Building a performing Machine Learning model from A to Z

A deep dive into fundamental concepts and practices in Machine Learning

WHAT IS **MACHINE LEARNING** (ML)?

"A computer program is said to **learn from experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, **improves with experience E**."

Tom M. Mitchel, computer scientist, 1997

WHAT IS **MACHINE LEARNING** (ML)?

Machine Learning is everywhere….

SHAZAM uses it to **recognize** the music you are listening to.

amazon uses it to always **recommend** you more products.

Some Japanese farmers use it to **classify** the different types of cucumbers they harvest.

(read the story here)

WHAT IS A MACHINE LEARNING **MODEL**?

A **Machine Learning model** intends to determine **the optimal structure in a dataset** to achieve **an assigned task.**

It results from **learning algorithms** applied on a **training dataset.**

DATA

\+

ALGORITHMS

=

MODEL

WHAT IS A MACHINE LEARNING **MODEL**?

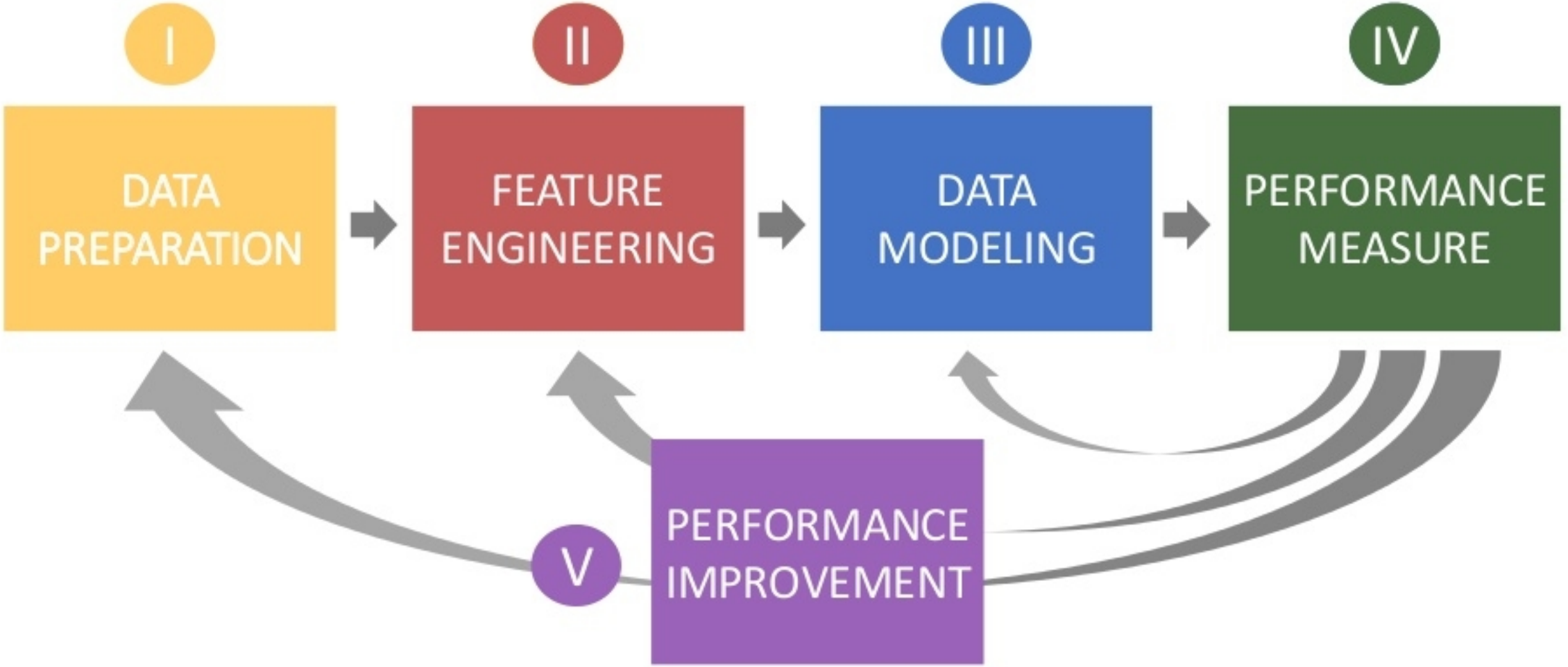
There are **4 steps** to build a machine learning model...

I DATA PREPARATION → II FEATURE ENGINEERING → III DATA MODELING → IV PERFORMANCE MEASURE

DATA + ALGORITHMS = MODEL

WHAT IS A MACHINE LEARNING **MODEL**?

This is a **highly** iterative process, to repeat…

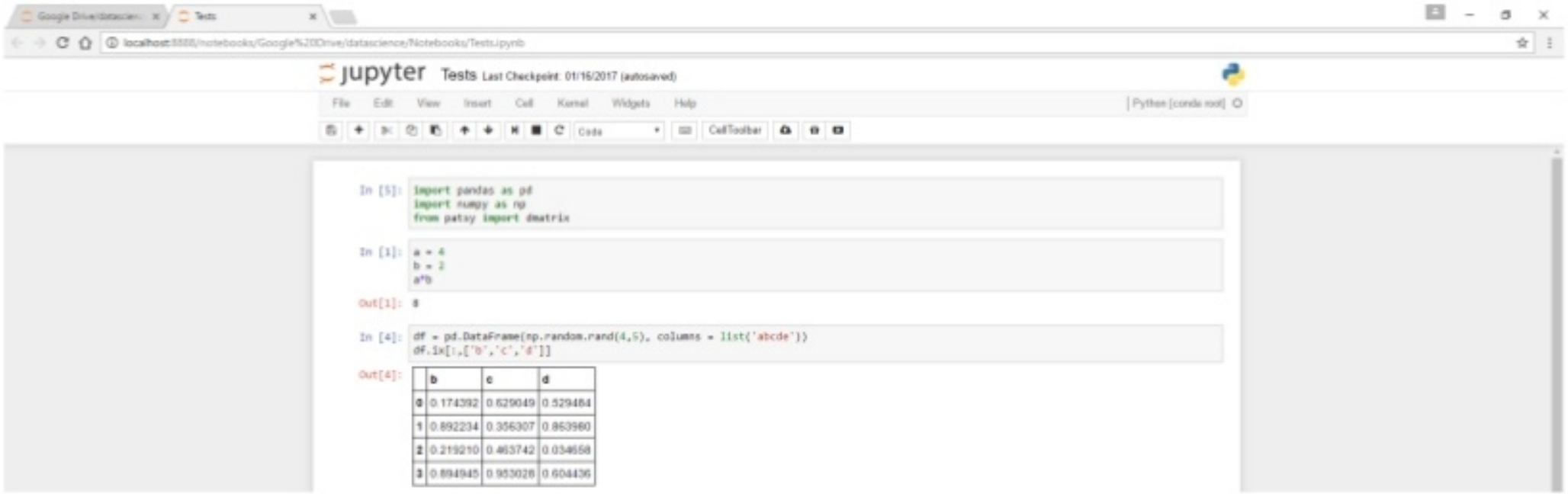


…until your model reaches a **satisfying performance!**

WHAT TOOLS WE WILL USE

You can code your ML model using programming language python™

You can write your code in jupyter, a popular interface for machine learning.



WHAT TOOLS WE WILL USE

uses the **notebook format**, with **input cells** containing code and **output cells** containing the result of your code.

```
In [1]: a = 4
        b = 2
        a*b

Out[1]: 8
```

You can iterate on your code very quickly,
instantly visualizing the result of your modifications.

WHAT TOOLS WE WILL USE

We will also use **Python modules**.
Modules are everywhere in Python, they enable to implement an infinity of actions with **just a few lines of code**.

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

pandas is the reference module to **efficiently manipulate millions rows of data** in Python.

---

scikit learn

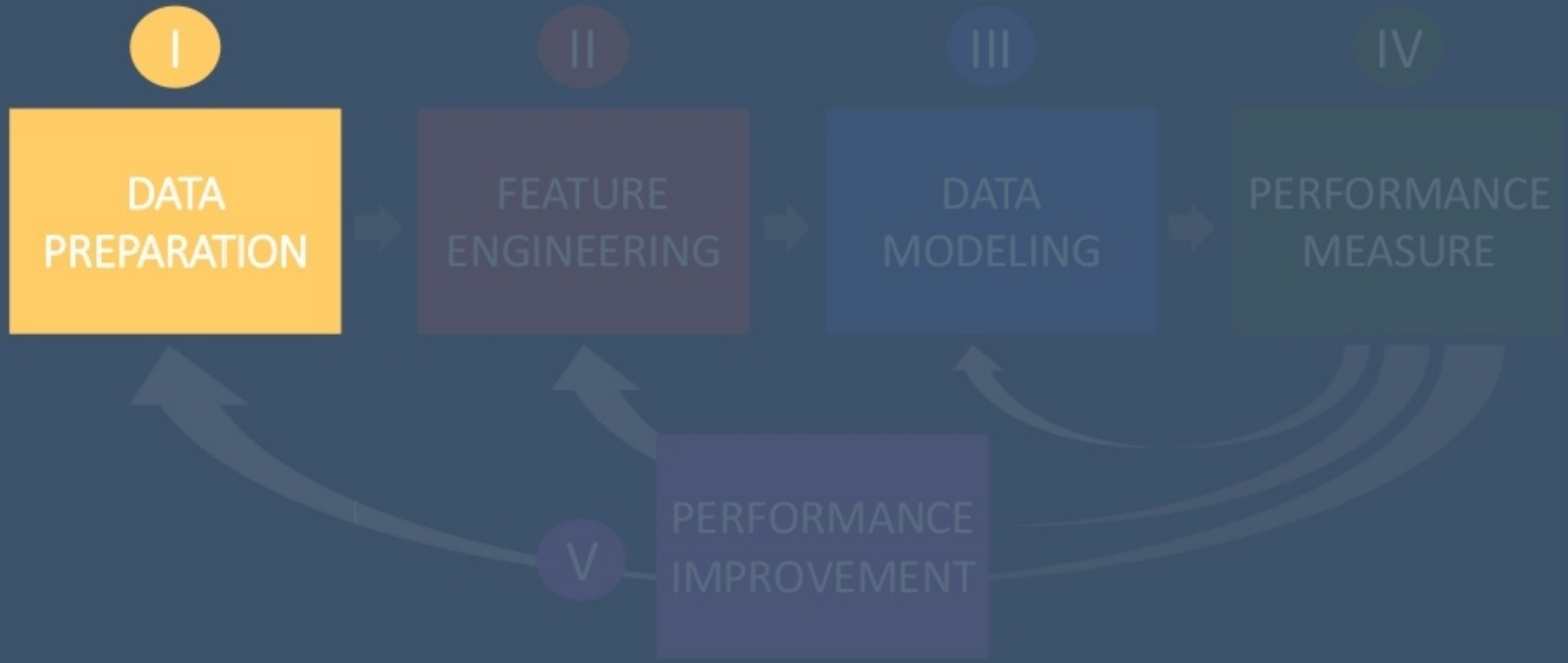
Scikit learn is one of **the reference module for machine learning** in Python.

NumPy SciPy

matplotlib

More convenient modules for **data computations and data visualization.**

WHAT YOU WILL LEARN IN THIS PRESENTATION


I
DATA PREPARATION
II
FEATURE ENGINEERING
III
DATA MODELING
IV
PERFORMANCE MEASURE
V
PERFORMANCE IMPROVEMENT

Preparing your data can be done in **3 steps**

1. 

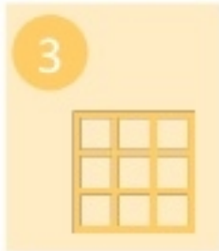
Query your data

2. 

Clean your data

a. Deal with **missing values**

b. Remove **outliers**

3. 

Format your data

You can query your data using pandas

If you have a database to connect to:

```
import pandas as pd

sql_query = """
SELECT * FROM table
LIMIT 100000
"""

dataframe = pd.read_sql(sql_query, connection_to_database)
```

If you have lots of data, you will find useful to **start working on a subset of your dataset**.
You will be able to iterate quickly since computations will be fast if there is not too much data.

If you have a csv (e.g. downloaded from the internet):

```
import pandas as pd
dataframe = pd.read_csv("dataset.csv")
subset = dataframe[:100000]
```

← Importing the whole file
← Slicing to work on a subset

This will give you a **dataframe** with your raw data

X =

	cat1	cat2	cat3	cont1	cat4	cat5	cat6	cont2	cont3	cat7	cont4	cat8	cat9	cont5	cont9
0	A	B	A	0.726300	B	A	A	0.245921	0.187583	A	0.789639	A	B	0.310061	0.67135
1	A	B	A	0.330514	A	A	A	0.737068	0.592681	A	0.614134	A	B	0.885834	0.35127
2	A	B	A	0.261841	A	B	A	0.358319	0.484196	A	0.236924	A	B	0.397069	0.26076
3	B	B	A	0.321594	B	A	A	0.555782	0.527991	A	0.373816	A	B	0.422268	0.32128
4	A	B	A	0.273204	B	A	A	0.159990	0.527991	A	0.473202	A	B	0.704268	0.22089
5	A	B	A	0.546670	A	A	A	0.681761	0.634224	A	0.373816	A	B	0.302678	0.46226
6	A	A	A	0.471447	A	B	A	0.737068	0.613660	A	0.189137	A	A	0.295397	0.40455
7	A	B	A	0.826591	B	A	A	0.488789	0.263570	A	0.623770	A	B	0.473767	0.84847
8	A	B	B	0.330514	B	B	A	0.555782	0.440642	A	0.473202	A	B	0.281143	0.38249
9	A	B	A	0.726300	A	B	B	0.358319	0.356819	A	0.802892	A	B	0.310061	0.67135
10	A	B	A	0.496063	A	A	A	0.358319	0.654310	A	0.284048	A	B	0.281143	0.53565

A **dataframe** is a common data format that will enable you to efficiently work on large volumes of data.

a Deal with missing values

Some of **your columns will certainly contain missing values**, often as 'NaN'.
You **will not be able** to use algorithms with NaN values.

Compute ratio $R_m = \frac{\text{Number of missing values}}{\text{Total number of values}}$

If R_m is high, you might want to **remove the whole column**

If R_m is reasonably low, to **avoid losing data**, you can impute the **mean**, the **median** or the **most frequent value** in place of the missing values.

scikit learn

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean')
```

b Remove outliers

Some of your columns will certainly **contain outliers**, i.e. a value that lies at an **abnormal distance** from other values in your sample.

Outliers are likely to mislead your future model, so you will have to **remove them**.

1 Remove them arbitrarily

For each of your column, you might guess arbitrarily thresholds above which your data don't make sense. **You have to be careful that you are not removing any insight !**

2 Use robust methods

Several methods, such as **robust regressions**, rely on robust estimators (e.g. the median) to remove outliers from an analysis.

```
from sklearn.linear_model import RANSACRegressor
from sklearn.linear_model import TheilSenRegressor
from sklearn.covariance import EllipticEnvelope
```

Examples of robust methods in Sklearn

FORMAT DATA

Your will have to **modify your data** so that they fit constraints of algorithms.

The most common transformation is the encoding of categorical variables.

	sex	pclass
0	male	3
1	female	1
2	female	3
3	female	1
4	male	3

We replace strings by numbers.

	sex	pclass
0	1	3
1	0	1
2	0	3
3	0	1
4	1	3

Because there is no hierarchical relationship between the 0 and 1, sex is **a "dummy variable".**
We create a new column for each of the values in the document.

	pclass	sex_female	sex_male
0	3	0	1
1	1	1	0
2	3	1	0
3	1	1	0
4	3	0	1

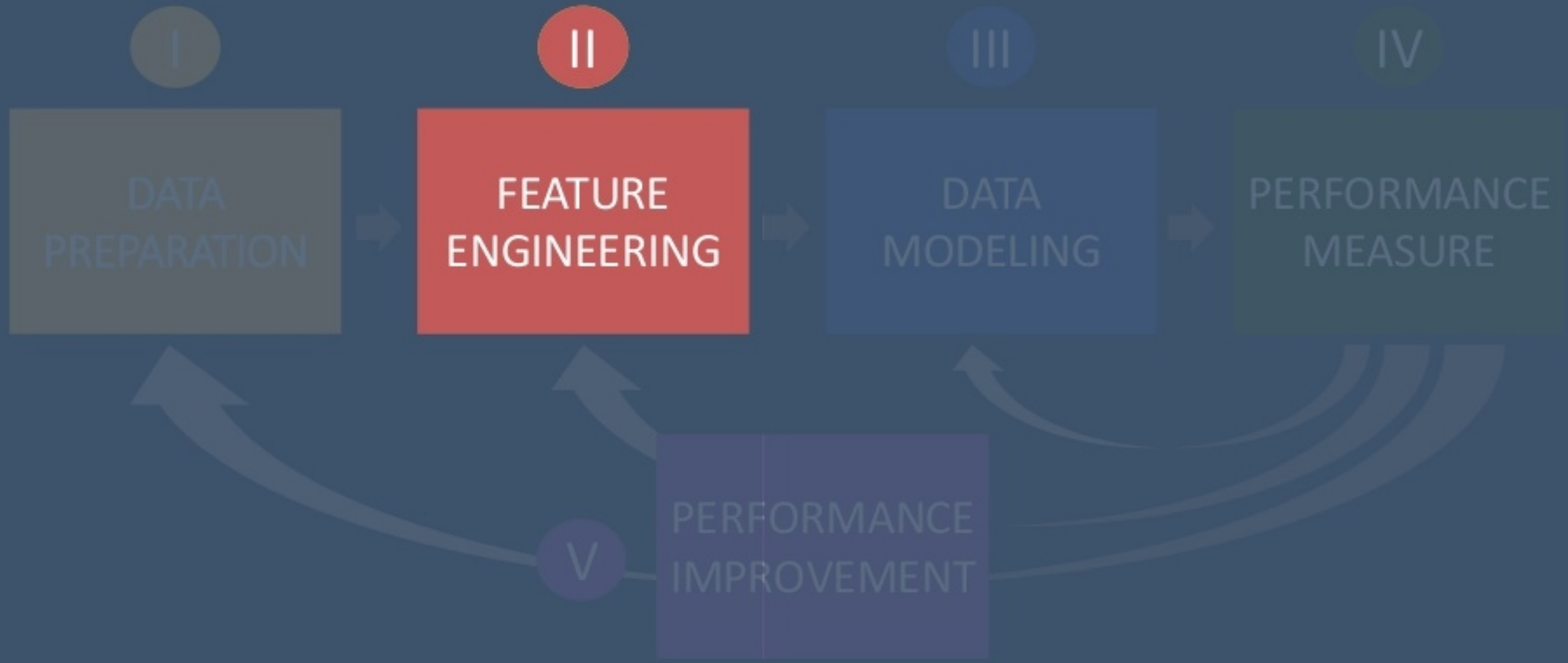
scikit learn

```
: from sklearn.preprocessing import LabelEncoder
number = LabelEncoder()
X['sex'] = number.fit_transform(X['sex'].astype('str'))
```

You can use the **patsy module** for this:

```
from patsy import dmatrix
X = dmatrix('pclass + C(sex)', X, return_type="dataframe")
```

Now that I have a clean dataset, I can start feature engineering.


I
II
III
IV
DATA PREPARATION
FEATURE ENGINEERING
DATA MODELING
PERFORMANCE MEASURE
V
PERFORMANCE IMPROVEMENT

WHAT IS A **FEATURE**?

"A **feature** is an **individual measurable property** of a **phenomenon** being observed."

Example: Predict the price of an apartment

Features *(Individual measurable properties)*	**Label** *(Phenomenon observed)*
Size: 33 sqm **Location:** Paris 6th **Floor:** 5th **Elevator:** No **# rooms:** 2 ...	400k€

The number of features you will be using is called the **dimension**.

WHAT IS FEATURE ENGINEERING?

Feature engineering is the process of transforming raw data into *relevant* features, i.e. that are:

- Informative *(it provides useful data for your model to correctly predict the label)*
- Discriminative *(it will help your model make differences among your training examples)*
- Non-redundant *(it does not say the same thing than another feature),*

resulting in improved model performance on unseen data.

WHAT IS FEATURE ENGINEERING?

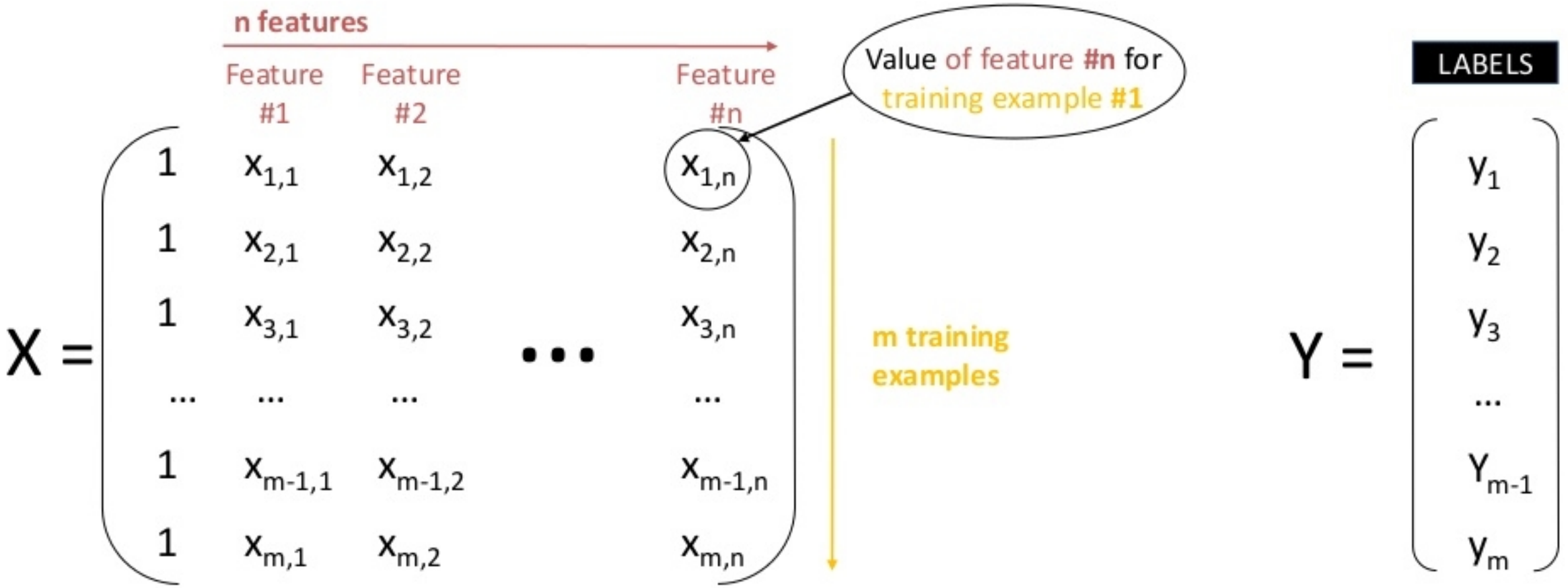
Example: Predict the price of an apartment in Paris

Is the feature ...	YES ✓	NO ✗
Informative?	Size in square meters	The name of your neighbor (unless it's Brad Pitt)
Discriminative?	Size in square meters	Simple or double glazed window? (99% is double glazing in Paris)
Non-redundant?	Size in square meters	Size in square inches

Obviously a good feature to predict the price of an apartment !

WHAT IS **FEATURE ENGINEERING**?

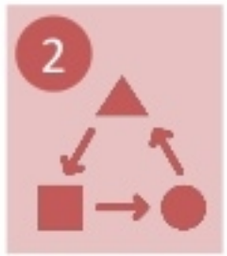
After feature engineering, your dataset will be a big matrix of numerical values.



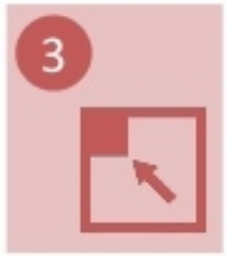
Remember that **behind "data"** there are two very different notions, training examples and features.

Feature engineering usually includes, successively:

Feature construction



Feature transformation



Dimension reduction

a. Feature **selection**

b. Feature **extraction**

Feature construction means turning raw data into **informative features** that **best represent the underlying problem** and that the **algorithm can understand**.

Example: Decompose a Date-Time

Same raw data	**Different problems**		**Different features**
2017-01-03 15:00:00 →	Predict how much hungry someone is	 →	**"Hours elapsed since last meal"**: 2
2017-01-03 15:00:00 →	Predict the likelihood of a burglary	 →	**"Night"**: 0 (numerical value for "False")

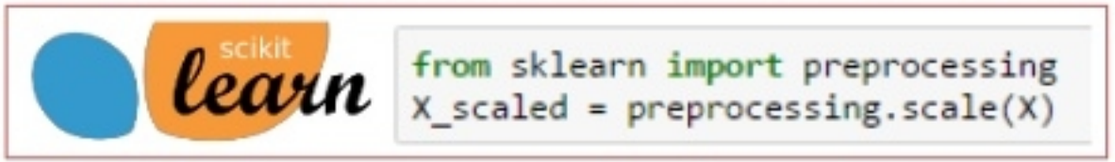
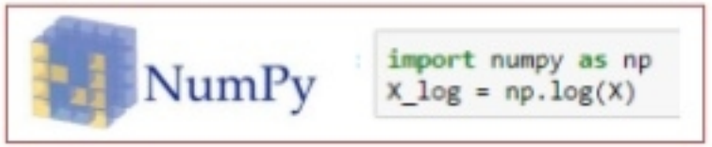
FEATURE CONSTRUCTION

Feature construction is where you will need all the domain expertise and is key to the performance of your model!

FEATURE TRANSFORMATION

Feature transformation is the process of **transforming a feature** into a new one with a specific function.

Examples of transformations:

Name	Transformation	Objectives
Scaling	$X_{new} = \frac{X_{old} - \mu}{\sigma}$	**The most important. Many algorithms need feature scaling** for faster computations and relevant results, e.g. in **dimension reduction** 
Log	$X_{new} = \log(X_{old})$	**Reduce heteroscedasticity** *(learn more)*, which can be an issue for some algorithms 

DIMENSION REDUCTION

Dimension reduction is the process of **reducing the number of features** used to build the model, with the goal of keeping only **informative, discriminative non-redundant features**.

The main benefits are:

- Faster computations
- Less storage space required
- Increased model performance
- Data visualization *(when reduced to 2D or 3D)*

a Feature selection

Feature selection is the process of **selecting the most relevant features** among your existing features.

To keep "relevant" features only, we will remove features that are:

i. Non informative

ii. Non discriminative

iii. Redundant

a Feature **selection**

i

REMOVE **NON INFORMATIVE** FEATURES

Method: Recursive Feature Elimination (RFE) *(among others)*

Principle: We eliminate a single feature **in turn**, run the model each time and note impact on the performance of the model.

Example: Predict the price of an apartment

Size: 3 sqm
Location: Paris 6th
Floor: 5th
Elevator: No
...

Test model

What is the impact on model performance?

The lower the impact, the less informative the feature is, and vice-versa.

scikit learn

```
from sklearn.feature_selection import RFE
rfe = RFE(estimator=estimator, n_features_to_select=1, step=1)
rfe.fit(X, y)
```

a Feature selection

ii

REMOVE **NON DISCRIMINATIVE** FEATURES

Method: Variance threshold filter filter

Principle: We remove any feature whose values are close across all the different training examples (i.e. that have **low variance**)

Ex: Predict the price of houses that are **all white**

A feature that always says the same thing won't help your model!

scikit learn

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=threshold_value)
sel.fit_transform(X)
```

a Feature selection

iii

REMOVE **REDUNDANT** FEATURES

Method: High correlation filter

Principle: We remove features that are similar or highly correlated with other feature(s).

Ex: **Same size** in square meters and square inches

Your model doesn't need the **same information twice!**

You can detect **correlated features** computing the Pearson product-moment correlation coefficients matrix.

NumPy

```
import numpy as np
matrix = np.corrcoef(X)
```

a Feature **selection**

Identifying the most relevant features will help you get a better general understanding of the drivers of the phenomenon you are trying to predict.

Learn more on feature selection

b Feature **extraction**

Feature extraction starts from an initial set of measured data and **automatically** builds **derived features** that are more relevant.

Automated dimension reduction that is **efficient** and **easy to implement.**

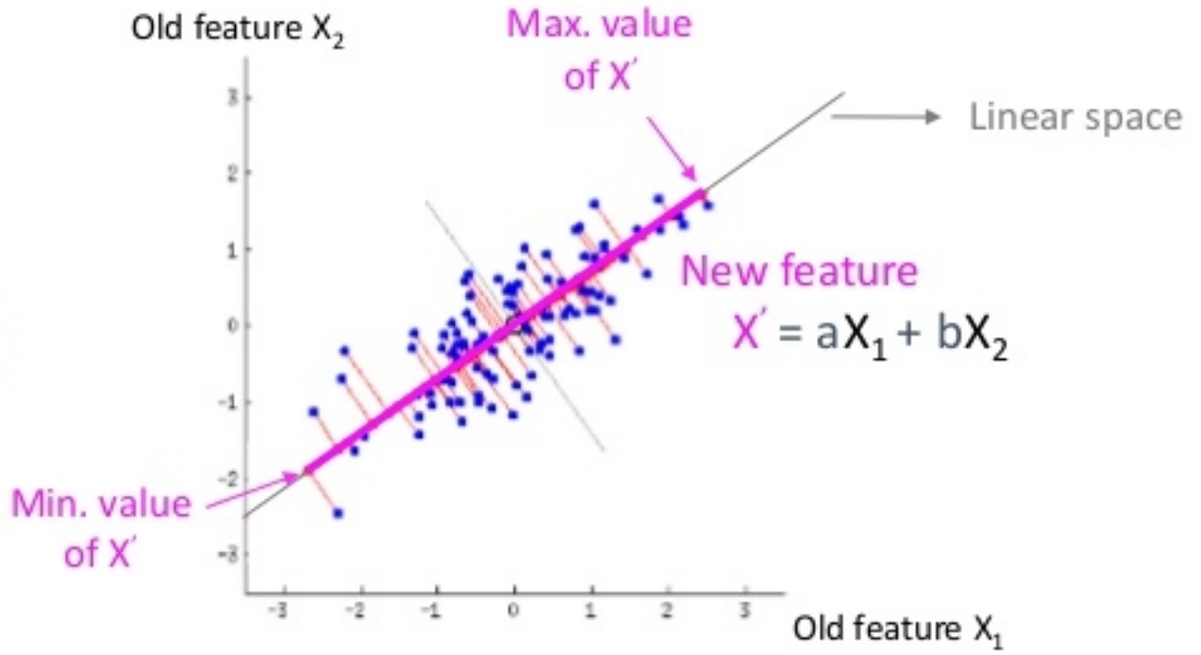
The new features given by the algorithms are **difficult to interpret**, unlike feature selection.

b Feature **extraction**

The most common algorithm for feature extraction is **Principal Component Analysis (PCA).**

PCA makes an orthogonal **projection on a linear space** to determine new features, called **principal components**, that are a linear combination of the old ones.

Example of reduction of 2 features into a single one

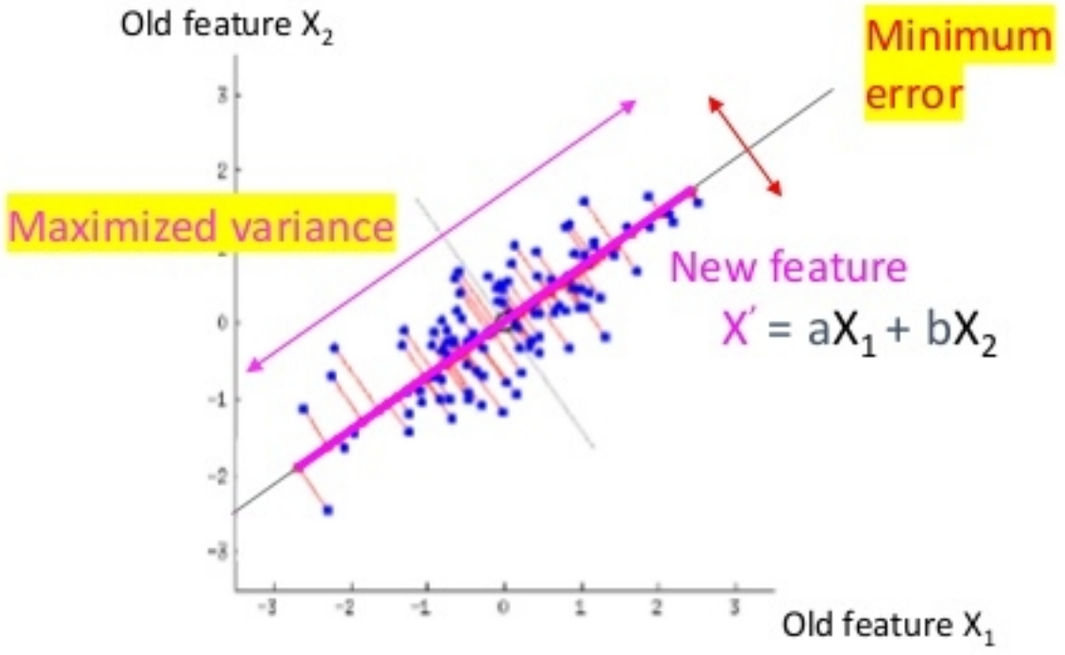


b Feature **extraction**

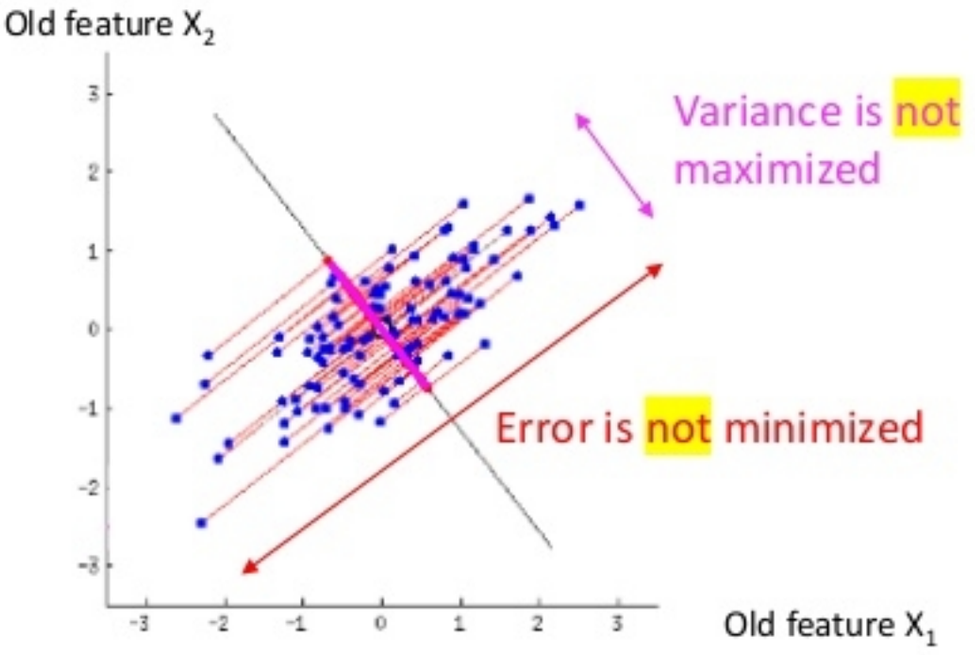
The principal component (PC) is built along an axis so that it is, as much as possible:

- **Discriminative** (its **variance** is **maximized**)
- **Informative** (the **error** to original values is **minimized**)

PCA ✓



Example of projection that is not a PCA ✗

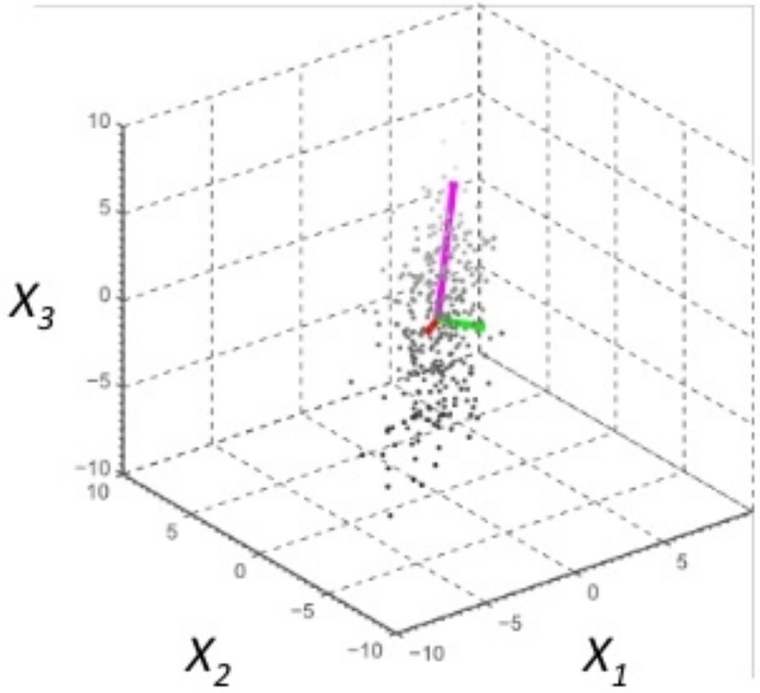


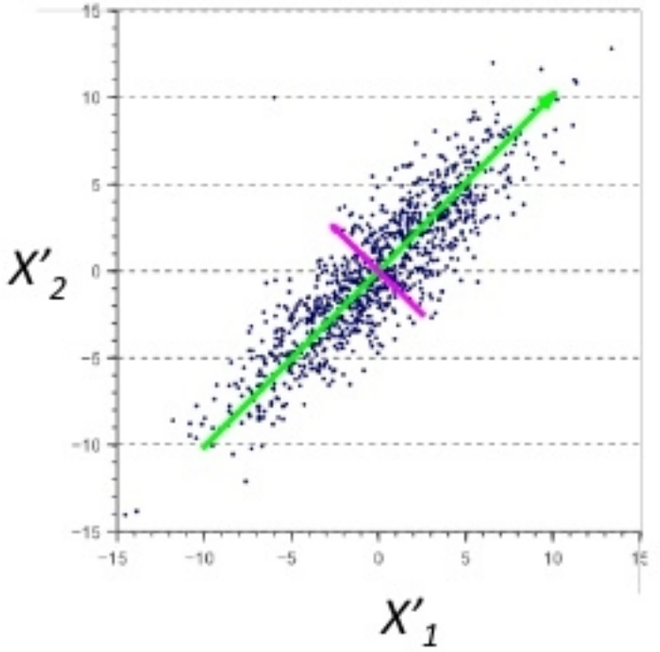
b Feature **extraction**

The principal components (PC) are built along axes so that they are, as much as possible:

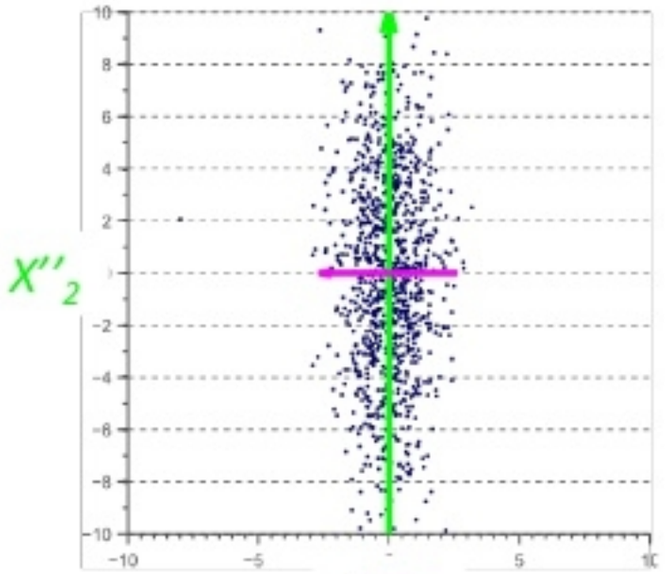
- **Independent** (i.e. non-redundant) from other PCs

Example of reduction of 3 features into 2 PCs





Correlated features

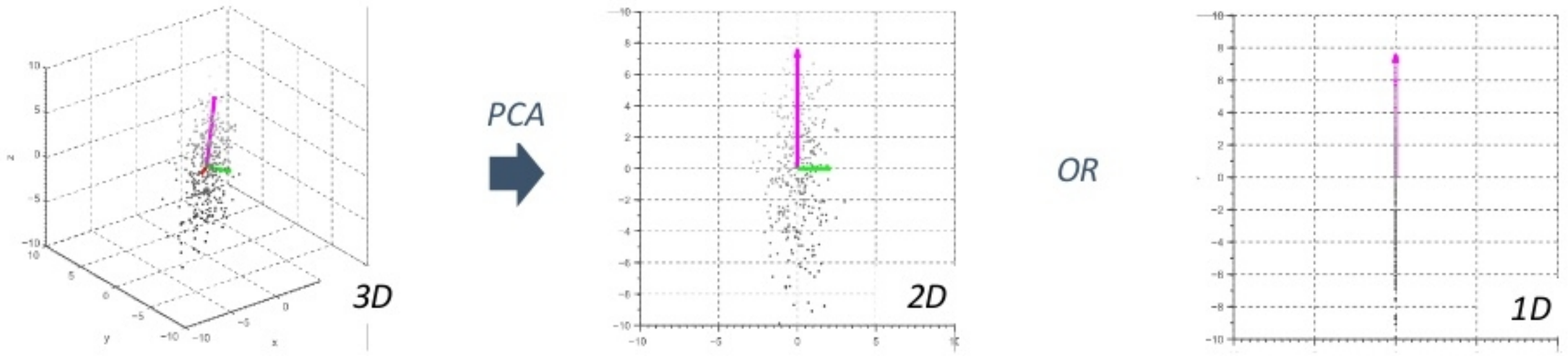


Independent features (principal components)

b Feature extraction

IN A NUTSHELL: Principal Component Analysis (PCA)

Given a desired number of final features, PCA will create these features called principal components **minimizing the loss of information** from initial data and thus **maximizing their relevance** (i.e. informative, discriminative, non-redundant).




```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
Principal_components = pca.fit(X).transform(X)
```

Desired number of final features

Learn more about PCA

b Feature **extraction**

A famous implementation of PCA is in face recognition.

IN A NUTSHELL

Feature engineering is the process of transforming raw data into *i)* informative *ii)* discriminative *iii)* non-redundant features.

Methods

1

Feature construction

Turn raw data into relevant features that best represent the underlying problem.

2

Feature transformation

Transform features so that they **fit some algorithms constraints**.

3

Dimension reduction

Eliminate less relevant features either by **selecting** them or by **extracting** new ones automatically.

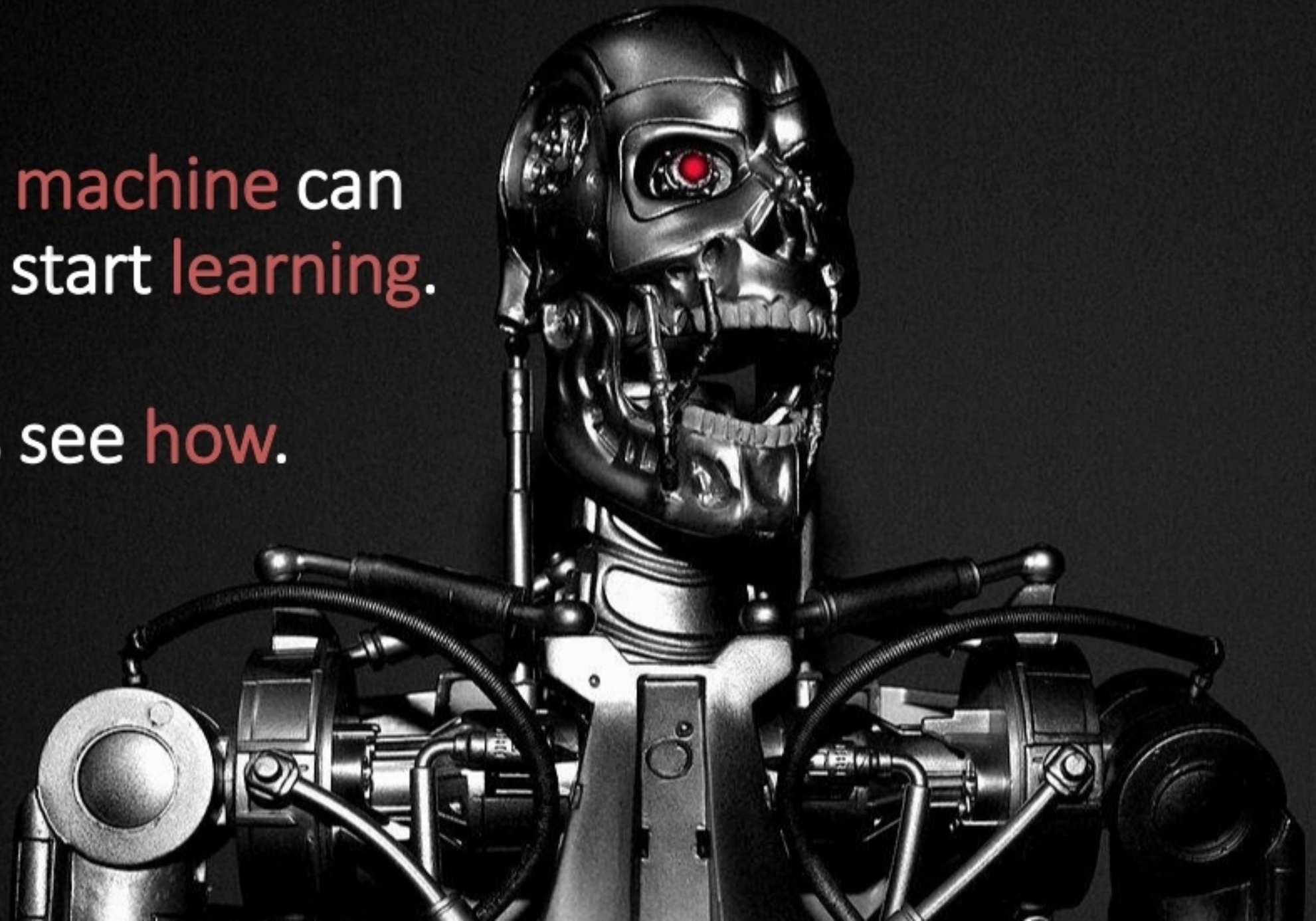
Benefits

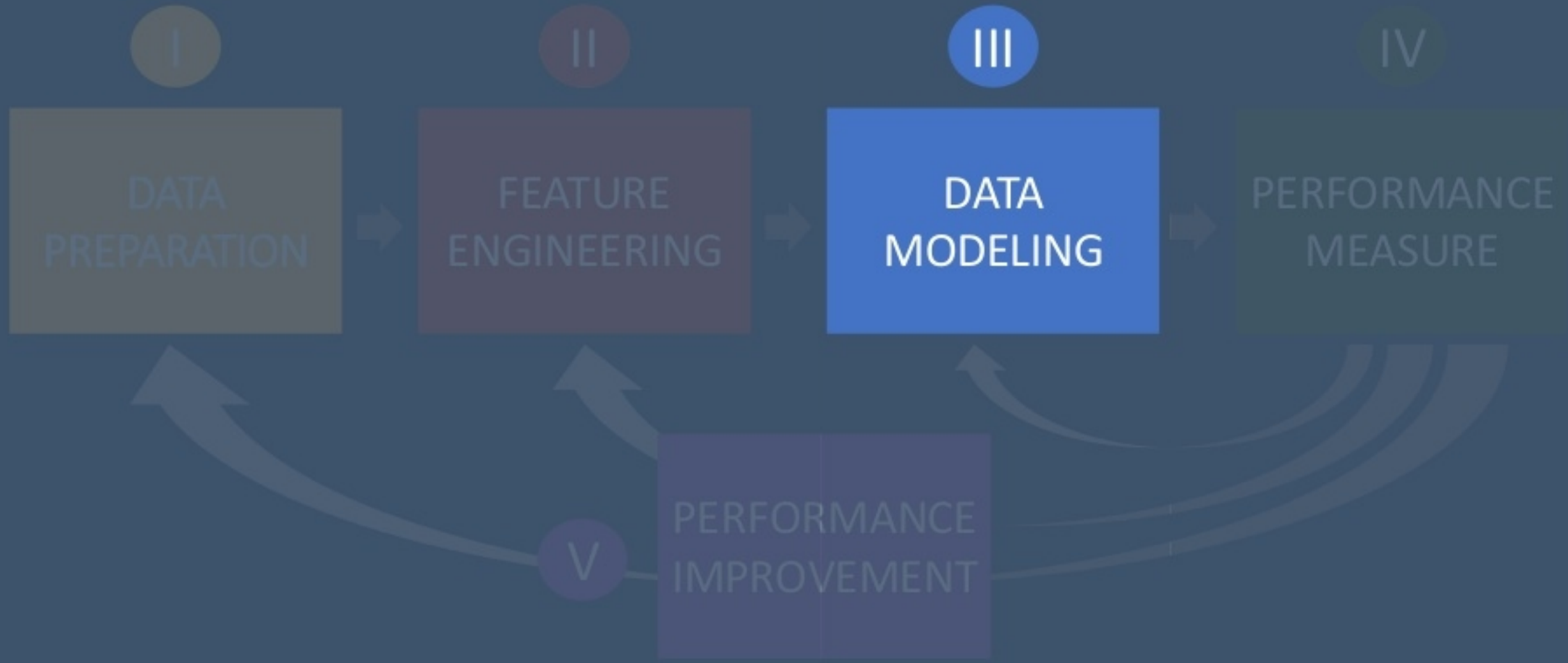
- Faster computations
- Less storage space required
- Increased model performance
- Data visualization

Feature engineering is a very important part (if not the most) to build a performing Machine learning model.

Your machine can now start learning.

Let's see how.




I
II
III
IV
DATA PREPARATION
FEATURE ENGINEERING
DATA MODELING
PERFORMANCE MEASURE
V
PERFORMANCE IMPROVEMENT


"The goal is to turn
data into information, and
information into insight."
Carly Fiorina, former CEO of Hewlett-Packard

You are going to train a model on your data using a learning algorithm.

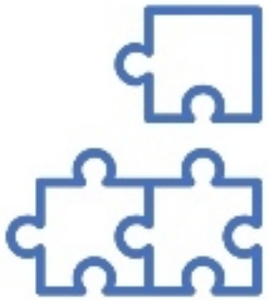
Remember:

DATA

+

ALGORITHMS

=



MODEL

Supervised vs. unsupervised learning

a **Regression** with Linear Regression

b Cost function and Gradient Descent

c **Classification** with Random Forests

d **Clustering** with K-means

Parametric vs. nonparametric algorithms

What are the best algorithms?

Supervised vs. Unsupervised

Supervised vs. unsupervised learning

SUPERVISED LEARNING

When the training set contains labels (i.e. outputs/target)

Example: Predict the price of an apartment

FEATURES					LABEL
Size (m²)	**# rooms**	**Location**	**Floor**	**Elevator**	**Price (k€)**
62	3	Paris	3	Yes	500
92	4	Lyon	4	No	400
43	2	Lille	5	Yes	200

UNSUPERVISED LEARNING

When the training set contains no label, only features

Example: Define client segments within a customer base

FEATURES				
Name	**Gender**	**Age**	**Location**	**Married**
John	M	46	New-York	Yes
Sarah	F	42	San Francisco	No
Michael	M	18	Los Angeles	Yes
Danielle	F	54	Atlanta	Yes

LABEL

Supervised vs. unsupervised learning

Supervised learning algorithms are used to build two different kind of models.

a **REGRESSION**

When the label to predict is a continuous value

Example: Predict the price of an apartment

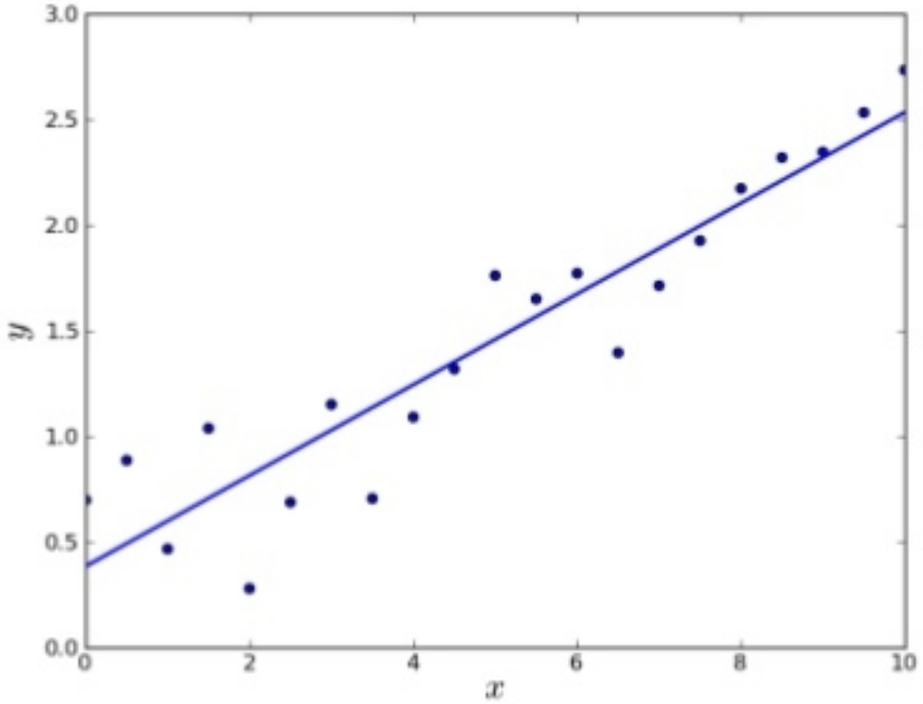
c **CLASSIFICATION**

When the label to predict is a discrete value

Example: Predict how many stars I am going to rate a movie on Netflix (0,1,2,3,4,5)

a Regression with Linear Regression

The output of a linear regression on some data will look like this:



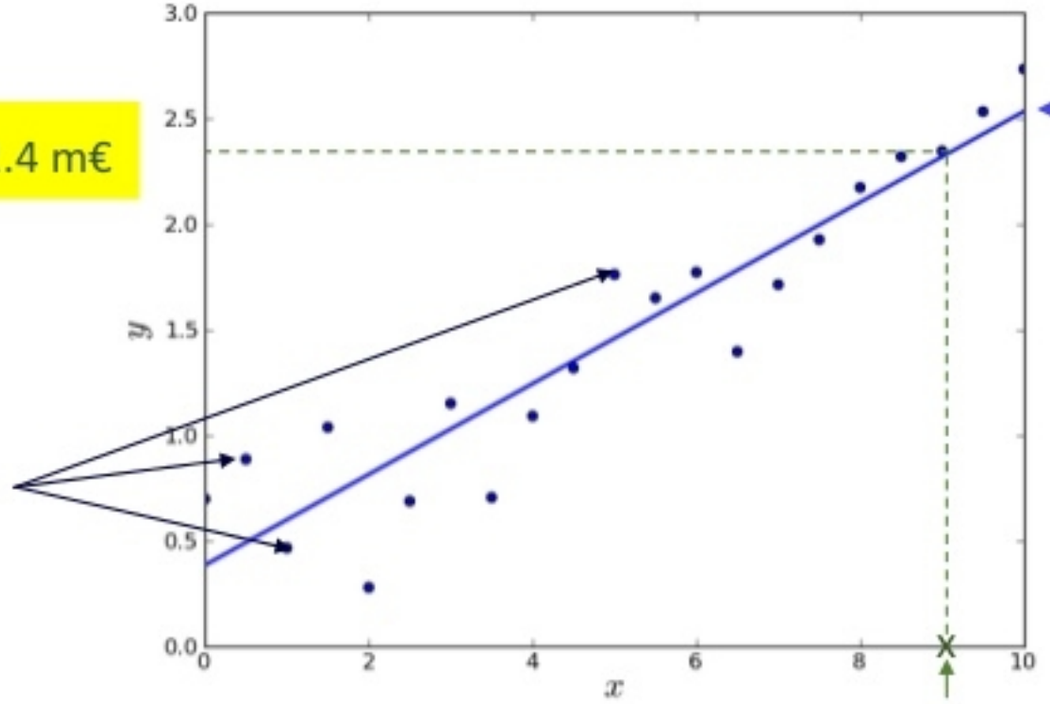
How is this a machine learning model?

a Regression with Linear Regression

Example: Predict the price of an apartment with 1 feature

Continuous Label Y: price (m€)

Output: Predicted price = 2.4 m€



Trained model
Y = 0.2X + 0.4

Training examples (training dataset)

Feature X: # of rooms

Input: Unknown apartment with 9 rooms

a Regression with Linear Regression

Using a Linear Regression assumes there is a linear relationship between your features X and the labels Y.

For all i = 1, …, m:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_n x_{i,n} + \varepsilon_i$$

Variable representing **random error (noise)** in the data, assumed to follow a standard normal distribution.

which gives, in matrix form:

$$Y = X\beta + \varepsilon \quad \text{where} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \ldots \\ \beta_m \end{pmatrix} \quad \varepsilon = \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \ldots \\ \varepsilon_m \end{pmatrix}$$

and X, Y as defined in slide 23

b Cost function and Gradient descent

The basic Linear regression method is called Ordinary Least Squares and will try to **minimize** the following function,

$$J(\beta) = ||Y - X\beta||_2^2 = \sum_{i=0}^{m} (Y_i - X_i\beta)^2$$

(where X_i is the feature vector of the i-th training example, and Y_i the corresponding label)

called "cost function" or "loss function", representing the difference between your predictions and the true labels.

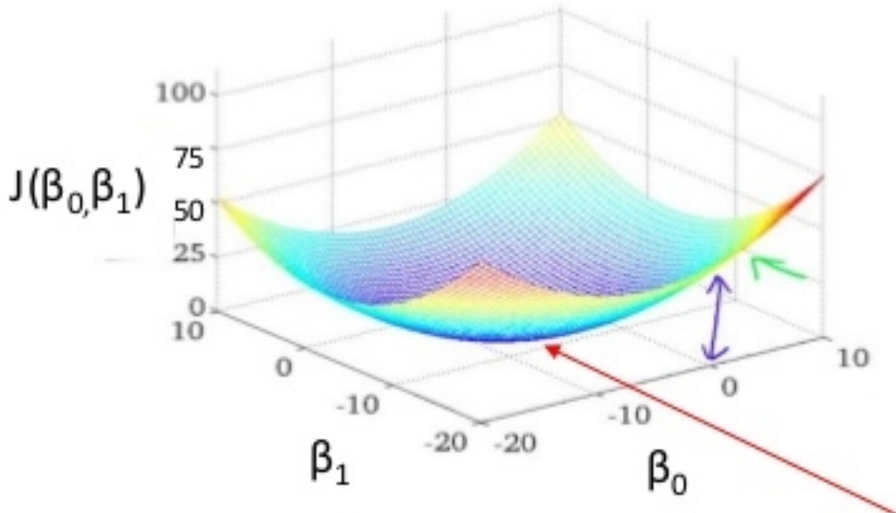
All learning algorithms are about minimizing a certain cost function.

b Cost function and Gradient descent

Cost functions are often minimized using an algorithm called **Gradient descent**.

Gradient Descent is an **iterative optimization algorithm** that will look step by step for β values where the gradient (i.e. the derivative) equals to 0, thus finding a local minimum of the cost function.

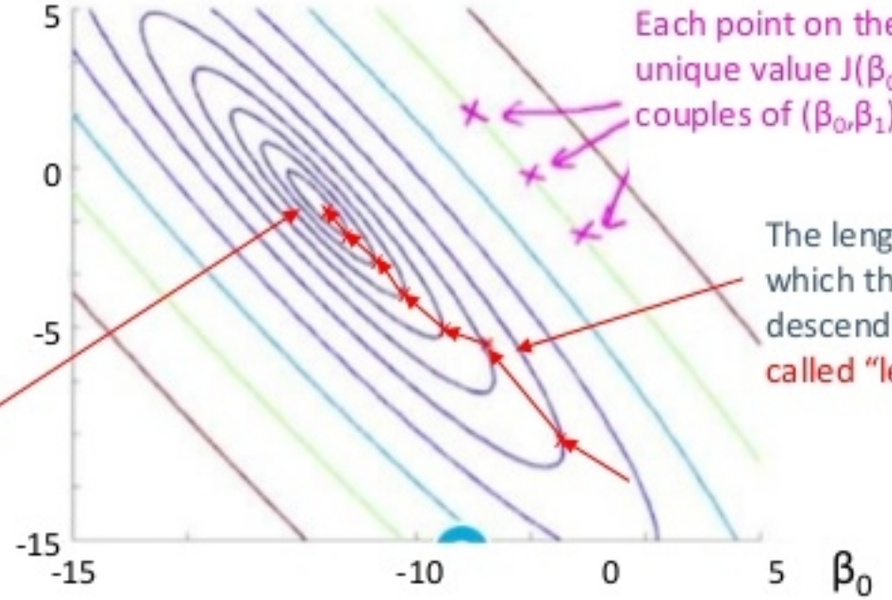
Cost function of a Linear regression with 2 params (β_0, β_1) only



3D representation

Minimum

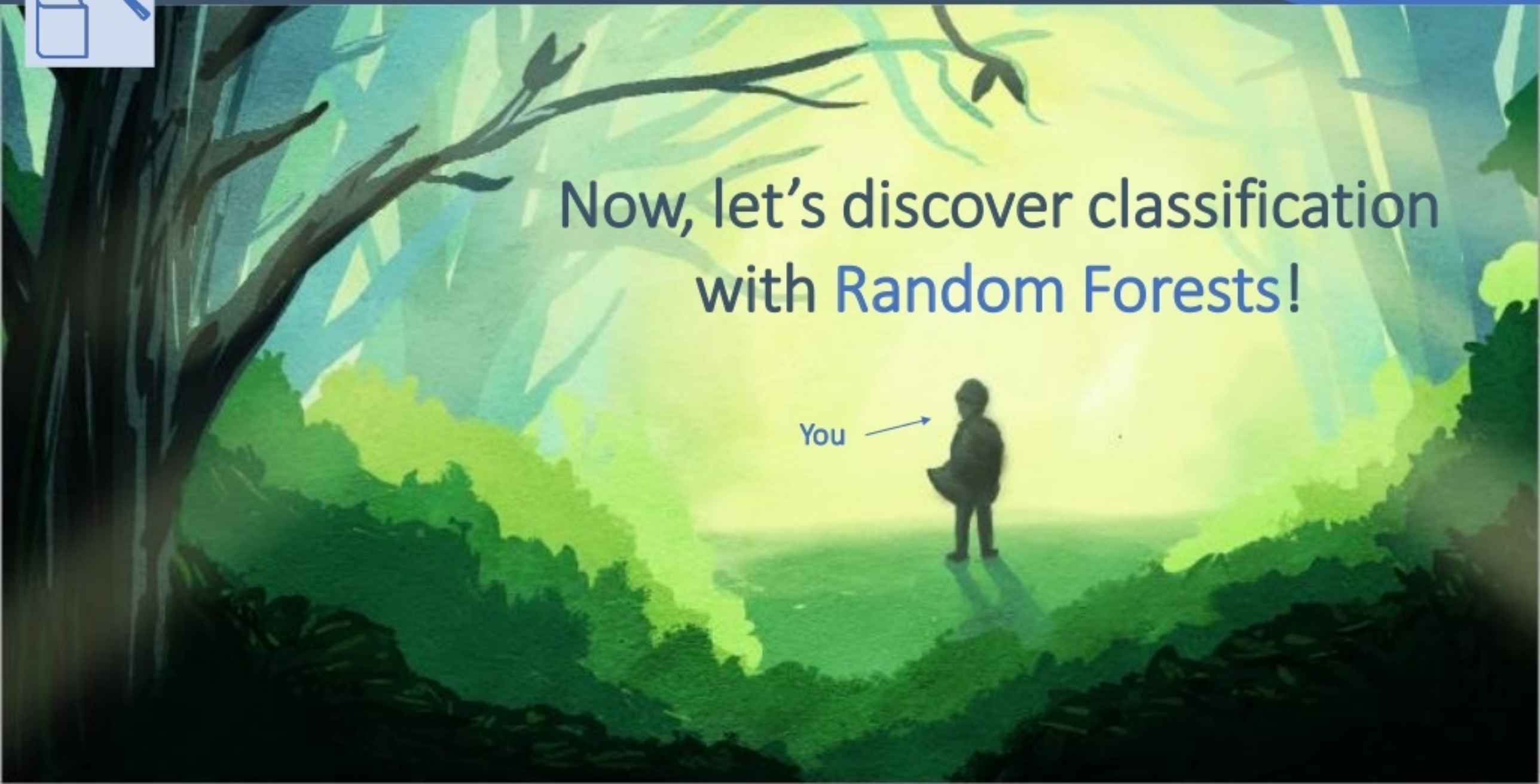
2D representation



Each point on the green line is a unique value $J(\beta_0,\beta_1)$ for multiple couples of (β_0,β_1) values

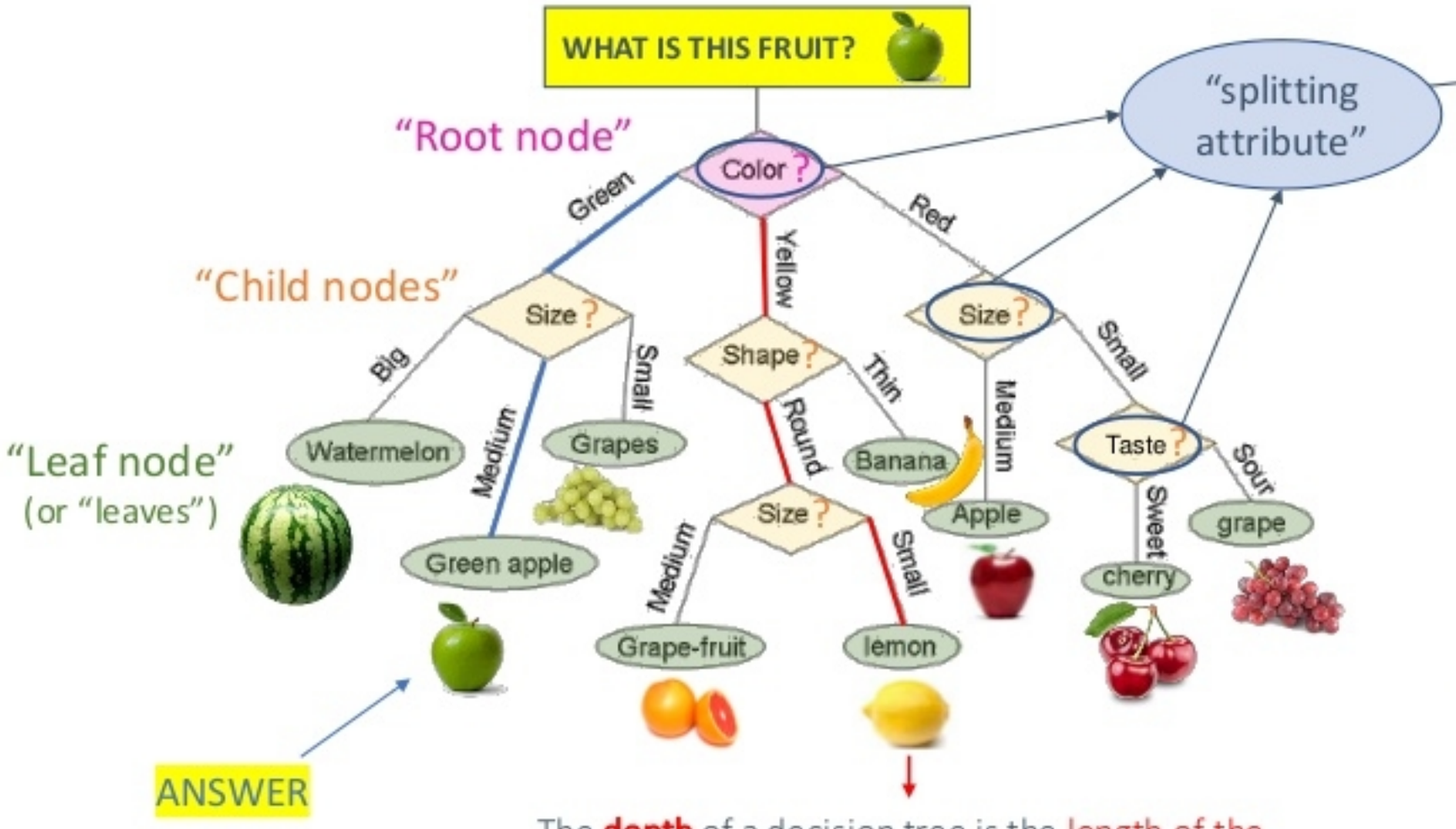
The length of the steps at which the gradient is descending is a parameter called "learning rate"

Supervised vs. unsupervised learning



c Classification with Random Forests

Let's talk first about Decision tree algorithms



The **depth** of a decision tree is the length of the longest path from a root to a leaf (here, depth = 3)

How is the splitting attribute is selected at each node?

NB: "attribute" = "feature"

➡ The selected attribute is the one maximizing the **purity** (i.e. the % of a unique class) in each output sub-groups after the split.
For example, "Shape" as root node would be bad as most fruits are rather round.

➡ Making the decision can rely on two different indicators:
- "Gini impurity", to minimize or
- "Information gain", to maximize

c Classification with Random Forests

What are the problems with Decision tree algorithms?

1 Imagine you add in your dataset some green lemons and not yet ripe bananas and tomatoes. Now we have many green fruits to classify!

"Color" is not the attribute with the most information gain anymore, so it will not be the splitting attribute in the root node, the structure of the tree is going to change drastically.
Decision trees are very sensitive to changes in training examples.

2 If there is a non-informative features that happens to provide good information gain, coincidentally or because it is correlated with an informative feature, decision trees will wrongly use it as a splitting attribute.
Decision trees are very sensitive to changes in the features.

In a nutshell, decision trees are very sensitive to changes in the data and won't generalize well.
They are considered as weak learners.

c Classification with Random Forests

General idea

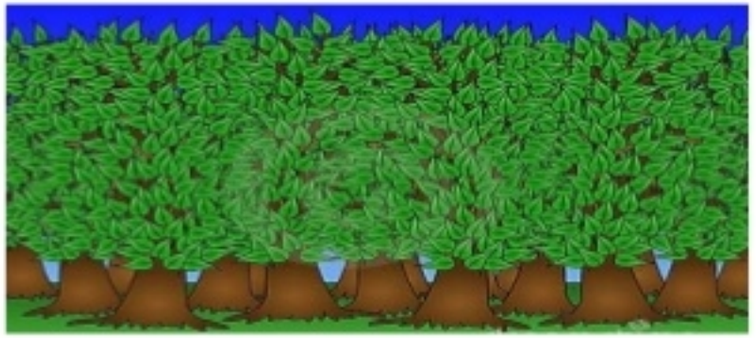
Random Forests algorithm is using randomness at 2 levels, that is *i)* in data selection and *ii)* in attribute selection, and relies on the Law of Large Numbers to discard error in the data.

Let's see how it works!

c Classification with Random Forests

TRAIN THE MODEL

1. Take different random subsets of your data *(method known as bootstrapping)*



RANDOM SUBSET #1 | RANDOM SUBSET #2 | RANDOM SUBSET #3 | ... | RANDOM SUBSET #N

2. Build different decision trees with each of them
When building the trees, splitting attributes are chosen among a random subset of features, just like for the data

RUN THE MODEL

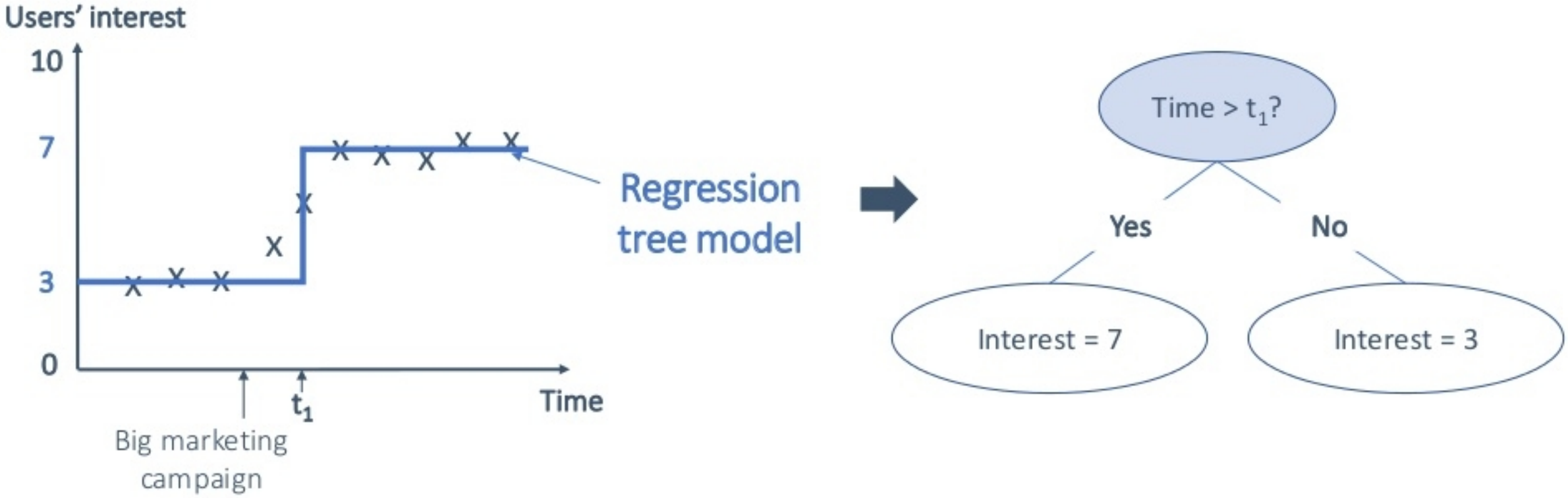
c Classification with Random Forests

Our Random Forests model is
99.33214% sure Magritte was wrong!

c Classification with Random Forests

Note that there Decision Trees and Random Forests can also solve regression problems.

Example of the evolution of users' interest in an app over time

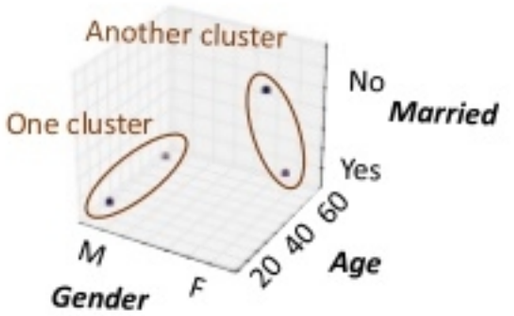


Unsupervised learning algorithms are used for clustering methods.

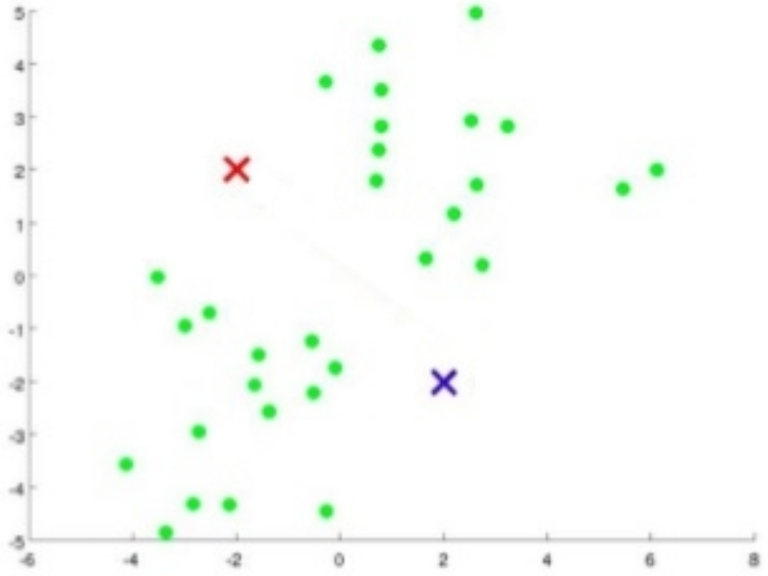
CLUSTERING
(= unlabelled classification)

When we cluster examples with no label

Name	Gender	Age	Location	Married
John	M	46	New-York	Yes
Sarah	F	42	San Francisco	No
Michael	M	18	Los Angeles	Yes
Danielle	F	54	Atlanta	Yes

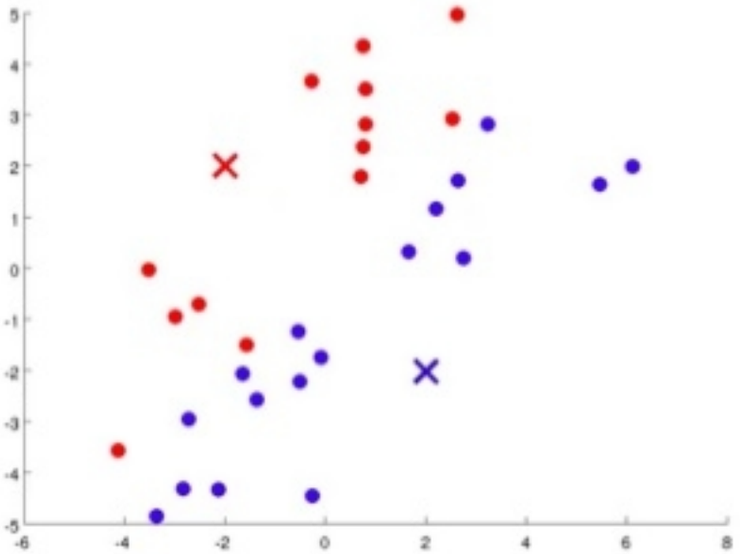


d Clustering with K-means



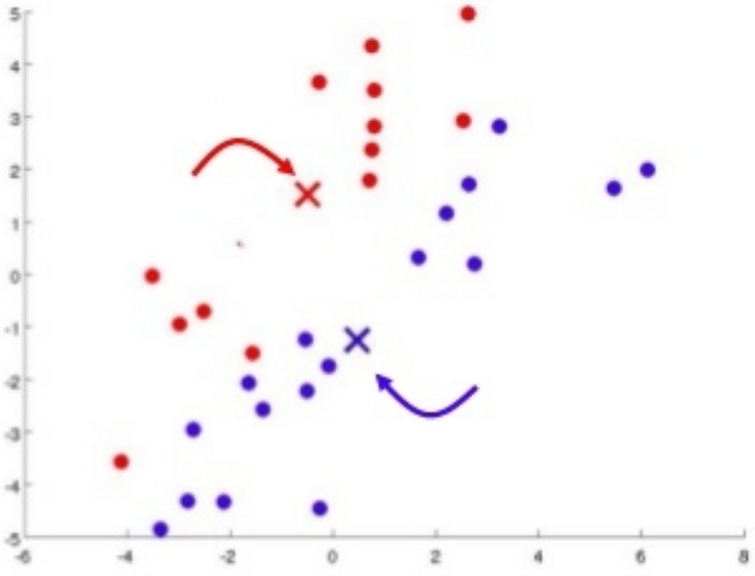
Step 1

All data points are unlabeled.
We randomly initiate two points called "cluster centroids".



Step 2

Data points are labeled according to which centroid they are the closest from.

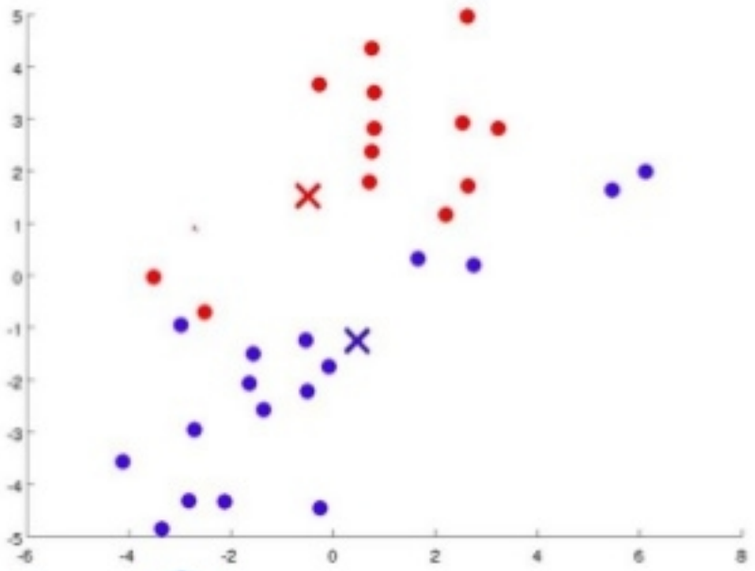


Step 3

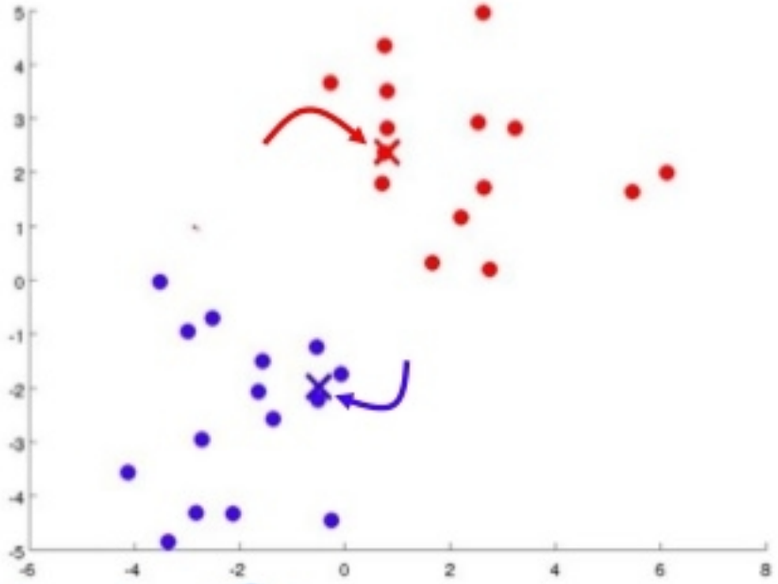
Each centroid is moved to the center of the data points that were labeled in step 2.

d Clustering with K-means

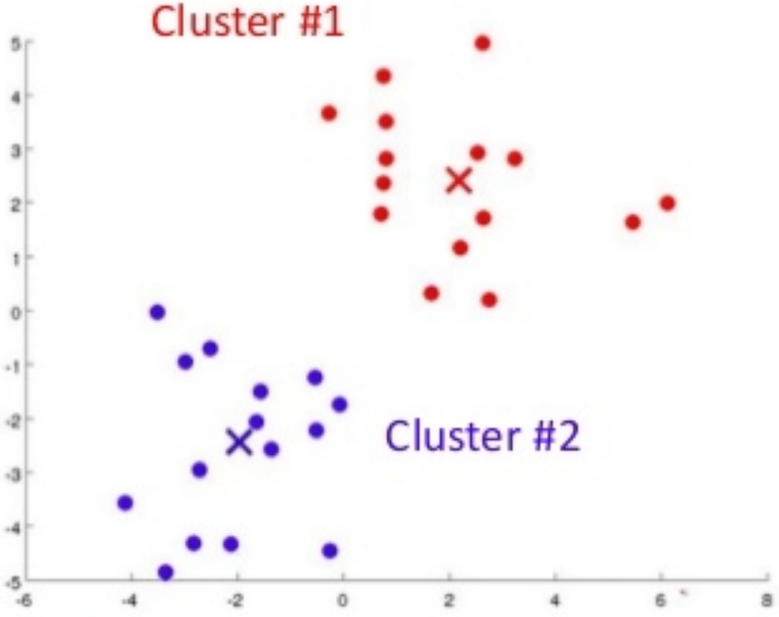
Steps 2 and 3 are repeated....



Step 2



Step 3



End

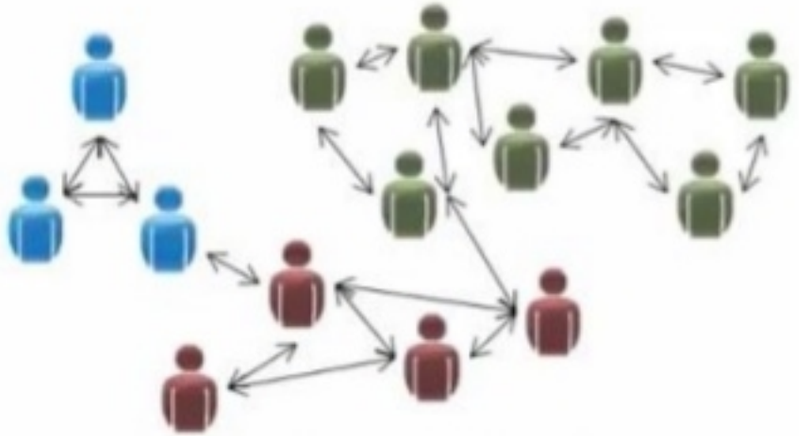
...until convergence.

(i.e. no data is relabeled after centroids have been recentered)

d Clustering with K-means

Examples of applications of clustering

Market segmentation



Social network analysis

Astronomical data analysis

These algorithms can easily be implemented in scikit learn

LINEAR REGRESSION

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
model = regr.fit(X, y)
```

RANDOM FORESTS

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
model = clf.fit(X, Y)
```

K-MEANS

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
model = kmeans.fit(X)
```

Desired number of clusters

Parametric vs. Nonparametric

Parametric vs. nonparametric algorithms

In a Linear regression, the vector $\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_m \end{pmatrix}$ contains the parameters of the **model** that are fitted to your data by the Linear regression algorithm.

Because it assumes a pre-defined form for the function modelling the data, with a set of parameters of fixed size, the linear regression is said to be a parametric algorithm.

Parametric vs. nonparametric algorithms

Algorithms that do not make strong assumptions about the form of the mapping function are **nonparametric algorithms**.

By not making assumptions, they are free to learn any functional form (with an unknown number of parameters) from the training data.

A decision tree is, for instance, a nonparametric algorithm.

Parametric vs. nonparametric algorithms

	Pros	Cons
Parametric algorithms	**Simpler** Easier to understand and to interpret **Faster** Very fast to fit your data **Less data** Require "few" data to yield good perf.	**Limited complexity** Because of the specified form, parametric algorithms are more suited for "simple" problems where you can guess the structure in the data
Nonparametric algorithms	**Flexibility** Can fit a large number of functional forms, which doesn't need to be assumed **Performance** Performance will likely be higher than parametric algorithms as soon as data structures get complex	**Slower** Computations will be significantly longer **More data** Require large amount of data to learn **Overfitting** We'll see in a bit what this is, but it affects model performance

What are the best algorithms?

WHAT ARE THE BEST ALGORITHMS?

We'll let the cat out of the bag now.
There is no such thing as "best algorithms".

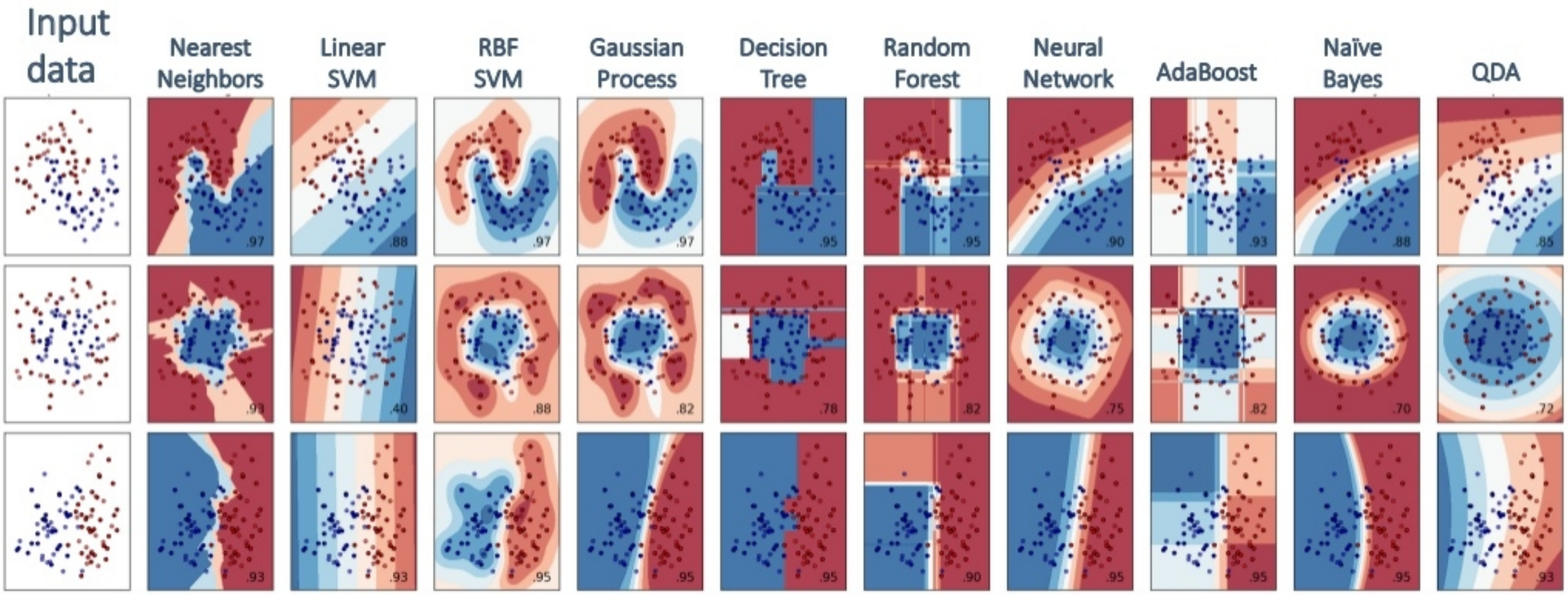
Which is why choosing the right algorithm is one tricky part in machine learning.

WHAT ARE THE BEST ALGORITHMS?

source

Look at the different models obtained from classification algorithms trained on the same data.

*(color shades represent the "**decision function**" guessed by the algorithm for each class)*



WHAT ARE THE BEST ALGORITHMS?

Moreover, every algorithm has parameters called **hyperparameters**.

They have default values in scikit learn, but how your model performs will also depend on your ability to **fine-tune** them.

WHAT ARE THE BEST ALGORITHMS?

Hyperparameters are parameters of the algorithm, they are not to be confused with the parameters of the model.

Examples of hyperparameters, as implemented in scikit learn

	Hyperparameters	Model parameters
Linear Regression	**"fit_intercept"** Whether to include or not the term β_0 in the functional form to fit	β
Random Forests	**"n_estimators"** Number of trees in the forest **"criterion"** Indicator to use to determine the splitting attribute at each node when building the trees in the forest ("gini" or "information gain")	Undefined *(nonparametric model)*
K-Means	**"init"** Initialization method for the centroids	Undefined *(nonparametric model)*

WHAT ARE THE BEST ALGORITHMS?

These algorithms and the possible values for their hyperparameters are all equivalent in absolute.

They will only perform differently because of the specificities of your data.

WHAT ARE THE BEST ALGORITHMS?

This is what the
"No Free Lunch" theorem states:

"When averaged across all possible situations, every algorithm performs equally well."

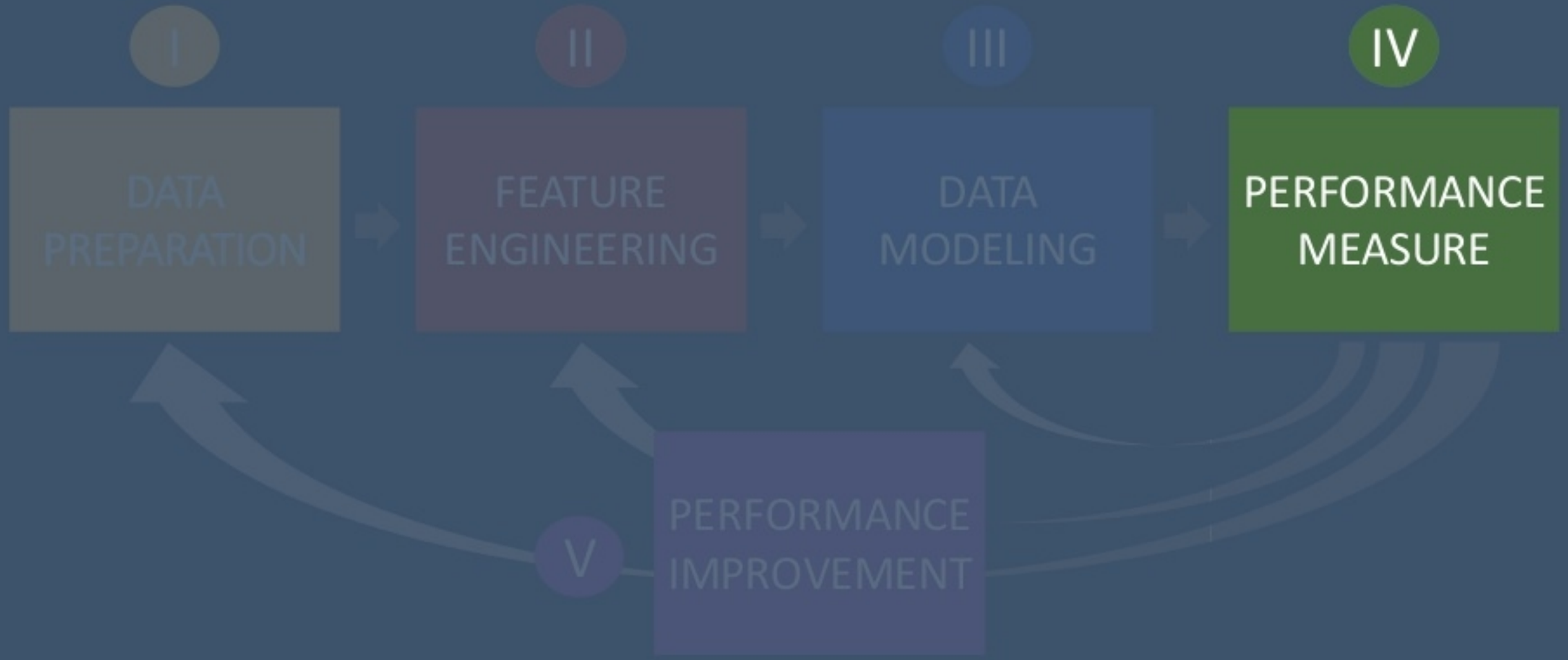
Wolpert and Macready, 1997

Learn more with an illustration of No Free Lunch theorem on K-means algorithm

Building a performing ML model is all about:

- making the right assumptions about your data
- choosing the right learning algorithm for these assumptions

But how do I know if my model is performing?


I
DATA PREPARATION
II
FEATURE ENGINEERING
III
DATA MODELING
IV
PERFORMANCE MEASURE
V
PERFORMANCE IMPROVEMENT

Assessing your model performance is a 2-step process

1. Use your model to predict the labels in your own dataset

```
Y_predicted = model.predict(X_test)
```

2. Use some indicator to compare the predicted values with the real values

```
comparison = some_indicator(Y, Y_predicted)
```

Predicting your dataset labels

a. Training set and test set

b. Cross-validation

Choosing the right performance indicator

a. Regression

b. Classification

a Training set and test set

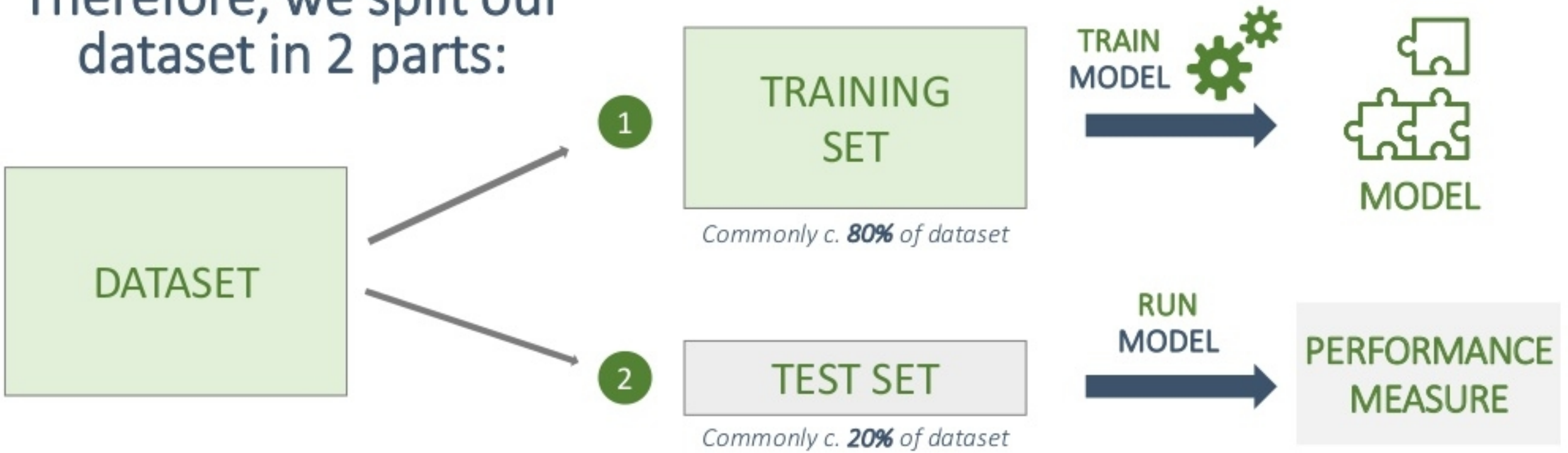
You never train your model and test its performance on the same dataset.

It's a bit like sitting for an exam where you already know the answer.

That would deeply bias the performance measure.

a Training set and test set

Therefore, we split our dataset in 2 parts:

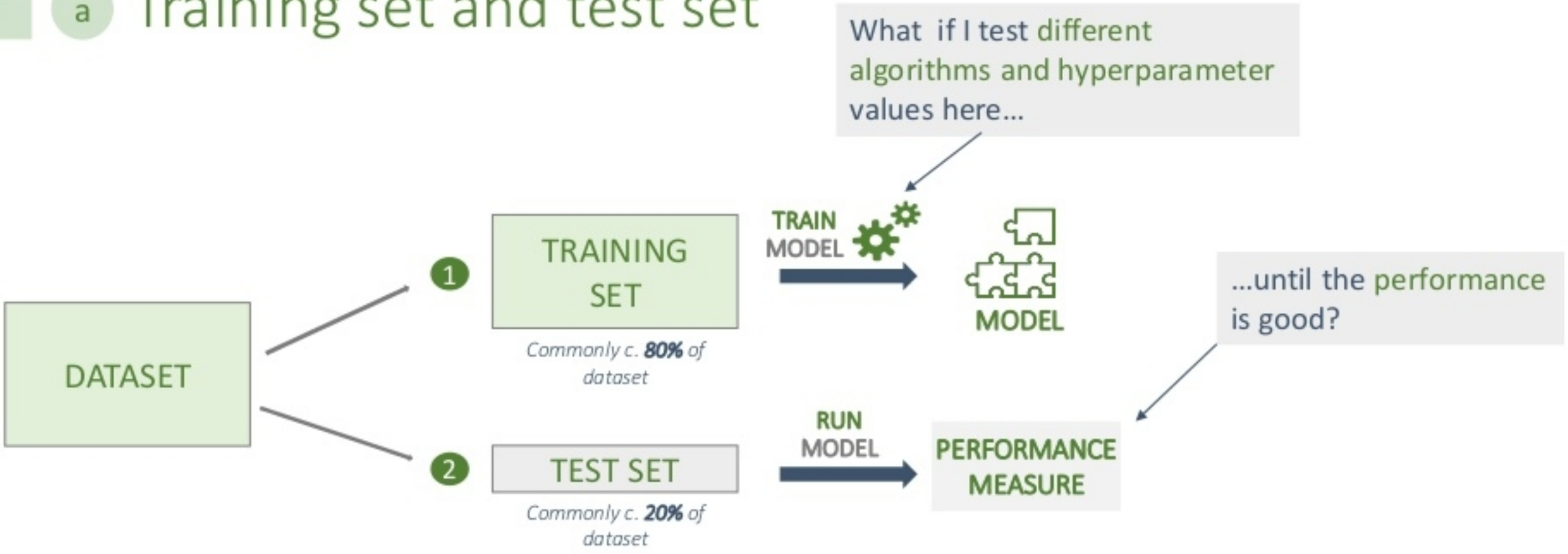


A test set enables to test our model on unseen data.

scikit learn

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

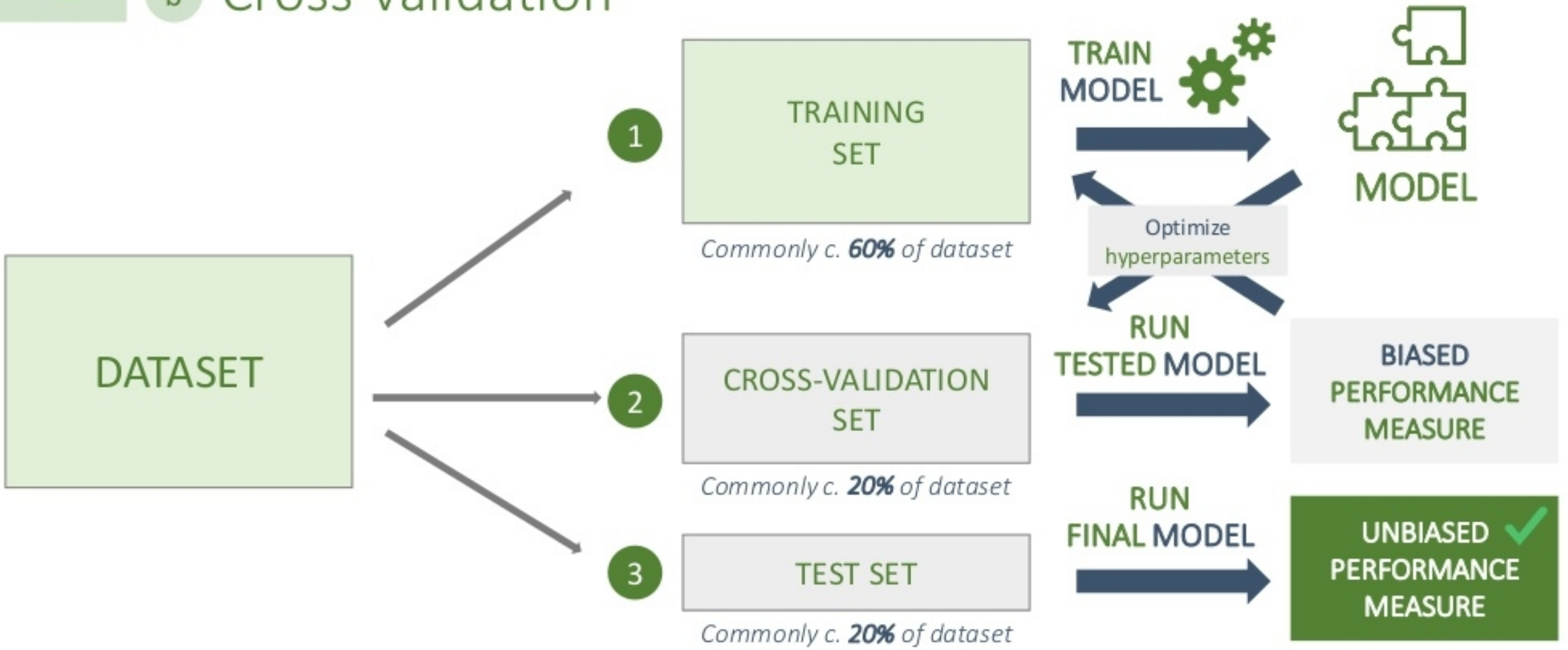
a Training set and test set



➡ Such a method is often used to quickly test different algorithms at the beginning or to fine-tune hyperparameters.

However, the performance measure will be biased again, because it highly depends on the data in the test set, which is why you will have to use cross-validation.

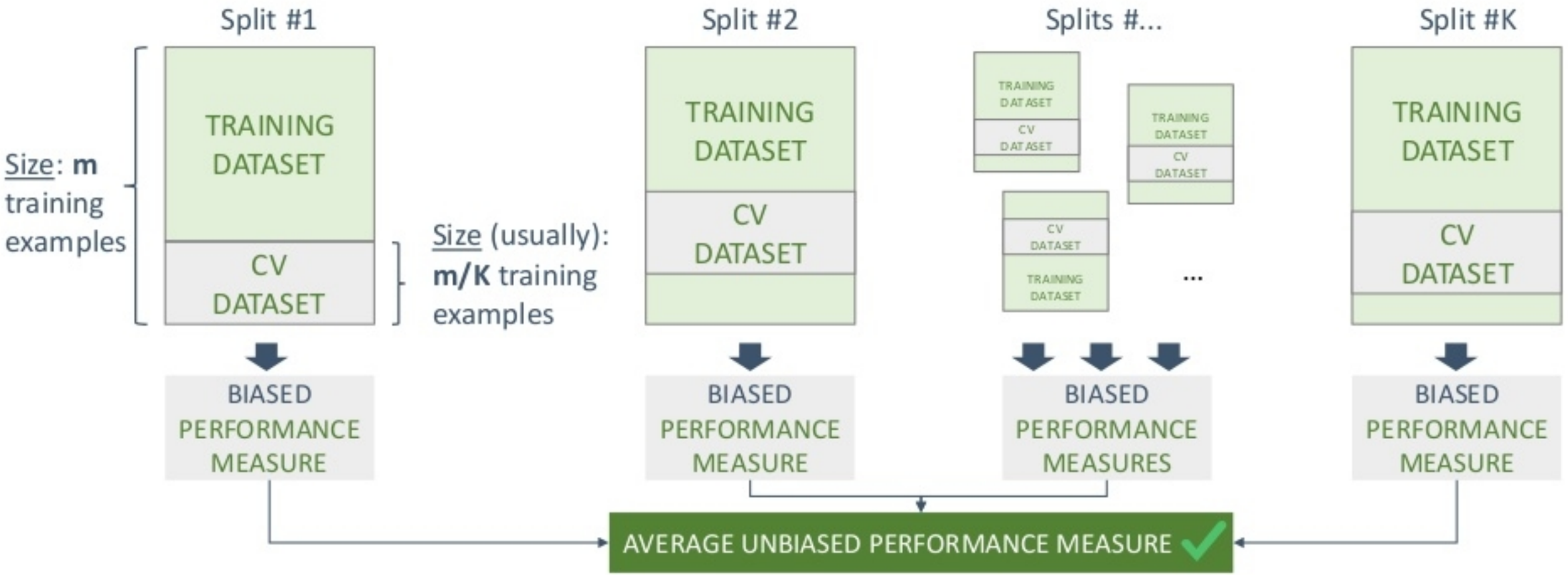
b Cross-validation



PROBLEM: You will **lose c. 20% of the data** to train your algorithm.
If you don't have lots of data, you might prefer KFold cross-validation

b Cross-validation

KFold cross-validation consists in repeating the training / CV random splitting process K times to come up with an average performance measure.



b Cross-validation

There are several ways to use KFold cross-validation in scikit learn

Learn more

1. **Simple performance measure with K=10**

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, scoring=indicator, cv=10)
```

2. **Fine-tuning hyperparameters with GridSearchCV**

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
regr = LinearRegression()
parameters = {'fit_intercept': [True, False]}
regr = GridSearchCV(regr, parameters, cv=10)
regr.fit(X, y)
```

KFold cross-validation is quickly very computationally expensive.

3. **CV is internally implemented in some algorithms and computations are optimized, e.g.** `from sklearn.linear_model import RidgeCV`

Now that I know the method to rigorously measure the performance, which indicator will I use?

a Regression

y_i is the **true label** for the *i-th* example in the test set

$\hat{y}_i$ is the **predicted label** for the *i-th* example in the test set

$\bar{y}$ is the **average** of the label values in the test set

Examples of two commonly used indicators

	Mean squared error	Coefficient of determination (R^2)
Formula	$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$	$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \bar{y})^2}$
Pros	Easy to understand	Absolute value Very roughly, a model with $R^2 > 0.6$ is getting good (1 being the best), $R^2 < 0.6$ is not so good
Cons	Relative value You need the scale of your labels to interpret MSE	Difficult to explain

b Classification

$$\text{Accuracy} = \frac{\text{Number of correctly predicted labels}}{\text{Total number of labels in test set}}$$

Accuracy is **very easy to understand but often too simple** to correctly interpret the performance of your model

Confusion matrix

$$\text{Precision} = \frac{TP}{TP + \mathbf{FP}}$$

Precision is a % expressing the **precision with which the positive values where recalled** by your model.

		PREDICTED LABELS: POSITIVE	PREDICTED LABELS: NEGATIVE
ACTUAL LABELS: POSITIVE		✓ TRUE POSITIVE (TP)	✗ FALSE NEGATIVE (FN)
ACTUAL LABELS: NEGATIVE		✗ FALSE POSITIVE (FP)	✓ TRUE NEGATIVE (TN)

$$\text{Recall} = \frac{TP}{TP + \mathbf{FN}}$$

Recall is a % expressing the **capacity of your model to recall positive values**.

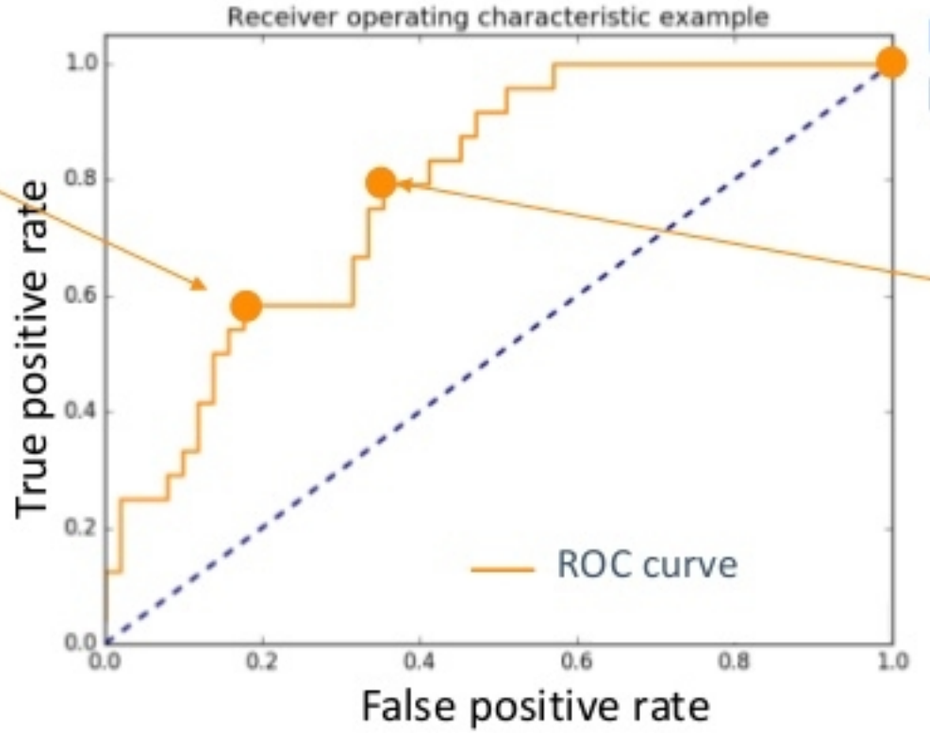
b Classification

ROC curve

There is always a trade-off in function of whether you want to prefer precision or recall. You can visualize how the TP and FP rates evolve according to different discrimination thresholds of your model with the **ROC curve**.

Precision > Recall

If you are running a marketing campaign but don't have too much money, you might want to focus on a smaller target (low recall) where your probability to convert is high (high precision)



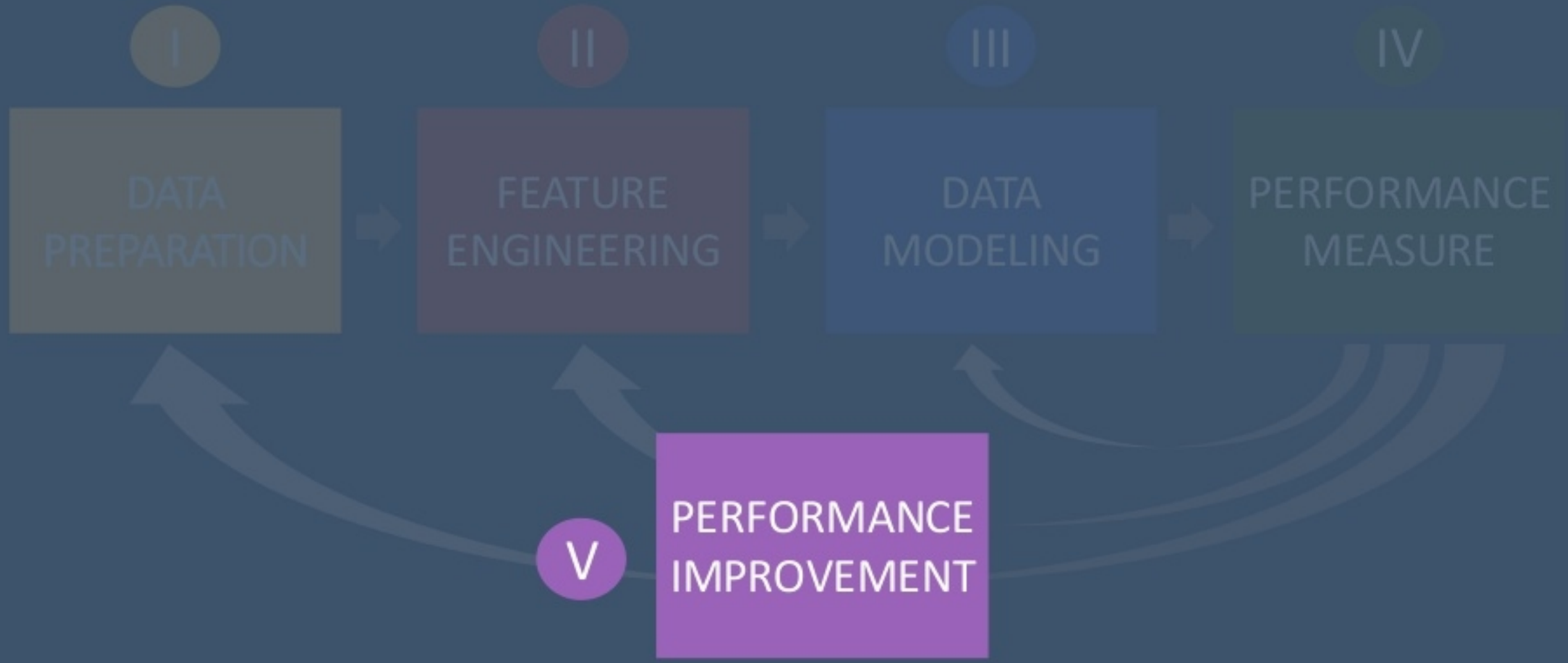
Recall = 100%
Precision = % of positive examples in your dataset

Recall > Precision

If you are running a marketing campaign and have lots of budget, you will rather focus on a large target where your probability to convert is lower (low precision), but on a greater number of people (high recall)

Create a dirty but complete model as quick as possible to iterate on it afterwards.

This is the right way to go!


I
II
III
IV
DATA PREPARATION
FEATURE ENGINEERING
DATA MODELING
PERFORMANCE MEASURE
V
PERFORMANCE IMPROVEMENT

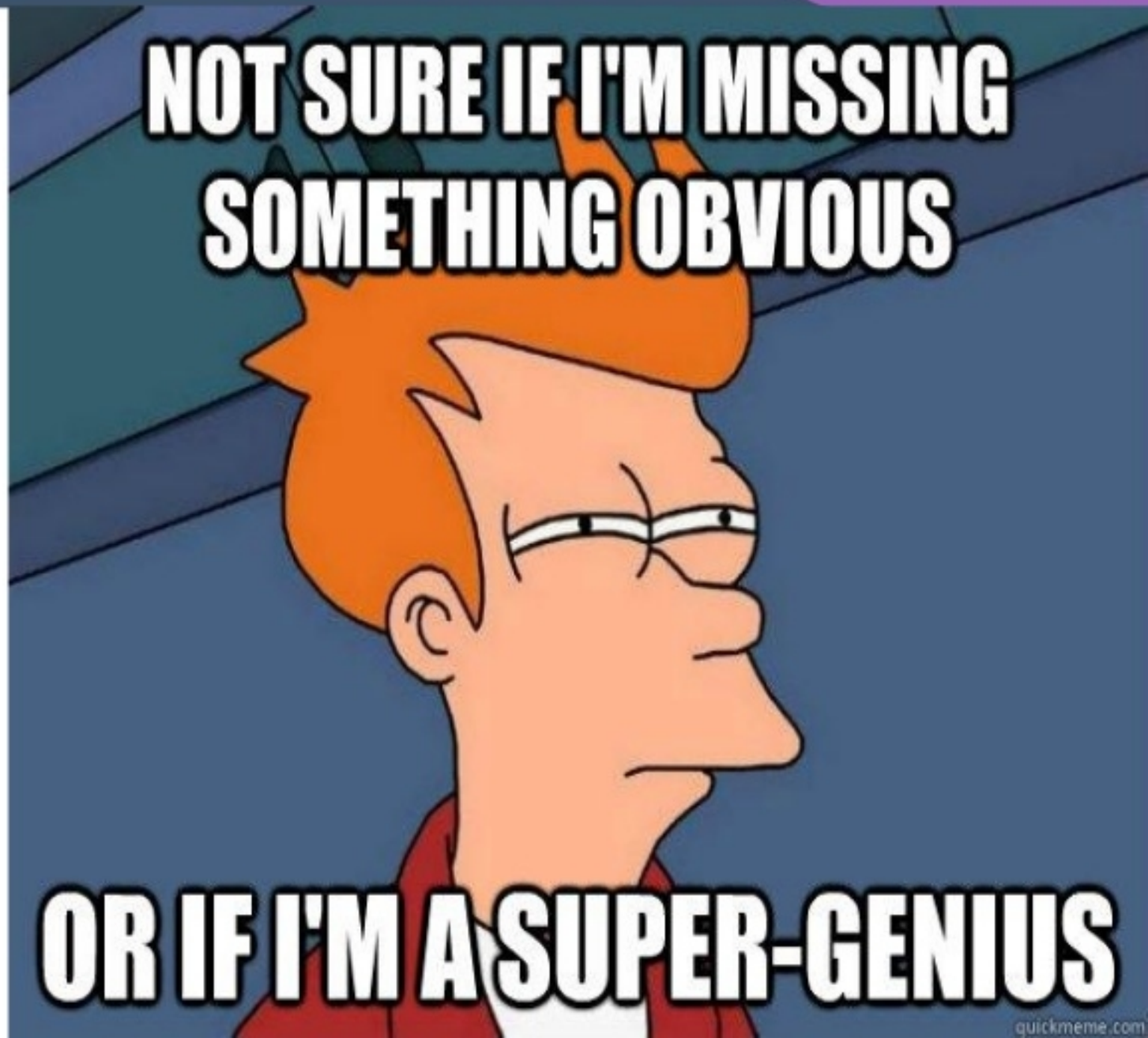
Reasons for underperformance

a. Underfitting

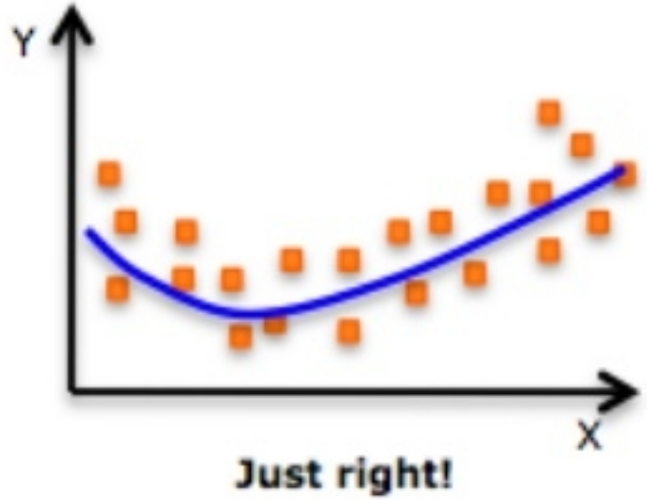
b. Overfitting

Solutions to increase performance

What are the reasons why your model is not performing well?



A performing model will fit the data in a way that it generalizes well to new inputs.



It should reproduce the underlying data structure but leave aside random noise in the data.

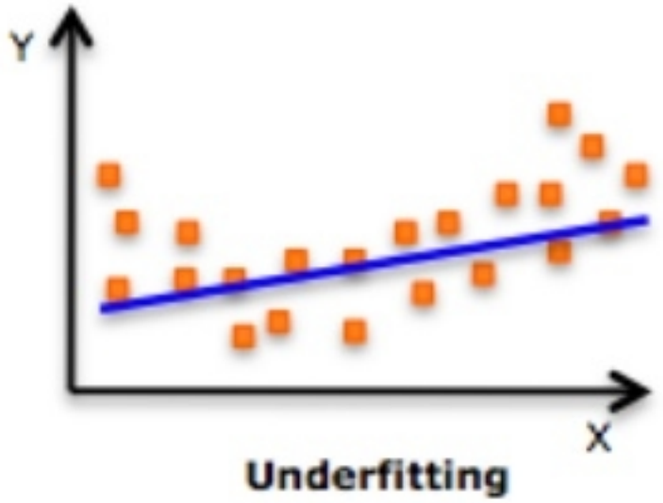
REASONS FOR UNDERPERFORMANCE

There are **two reasons** why a model would not generalize and thus not perform correctly:

a. Underfitting

b. Overfitting

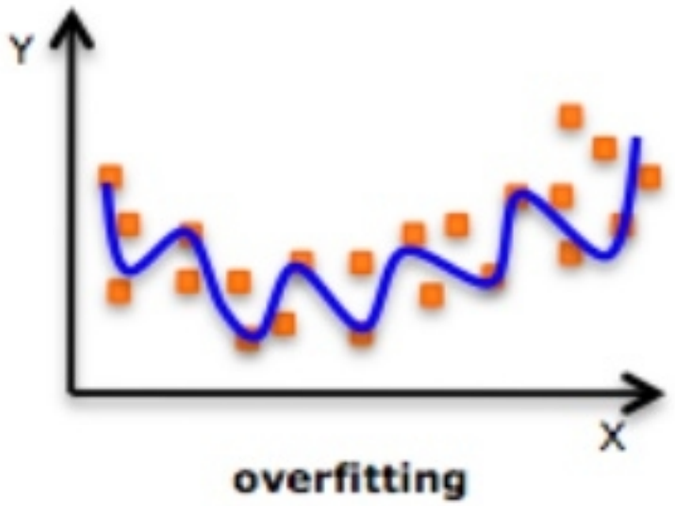
a Underfitting



Underfitting happens when your model is too simple to reproduce the underlying data structure.

When underfitting, a model is said to have high bias.

b Overfitting



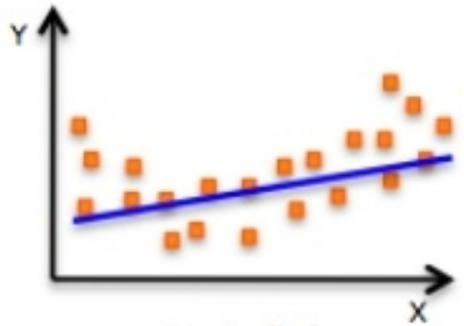
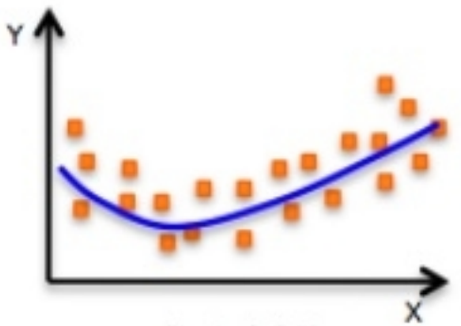
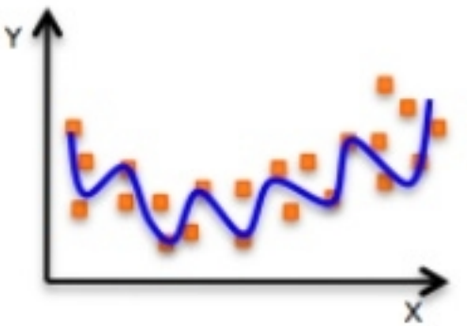
Overfitting happens when your model is too complex to reproduce the underlying data structure.

It captures the random noise in the data, whereas it shouldn't.

When overfitting, a model is said to have high variance.

IN A NUTSHELL

You want to select a model at the sweet spot between underfitting and overfitting. This is not easy!

Performance on...	 Underfitting	 Just right!	 overfitting
Training set	Bad	Good	Very good
Test set	Bad	Good	Bad

Overfitting can easily be spotted with this performance difference on training and test sets !

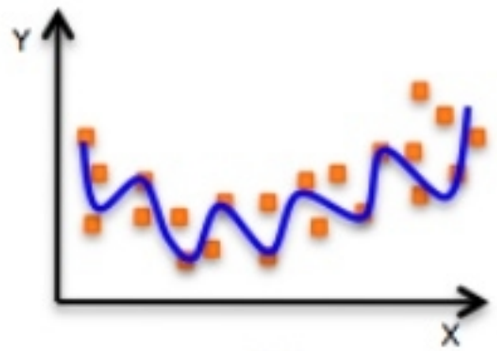
Ok, I got the point…

So, how do I address these overfitting and underfitting issues?

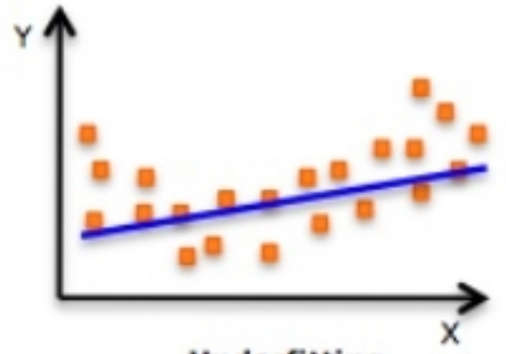
SOLUTIONS TO INCREASE PERFORMANCE

ISSUE OF THE

MODEL



overfitting



Underfitting

POTENTIAL SOLUTIONS ON …

DATA

- A — More training examples
- B — a Less features; b More features

ALGORITHMS

- C — Simpler algorithms; More complex algorithms
- D — Regularization
- E — Bagging; F — Boosting

"ENSEMBLE METHODS"

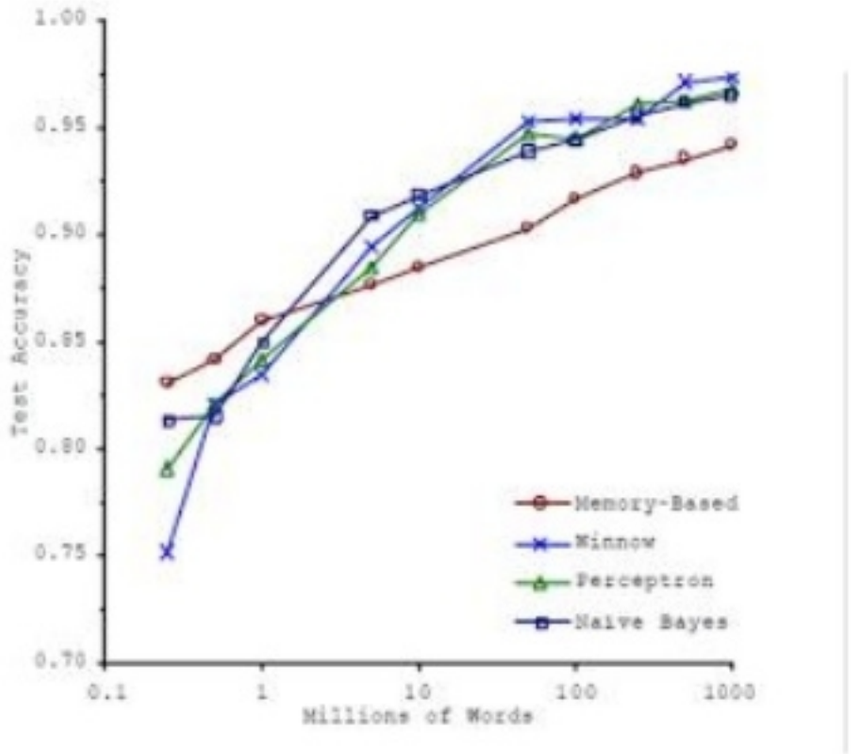
SOLUTIONS TO INCREASE PERFORMANCE

Let’s discuss the mentioned solutions one by one.

(Final stretch, I promise !)

A More training examples

The study below shows how different algorithms perform similarly for a given problem as the amount of training examples increases.



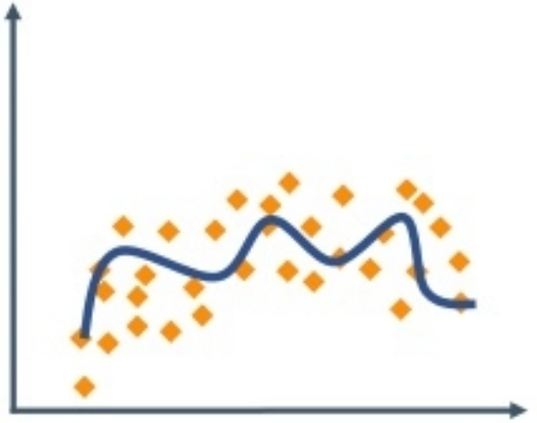
From "Scaling to Very Very Large Corpora for Natural Language Disambiguation" by Microsoft researchers Banko and Brill, 2001

A More training examples

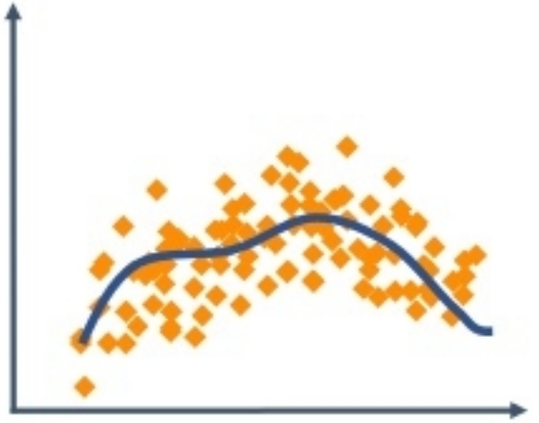
The more training examples there are, the more complex it is for an algorithm to fit the noise in the data.

Therefore, the fitted model will be less sensitive to noise and will better generalize.

A few samples



A bit more samples



A lot of samples

B a Less features

Some features might contain more noise than informative data for your model.

This is especially the case when the features are non-informative or correlated with other features.

Remove them and your model will not take this noise into account anymore.

B b More features

If your model is underfitting, it might be because you did not give it enough informative features.

We are talking about science, not divination!

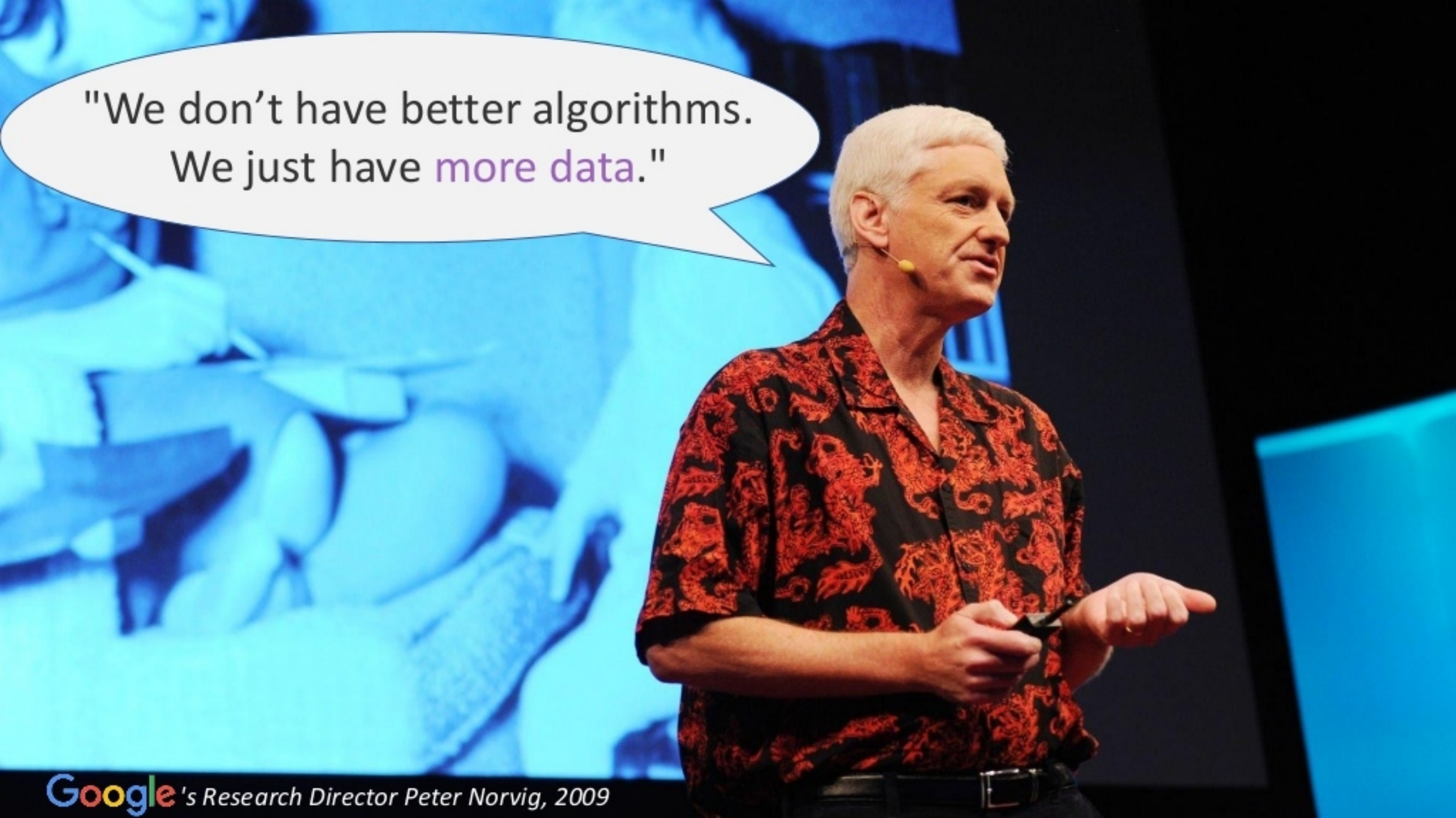
B In a nutshell: more data!

Data is key because it can help you both:

- Reduce variance (overfitting) with more training examples
- Reduce bias (underfitting) with more features

However, you must know how to make good use of these data with good feature engineering.

More details



Google's Research Director Peter Norvig, 2009

B In a nutshell: more data!

If you're convinced you need more data,
Turk might help you. Check it out !

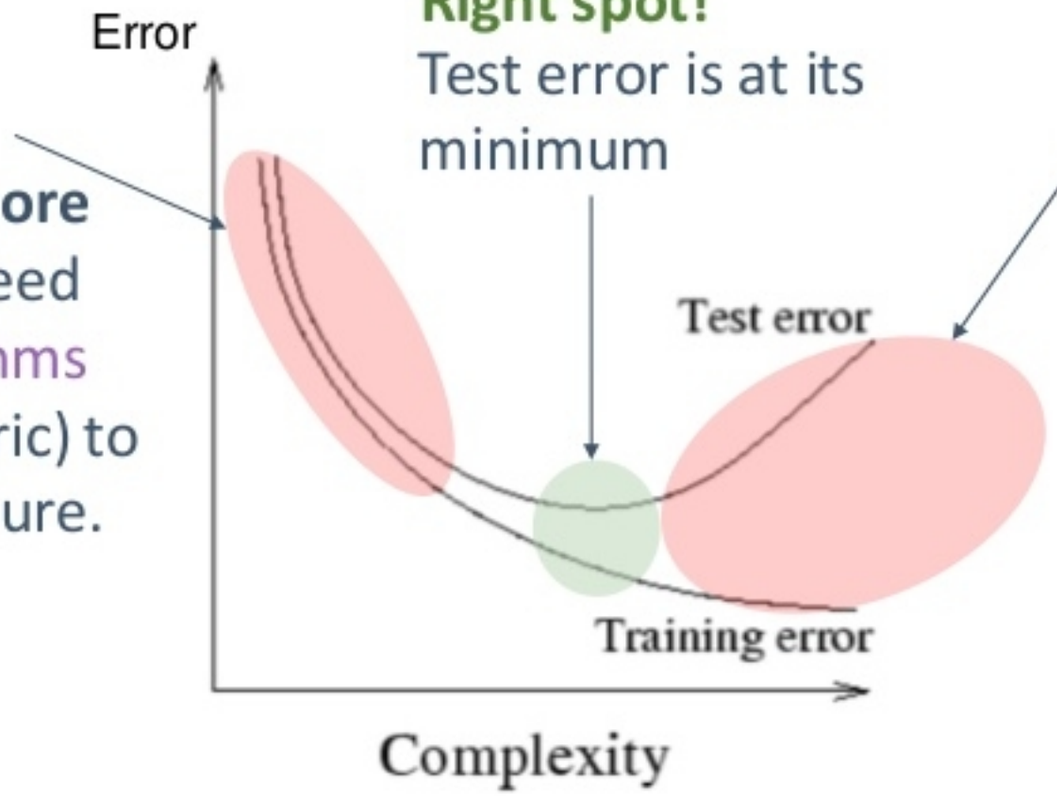
amazon mechanical turk
beta

C Algorithms complexity

Below is how a model error will theoretically evolve as its complexity increases (i.e. more complex algorithms, more features).

Underfitting area

In addition/place of **more** features, you might need more complex algorithms (nonlinear, nonparametric) to model your data structure.



Right spot!

Test error is at its minimum

Overfitting area

In addition/place of **less** features, you might need simpler algorithms. Certain algorithms, such as deep neural networks, are very powerful and **easily tend to overfit** if you don't have millions of rows of data.

D Regularization

Regularization aims at reducing overfitting by adding a complexity term to the cost function.

Explanation

Let's see how it works for Linear Regression.
The **loss function** will be:

Regularization term

$$J(\beta) = \sum_{i=0}^{m} (Y_i - X_i\beta)^2 + \alpha \sum_{j=0}^{n} \beta_j^2$$

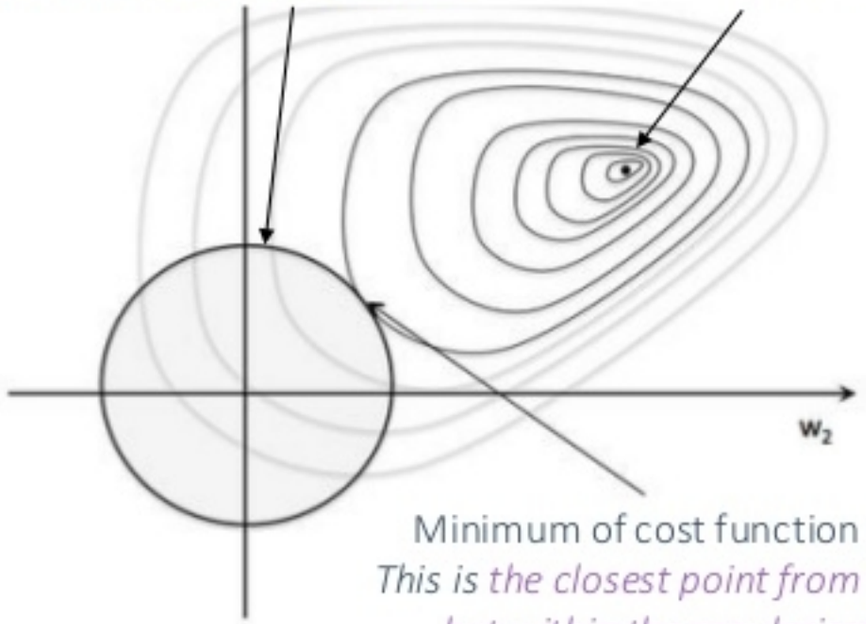
α = "Complexity" / "Penalty" / "Regularization" parameter

Because of the regularization term, the algorithm will find smaller β values when minimizing the cost function, resulting in **lower variance**.

Illustration

Limits set by the regularization term.
The greater α, the smaller more restricted these limits will be.

Minimum of cost function when α = 0 (no regularization)



Minimum of cost function when $\alpha > 0$.
This is the closest point from the minimum, but within the regularization limits.

D Regularization

This regularized linear regression is called a Ridge regression, and can be found in scikit learn

There even is an implementation with a fast built-in cross-validation that enables you to quickly optimize the α parameter.

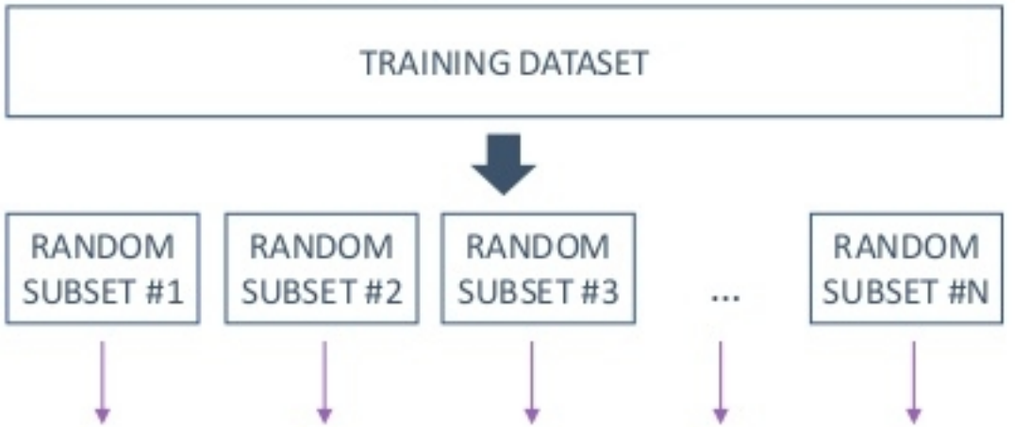
```
from sklearn.linear_model import RidgeCV
reg = linear_model.RidgeCV(alphas=[0.1, 1.0, 10.0])
reg.fit(X,y)
reg.alpha_ #Returns the best alpha value between [0.1, 1.0, 10.0]
```

If α is too large, your model will underfit. It's a **constant trade-off**!

E Bagging

Bagging

= Take N different random subsets of the training dataset



1 Bootstrap

\+

2 Aggregating

Note

You can also apply bagging on your set of features.

Did you notice ?

Random Forest is simply bagging applied on decision tree classifiers (week learners).

F Boosting

Bagging

Aggregating equally the results of weak learners built independently on random samples to create a strong learner.

Strong learner

Weak learner weight, = 1 for all

$$D(x) = \sum_j \alpha_j d_j(x)$$

Weak learners (usually decision trees) built **independently**

Boosting

Combining differently the results of weak learners built sequentially on the whole dataset to create a strong learner.

Strong learner

Different weight for each weak learner d_j

$$D(x) = \sum_j \alpha_j d_j(x)$$

Weak learners (usually decision trees) built **sequentially**

F Boosting

Illustration of Gradient Boosted Trees (GBT) algorithm.

GBT = Boosting + Gradient Descent + Trees

a Boosting – we start with **constant regression tree d_0** and **model error term at each iteration**

Initialization: $Y_{true} = d_0(x) + \varepsilon_0$ with $d_0(x) = 0$ and ε_0 the error term ("residual")

Boosting:

$$\varepsilon_0 = \alpha_1 d_1(x) + \varepsilon_1$$
$$\varepsilon_1 = \alpha_2 d_2(x) + \varepsilon_2$$

We model the residual with a **regression tree d_j** (weak learner), slowly decreasing total error amount iteration after iteration

...

$$\varepsilon_{t-1} = \alpha_t d_t(x) + \varepsilon_t$$

We stop when $\varepsilon_t \sim \varepsilon_{t-1}$ ➡

$$Y_{pred} = \sum_{j=1}^{t} \alpha_j d_j(x)$$

and **ε_t is the remaining error** between our prediction Y_{pred} and the true label Y_{true}

But how do we know which regression tree d_j to add at each iteration?

F Boosting

b Gradient Descent – find optimal regression tree d_j

At the *j-th iteration*, the steps are as follow:

1. For i = 0, …, m, **compute residual** $\varepsilon_{j,i} = Y_{true,i} - \sum_{k=0}^{j-1} \alpha_k d_k(X_i)$ — *Model obtained from previous iteration j-1*

2. With Gradient Descent, **find d_j minimizing cost function** $\sum_{i=0}^{m} (d_j(X_i) - \varepsilon_{j,i})^2 + \text{regularization term}$

   NB: d_j belongs to the **space of functions** containing all regression trees.

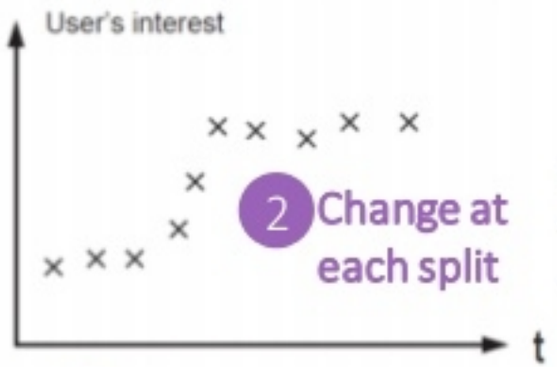
   > Because boosted trees do gradient descent in a space of functions, they are very good when the structure of the data is **unknown**.

3. **Find optimal α_j** to minimize total residual $\varepsilon_{j+1} = Y_{true} - \sum_{k=0}^{j} \alpha_k d_k(X_i)$

F Boosting

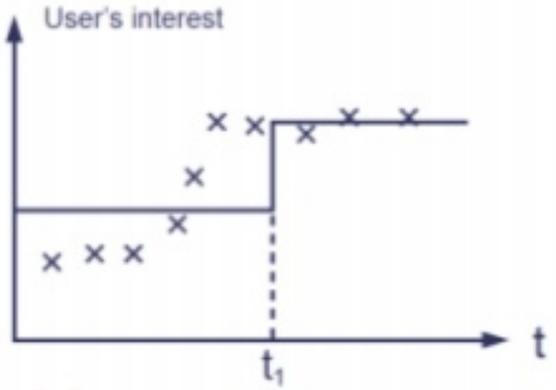
C Regression Tree – Gradient descent will find the optimal regression tree d_j

Two kinds of parameters to determine:

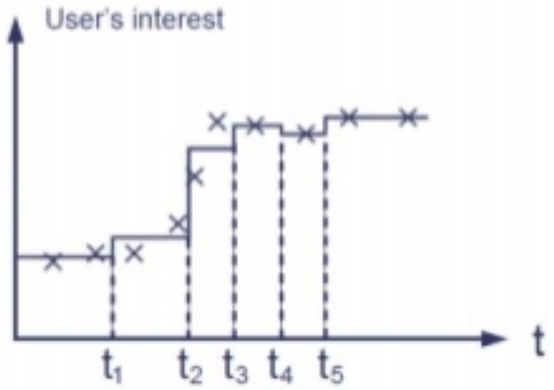


2 Change at each split

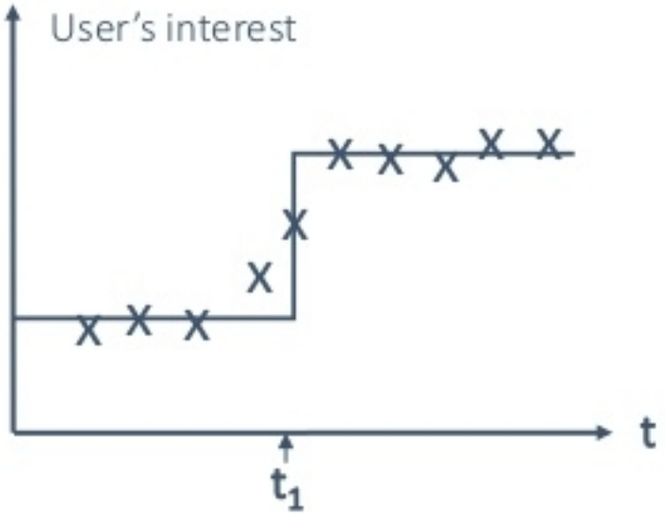
1 Splitting positions



✗ Underfit



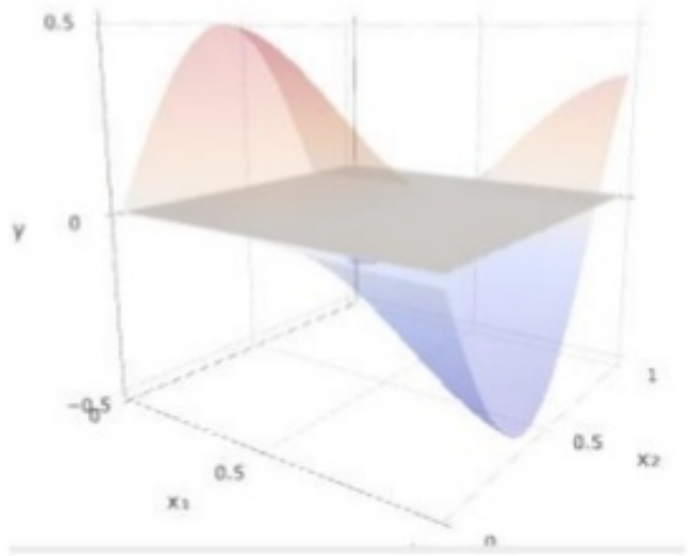
✗ Overfit



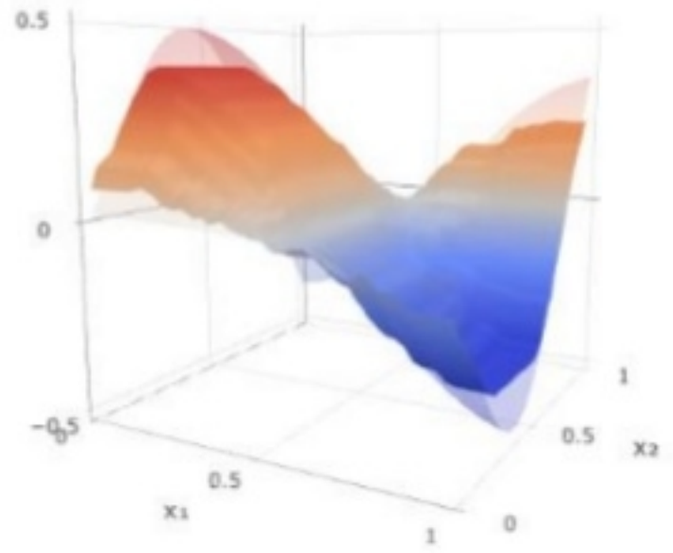
✓ Good

F Boosting

You can visualize the output of Gradient Boosting Trees below:



3D function to modelize



GBT output combining 100 decision trees with depth = 3

More visualization

F Boosting

Note that even for classification tasks, Gradient Boosted Trees will be using regression trees.

The algorithm will compute continuous values between 0 and 1 that are probabilities of belonging to a certain class.

F Boosting

Tree ensemble methods (i.e. bagging/boosting used with decision trees) are very popular algorithms in Machine Learning.
They often enable a good performance with little effort.

The most popular implementation of Gradient Boosted trees is the module XGBoost

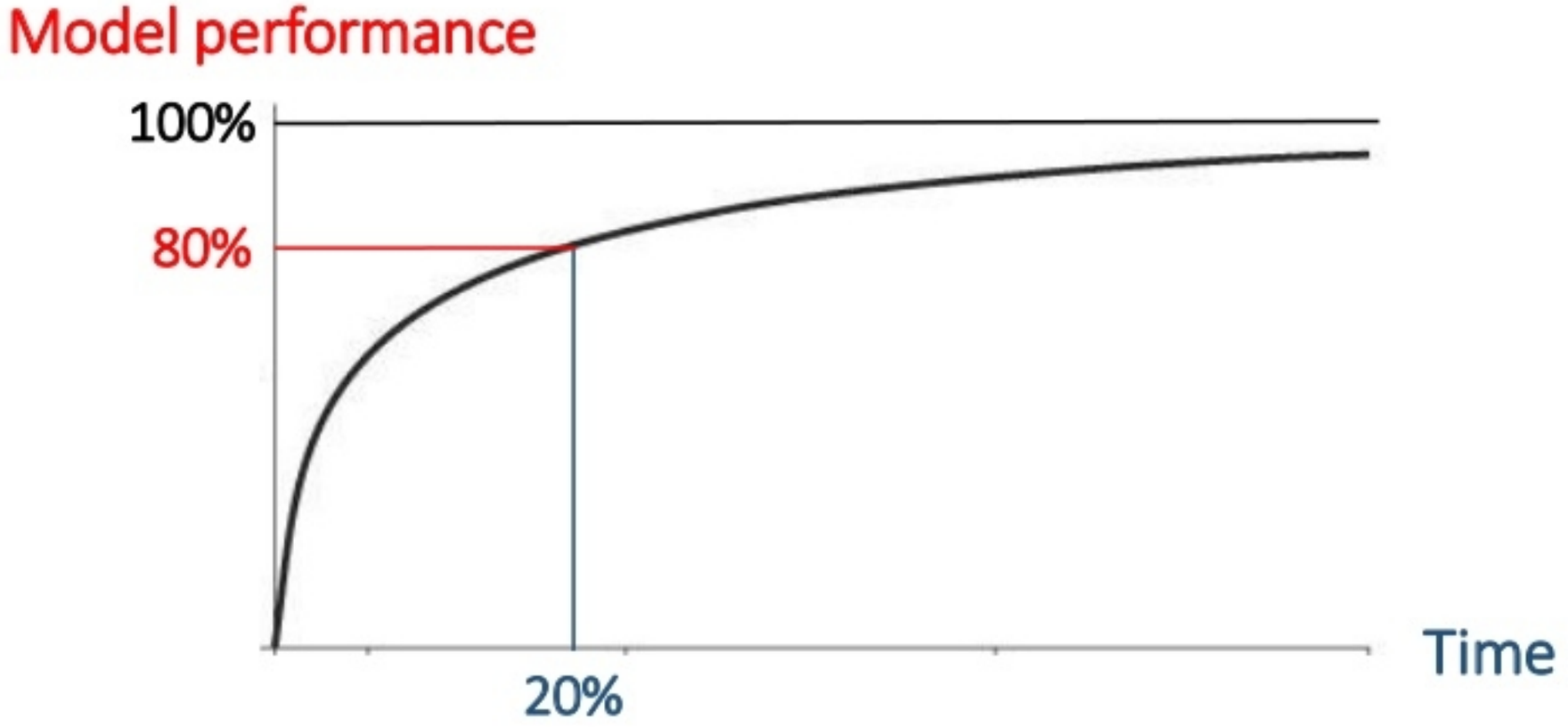
It includes regularization that helps the algorithm not to overfit, which is the big risk with boosting!

Learn more on XGBoost Gradient Boosted trees


You now have all you need to build a performing machine learning model!

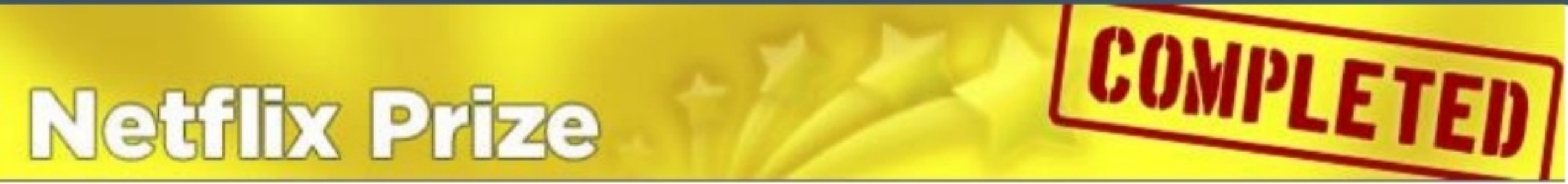
CONCLUSION

This is how the performance of your model will most likely evolve



You must assess when the effort is not worth it anymore!

CONCLUSION



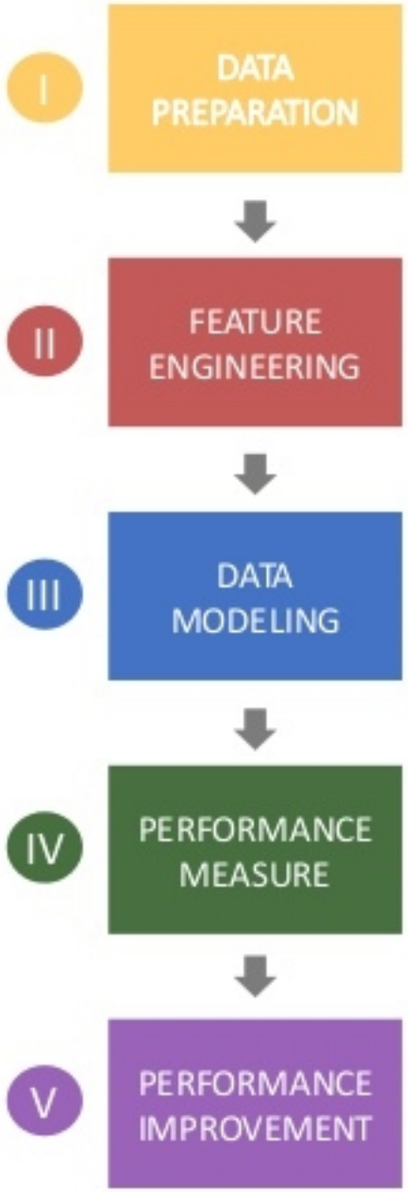
In 2006, Netflix offered a $1m prize for anyone who would **improve the accuracy of their recommendation system by 10%.**

The 2nd team, which achieved a 8.43% improvement, reported **more than 2000 hours of** work to come up with a combination of **107 algorithms!**

The 10% improvement was **only achieved in 2009**, and the algorithm **never went into production…**

Learn more

BUILDING A MACHINE LEARNING MODEL: SUMMARY



First, you will apply some of the most common data cleaning actions on your **raw data**, including **removing outliers** and dealing with **missing values** and **categorical variables**.

You then **turn raw data into individual measurable properties (features)** that will help your model complete its task. They must be as **informative**, **discriminative** and **non-redundant** as possible. This step is **commonly acknowledged as the most important part** in building a ML model.

You can now apply either **supervised or unsupervised** machine learning algorithms. Their **complexity vary** but how they correctly model your data will **solely depend on your assumptions**.

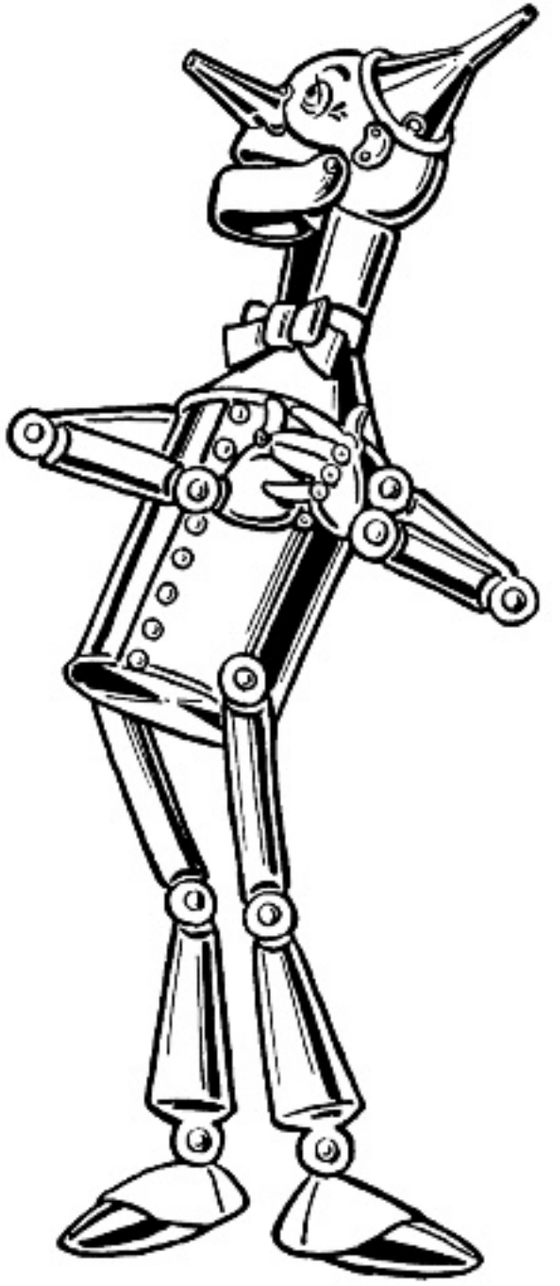
To assess the performance of your model, you will pick a **relevant indicator that you understand** and measure it on **unseen test data** that you will **have set aside before training your model**.

Your model can underperform for **only two reasons: underfitting or overfitting**. Many solutions exist, including two popular techniques: **regularization** (overfitting) and **boosting** (underfitting.)

Thank you!

If you wanna talk about Machine Learning:

charles.vestur@gmail.com



GOOD RESSOURCES

Main ressources:

The very famous MOOC from Andrew Ng. An excellent introduction to Machine Learning, in which you will learn different algorithms and a bit of the maths behind it.

Kaggle, a reference for data science, which provides many public datasets, organizes competitions in which you can take part or get inspired by the code published by the winners!

Quora: A lot of people put in the effort to clearly explain a lot of concepts in Machine Learning. You can quickly spot the best answers with the upvotes.

CrossValidated: The equivalent of StackOverflow for Data Science. Just like in Quora, you will find very good quality answers on numerous topics.

Blogs:

Analytics Vidhya: Many articles on a ML topic with explanations and code samples

Machine Learning Mastery: Similar to Analytics Vidhya, you will find nice articles on this blog.

Newsletter:

Data Elixir: You don't need a thousand newsletter, this one is a very good one with both technical and high-level articles.