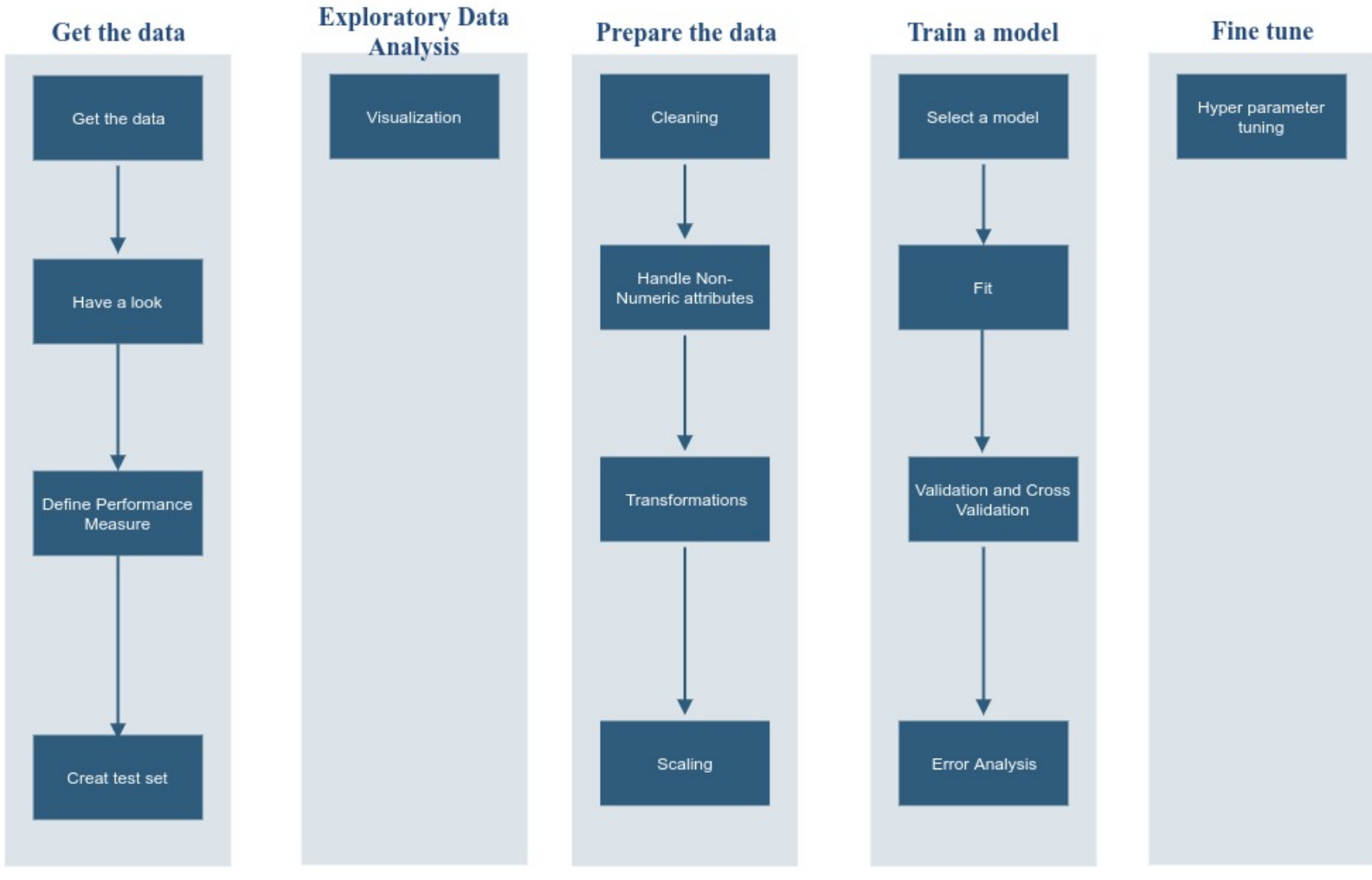
What is the ”Recipe” for Machine Learning

We will define a methodical approach to solve problems using Machine Learning.

This is the *Recipe for Machine Learning*


Get the data
Get the data
Have a look
Define Performance Measure
Creat test set
Exploratory Data Analysis
Visualization
Prepare the data
Cleaning
Handle Non-Numeric attributes
Transformations
Scaling
Train a model
Select a model
Fit
Validation and Cross Validation
Error Analysis
Fine tune
Hyper parameter tuning

There are no short-cuts !

Each step in the Recipe both prepares you for the next and, crucially, gives you *deeper insight* which improves the result.

Plan

- Will illustrate the recipe using a model familiar to you: Linear Regression
- **This will be a sprint**
    - illustrate the steps in the Recipe
    - will be followed by a deeper dive into the steps and the mathematics

Since this is our first lecture involving code

- I will show a lot more code within this notebook than in subsequent notebooks
- In the future, most of this code will be moved to separate modules
    - re-usable as a module, rather than copy/paste between notebooks
    - notebooks are less cluttered; maintain focus on the problem, not the code
- I will digress from the problem in order to solidify your understanding of the tools:
    - Jupyter, Pandas, etc.

The particular task with which we illustrate the Recipe is Regression, a form of Supervised Learning.

- Using numerical features and target
- We will introduce categorical variables (features/targets) in a following lecture
- Focus is on the *concept*, not the code
- Let's jump in ! Start *doing* ML

Disclaimer:

The purpose of this lecture is not to make you an expert in sklearn.

It is to introduce you to concepts that you can apply no matter what toolkit you use in the future.

Get the data

The first step is obtaining data for training and evaluation.

This is often the most challenging part !

- Interesting data is scattered: requires collection
- Supervised Learning requires labelled data; where do the labels come from ?

In this course, we will usually provide you with data so you will be mostly insulated from this challenge.

- Learning how to obtain data is a good skill to learn
    - Web scraping

Let's visit the notebook section Get the Data (Recipe_for_ML.ipynb#Recipe-step-A:-Get-the-data)

Look at the data

Always put your eyes on the data !

- You will learn about its "shape":
    - tabular ? What are the attribute names ?
    - What are the types of the attributes ? Numeric ? Text ?

- You will learn about potential data problems
    - missing data
    - strange values

Don't even try to do anything with your data until you have at least the most basic understanding by performing an inspection.

Let's visit the notebook and Look at the data (Recipe_for_ML.ipynb#Recipe-A.2:-Have-a-look-at-the-data)

Define a Performance Measure

Our model "learns" from training data, so we might expect it to predict well on training examples

- the training examples are *in sample*: used by the model to learn Θ

How well should I expect the model to predict on examples not encountered during training ?

- "test" examples never seen during training, called *out of sample* examples

We define a *Performance Measure* to measure how well the model performs out of sample.

Let's visit the notebook and Define a Performance Measure (Recipe_for_ML.ipynb#select_performance_measure)

Performance Measure versus Loss Function

There may some confusion between the Performance Measure and Loss functions

- they are both evaluated over a set of examples
- they both measure performance of some sort

A Performance Measure can be thought of as the promise you make

- to a client/customer/boss
- on how well your model will perform on arbitrary, yet to be seen examples (non-training, out-of-sample)

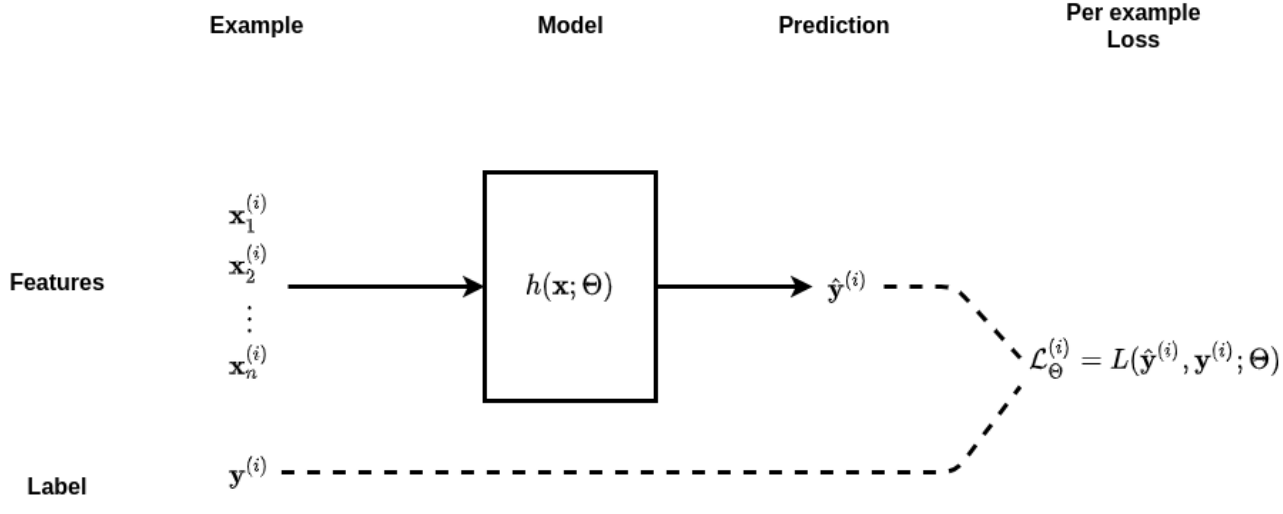
In order for you to have confidence in your promise

- you evalute the Performance Measure on *out of sample* examples
- using the out of sample examples *once* so that your model doesn't learn from them (i.e., become in-sample)

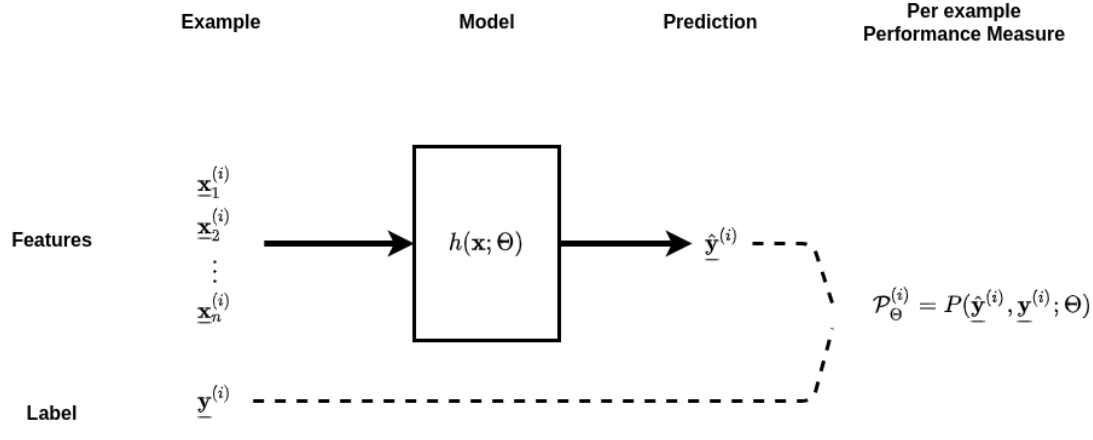
To illustrate, let

- $\mathbf{X}$ denote our set of training examples, $\mathbf{x^{(i)}} \in \mathbf{X}$
- $\underline{\mathbf{X}}$ denote a set of test examples (out of sample: not used in training), $\underline{\mathbf{x}}^{\mathbf{(i)}} \in \underline{\mathbf{X}}$

Loss on Training example



Performance on **Test example**



- A Performance Measure
    - is a property of the *problem* (not the model used to solve the problem)
    - you may have more than one Performance Measure
        - each expressing some desired quality of the prediction
    - is evaluated *out of sample*, that is, on non-training examples

- The Loss Function
    - is a property of a *model*: it guides a particular model's search for the best Θ
        - different models may have different Loss Functions
            - but the *problem's* Performance Metric is the same
    - is evaluated *in sample*, that is, on training data

Create a test set

Let's visit the notebook and Create a test set (Recipe_for_ML.ipynb#Recipe-A.4:-Create-a-test-set-and-put-it-aside-!)

Exploratory Data Analysis

This is one of the key steps of a good Data Scientist.

Besides "seeing" the data, we need to hear it: what is it telling us that may aid prediction

- Any problems with the data that would inhibit learning ?
- Any obviously useful predictive features ?
    - relationship between target and a single feature ?
    - relationship between target and combinations of features ?

Any more complex relationships that may be usefuls ?

- Relationship *between* features ?
- Relative magnitudes of features ?

Often, understanding the data intimately can lead to

- transformations of the features that will aid prediction
- improved models

Let's visit the notebook and perform Exploratory Data Analysis (Recipe_for_ML.ipynb#Recipe-Step-B:-Exploratory-Data-Analysis-(EDA))

Prepare the data

It is not always the case that the data in "raw" form is adequate for modelling

- Cleanliness
    - dealing with missing data or anamolous values
- Numericalization
    - Converting non-numeric/categorical data into appropriate numbers
- Scaling, normalization
    - putting features on compatible scales

A key part of the Prepare the Data step is *feature engineering* or *transformations*

- Creating new, synthetic features from the "raw" features
- Knowing when/how to do this is what separates a good Data Scientist from an average one

We will devote parts of several subsequent modules to the important topic of Transformations.

- Our initial pass over this subject will be minimal
- So that we have enough time to explore the other steps in the Recipe

Let's visit the notebook Prepare the data (Recipe_for_ML.ipynb#Recipe-Step-C:-Prepare-the-data)

Train a model

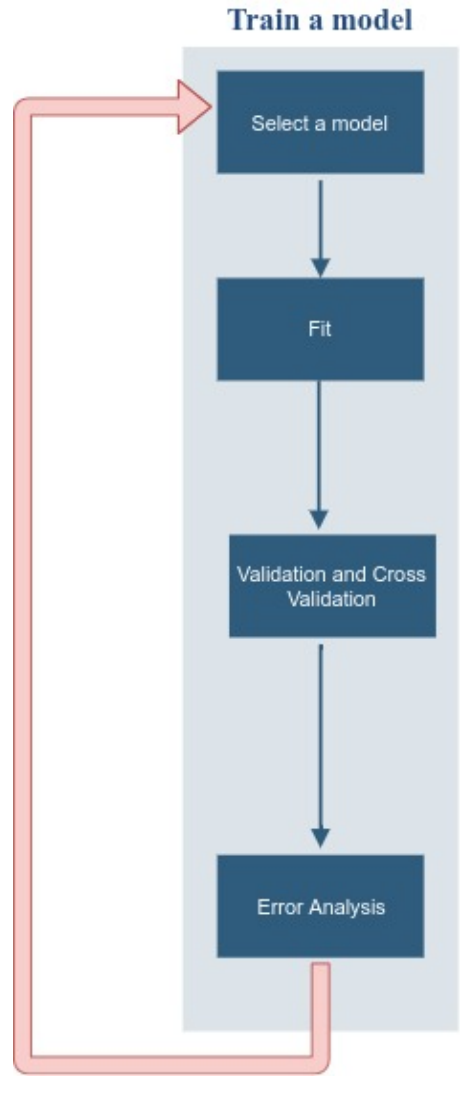
The model is our "predictor": the machine that takes features and produces predictions.

All the prior steps of the recipe were "prep-work": preparing the ingredients (data) for cooking (modelling)

Unlike actual cooking, this step is *iterative*

- Try a simple model, fit it to the data and evaluate the results
- Perform Error Analysis: Examine the results critically
    - Have our changes improved the Loss ? Is the Loss value acceptable ?
    - Is there some commonality among the examples with "bad" predictions ?
        - Can we change the model or perform Feature Engineering to compensate for common errors ?
        - Perform Feature Engineering, modfiy the model.
- Repeat !

Iterative training



The iterative nature is often overlooked in the rush to learn as many models as possible.

The lessons learned from the Error Analysis is how we systematically progress from

- Simple but poor models
- To better models of increasing complexity

Select a model and train it

Let's move to the notebook and Select and Train a model (Recipe_for_ML.ipynb#Recipe-Step-D:-Train-a-model)

Validation and Cross Validation

We have just fit our first model and evaluated the Performance Measure on the test examples.

Can we continue trying to improve the model and re-evaluate the Performance metric on the same test examples ?

No ! By seeing the "out of sample" examples, we have made them "in-sample"

- We can improve our model by causing it to perform well on the test examples
- Result won't be a realistic measure of performance on unseen examples
    - Like seeing the questions before the exam !

Fortunately, there is a way to both

- save your test examples for single use
- create pseudo-test examples that can be reused

Let's return to the notebook and explore Validation and Cross Validation (Recipe_for_ML.ipynb#Recipe-D.3:--Validation-and-Cross-Validation)

Error analysis

Now that we have fit a model, we have

- an estimate of Performance Measure using Validation/Cross Validation

If this measures is not "good enough", we will want to improve predictions

- we might improve prediction by *changing* to a different model
- we might improve prediction by *adding features*

How do we know if the Performance Measure is "good enough" and how to improve our model ?

Unfortunately, many people (and courses!) don't explore this enough.

Let's move to the notebook to Examine the errors (Recipe_for_ML.ipynb#Recipe-D.4:--Error-analysis) to see why a deeper analysis may be warranted.

Iterate: Linear Regression with higher order features

The Error Analysis we performed on our first model (single non-constant feature) suggested a need for improvement.

Two types of improvement come to mind

- Hypothesis iteration: try a different model
- Feature iteration: change/add to the features of the current model
    - adding a previously discarded feature
    - creating a synthetic feature

We will take the approach of adding a feature.

Let's extend Linear Regression with higher order features (Linear_Regression_HigherOrderFeatures.ipynb) (separate notebook).

When to stop iterating

Adding second order features resulted in a perfect in-sample fit, so there is no point iterating further.

In general, this will not be the case.

How do we know that our model is "good enough" ?

We encourage you to do a Deeper Dive by examining the topic of Bias and Variance (Bias_and_Variance.ipynb)

Fine tune

There are often "tweaks" that can be applied to a near-final model in order to squeeze out increase performance.

For example: many models have *hyper parameters*.

These are values that are *chosen* at model construction, rather than *discovered* by fitting during training (Θ)

- the degree d of the polynomial when constructing higher order features $\mathbf{x}^d$
- whether to include/exclude the interecept Θ_0 in a Linear Regression
- strength of the regularization penalty (coming attraction: to be discussed together with the Loss function)
- the k in K Nearest Neigbhors

Perhaps a different choice of a hyper-parameter would improve the model ?

We can try many choices before settling on the one giving the best Performance Metric.

Hyper parameters search is another reason for using Cross Validation

- we can't use the Test set more than once
- with a single Validation set: we might overfit to the validation set
    - that is, choose a value for the hyper parameter that is best for this *single* validation set

We encourage you to do a Deeper Dive by examining the topic of Fine Tuning (Fine_tuning.ipynb).

Recap

- We have briefly detailed the multi-step process for Machine Learning
- This should be a model for you (and your assignments !)
- We will explore some of the steps in greater depth in future modules.

In [2]:
```
print("Done !")
```

```
Done !
```