MACHINE LEARNING COURSE

PRESENTED BY ABDEL RAHMAN ALSABBAGH

LECTURE #2 – SAT - 13.5.2023

In the name of Allah, the most gracious, the most merciful, we start :)

Today's Quote

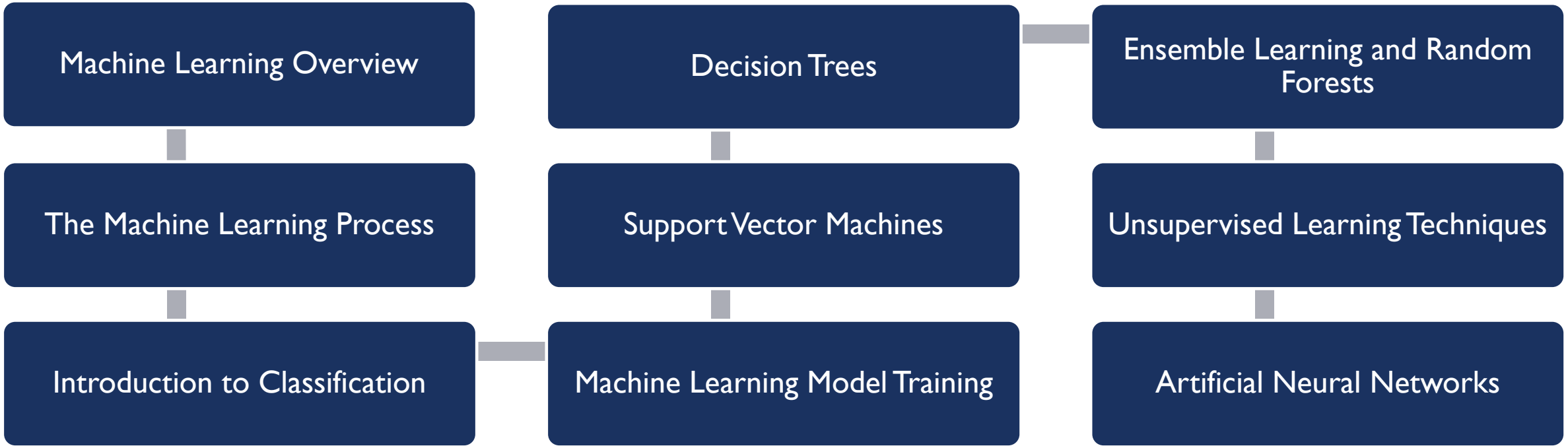
"You don't have to see the whole staircase,
just take the first step."

- Martin Luther King Jr.

So, What is Machine Learning in a sentence?

It's the science of getting computers to learn without being **explicitly** programmed.

Course Outline



Resources used:

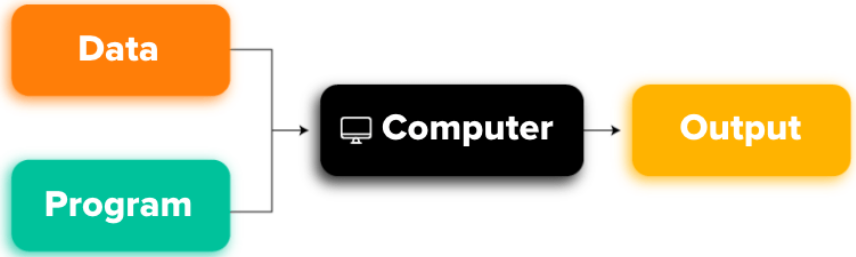
- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron
- Machine Learning Specialization by Andrew Ng and Stanford Online

Machine Learning Overview

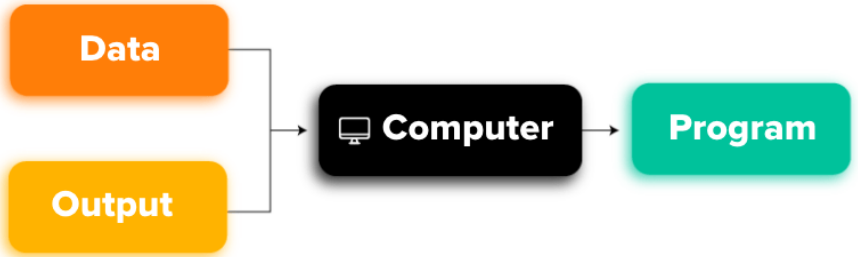
Topics covered in this section include:

- What Is Machine Learning.
- Why Use Machine Learning?.
- Types of Machine Learning Systems.
- Main Challenges of Machine Learning.
- Testing and Validating.
- Exercises.

Why Use Machine Learning?

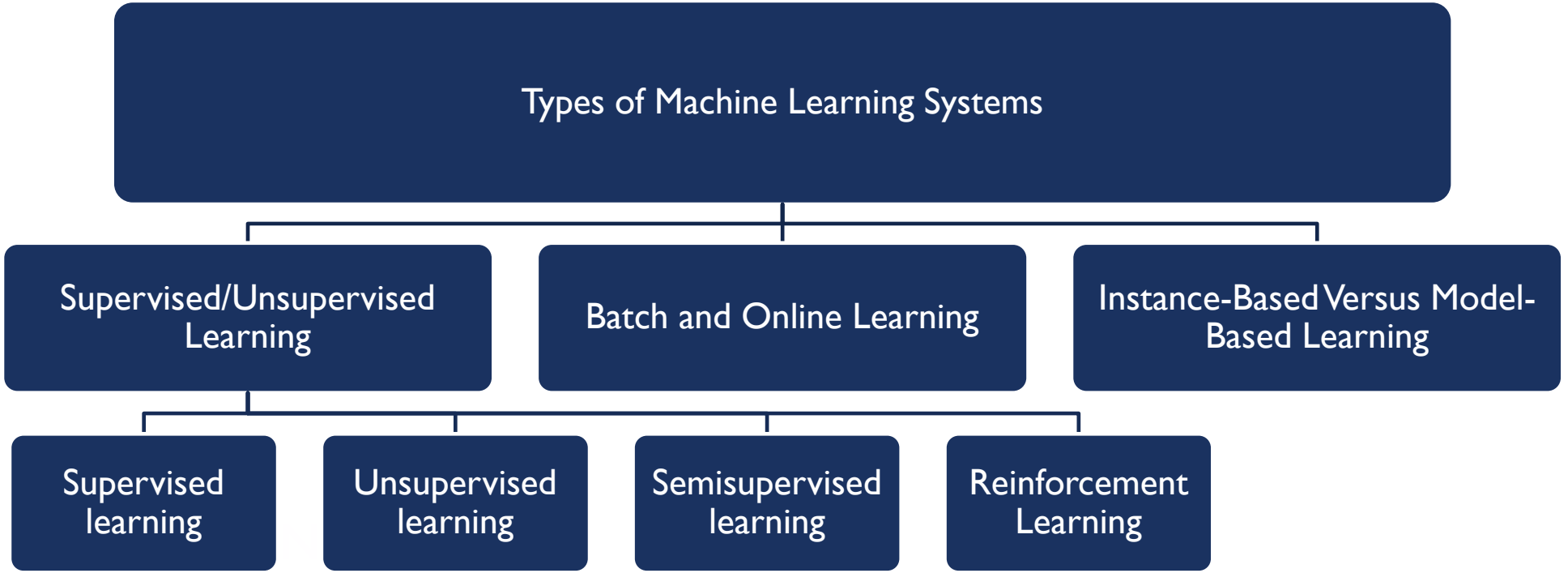


Traditional approach



Machine Learning approach

Types of Machine Learning Systems



Instance-Based Versus Model-Based Learning

One more way to categorize Machine Learning systems is by how they *generalize*.

Most Machine Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to generalize to examples it has never seen before.
Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.

There are two main approaches to generalization: **instance-based** learning and **model-based** learning.

Instance-Based Versus Model-Based Learning

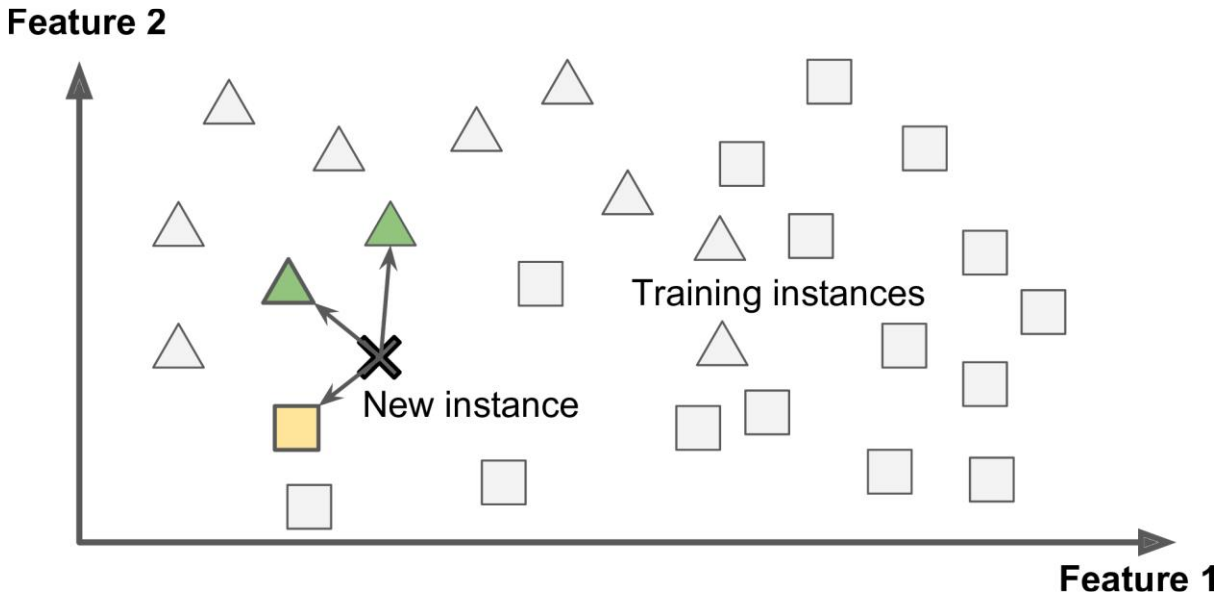
One more way to categorize Machine Learning systems is by how they *generalize*.

Most Machine Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to generalize to examples it has never seen before.
Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.

There are two main approaches to generalization: **instance-based** learning and **model-based** learning.

Instance-Based Learning

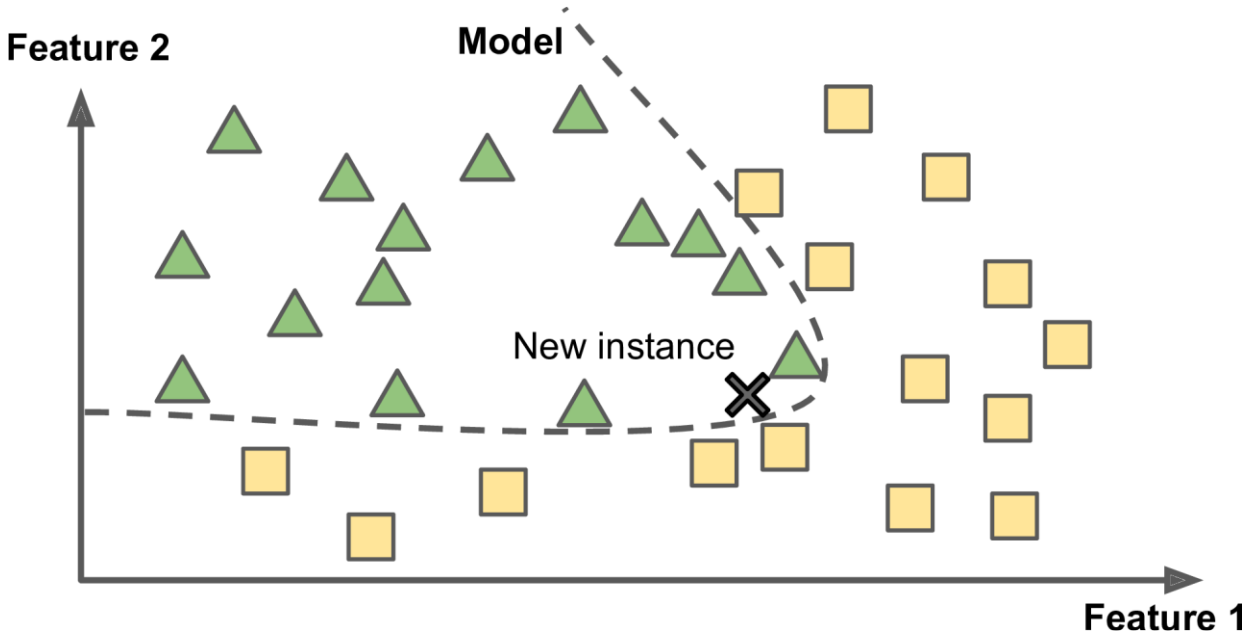
The system learns the examples by heart, then generalizes to new cases by comparing them to the learned examples (or a subset of them), using a similarity measure.



Model-Based Learning

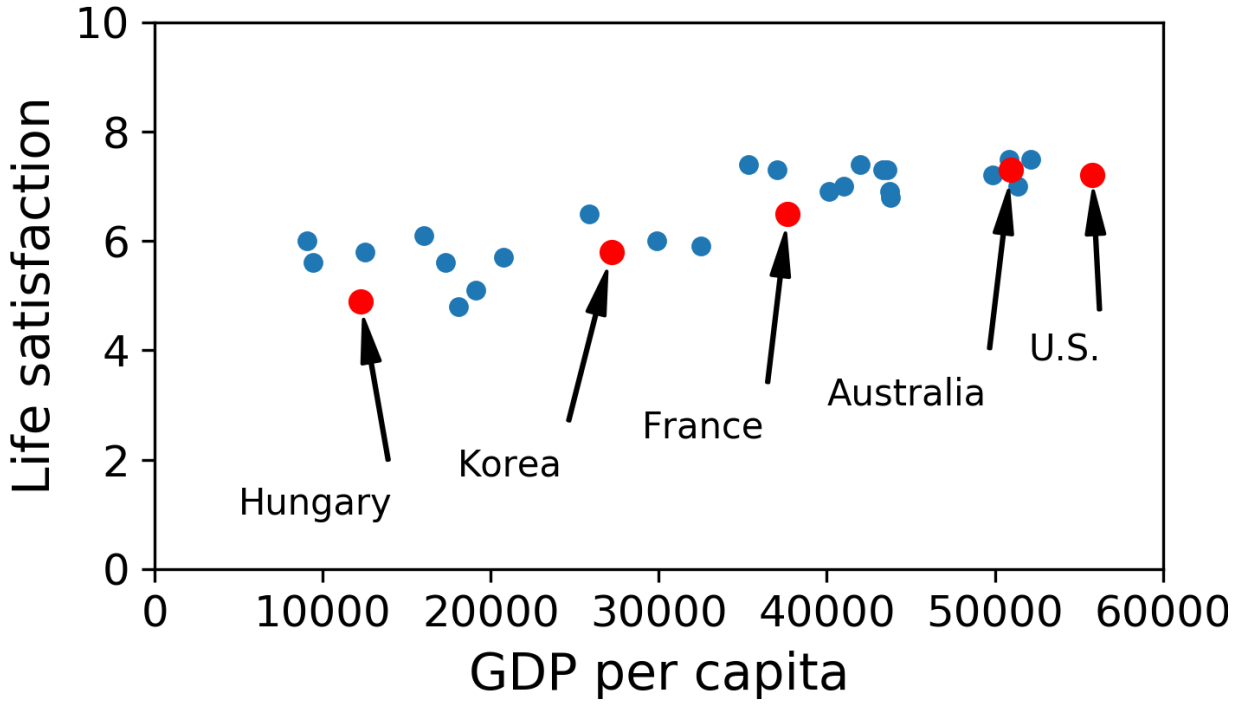
Another way to generalize from a set of examples is to build a model of these examples, then use that model to make **predictions**. This is called **model-based** learning.

A **model** is the output of a machine learning algorithm run on data. A model represents what was learned by a machine learning algorithm.



Model-Based Learning - Example

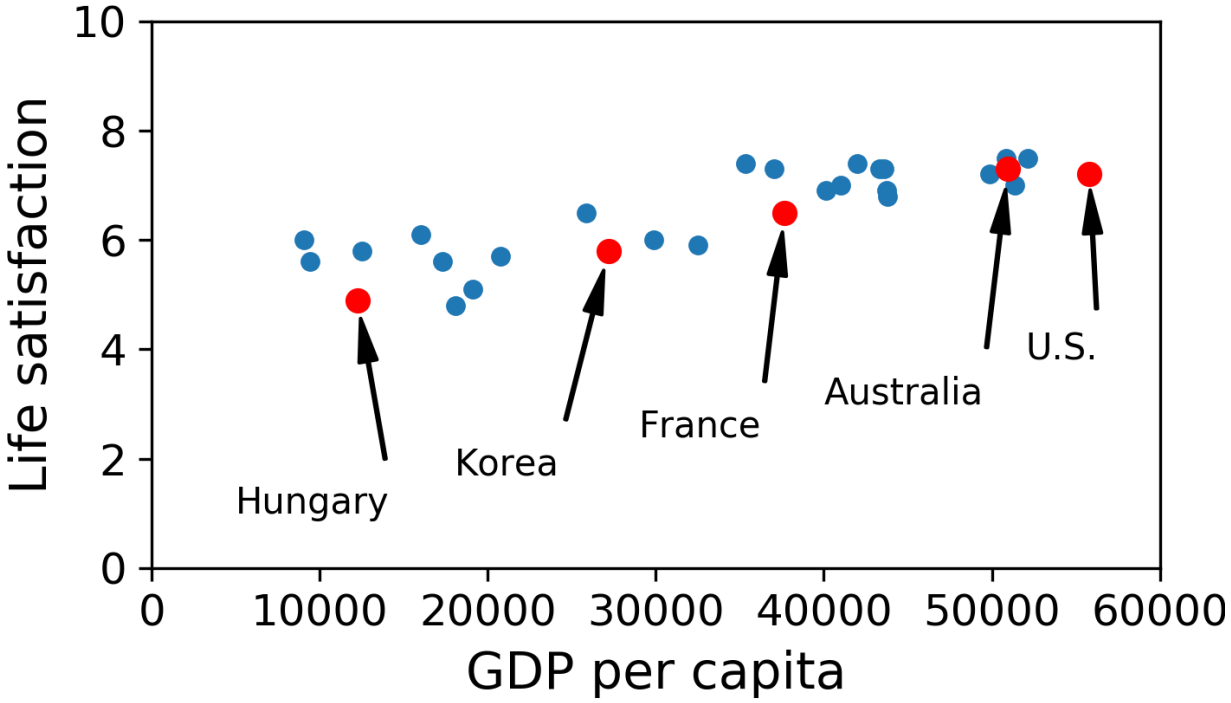
Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2



Model-Based Learning - Example

There does seem to be a trend here! Although the data is noisy (i.e., partly random), it looks like life satisfaction goes up more or less linearly as the country's GDP per capita increases.

So you decide to model life satisfaction as a linear function of GDP per capita.



Model-Based Learning - Example

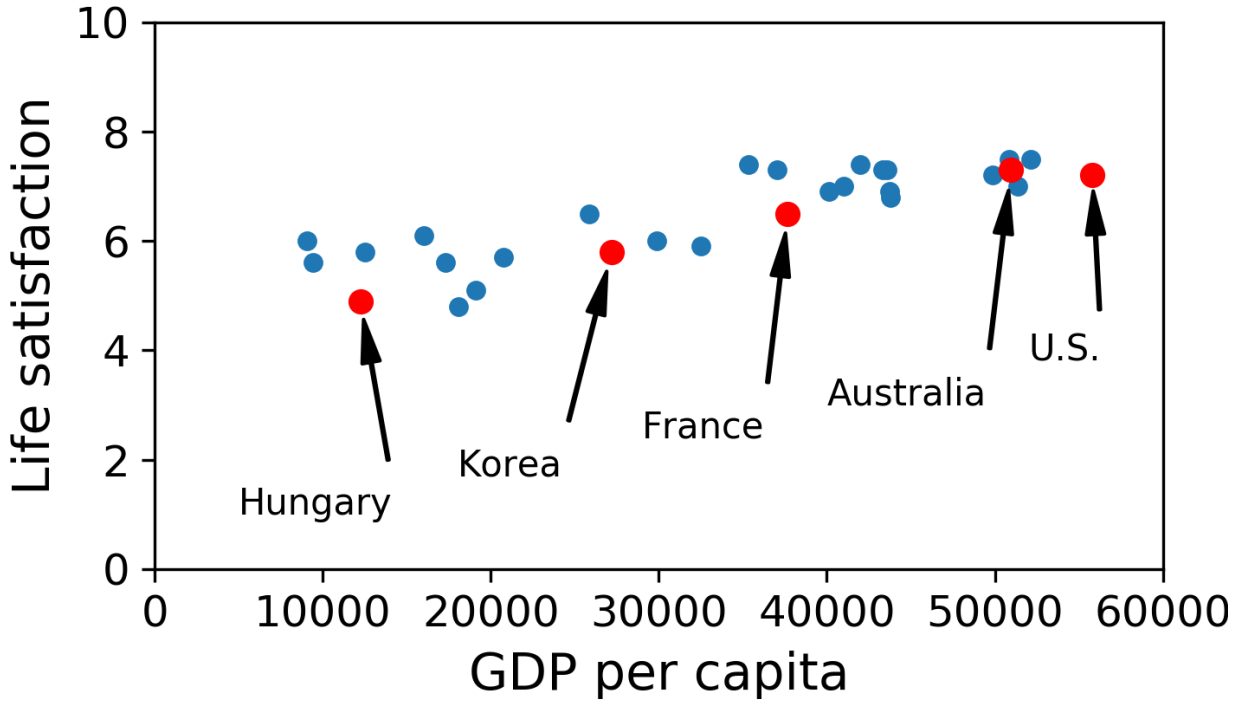
This step is called model selection: you selected a linear model of life satisfaction with just one attribute, GDP per capita.

A simple linear model:

Life_satisfaction = $\theta_0 + \theta_1 \times$ GDP_per_capita

Another look:

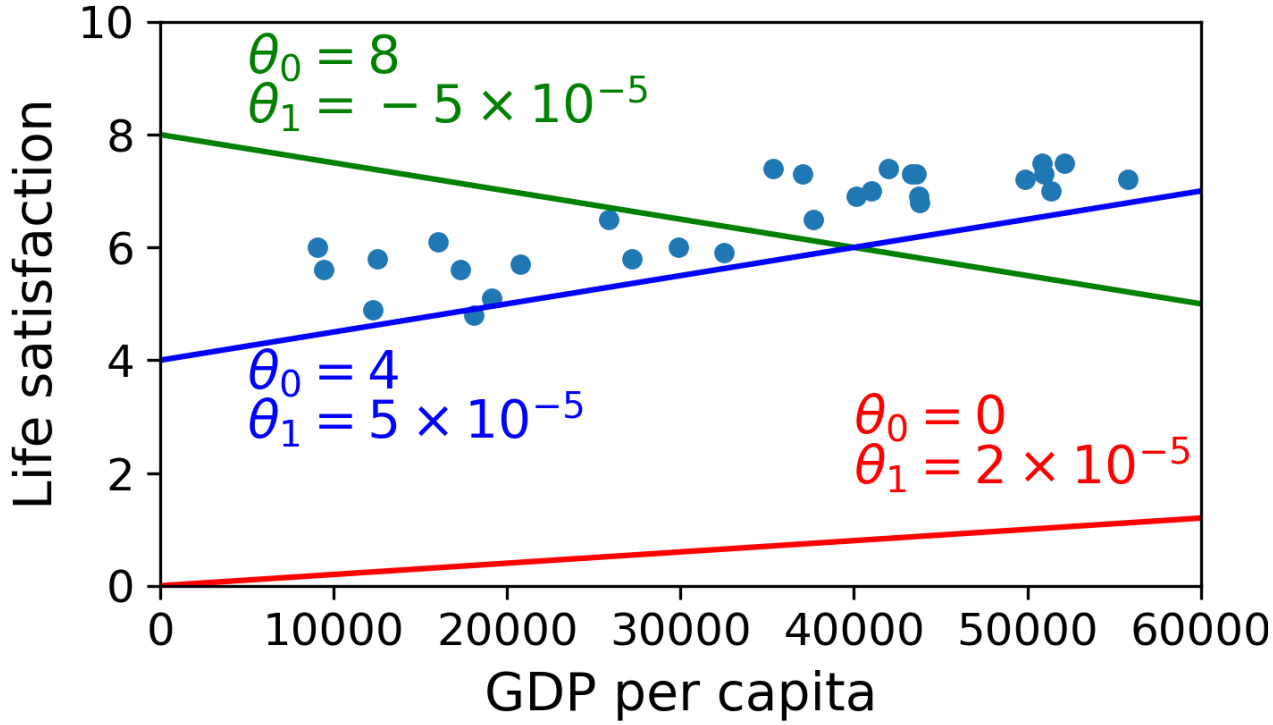
Life_satisfaction = $b + w \times$ GDP_per_capita



Model-Based Learning - Example

This model has two model **parameters**, θ_0 and θ_1. By tweaking these parameters, you can make your model represent any linear function, as shown.

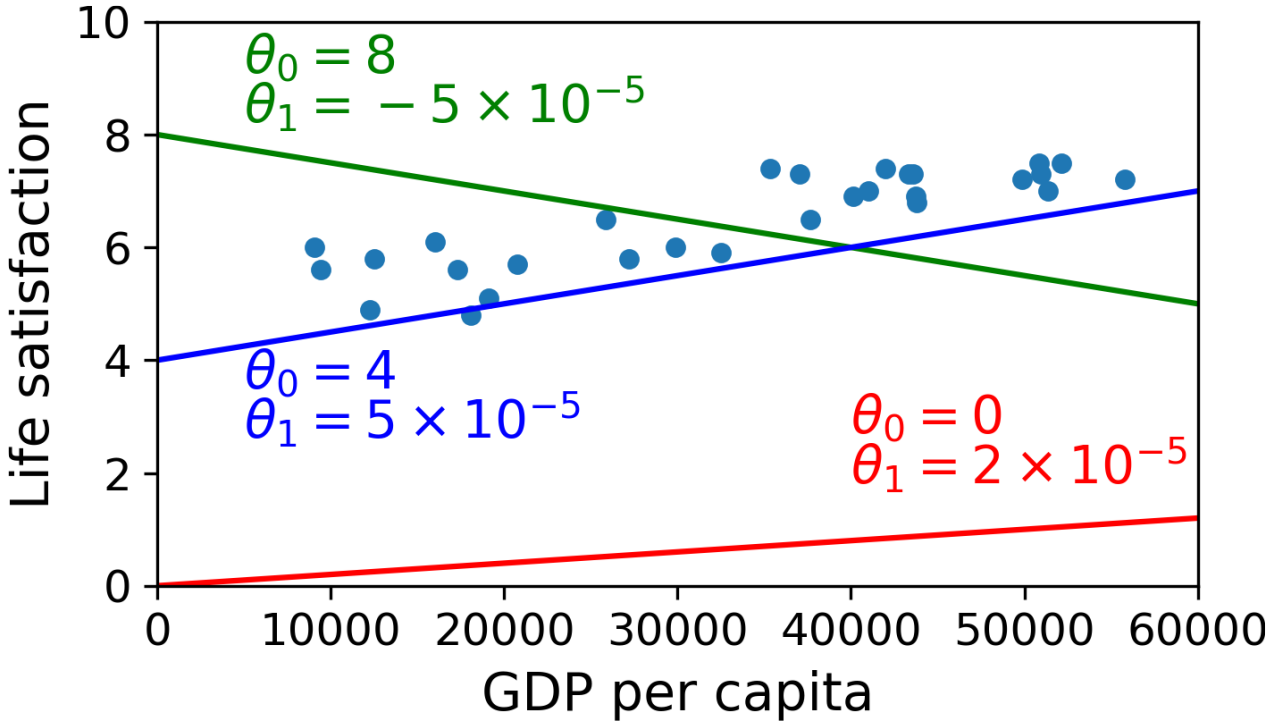
How can you know which values of values θ_0 and θ_1 will make your model perform best? To answer this question, you need to specify a **performance measure**.



Model-Based Learning - Performance Measure

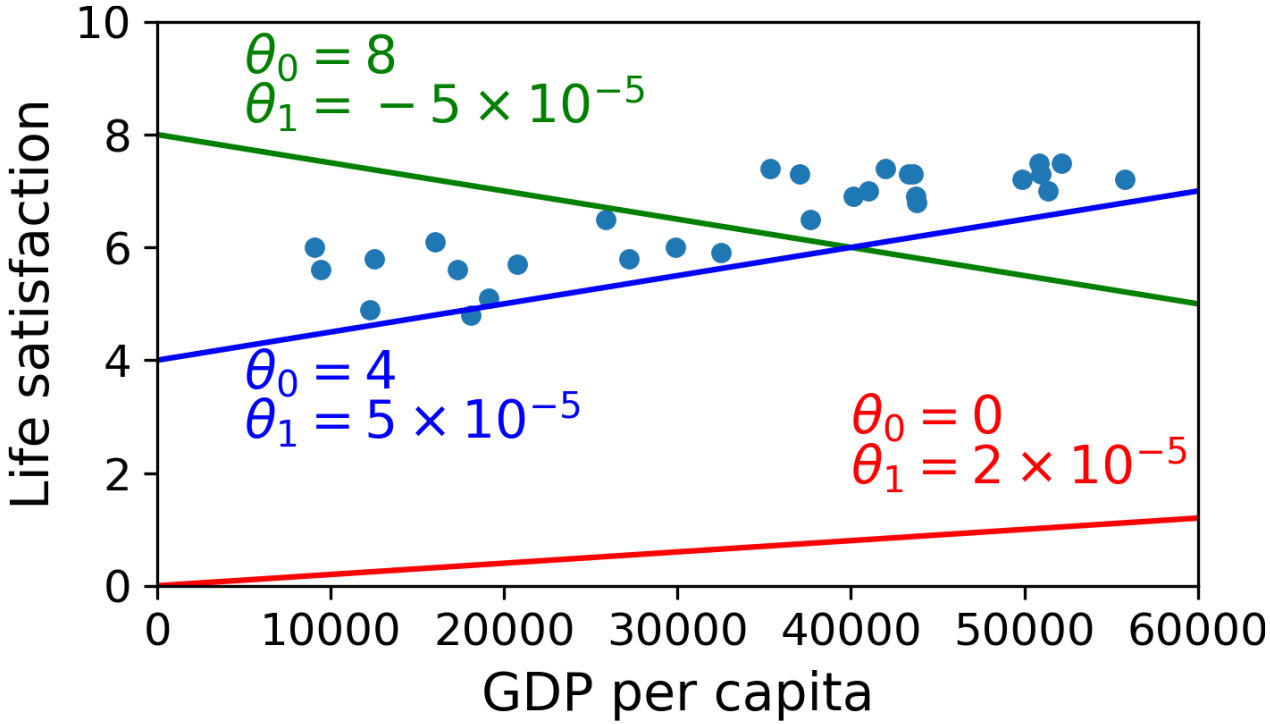
You can either define a **utility function** (or fitness function) that measures how good your model is.

Or you can define a **cost function** that measures how bad it is.



Model-Based Learning - Performance Measure

For **linear regression** problems, people typically use a **cost function** that measures the distance between the linear model's predictions and the training examples; **the objective is to minimize this distance**.

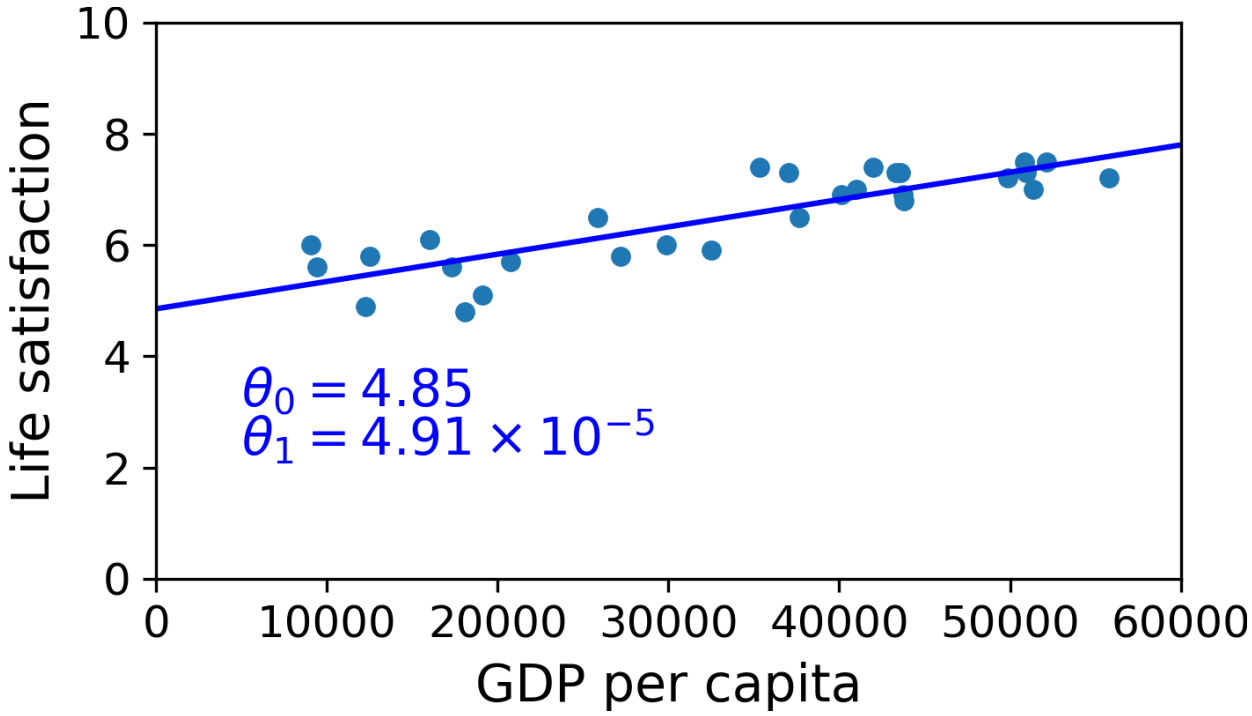


Model-Based Learning - Linear Regression

This is where the **Linear Regression** algorithm comes in: you feed it your training examples and it finds the parameters that make the linear model fit best to your data.

This is called training the model. In our case the algorithm finds that the optimal parameter values are:

- θ_0 = 4.85
- θ_1 = 4.91 × 10–5

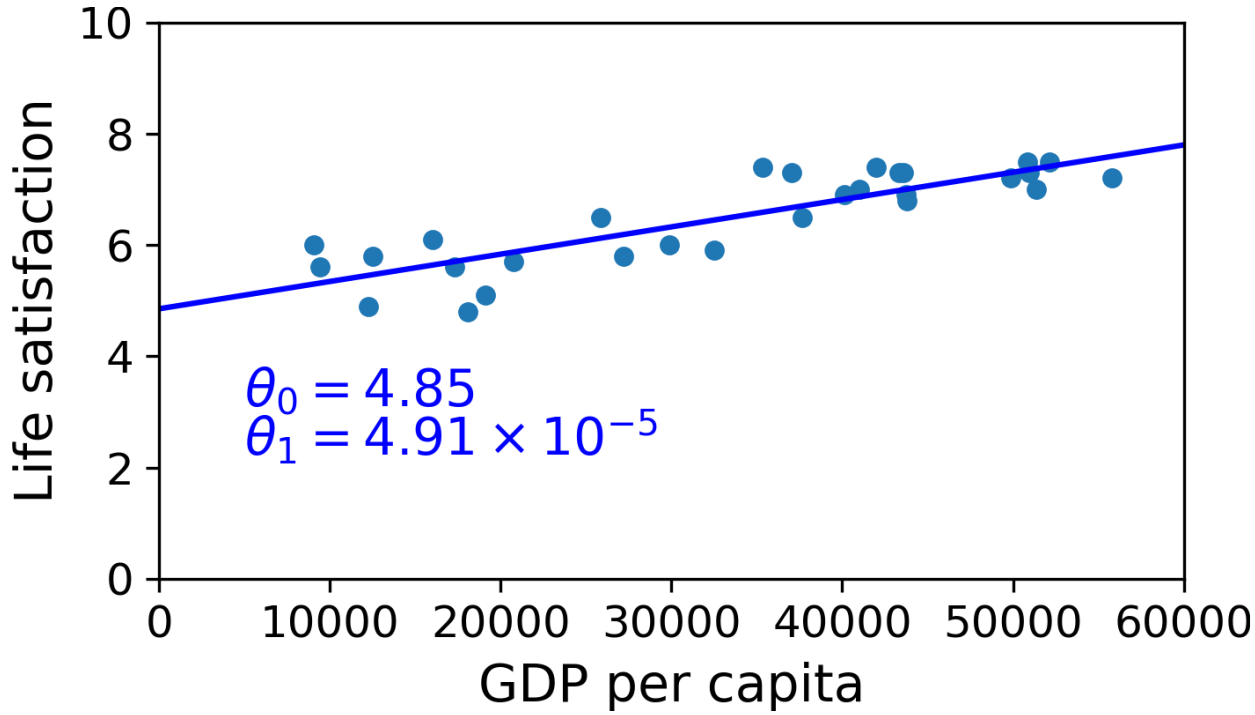


Model-Based Learning - Linear Regression

Say you want to know how happy Cypriots are, and the OECD data does not have the answer.

Fortunately, you can use your model to make a good prediction: you look up Cyprus's GDP per capita, find $22,587, and then apply your model and find that life satisfaction is likely to be somewhere around:

4.85 + 22,587 × 4.91 × 10-5 = 5.96.



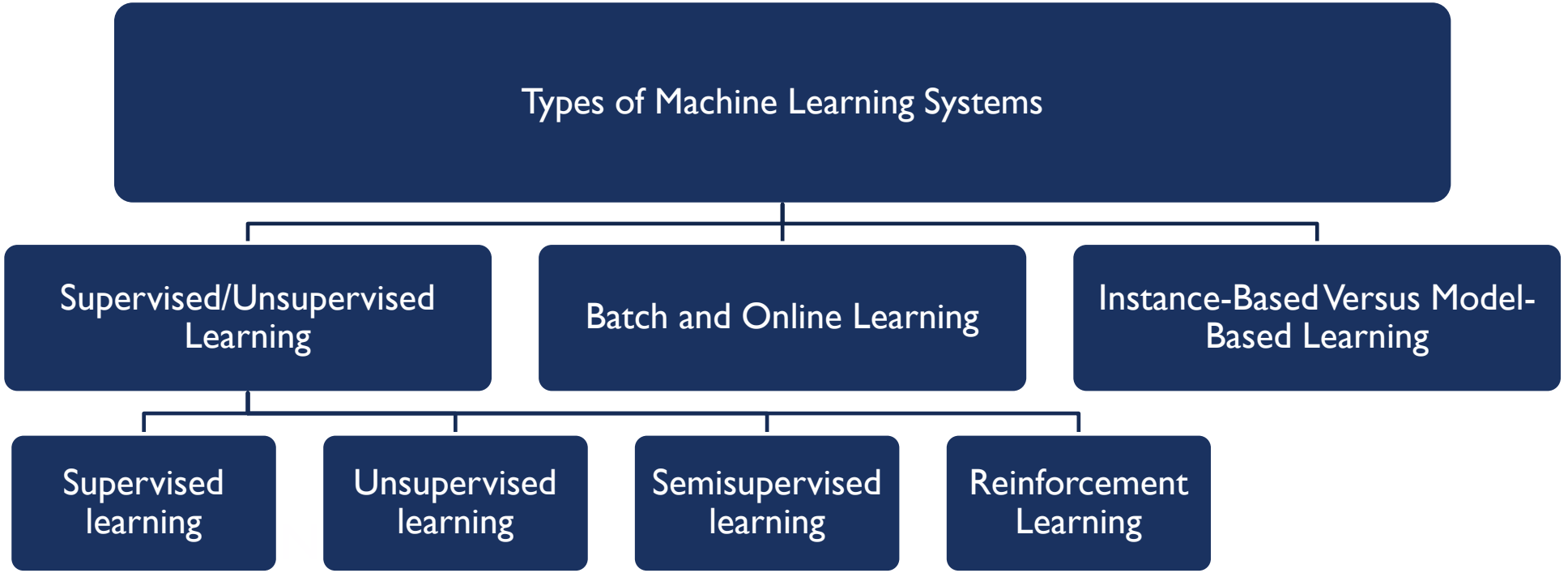
Model-Based Learning

If all went well, your model will make good predictions. If not, you may need to use more attributes (employment rate, health, air pollution, etc.), get more or better quality training data, or perhaps select a more powerful model (e.g., a Polynomial Regression model).

In summary:

- You studied the data.
- You selected a model.
- You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
- Finally, you applied the model to make predictions on new cases (this is called inference), hoping that this model will generalize well.

Recap


Types of Machine Learning Systems
Supervised/Unsupervised Learning
Batch and Online Learning
Instance-Based Versus Model-Based Learning
Supervised learning
Unsupervised learning
Semisupervised learning
Reinforcement Learning

Main Challenges of Machine Learning

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are "bad algorithm" and "bad data."

Bad algorithm:

- Overfitting the Training Data.
- Underfitting the Training Data.

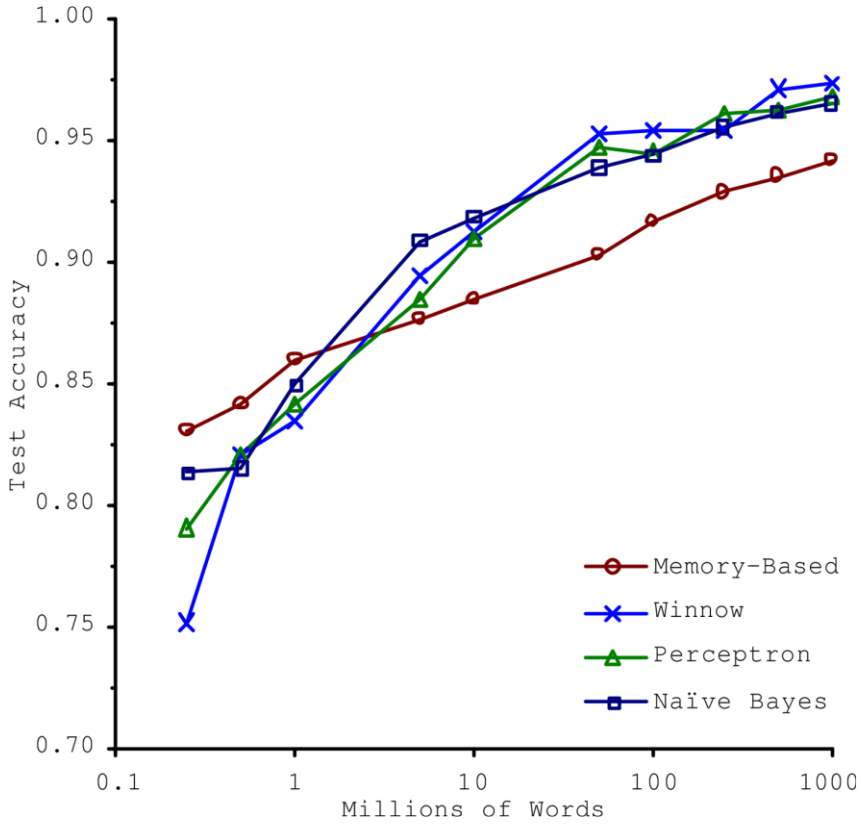
Bad data:

- Insufficient Quantity of Training Data.
- Nonrepresentative Training Data.
- Poor-Quality Data.
- Irrelevant Features.

Insufficient Quantity of Training Data

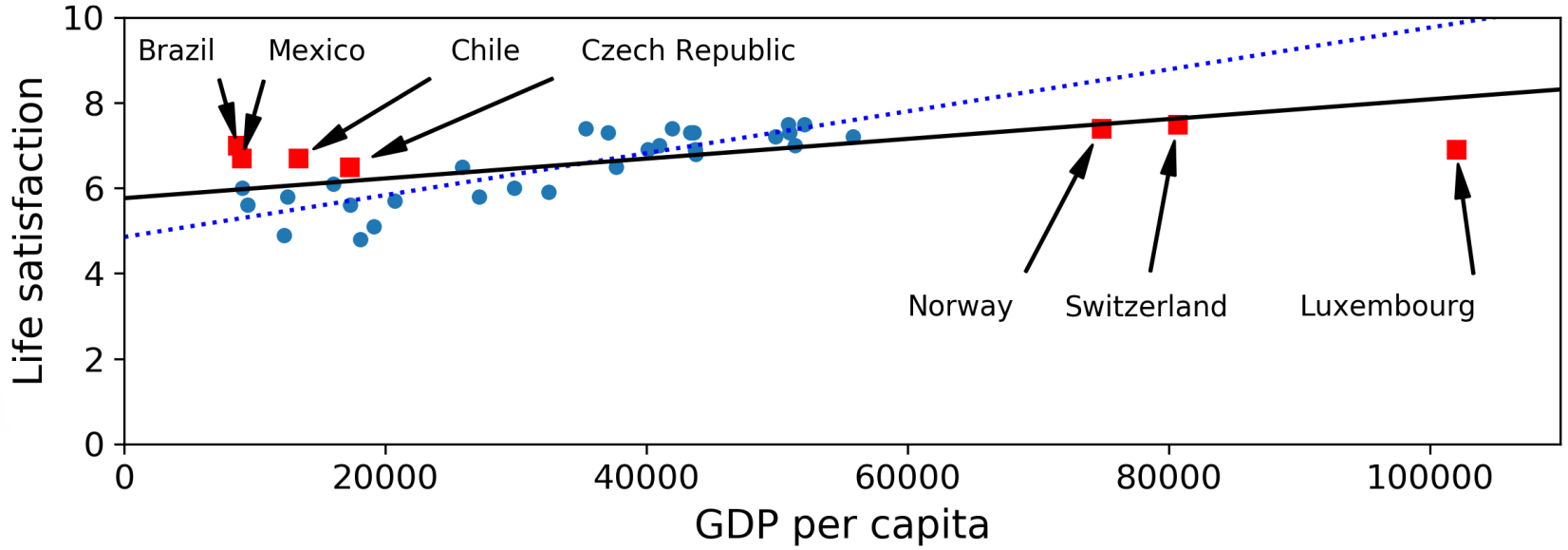
It takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

Microsoft researchers showed that very different ML algorithms, including fairly simple ones, performed almost identically well on a complex problem once they were given enough data.



Nonrepresentative Training Data

In order to generalize well, it is crucial that your training data be representative of the new cases you want to generalize to. This is true whether you use instance-based learning or model-based learning.



Poor-Quality Data

Obviously, if your training data is full of errors, outliers, and noise (e.g., due to poor quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well.

It is often well worth the effort to spend time cleaning up your training data.

The truth is, most data scientists spend a significant part of their time doing just that.

Irrelevant Features

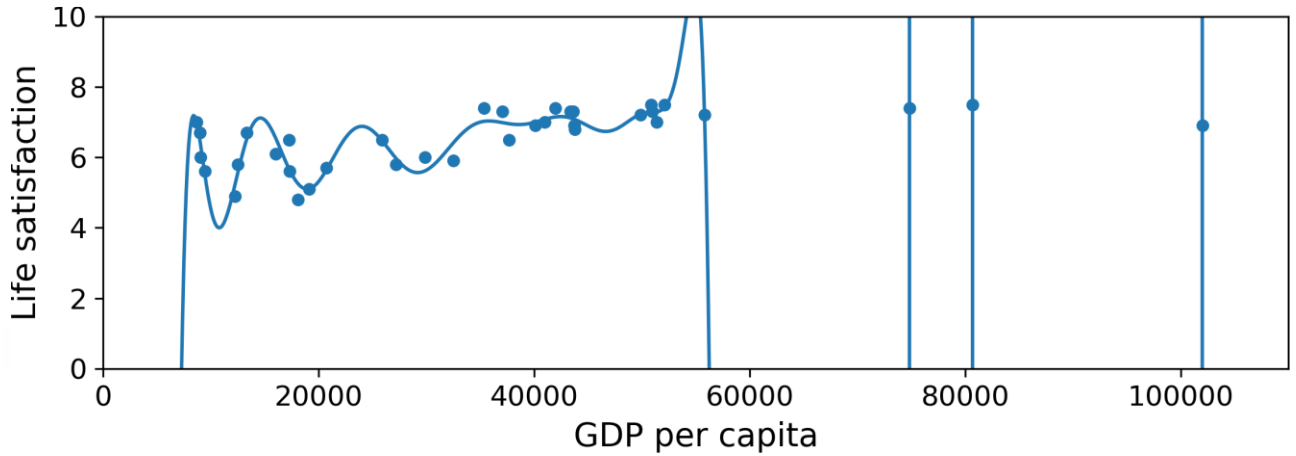
As the saying goes: garbage in, garbage out.

Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones. A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called **feature engineering**, involves:

- Feature selection: selecting the most useful features to train on among existing features.
- Feature extraction: combining existing features to produce a more useful one (as we saw earlier, dimensionality reduction algorithms can help).
- Creating new features by gathering new data.

Overfitting the Training Data

The model performs well on the training data, but it does not generalize well.
Complex models can detect subtle patterns in the data, but if the training set is noisy, or if it is too small (which introduces sampling noise), then the model is likely to detect patterns in the noise itself. Obviously these patterns will not generalize to new instances.



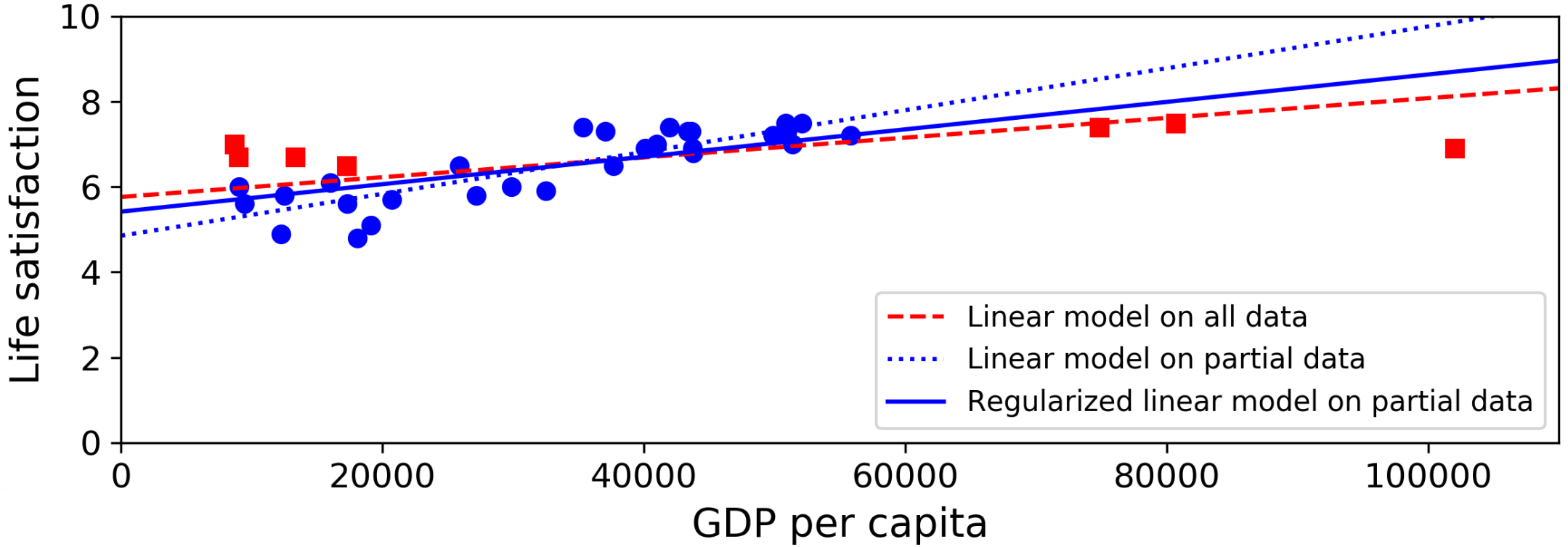
Overfitting the Training Data

Overfitting happens when the model is too complex relative to the amount and noisiness of the training data.
The possible solutions are:

- To simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model.
- To gather more training data.
- To reduce the noise in the training data (e.g., fix data errors and remove outliers).

Constraining a model to make it simpler and reduce the risk of overfitting is called **regularization**.

Overfitting the Training Data



Overfitting the Training Data

The amount of regularization to apply during learning can be controlled by a **hyperparameter**. A hyperparameter is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training.

If you set the regularization hyperparameter to a very large value, you will get an almost flat model (a slope close to zero); the learning algorithm will almost certainly not overfit the training data, but it will be less likely to find a good solution.

Tuning hyperparameters is an important part of building a Machine Learning system.

Underfitting the Training Data

As you might guess, underfitting is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data.

The main options to fix this problem are:

- Selecting a more powerful model, with more parameters.
- Feeding better features to the learning algorithm (feature engineering).
- Reducing the constraints on the model (e.g. reducing the regularization hyperparameter).

Recap

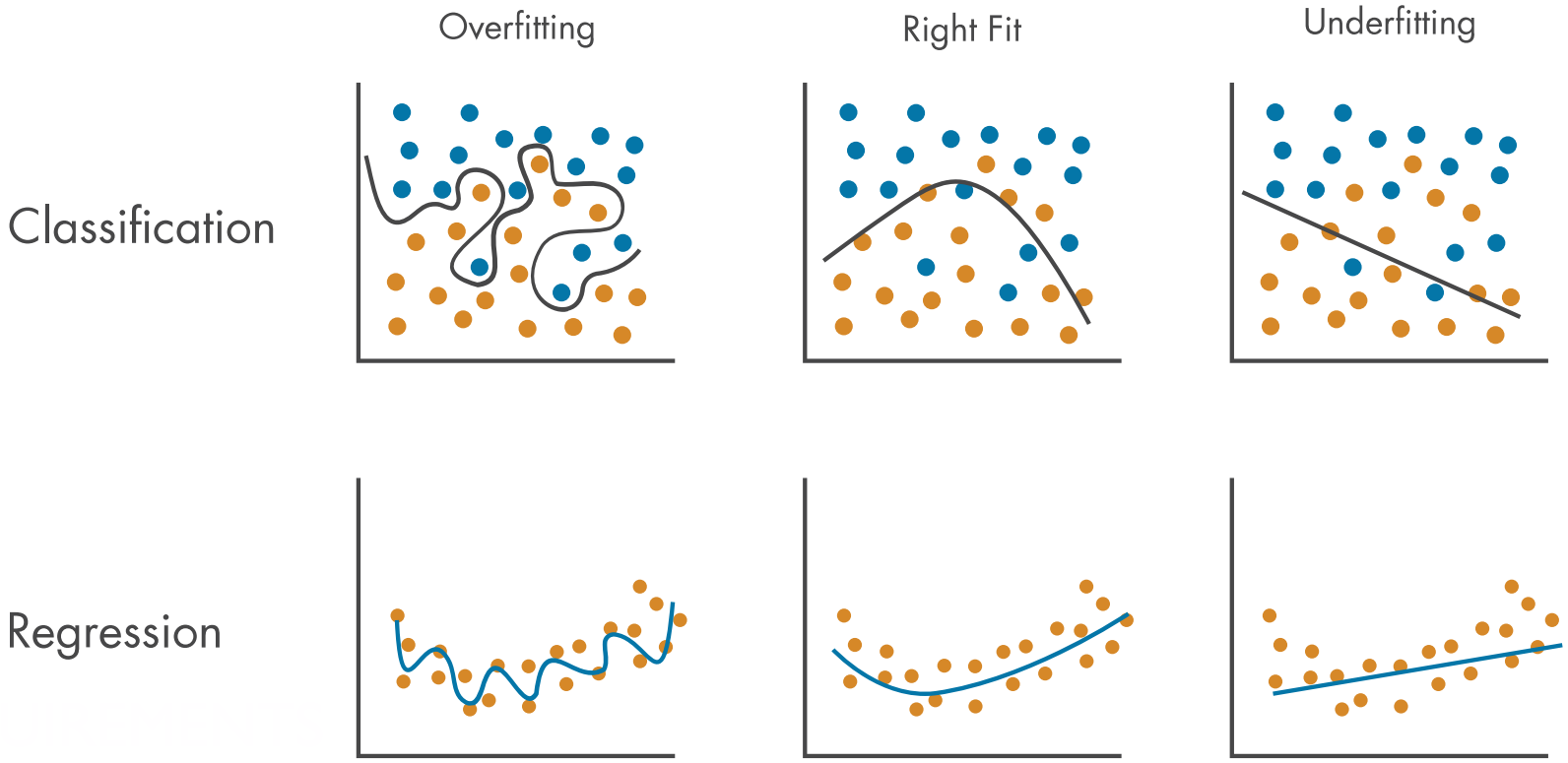


Image source: MathWorks

Stepping Back

By now you already know a lot about Machine Learning.

However, we went through so many concepts that you may be feeling a little lost, so let's step back and look at the big picture:

- Machine Learning is about making machines get better at some task by learning from data, instead of having to explicitly code rules.

- There are many different types of ML systems: supervised or not, batch or online, instance-based or model-based, and so on.

Stepping Back

- In a ML project you gather data in a training set, and you feed the training set to a learning algorithm. If the algorithm is model-based it tunes some parameters to fit the model to the training set (i.e., to make good predictions on the training set itself), and then hopefully it will be able to make good predictions on new cases as well. If the algorithm is instance-based, it just learns the examples by heart and generalizes to new instances by comparing them to the learned instances using a similarity measure.

- The system will not perform well if your training set is too small, or if the data is not representative, noisy, or polluted with irrelevant features (garbage in, garbage out). Lastly, your model needs to be neither too simple (in which case it will underfit) nor too complex (in which case it will overfit).

Testing and Validating

There's just one last important topic to cover:

Once you have trained a model, you don't want to just "hope" it generalizes to new cases.
You want to evaluate it, and fine-tune it if necessary.
Let's see how.

Testing and Validating

- The only way to know how well a model will generalize to new cases is to actually try it out on new cases. Not the best idea.

- A better option is to split your data into two sets:
  - The training set.
  - The test set.

- As these names imply, you train your model using the training set, and you test it using the test set.

Testing and Validating

- The error rate on new cases is called the generalization error (or out-of-sample error), and by evaluating your model on the test set, you get an estimate of this error.

- This value tells you how well your model will perform on instances it has never seen before.

- If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

Testing and Validating

Hyperparameter Tuning and Model Selection

- So evaluating a model is simple enough: just use a test set.

- Now suppose you are hesitating between two models (say a linear model and a polynomial model): how can you decide? One option is to train both and compare how well they generalize using the test set.

- Now suppose that the linear model generalizes better, but you want to apply some regularization to avoid overfitting. The question is: how do you choose the value of the regularization hyperparameter?

Hyperparameter Tuning and Model Selection

- One option is to train 100 different models using 100 different values for this hyperparameter. Suppose you find the best hyperparameter value that produces a model with the lowest generalization error, say just 5% error.

- So you launch this model into production, but unfortunately it does not perform as well as expected and produces 15% errors. What just happened?

- The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that particular set. This means that the model is unlikely to perform as well on new data.

Hyperparameter Tuning and Model Selection

Hyperparameter Tuning and Model Selection

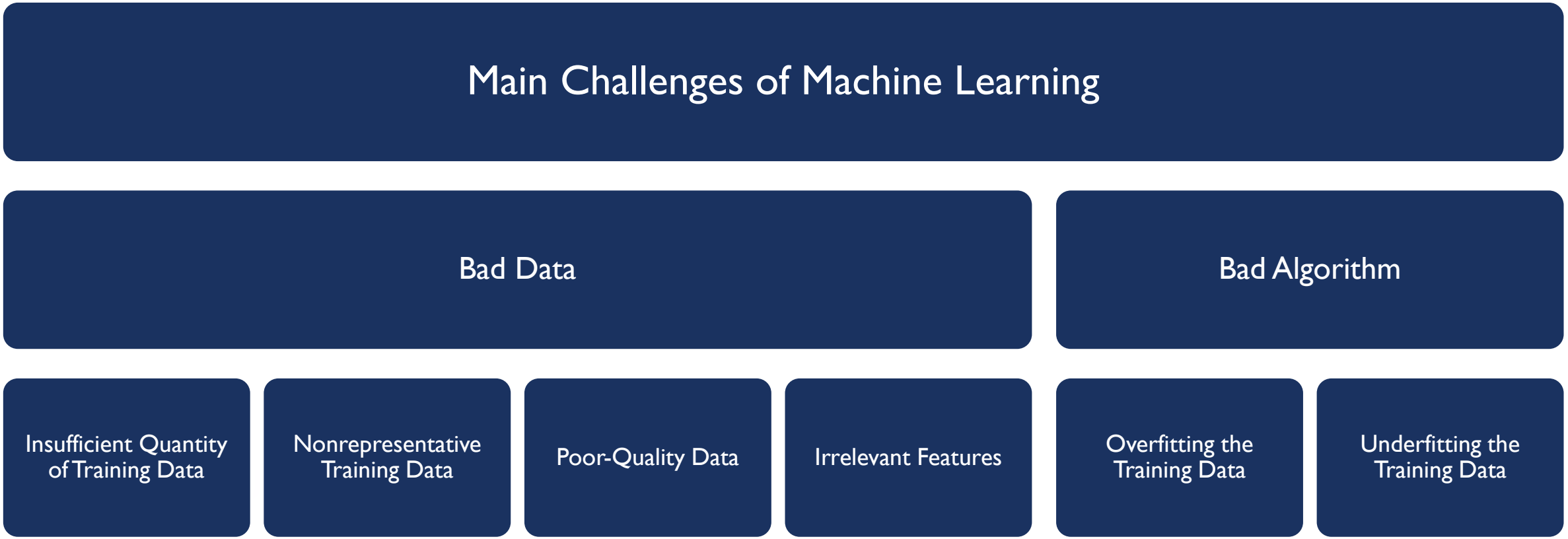
Data Mismatch

- In some cases, it is easy to get a large amount of data for training, but it is not perfectly representative of the data that will be used in production.

- The most important rule to remember is that the validation and the test set must be as representative as possible of the data you expect to use in production.

Data Mismatch

Recap


Main Challenges of Machine Learning
Bad Data
Bad Algorithm
Insufficient Quantity of Training Data
Nonrepresentative Training Data
Poor-Quality Data
Irrelevant Features
Overfitting the Training Data
Underfitting the Training Data

Recap

Testing and Validating

Hyperparameter Tuning and Model Selection

Data Mismatch

Finally

Let's write some code!

Want to get better at coding?
Try LeetCode (https://leetcode.com/studyplan/)

On another note, there's a CV workshop with an HR from the prestigious company RealSoft.
This Tuesday, 16.5, 12:30, KASIT Lab #107. Join us!

THANK YOU

NEXT LECTURE WILL BE ONLINE
ON MON, 15.5.2023, IN SHAA ALLAH!

SABBAGH@IEEE.ORG

CONNECT ON LINKEDIN