{tree}_1 \longrightarrow \text{acc}_{1,dev}$

$\text{tree}_2 \longrightarrow \text{acc}_{2,dev}$

$\text{tree}_3 \longrightarrow \boxed{\text{acc}_{3,dev}} \dashrightarrow \text{acc}_{3,test}$

$\text{tree}_4 \longrightarrow \text{acc}_{4,dev}$

The general approach …

1. Split your data into 70% training data, 10% development data and 20% test data.
2. For each possible setting of your hyperparameters:
   (a) Train a model using that setting of hyperparameters on the training data.
   (b) Compute this model's error rate on the development data.
3. From the above collection of models, choose the one that achieved the lowest error rate on development data.
4. Evaluate that model on the test data to estimate future test performance.

What You Should Know So Far …

- Decision Trees
  - What is a decision tree, and how to induce it from data
- Fundamental Machine Learning Concepts
  - Difference between memorization and generalization
  - What inductive bias is, and what its role in learning is
  - What underfitting and overfitting mean
  - How to take a task and cast it as a learning problem
- Why you should never ever touch your test data!!