Big data science Day 3

F. Legger - INFN Torino

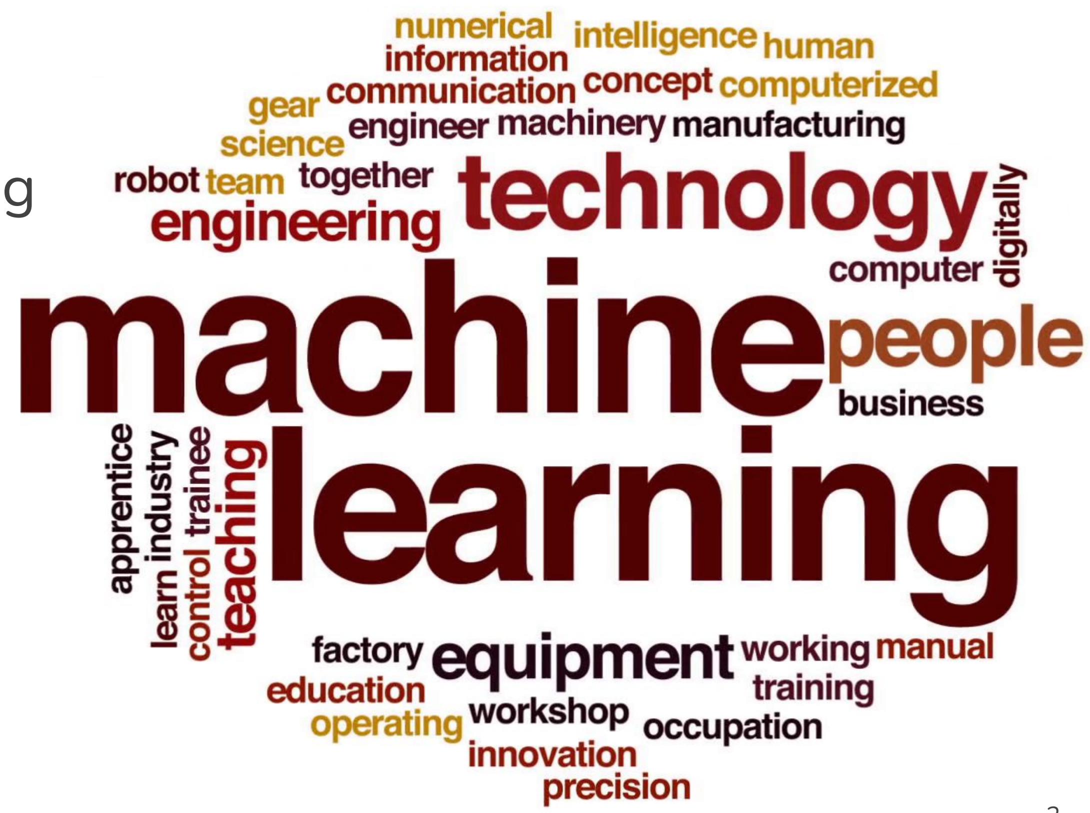
https://github.com/Course-bigDataAndML/MLCourse-2324

Yesterday

- Big data, analytics
- Distributed computing
- ML: Feature engineering

Today

- **Machine learning**
    - Architectures
    - Train model and evaluate

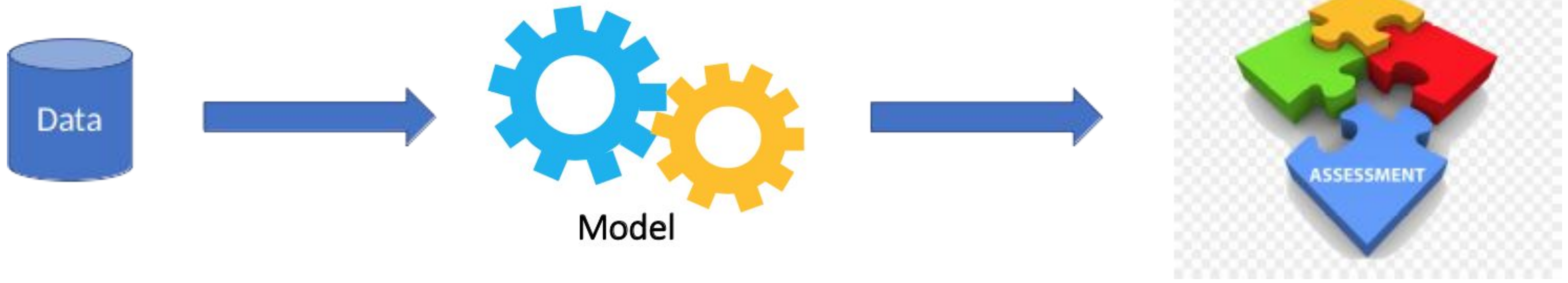


Remember:

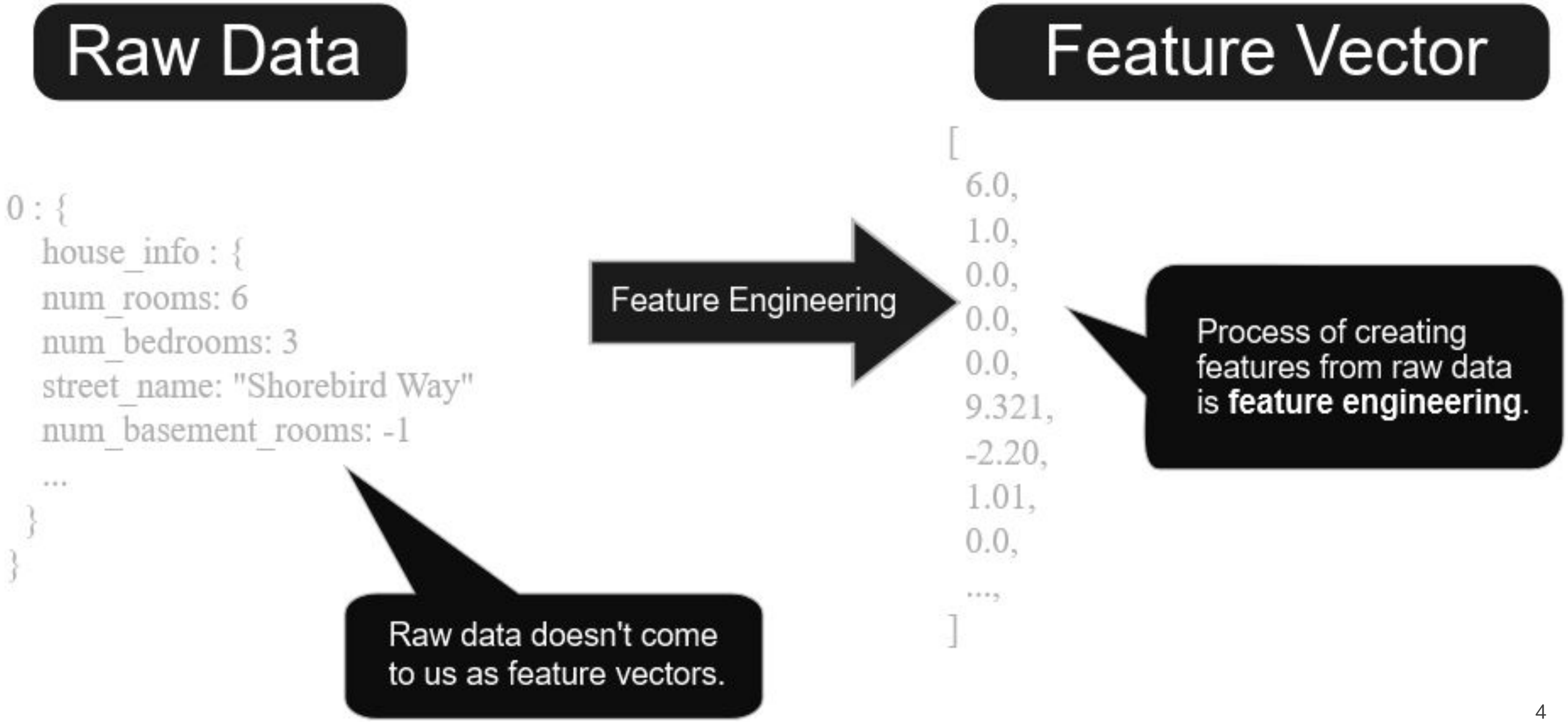
Machine learning involves **two mathematical entities**

Model: a mathematical model describes the relationship between different aspects of the data

Features: a feature is a representation of raw data



Feature engineering


Raw Data
Feature Vector
Feature Engineering
Process of creating features from raw data is **feature engineering**.
Raw data doesn't come to us as feature vectors.

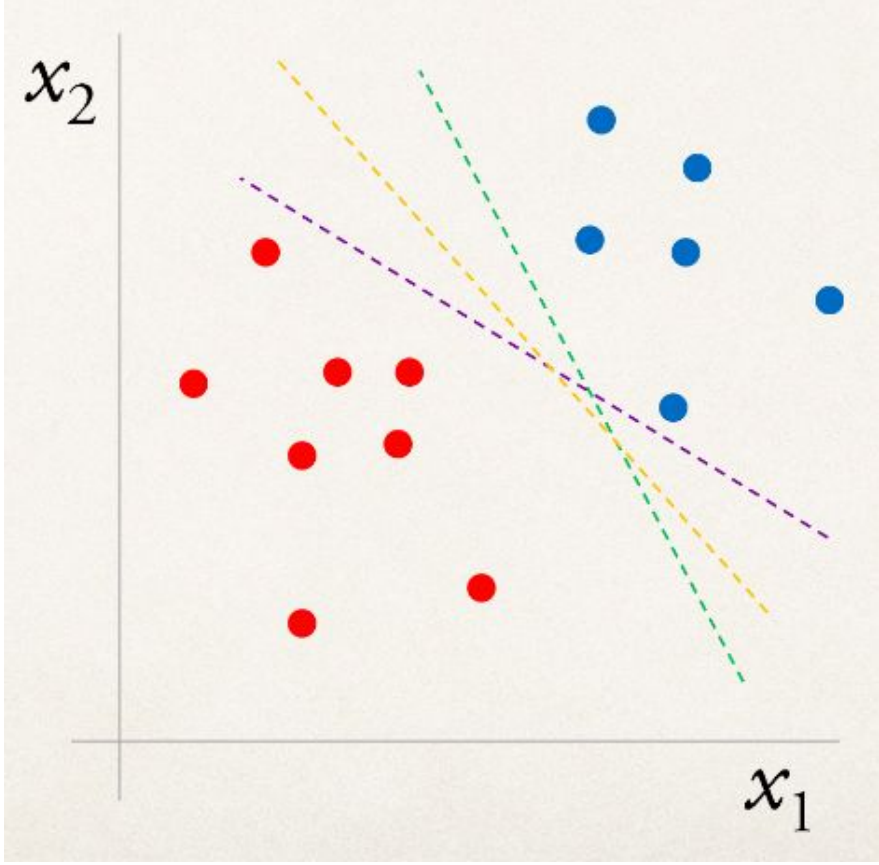
```
0 : {
  house_info : {
  num_rooms: 6
  num_bedrooms: 3
  street_name: "Shorebird Way"
  num_basement_rooms: -1
  ...
 }
}
```

```
[
  6.0,
  1.0,
  0.0,
  0.0,
  0.0,
  9.321,
  -2.20,
  1.01,
  0.0,
  ...,
]
```

Example: supervised classification

Ingredients

- **Inputs: X**, is a matrix of size **n** (number of samples) x **m** (number of features)
- **Features: X**, transformed inputs, matrix **nxm**
- **Labels: y,** vector size **n**



Recipe:supervised classification

- For each input vector $\mathbf{X}_i$ predict $\mathbf{z}_i$, **i=1...n**

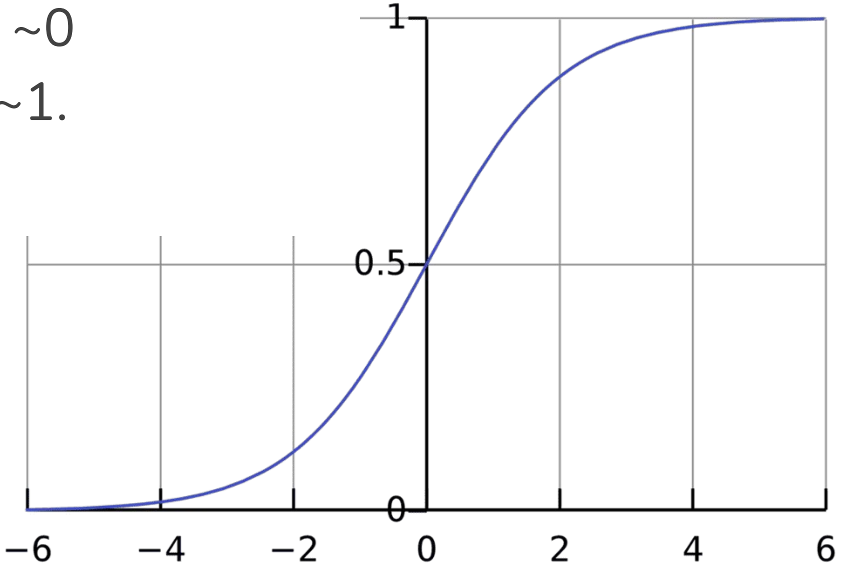
- **Predictions:**
  - $\mathbf{z}_i = \boldsymbol{\varphi}(\mathbf{W}^T X_i)$ yields (0,1)
  - **Weights: W** (matrix) contains the model parameters
  - **Activation function: φ** (step function, sigmoid)

- **Cost function == loss function == prediction error**, function of the model parameters **W**
- Aim: find weights W that <u>minimize cost function</u>

Activation function

- Turns unbounded output into a known range/shape
- For example, **sigmoid** function only outputs numbers in the range (0, 1)
  - big negative numbers become ~0
  - big positive numbers become ~1.

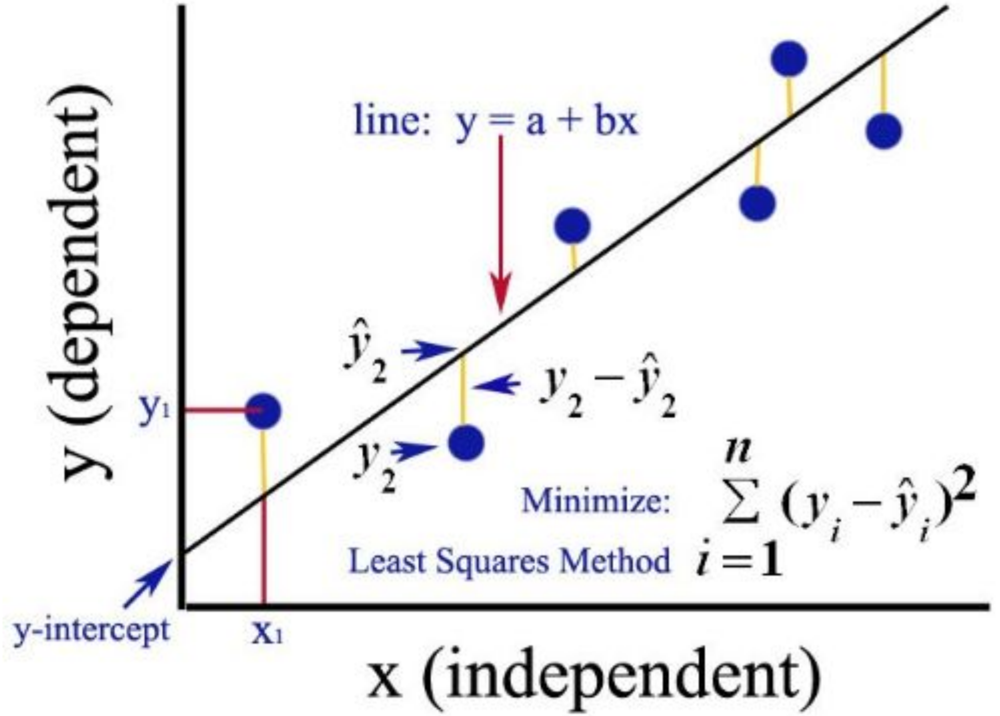
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$



Another example, linear regression

- Inputs (features): $\mathbf{x_i}$
- Labels: $\mathbf{y_i}$

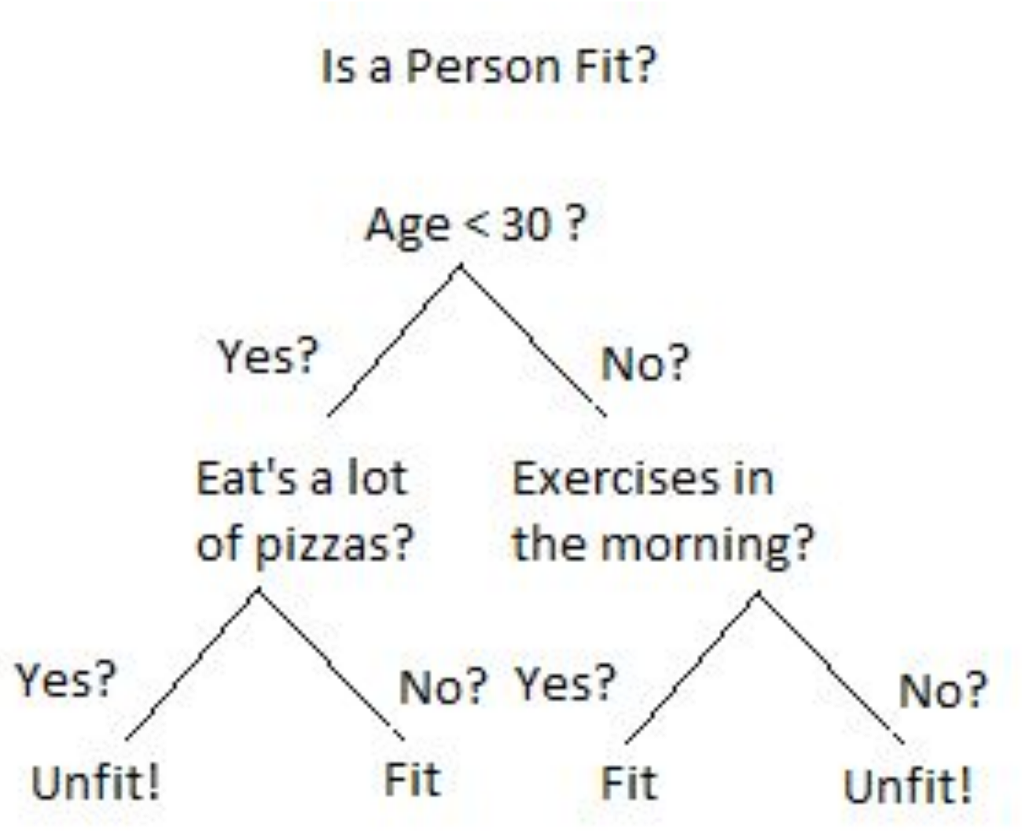
- Model: **y = a + bx**
- Weight+bias (parameters to be found): **a, b**
- Cost function: **Mean Square Error (MSE)**
- No **activation function:** problem is linear



$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

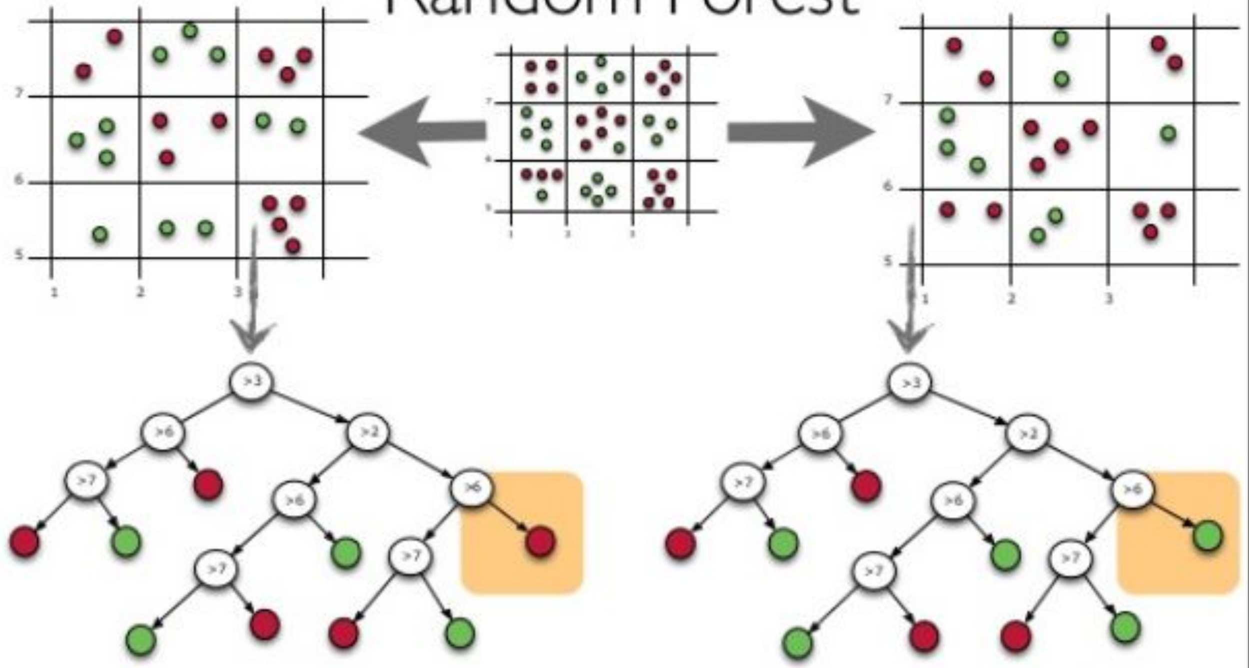
Supervised learning

Decision trees: supervised classification



Typically used in combinations (Random forest, Gradient Tree Boosting)

Random Forest



- Each tree sees part of the training sets and captures part of the information it contains

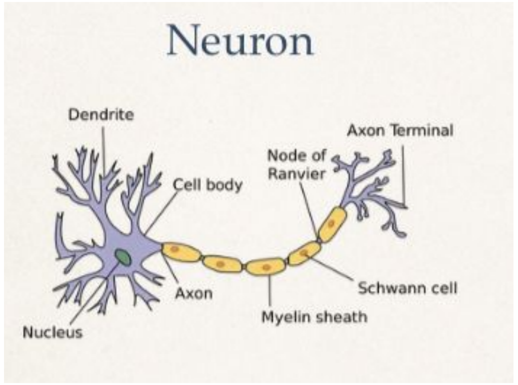
Ensembles

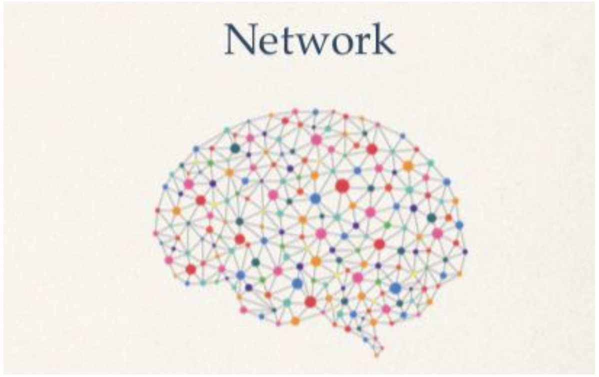
- **Bagging**
  - building multiple models (typically of the *same* type) from different subsamples of the training dataset
- **Boosting**
  - building multiple models (typically of the *same* type) each of which learns to fix the predictions errors of a prior model in the chain
- **Stacking**
  - building multiple models (typically of *different* types) and a supervisor model that learns how to best combine the predictions of the primary model
- **Weighting|Blending**
  - combine multiple models into single prediction using different weight functions

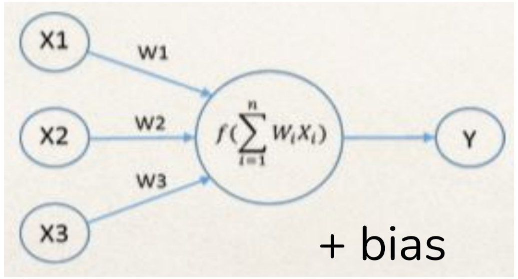
Neural networks: supervised classification

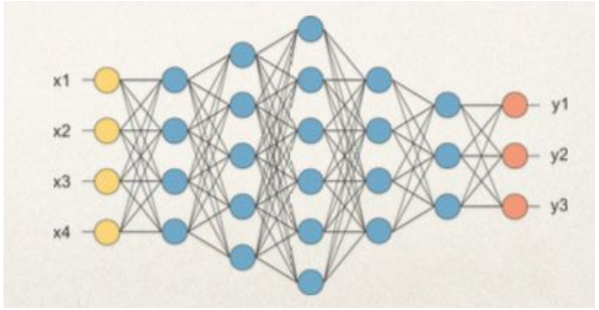
- Basic unit: **Neuron.** A neuron takes inputs, does some math with them, and produces **one** output

More on NN tomorrow!

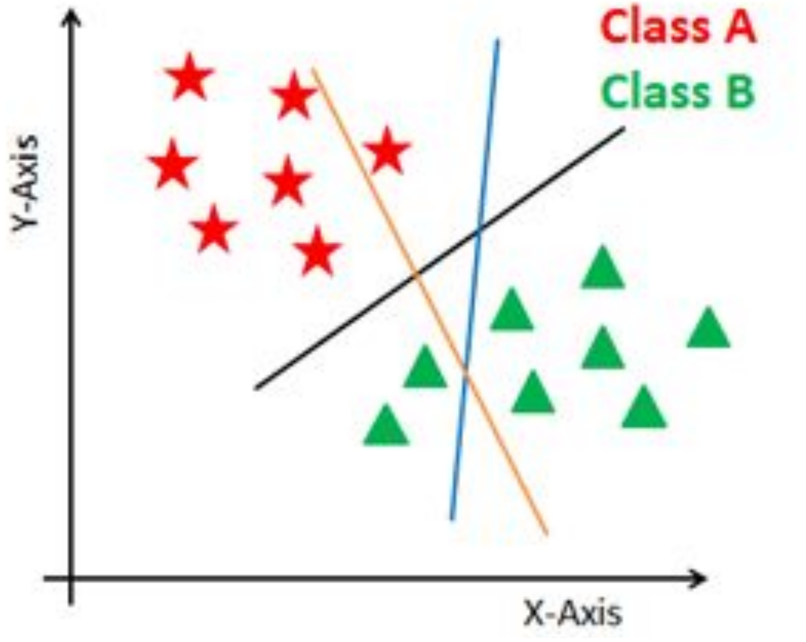


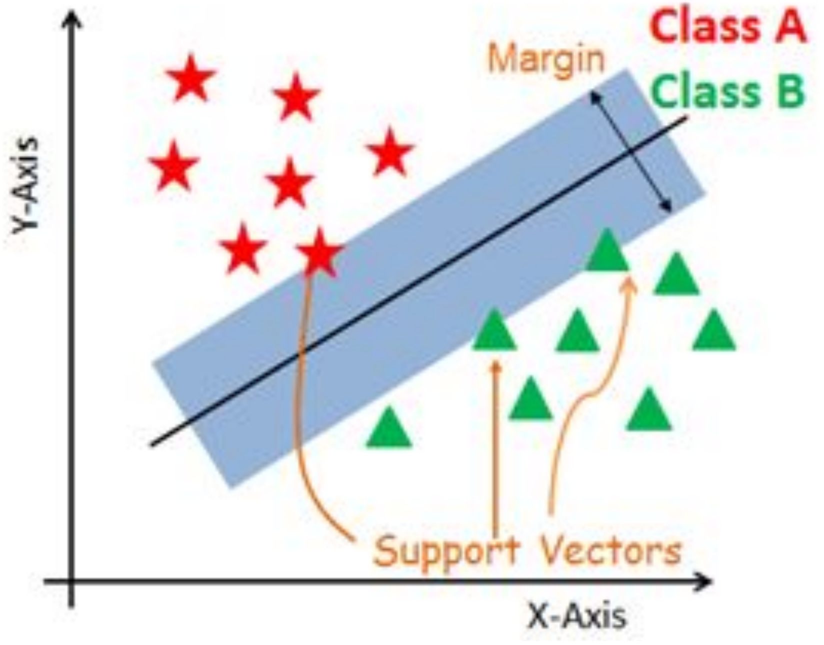






Support vector machines (SVG), supervised classification

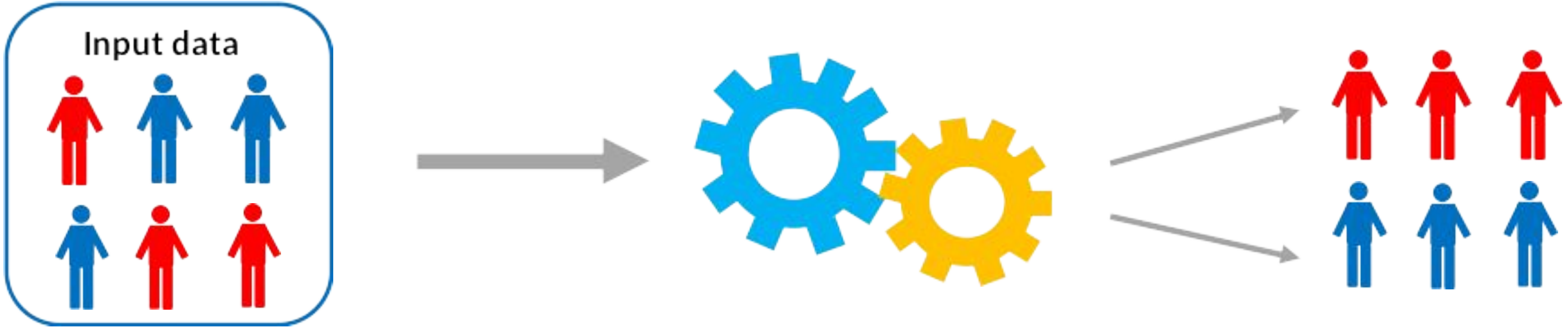




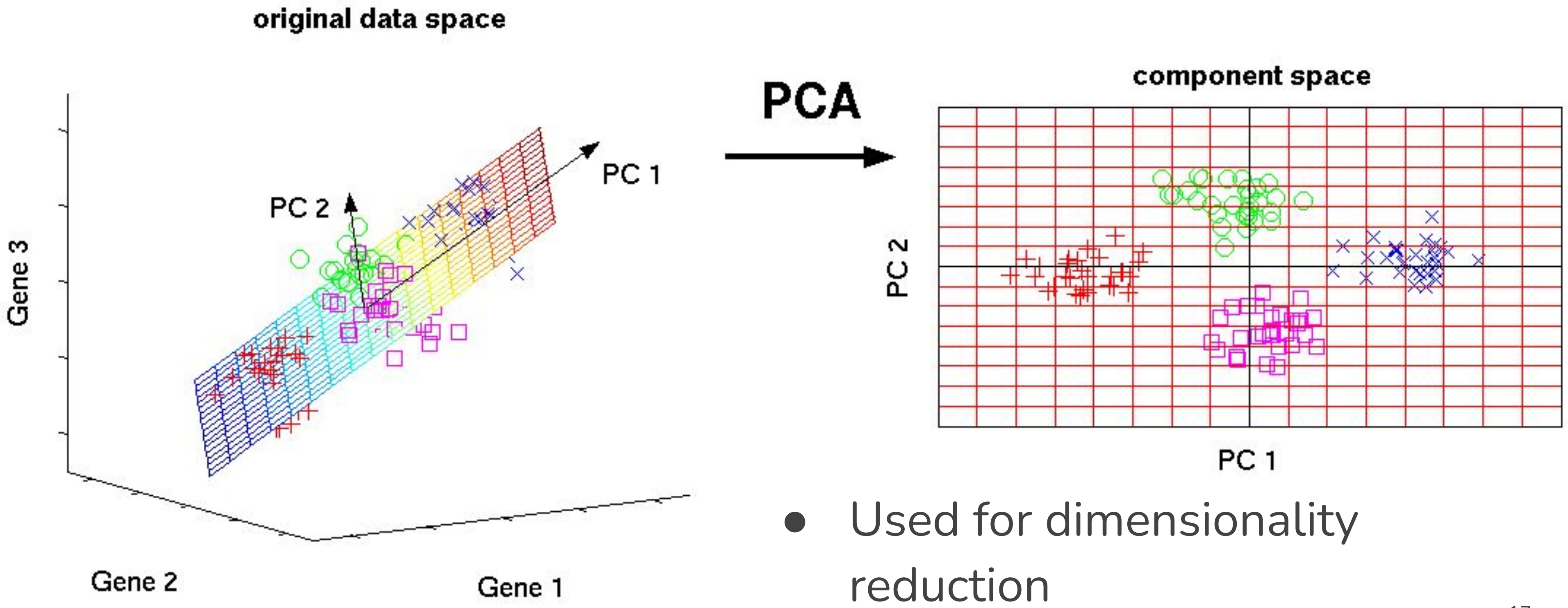
Unsupervised learning

Challenges

- No label (ground truth) in input dataset
- The system must have the ability to recognize patterns in the data without explicitly being told what patterns to identify

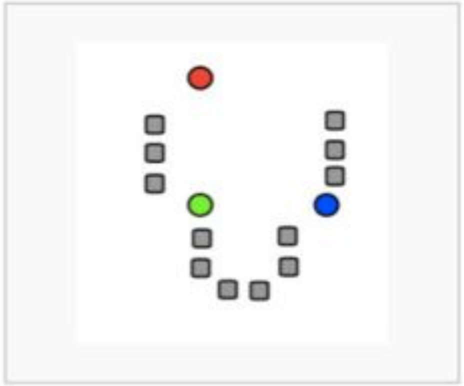


Principal Component Analysis (PCA), unsupervised

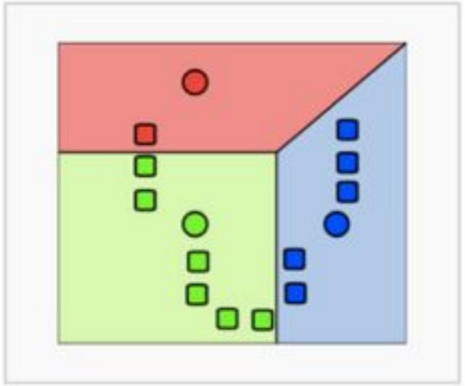


- Used for dimensionality reduction

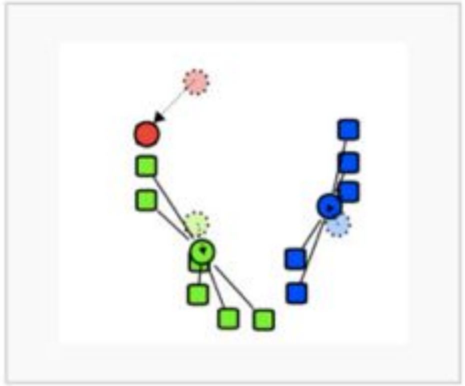
K-means clustering, unsupervised



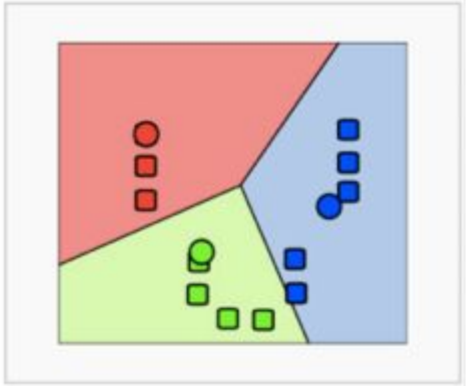
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the *k* clusters becomes the new mean.

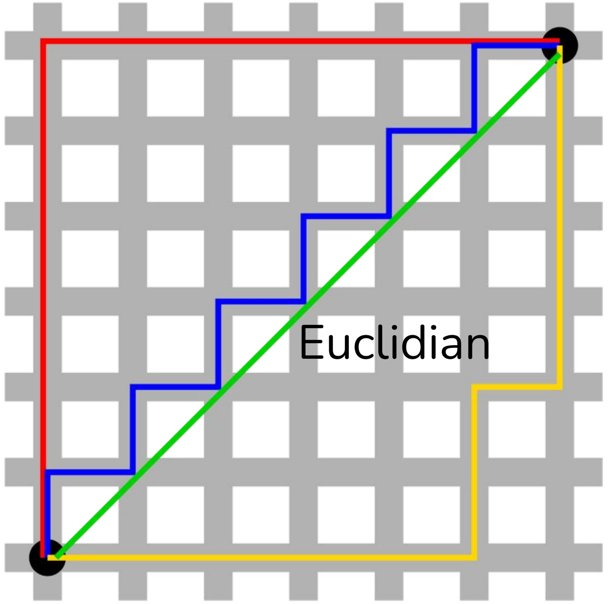


4. Steps 2 and 3 are repeated until convergence has been reached.

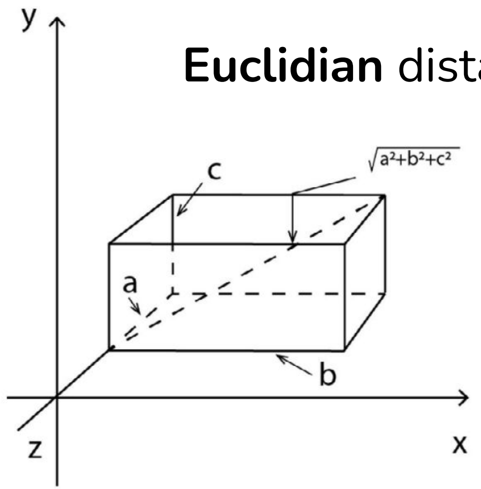
You must define **k, the number of clusters,** and which **distance** to use!

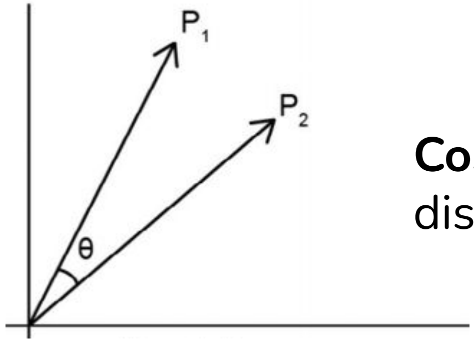
K-means: Distances

Taxicab or **Manhattan** distance: sum of the projections along all axis

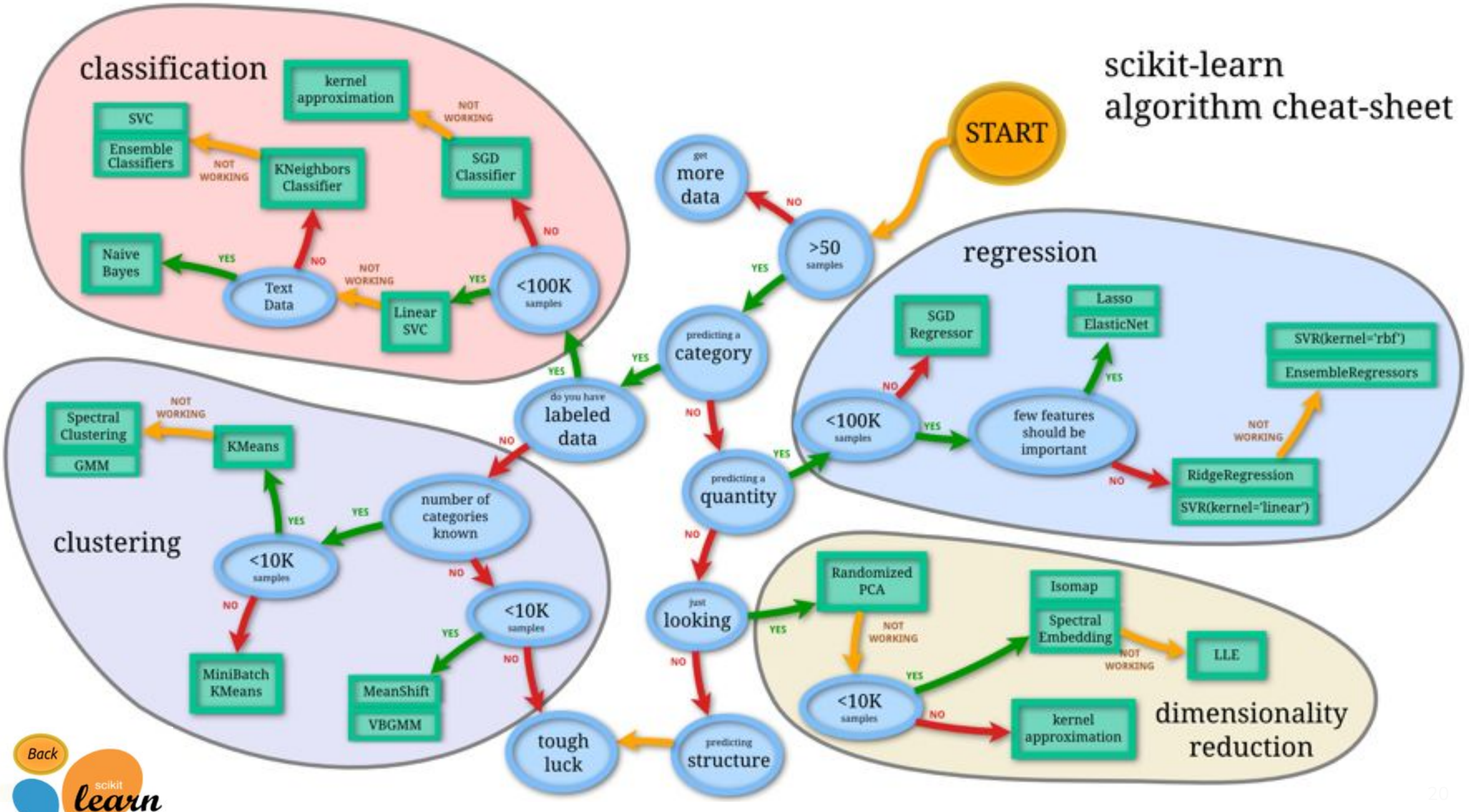


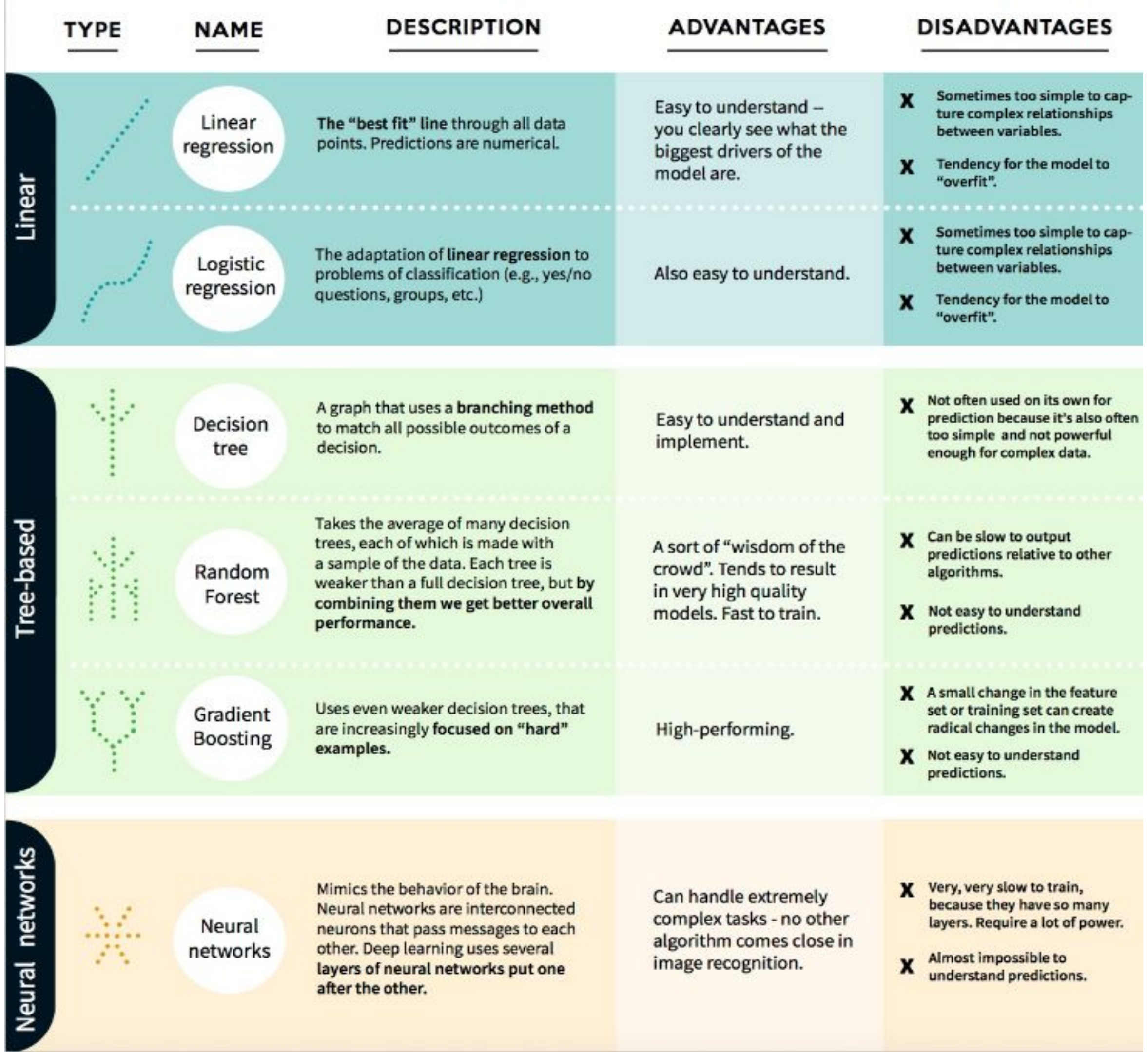
Euclidian distance





Cosine distance


scikit-learn
algorithm cheat-sheet
START
>50 samples
NO
get more data
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
YES
Linear SVC
NOT WORKING
Text Data
YES
Naive Bayes
NO
KNeighbors Classifier
NOT WORKING
SVC
Ensemble Classifiers
NO
SGD Classifier
NOT WORKING
kernel approximation
classification
NO
number of categories known
YES
<10K samples
YES
KMeans
NOT WORKING
Spectral Clustering
GMM
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
tough luck
clustering
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
few features should be important
YES
Lasso
ElasticNet
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
Back
scikit learn

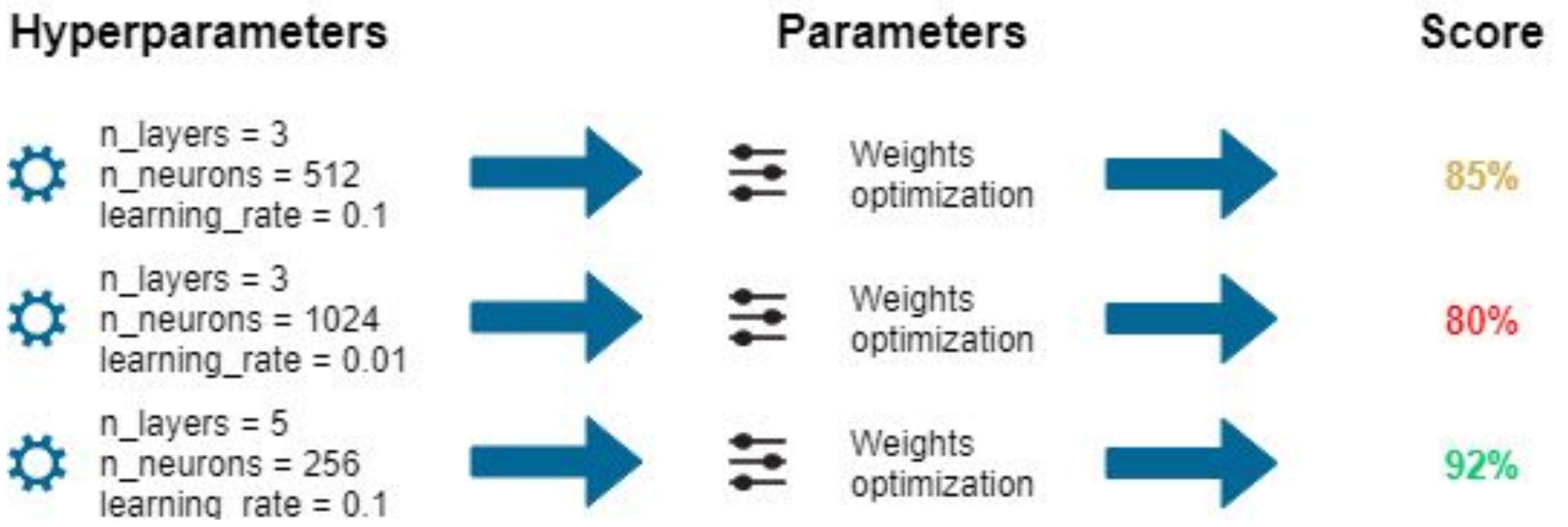


TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	The **"best fit" line** through all data points. Predictions are numerical.	Easy to understand – you clearly see what the biggest drivers of the model are.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	X Can be slow to output predictions relative to other algorithms. X Not easy to understand predictions.
	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X A small change in the feature set or training set can create radical changes in the model. X Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	X Very, very slow to train, because they have so many layers. Require a lot of power. X Almost impossible to understand predictions.

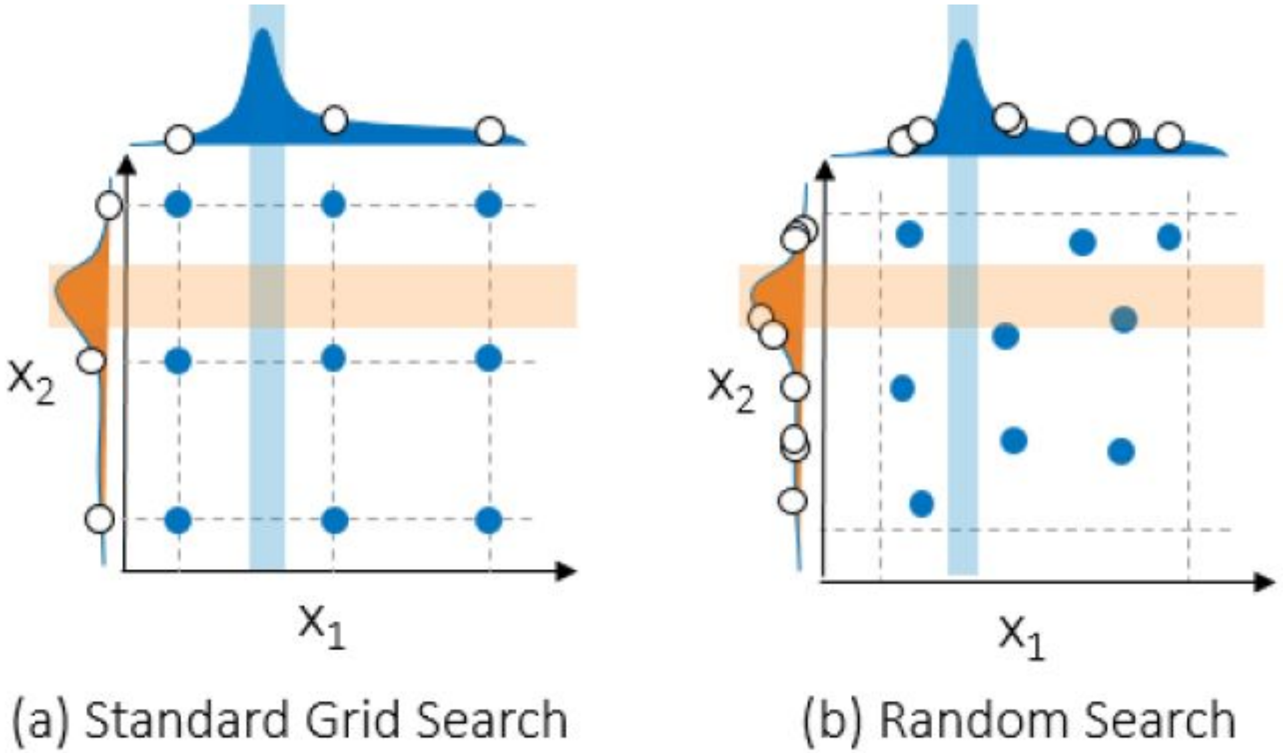
All models are wrong, but some are useful (George Box)

Hyperparameters vs parameters

- Model **parameters** are learned during **training** when we optimize a loss function -> *weights*
- **Hyperparameters** are not model parameters and they **cannot** be directly trained from the data -> *model architecture*

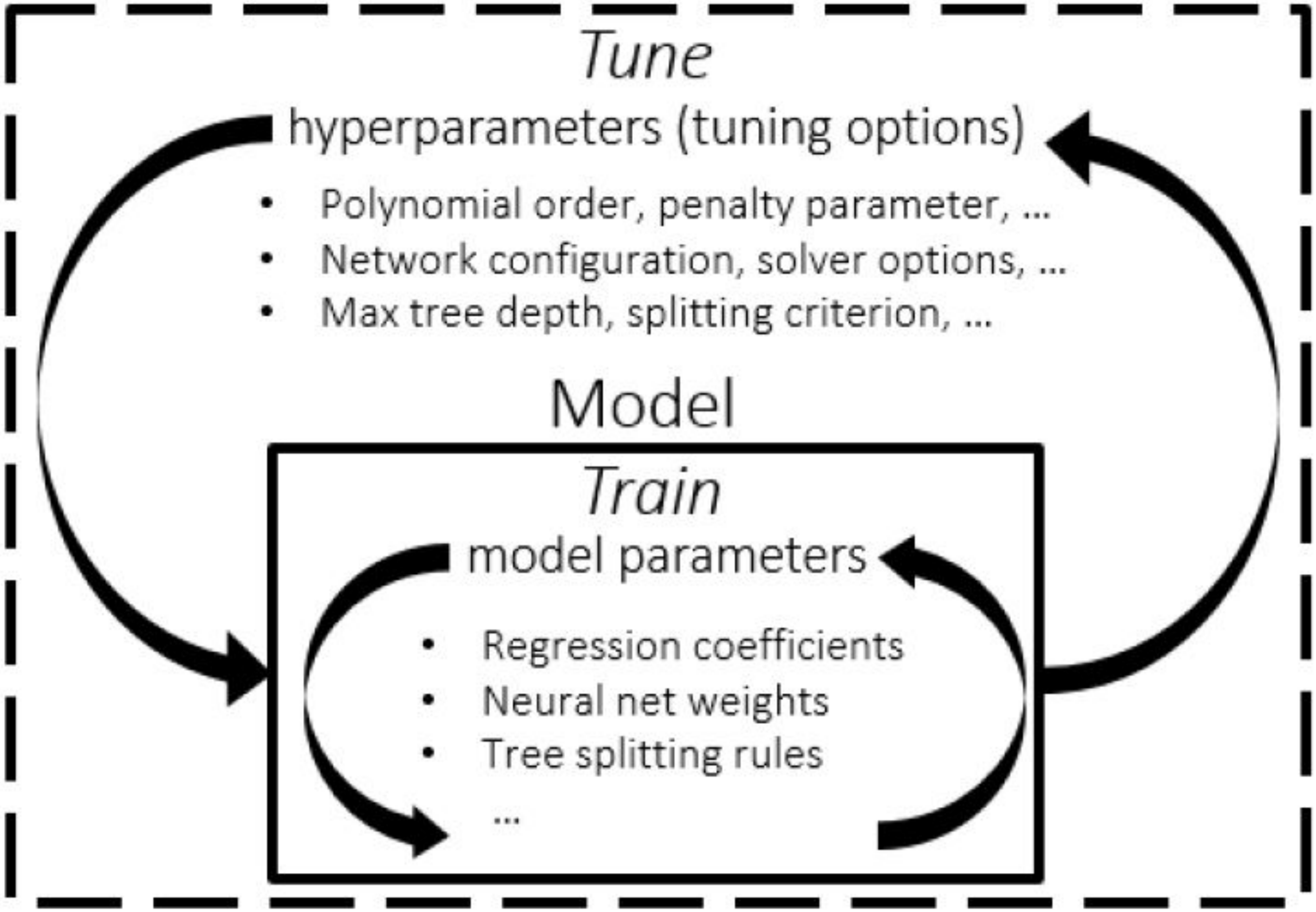


Hyperparameter tuning

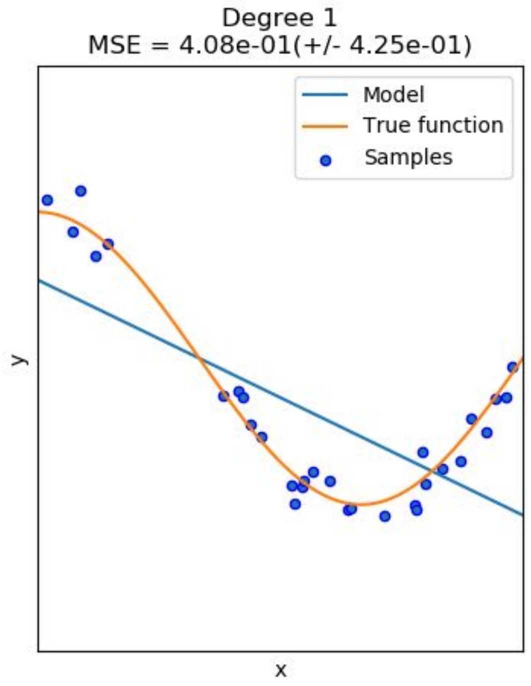


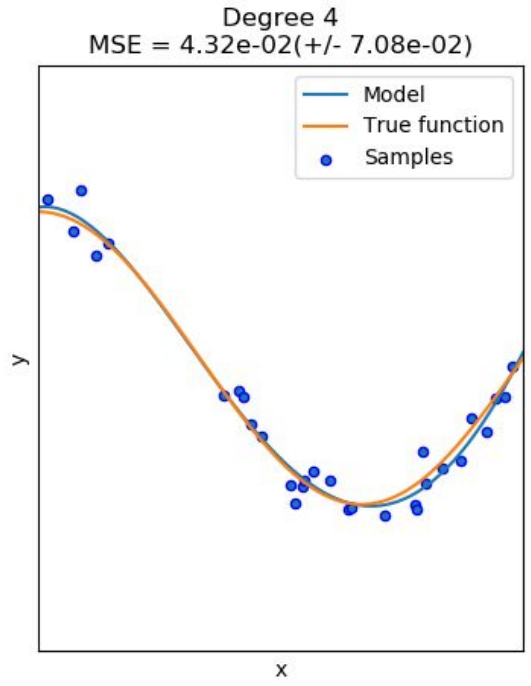
(a) Standard Grid Search (b) Random Search

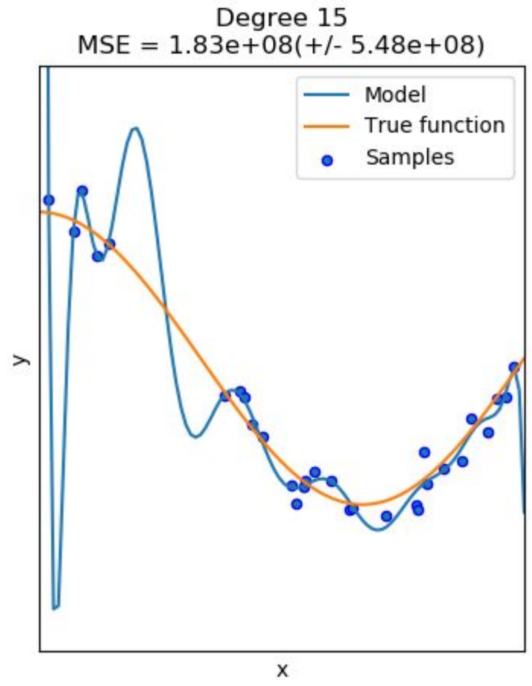
Modeling Algorithm


Tune
hyperparameters (tuning options)
• Polynomial order, penalty parameter, ...
• Network configuration, solver options, ...
• Max tree depth, splitting criterion, ...
Model
Train
model parameters
• Regression coefficients
• Neural net weights
• Tree splitting rules
...

Overfitting / underfitting





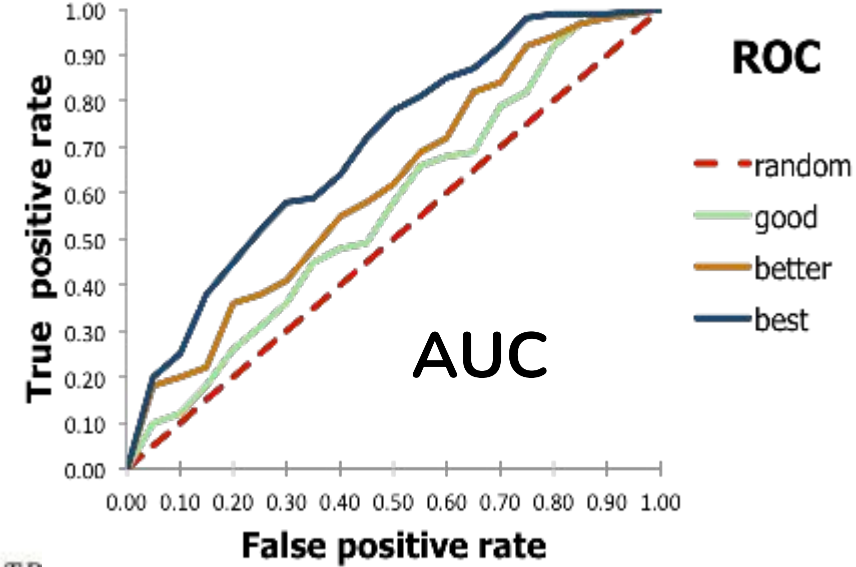


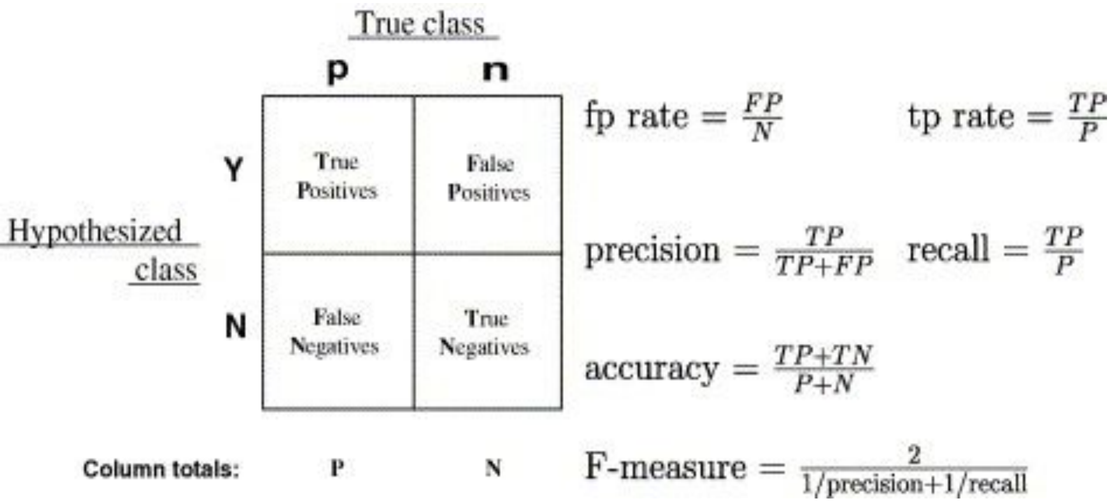
Underfitting
Model doesn't have enough (hyper-)parameters to describe data

Overfitting
Model has too many (hyper-)parameters

Classification metrics

- **ROC**: Receiver Operating Characteristics
- **AUC**: Area under the curve
- **TPR**: True positive rate
- **FPR**: False positive rate
- **TNR/FNR**: True/False negative rate





Hypothesized class		True class: p	True class: n
	Y	True Positives	False Positives
	N	False Negatives	True Negatives
Column totals:		P	N

$$\text{fp rate} = \frac{FP}{N} \qquad \text{tp rate} = \frac{TP}{P}$$

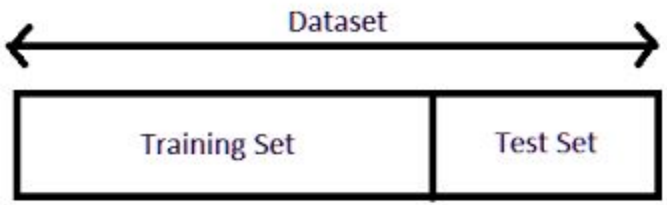
$$\text{precision} = \frac{TP}{TP+FP} \qquad \text{recall} = \frac{TP}{P}$$

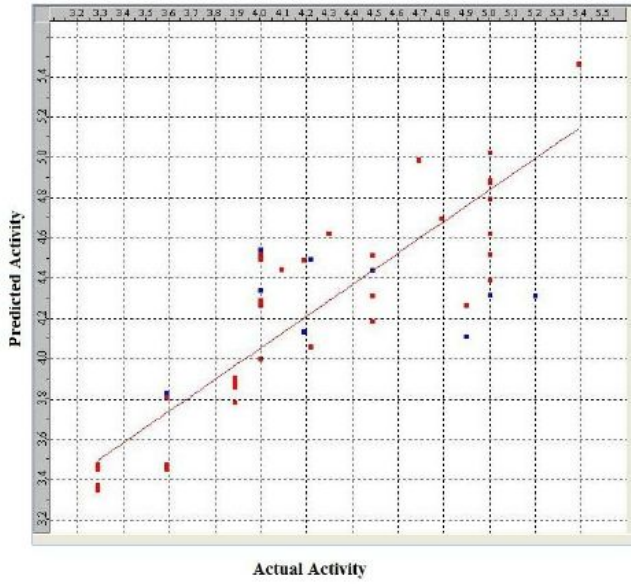
$$\text{accuracy} = \frac{TP+TN}{P+N}$$

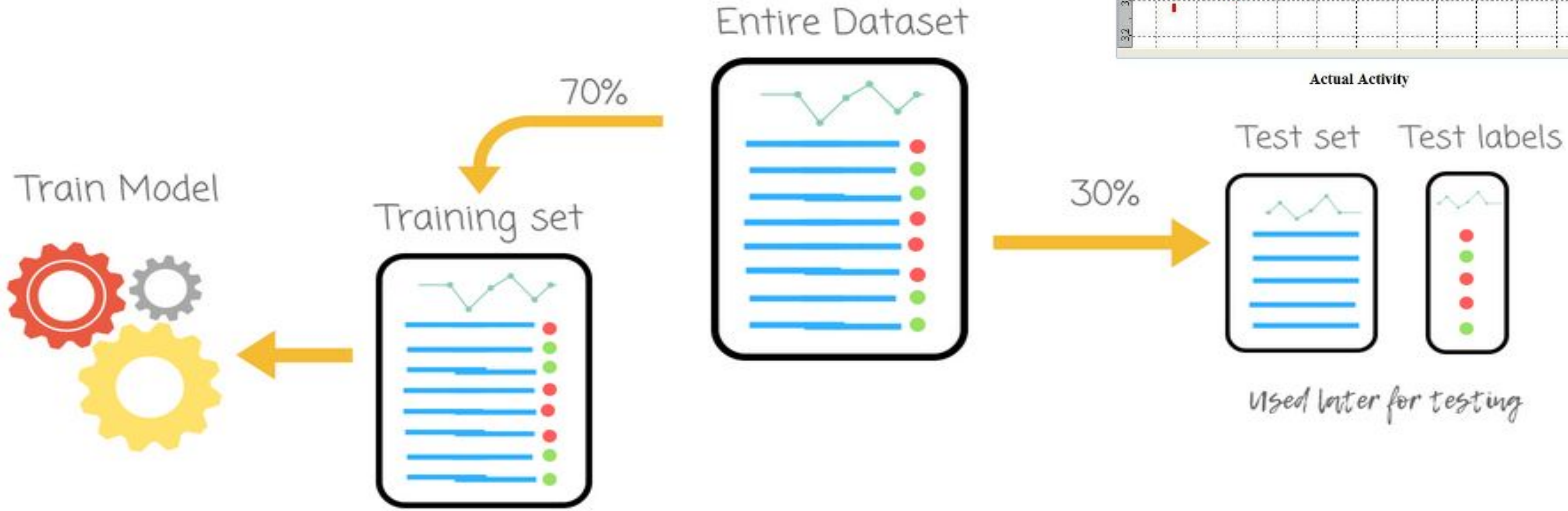
$$\text{F-measure} = \frac{2}{1/\text{precision}+1/\text{recall}}$$

Confusion matrix

Training and test set

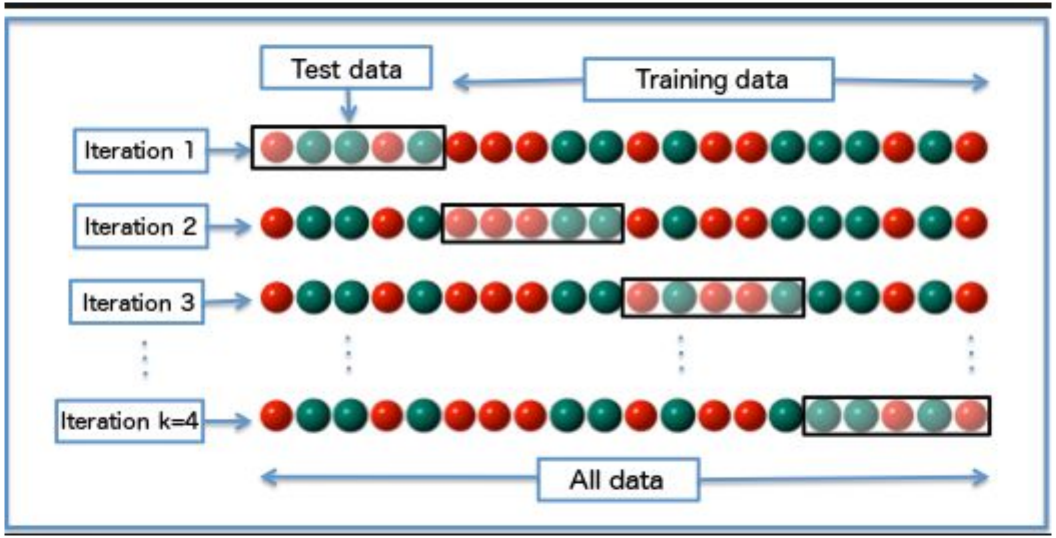


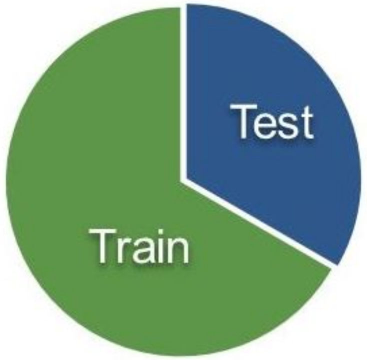




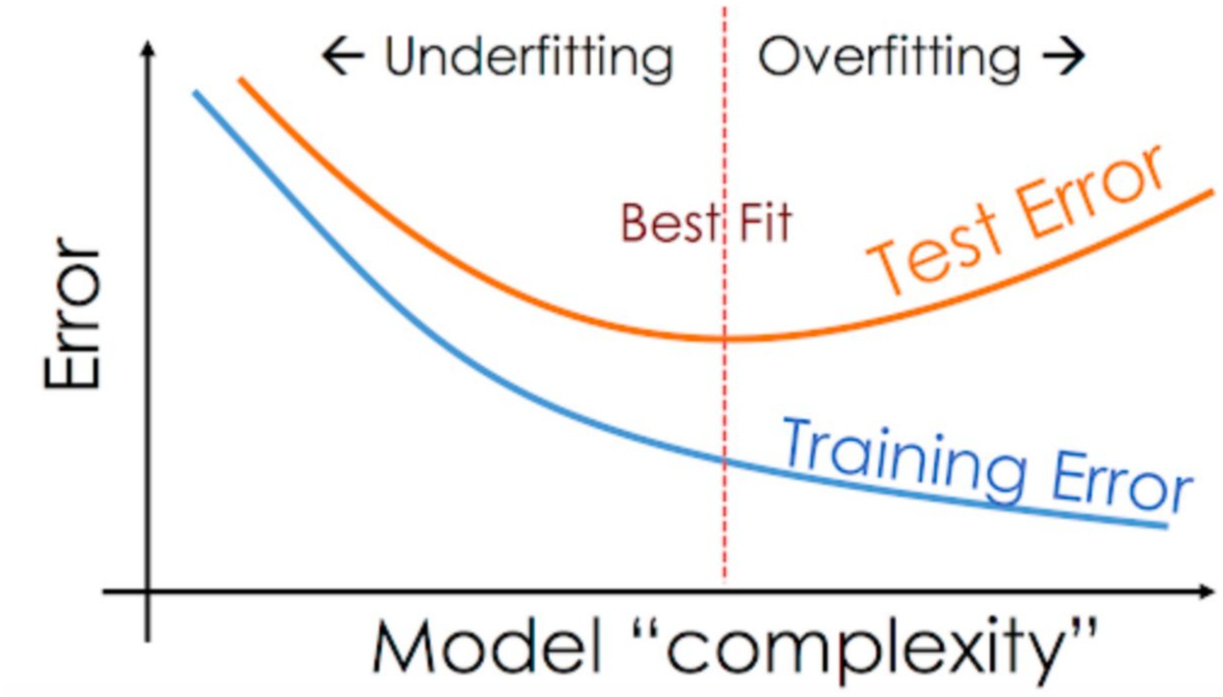
Cross-validation: is you model robust?

- Train/test split
- K-folds cross validation





Loss, or could be any other metrics of interest



Or training epochs (number of iterations)

Uncertainties

- **Covariate shift/domain shift**: change in the distribution of the input variables between the training and the test datasets
- **How to treat:**
  - **Propagate all errors** in your input variables to the output model
  - Use **cross-validation** and get an estimate of the variance of the model performances when changing hyperparameters
  - Compare model performances when **dropping/adding** features
  - If possible, compare your model results with another architecture
  - Ideally, measure error with independent model-data comparison (such as tag&probe method) just like any other classifier
- Reference