Pitfalls of applying Machine Learning to Scientific Software

and tips on how to avoid them

Davide Del Vento & Alessandro Fanfarillo *NCAR*

Disclaimer

- A funny speaker is needed

Disclaimer

- A funny speaker is needed
- Instead you've got me

Disclaimer - cont

- No new scientific results
- Not derogatory of the techniques
- Not criticism of researchers

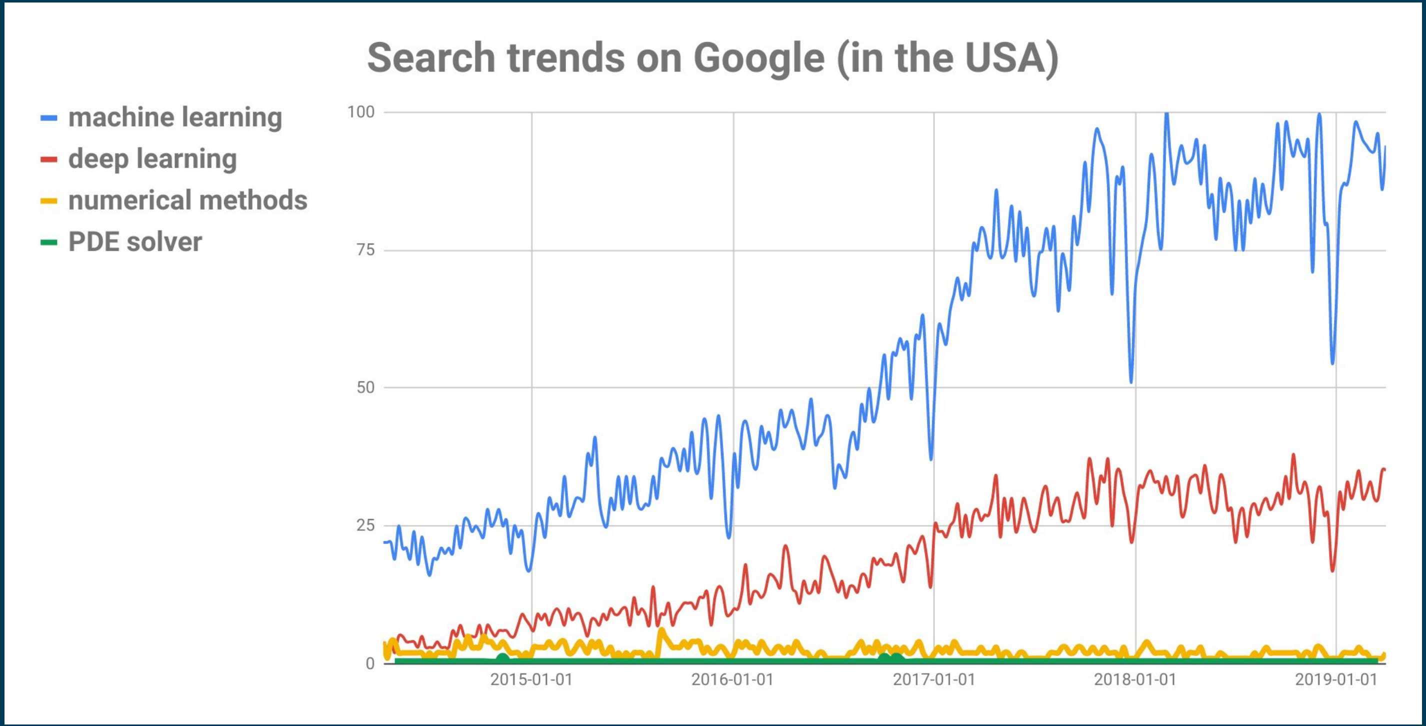
What this talk is about

- Survey / Summary of useful tips, especially for newcomers
- Mostly, but not exclusively, Deep Learning

Motivation of this presentation

- Recent remarkable ML results have piqued the interest of the scientists
- ML is a fast moving field
- Some of the results rely on details less known by the scientific community

Motivations for applying ML



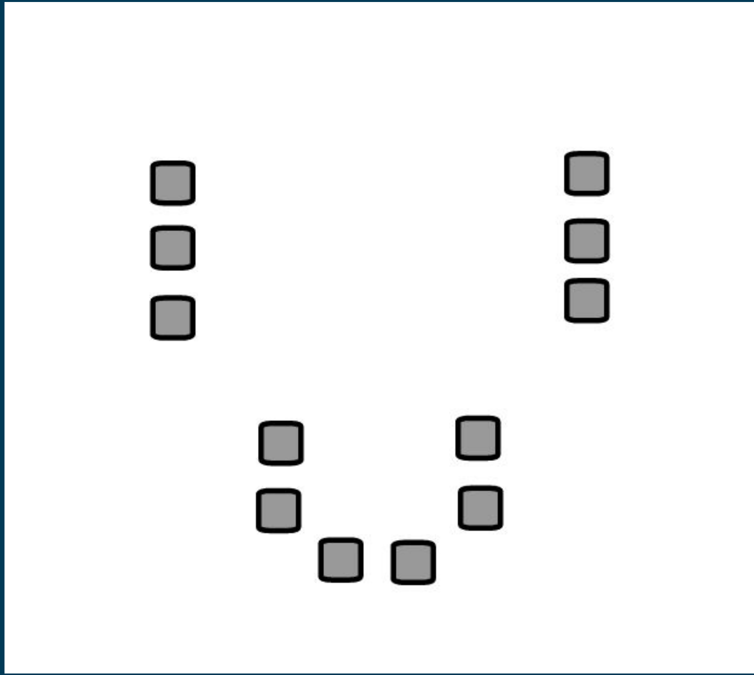
Better motivation, Analysis of goals

- If non-ML solution exists, what is unsatisfactory?
    - Accuracy of scientific results
    - Speed
    - Memory footprint, possibility to run on GPU
    - Accuracy/speed trade-off
- If non-ML solution does not exist, why?
    - “I can tell it, only when I see it” problems
    - Nobody tried, but seems possible
- What is the goal of applying ML to the project
    - Learning/teaching ML
    - Solving new scientific questions
    - Quicker or more agile implementation that a traditional one

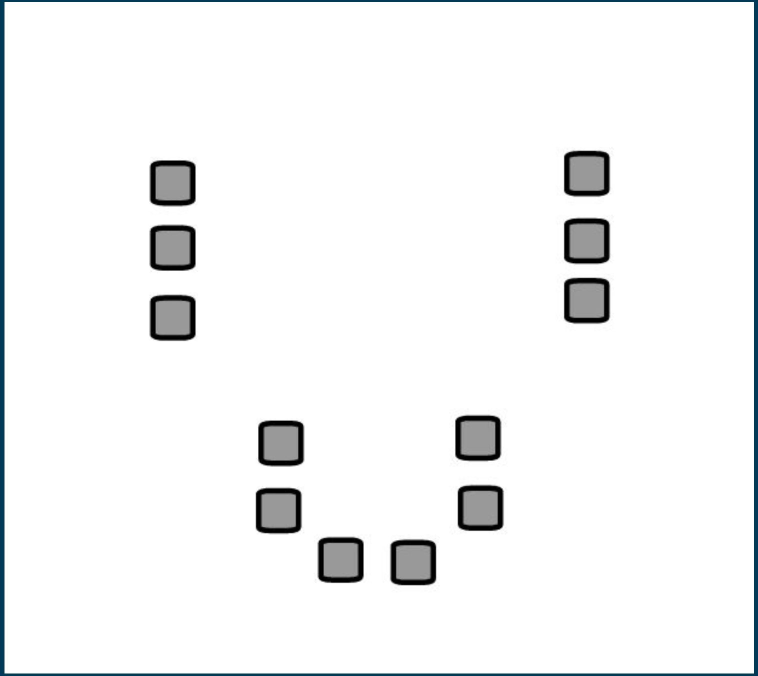
What ML is not

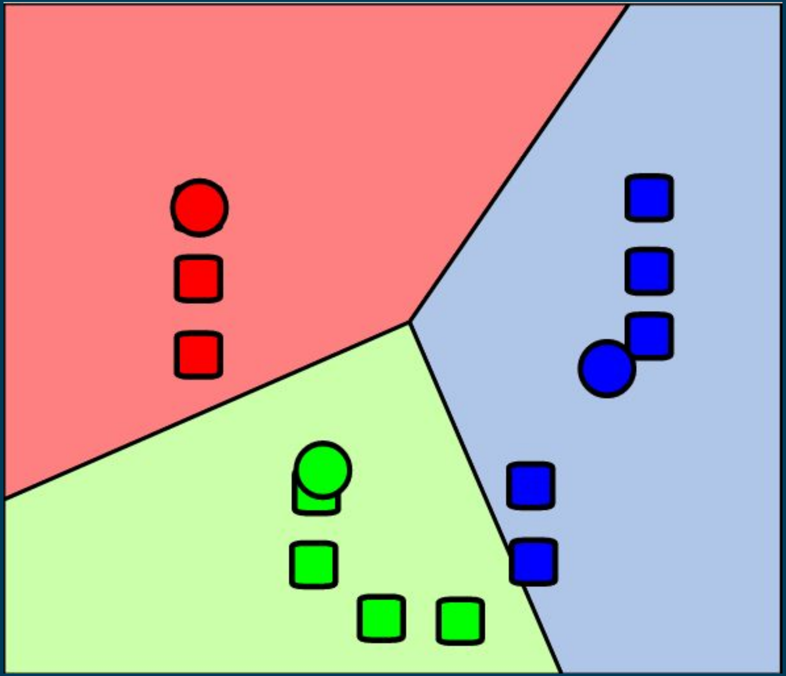
- ML is not a system able to understand semantic
- ML is not a system able to model causation
- ML is not a system with “insight” or “intelligence”
    - K-means example

K-means “insight”



K-means “insight”





What ML is

- An algorithm like many others
  - Everything that can be made with ML, can be made without it
  - However with different effort

- ... for which the practitioner needs to have insight (a theory)
  - Meaningful and most revealing features

- Something similar to curve fit
  - At least ignoring online, unsupervised and reinforcement learning

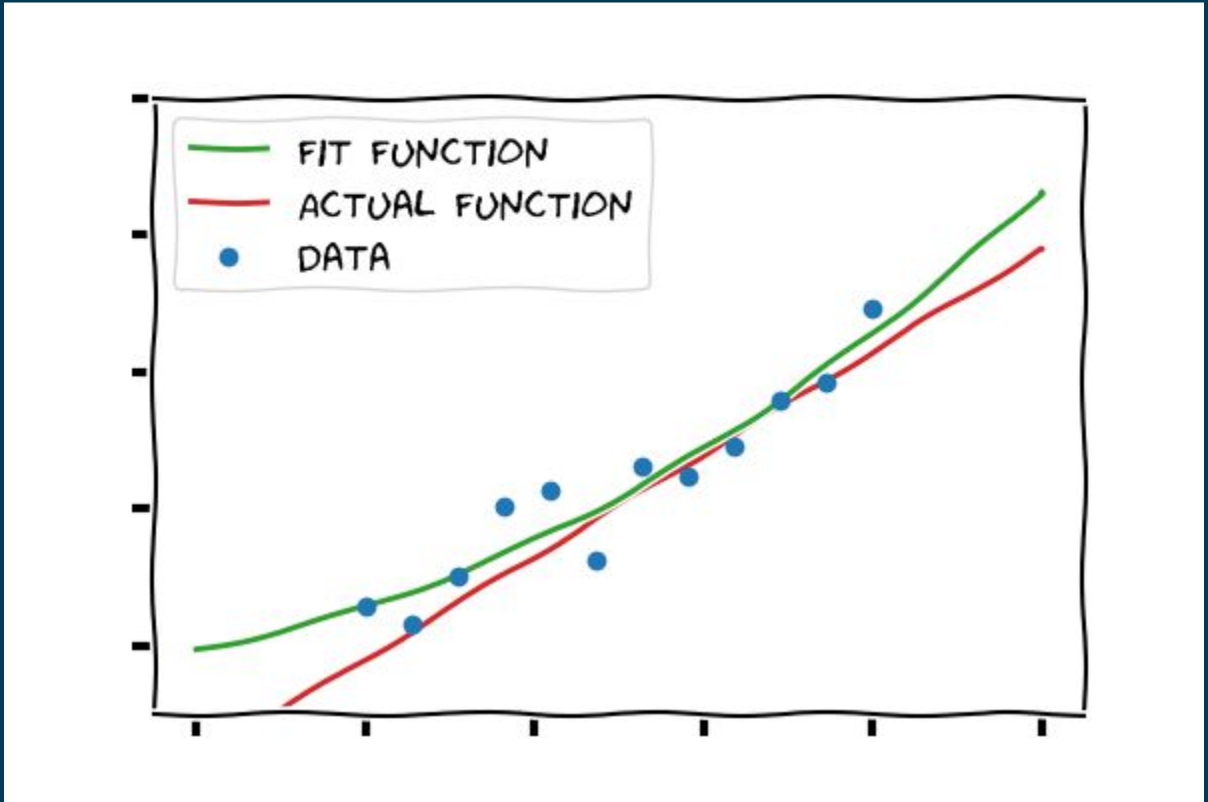
How ML compares to a Curve Fit

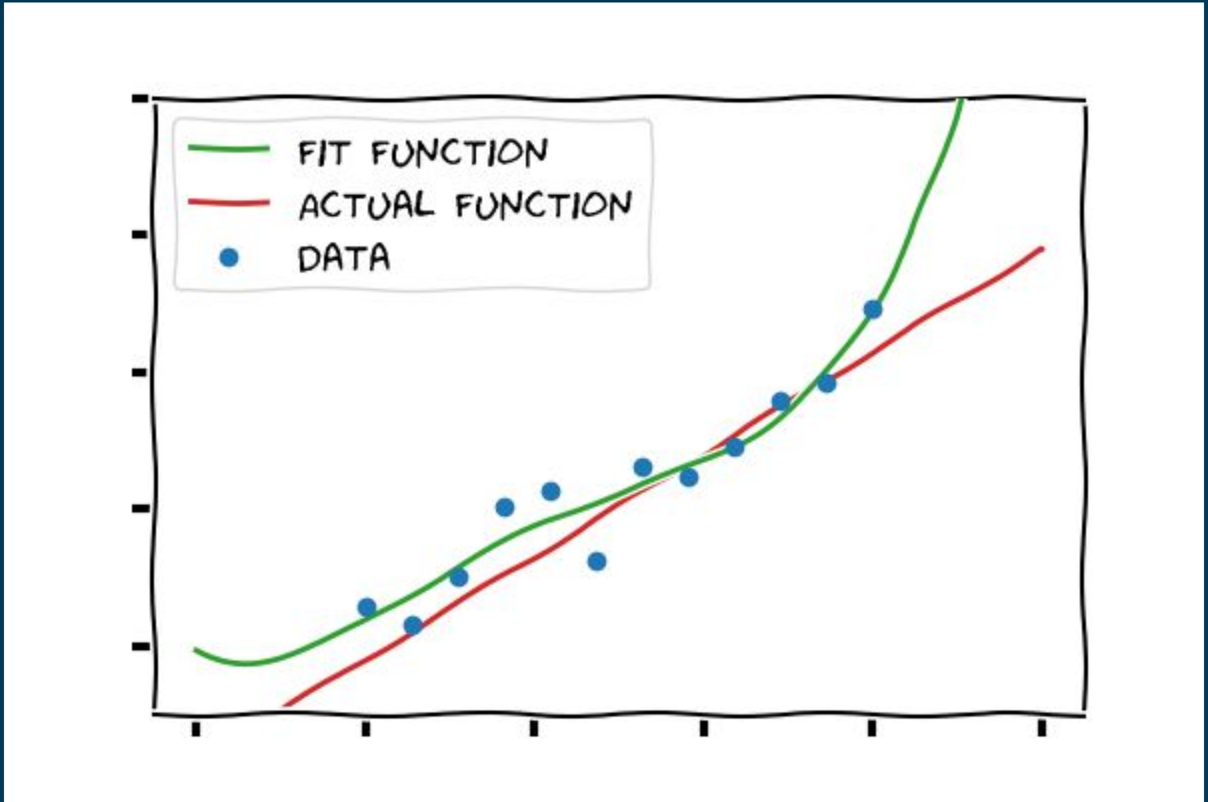
- Both use know data to predict unknown data

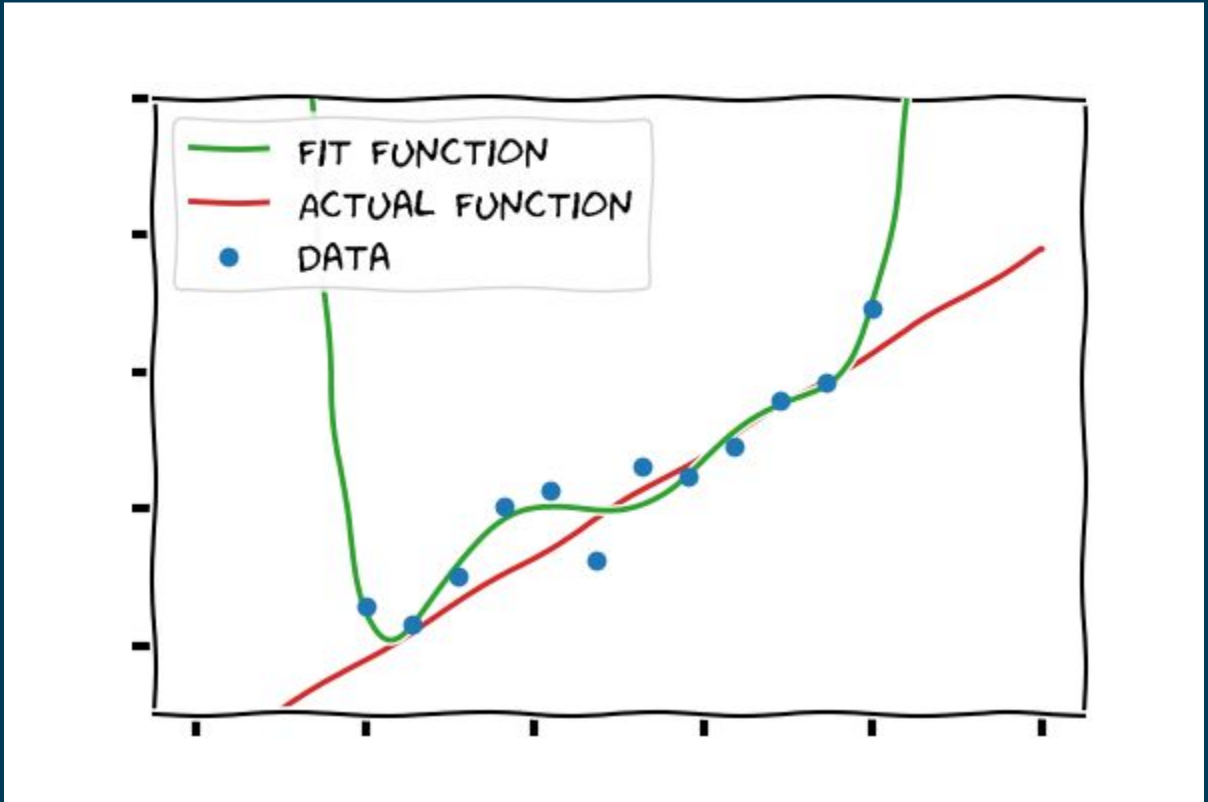
- Curve fit usually on a domain with few continuous variables

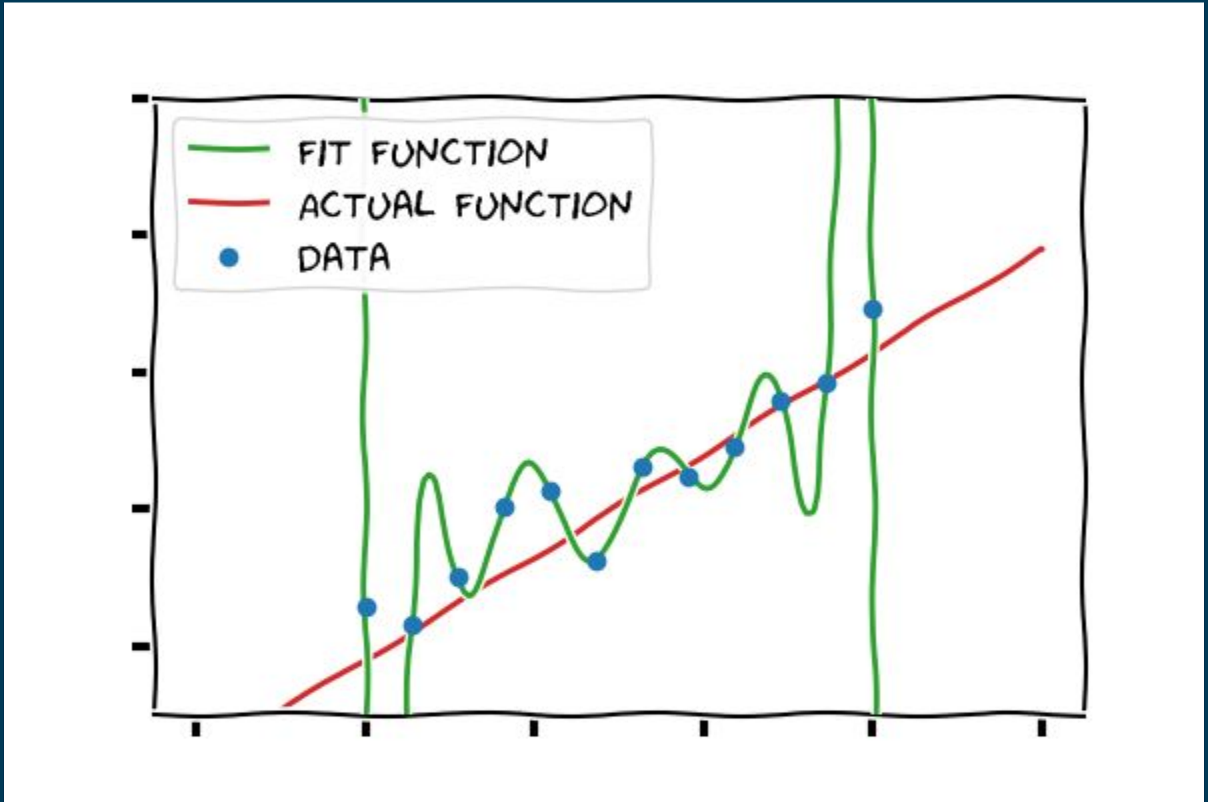
- ML usually on a feature space of very discrete variables

- For both need to make choices
  - Type of curve/architecture
  - Number of degrees of freedom
  - Occam’s razor vs ???


FIT FUNCTION
ACTUAL FUNCTION
DATA

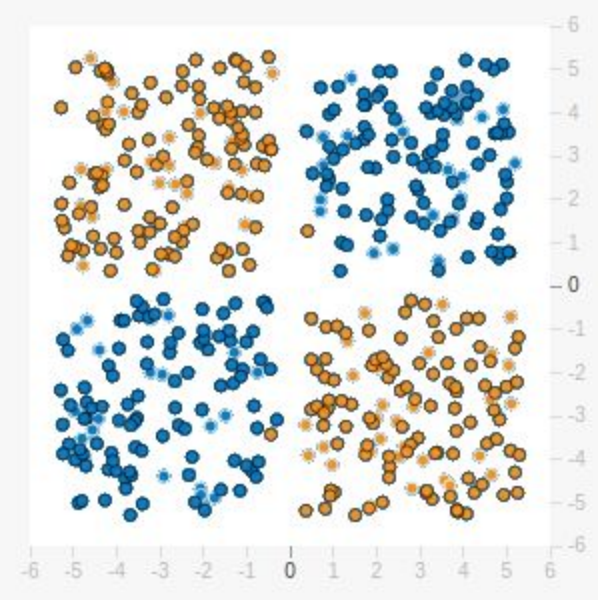

FIT FUNCTION
ACTUAL FUNCTION
DATA


FIT FUNCTION
ACTUAL FUNCTION
DATA


FIT FUNCTION
ACTUAL FUNCTION
DATA

Feature Engineering

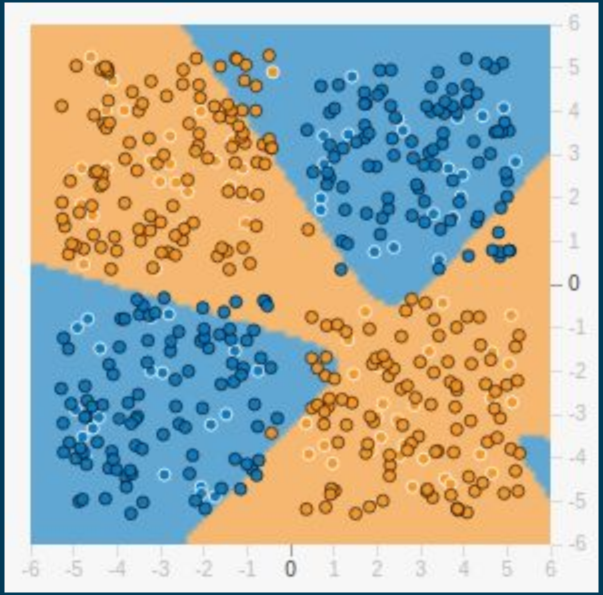
- Consider a classification problem modeling something like the triple point



Feature Engineering - cont

- Classification performance of a small, shallow network

- Fed raw features
- 2 hidden layers
- 5 units

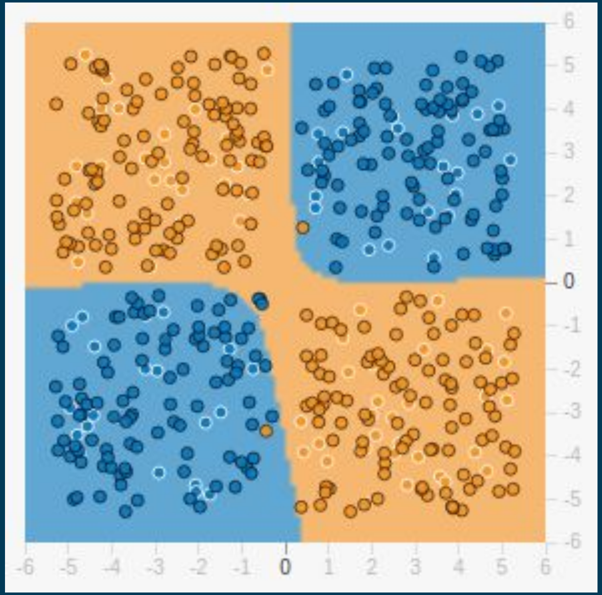


Feature Engineering - cont

- Classification performance of a somewhat large network

- Fed raw features
- 3 hidden layers
- 8 units

- Skittish

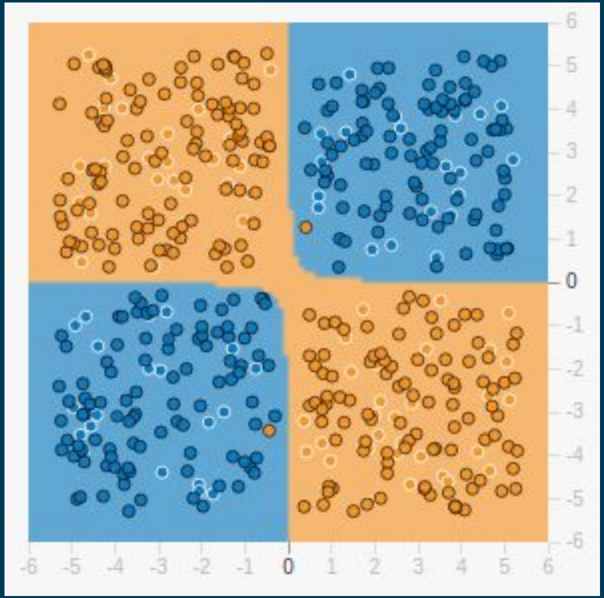


Feature Engineering - cont

- Compare with super-simple network

- Engineered feature
- 1 hidden layer
- 2 units

- Stable, easy to train



Feature Crossing

A **feature cross** is a synthetic feature that encodes nonlinearity in the feature space by combining two or more input features together.

$$x_3 = x_1 * x_2$$

A linear algorithm can learn a weight for x_3 just as it would for x_1 and x_2.

Thanks to x_3, a simple linear model can use nonlinear information.

Feature Engineering

- It was very common for old, small data, projects

- Slightly out of fashion now in big data era

- But it’s extremely important for many scientific applications

- feature scaling
- outliers removal
- features encoding
- feature crossing
- feature selection

Keep your model small and focused

- For one single task, one model might be too complicated
- Complicated models are big and need more data to be accurate
- Smaller/simpler model are easier to train and more portable
- Combine several, smaller models to accomplish a complicated task

ML cycle

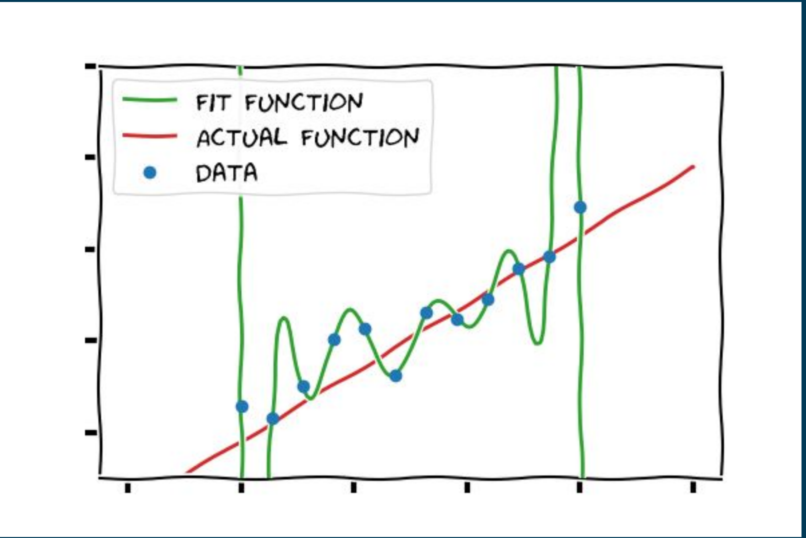
To guide these choices one has to know well…..

Variance & Overfitting

- Allowing the model to vary too much
- Not keeping it smooth enough
- Capturing too much information
  - which usually means random noise in the data

Variance & Overfitting

- Allowing the model to vary too much
- Not keeping it smooth enough
- Capturing too much information
    - which usually means random noise in the data

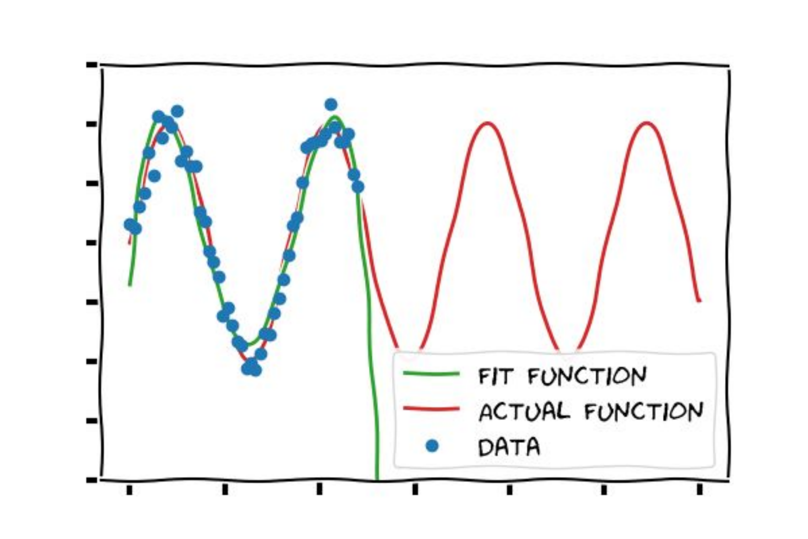


Bias & Underfitting

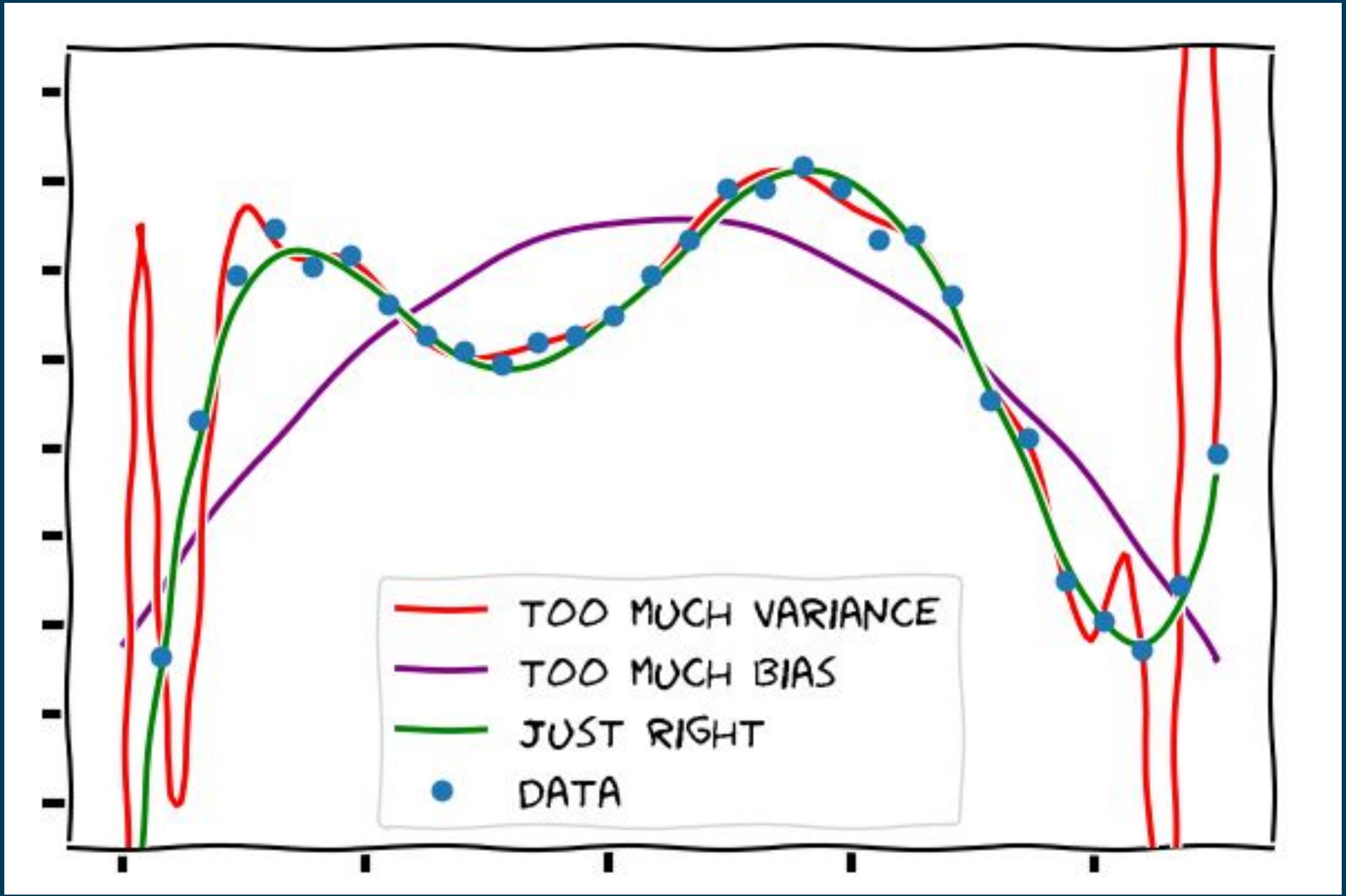
- Selecting a model too restrictive for the phenomenon
- Keeping the model too smooth (not enough ups-and-downs)
- Potentially not capturing enough information

Bias & Underfitting

- Selecting a model too restrictive for the phenomenon
- Keeping the model too smooth (not enough ups-and-downs)
- Potentially not capturing enough information



Bias-Variance trade-off


TOO MUCH VARIANCE
TOO MUCH BIAS
JUST RIGHT
DATA

Bias-Variance trade-off

- Could be a balancing act
- Not easy to achieve for highly dimensional data
- Hard to get satisfactory results in ML

Bias-Variance trade-off

- Could be a balancing act
- Not easy to achieve for highly dimensional data
- Hard to get satisfactory results in ML

- The good news is that for ML it can be defied!

Splitting the dataset into train/test sets

- Traditionally and still too commonly a labelled dataset is split in:
  - training set (usually 70%-80% of the available data)
  - test set (the rest)

- With the purpose of assessing the performance and iterate

Splitting the dataset into train/test sets

- Traditionally and still too commonly a labelled dataset is split in:
  - training set (usually 70%-80% of the available data)
  - test set (the rest)

- With the purpose of assessing the performance and iterate

Splitting the dataset into train/test sets

- Traditionally and still too commonly a labelled dataset is split in:
  - training set (usually 70%-80% of the available data)
  - test set (the rest)

- With the purpose of assessing the performance and iterate

- High risk of tuning the model to the test set (overfit)
- Little guidance on what to change

Alternative (better) splitting strategy

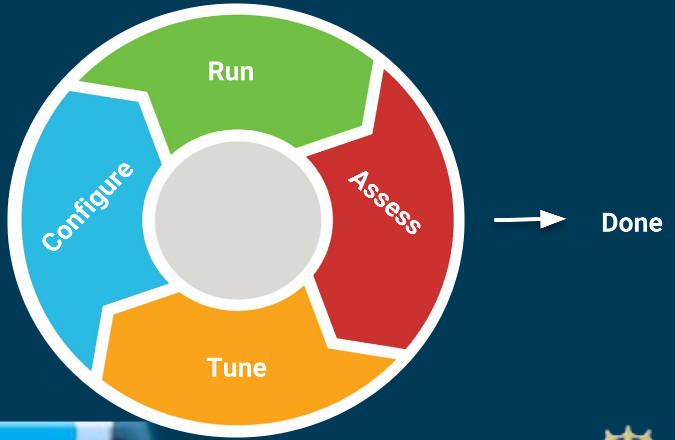
- dev set (thousands of instances, if possible)
- test set (thousands of instances, if possible)
- training set (the rest, still large)

Alternative (better) splitting strategy

- dev set (thousands of instances, if possible)
- test set (thousands of instances, if possible)
- training set (the rest, still large)

- training set and dev set are used to cycle
- test set is used very seldom, only to exit

- And here is the guidance on what to change!

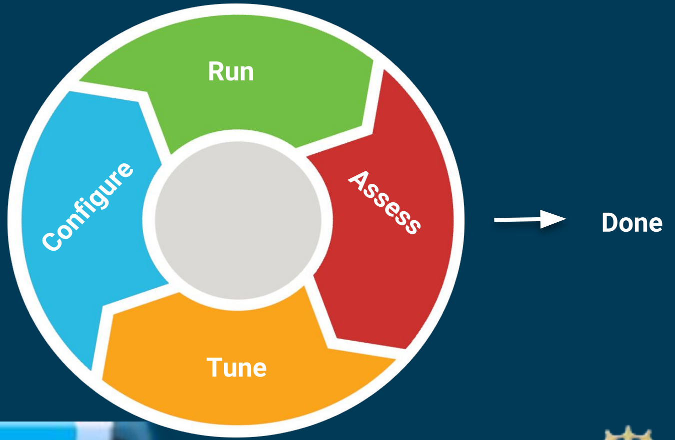


Alternative (better) splitting strategy - cont

- *dev* and *test* sets **<u>MUST</u>** come from the same distribution
- *train* set should too, but it is ok if it does not

Defying the Bias-Variance tradeoff - step 1

- Run the model (train using the training set)

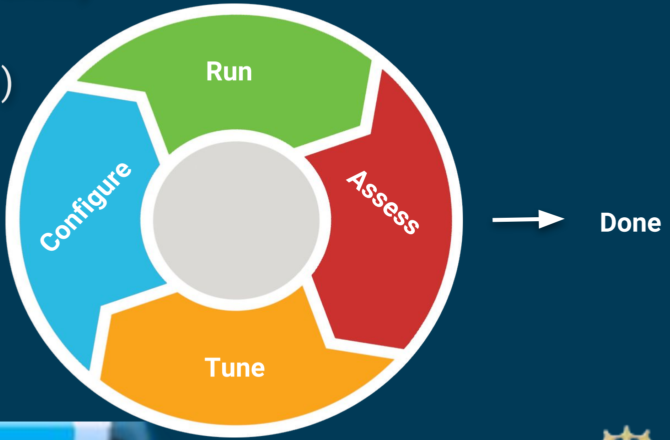


Defying the Bias-Variance tradeoff - step 1

- Run the model (train using the training set)

- Assess the performance (e.g. % of mislabels) on the training set
- If the performance are poor, it's high **<u>bias, i.e. underfit</u>**
  - Try bigger network (i.e. more layers and/or more hidden units)
  - Increase the number of training iterations
  - Use a more advanced optimization algorithm (e.g. Adam)
  - As a last resort, use a different architecture

- Train again and assess again until the performance on the training set are satisfactory



Defying the Bias-Variance tradeoff - step 2

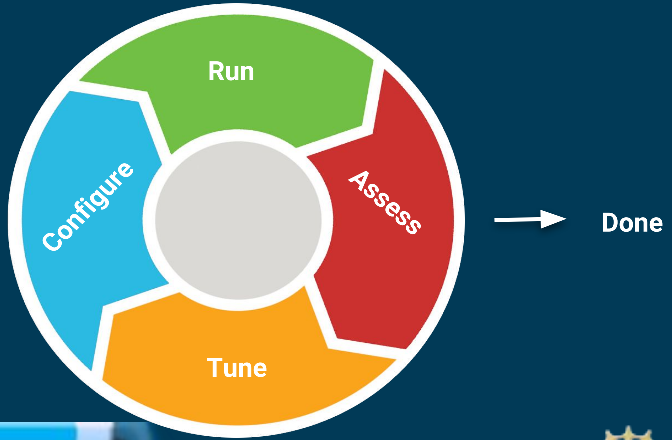
- Once the performance on the training set are good…

Defying the Bias-Variance tradeoff - step 2

- Once the performance on the training set are good…

- Assess the performance (e.g. % of mislabels) on the dev set
- If the performance are poor, it's high **<u>variance, i.e. overfit</u>**
    - Find more labeled data
    - Use a regularization technique such as L_2 or dropout
    - Avoid using Early Stopping
    - As a last resort, try using a different architecture

- Train again and assess again until the performance on the training set are satisfactory



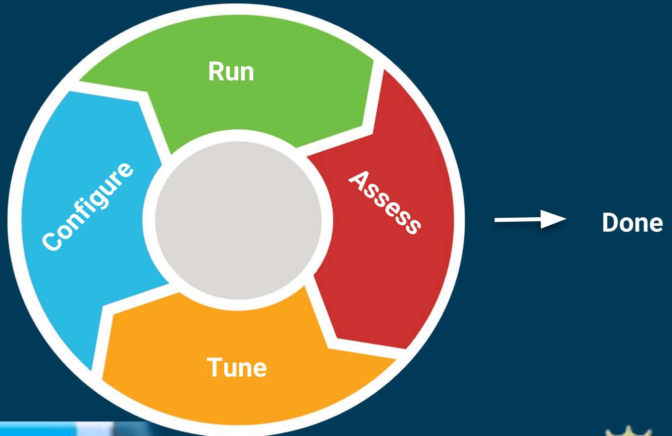
What is “high” for variance and bias?

- Percentage of mislabeled examples, but how many?
- Compare train and dev errors
  - If dev error is higher than train error and unacceptable, then is overfitting
  - If train error is higher than dev error, probably a spurious correlation
  - If the errors are same and unacceptable (e.g. 10% or more),
    - the model might be underfit
    - or what is needed is absent from the data
  - If same and acceptable, move to step 3

Defying the Bias-Variance tradeoff - step 3

- Once the performance on the dev set are good…

- Assess the performance (e.g. % of mislabels) on the test set
- If the performance are good ⟶ Done!



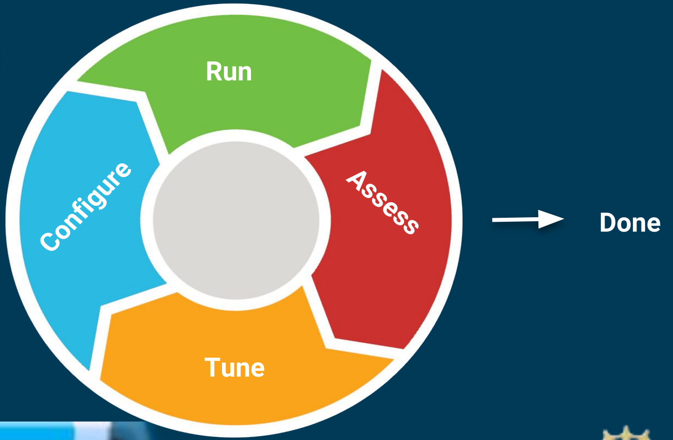
Defying the Bias-Variance tradeoff - step 3

- Once the performance on the dev set are good…

- Assess the performance (e.g. % of mislabels) on the test set
- If the performance are good ⟶ Done!

- If the performance are poor, the model has been fit to the dev set (or is not modeling correctly)
  - Restart from scratch with a larger dev set
  - Restart from scratch with a different cost function

- Try not to use the dev set too many times, to avoid fitting to the dev set ⚠



Possible other issues

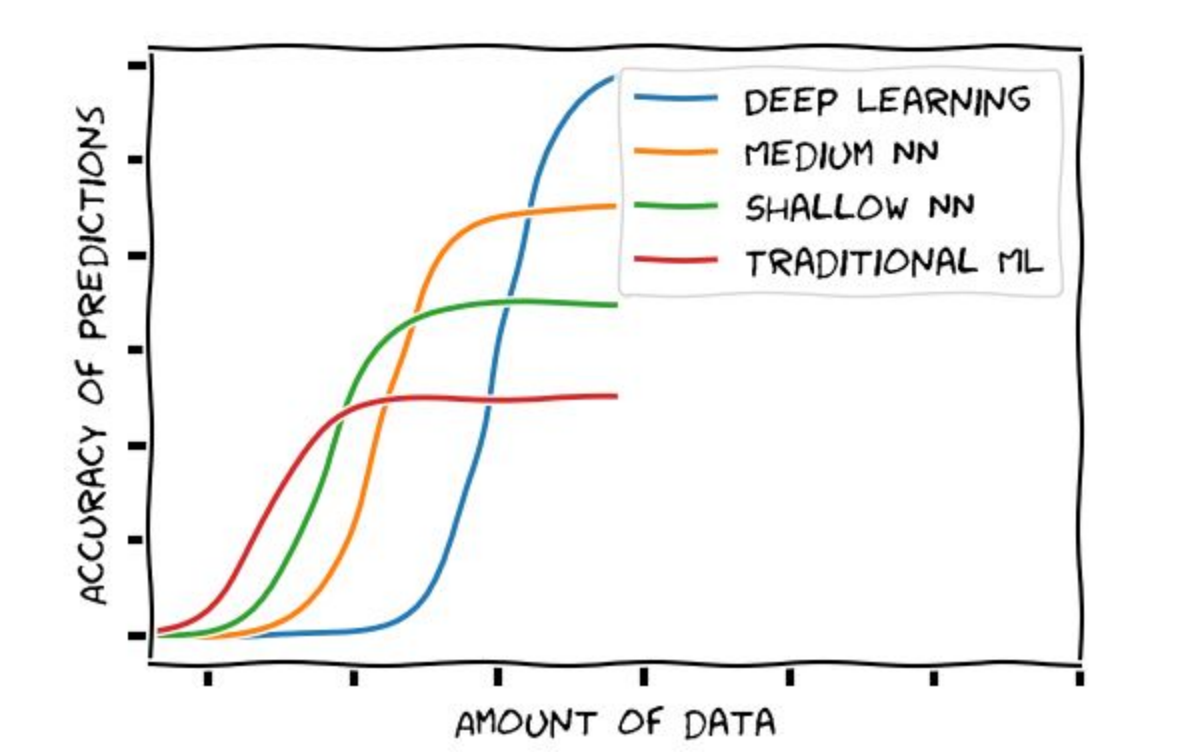
- The performance on the **train** set might never be satisfactory
  - what is needed is absent from the data

- The performance on the **test** set might never be satisfactory
  - dev and test sets do not come from the same distribution

Not enough labeled data is available

- Find more data!
- Do you have known equations?
- Start from a pre-trained model
- Do not use a too large neural network
- Clean your data, study feature engineering
- Data augmentation
- Do not use a neural network (use SVM, maybe semisupervised techniques)

Not enough labeled data is available



Trying over and over until it works

- ML is iterative, so you will try over and over

- However, you may experience spurious correlations

- Make sure you do not fit to the test set
- Make sure you do not “overfit the hyperparameters”

Questions?