MACHINE LEARNING

Arthur Samuel (1959) defined machine learning as a **field of study that gives computers the ability to learn without being explicitly programmed**.

Differently from the algorithms we are used to create, a machine learning algorithm can **learn how to solve a specific problem from a set of data**. This will shift our focus from designing all the steps needed to solve a problem to create algorithms which can effectively learn how to solve problem.

In some sense we can see classic programming as designing a procedure that everybody can follow to solve a specific task and machine learning as teaching a series of techniques to instruct a set of people to figure out by themselves how to solve a given problem.

We'll give a more formal definition in a moment.

WHY DO WE NEED MACHINE LEARNING?

As computer scientists, to solve a problem, we generally think about a logical way to approach it, then write a program, possibly broken into several different pieces, which allows to automate the logical approach we have devised.

This works for a series of problems (even complex ones) which we know how to solve by hand, such as:

- Computing the area of polygons.
- Solving partial differential equations.
- Creating an interface which allows to store and search some values in a database.
- Count the instances of a specific word in a long text.

If we think about this first class of problems, we can devise some logical process to carry out the task. If we are good at programming, we can generally build programs that solve this first class of problems.

Some problems however are particularly hard and involve **uncertainty**[1]. Some examples are:

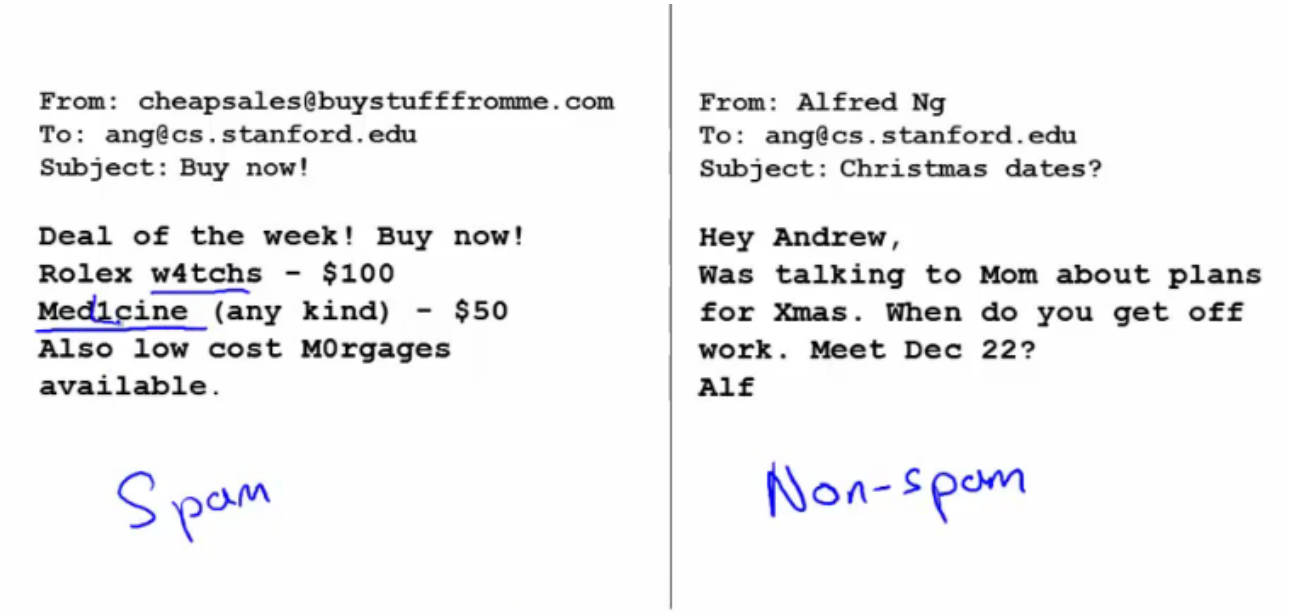
- Distinguishing legitimate e-mails form spam e-mails.
- Classifying an image to determine what object is depicted in it.
- Infer the correct colorization of a grayscale photo.

This second class of problems is much harder to deal with. On one hand, this is due to the fact that these phenomena are governed by **large uncertainty**, which does not allow us to make many assumptions on what data we are likely to see. On the other hand, even if, as humans, we can solve these tasks, **we do not always know how we do it**. For instance, we can understand what objects are present in an image, but the way our brain does it is still not very well known.

GUIDING EXAMPLE

To better understand all the definitions given in this lecture, we will consider a simple example in which we want to create a machine learning algorithm to determine if a given email is spam or not. We will say that each email can go into two bins (which we will later call "classes"): *spam* and *ham*. To know where this terminology come from, you can read the Wikipedia page for spamming (https://en.wikipedia.org/wiki/Spamming) and see this Monty Python's sketch (https://www.youtube.com/watch?v=_bW4vEo1F4E).

[1] Also in this context, uncertainty can be due to the fact that the process we are trying to model is genuinely stochastic, that it is too hard to model, or that we cannot observe it completely.



DEFINITION

Tom Mitchell (1997) defined the problem of learning as follows:

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

There are several components in this definition:

- Task: this is the problem to be solved. In example of determining if an email is *spam* or *ham*, **predicting** the label ($Y = spam$ or $Y = ham$) given the post **is the task**.
- Experience: this is **data**. Remember that we defined data as the values assumed by random variables. In our example, if X is the content of the email and Y i s its label (spam or ham), a set of pairs of values $\{ (X = x_i, Y = y_i)\}_{i=1}^{N}$ is the **experience**. The experience is generally seen as a collection of items (pairs of values in this examples). Each item is also called an "**example**".
- Performance measure: this is a function that evaluates how well the computer can solve the task T. Suppose our algorithm has *predicted* a set of labels for a given number of emails: $\{\hat{y}_i\}_i$ (here the 'hat' symbol indicates that this value has not been observed, but we have predicted it) and suppose that we know that the correct labels are the set $\{y_i\}_i$. To evaluate how good our method is, we should compare the two sets of predictions using a performance measure $P(\{y_i\}_i, \{\hat{y}_i\}_i)$. This function usually returns a real value. A large value (high performance) indicates that the predictions are accurate, whereas a small value (low performance) indicates that the predictions are not accurate. For instance, P could just compute the number of emails which have been classified correctly. We can define several performance measures. We will see the main ones in the future.

To solve these problems, we usually design machine learning algorithms which take the form of **statistical models**.

TASK, EXAMPLES AND LABELS

Machine learning enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings.

Machine learning tasks are usually described in terms of how the algorithm should process an **example**. An **example** is generally expressed as a collection of values that have been quantitatively measured from some observed event. Hence, an **example** is generally represented as a vector $\boldsymbol{x} \in \Re^n$. The vector **x** can also be written as: $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$. The

values of the input vector x_i are called **features** as they represent specific properties of the input examples. If the dimensionality of $\boldsymbol{x}$ is 10, we say that it has 10 features.

In most cases, each example $\boldsymbol{x}$ is also paired with a **desired output** y. Such desired outputs are also called **labels**. A task can hence be defined as *a certain way of processing an **input example** to obtain some **output***.

Examples of tasks include:

- Determining if an e-mail is spam or ham. In this case, the **input** is the email, the **features** may be characteristics of the email such as the number of orthographic errors or the presence of some keyworks, while the **expected output** is the label (spam or ham).
- Determining the price of a house given its position and number of rooms. In this case, the **input** a representation of the house, the **features** are position and number of rooms, while the **expected output** is the price.
- Translating a text from a language to another language. In this case, the **input** is the text in the source language, the **features** may be some characteristics of each word of the text such as a reference to a database reporting their semantic meaning, while the **expected output** is the text in the target language.
- Anomaly detection. In this case, the **input** is a set of data representing an event (e.g., the logs of accesses to a given machine from a given account), the **features** may be important characteristics extracted from the logs such as the frequency of the accesses or the list of IP addresses, while the **expected output** is a label "normal" or "abnormal".

DATA REPRESENTATION AND FEATURES

We have seen that a machine learning algorithm usually expects examples to be in the form of **feature vectors**. However, in most cases, the examples come in a form that is not very convenient to process.

For instance, e-mails are generally encoded as a sequence of characters. Also, each post has its own length, which makes the data difficult to work with.

To deal with entities such as "email messages" and "Facebook posts", we first need to turn them into some quantifiable entity. This is usually done by identifying some characteristics of the data **which are important for the given task**.

In practice, we are looking for a function f which transforms the "entity" from its original form to a destination form, which is good to solve a specific task:

$$\boldsymbol{x} = f(\boldsymbol{d})$$

Where $\boldsymbol{d}$ is the input raw data (e.g., the full e-mail message), f is the transformation function, and $\boldsymbol{x}$ is the output of the transformation, which will be the input of the machine learning algorithm.

The function f is called **representation**. The output of the transformation $\boldsymbol{x}$ is also called **representation**. When $\boldsymbol{x}$ is multi-dimensional (also called **multivariate**), it is often referred to as a **feature vector**. The fact that both f and $\boldsymbol{x}$ are called **«representation»** might generate some confusion. However, it is often clear from the context which of the two concepts we are referring to. Since by representing the data we obtain a feature vector, the data representation process is sometimes called **feature extraction**.

Representation functions are fundamental to work with machine learning. Without them, the data would often be in a form which is simply impossible to work with. **In general, representations are functions of the data which make some things explicit about the data, while neglecting other factors, depending on the considered task.**

Since we need to understand what to neglect and what to make explicit, when creating a representation, we need to keep a task in mind. There are no «universal representations», only representations which are good for some task.

FEATURES

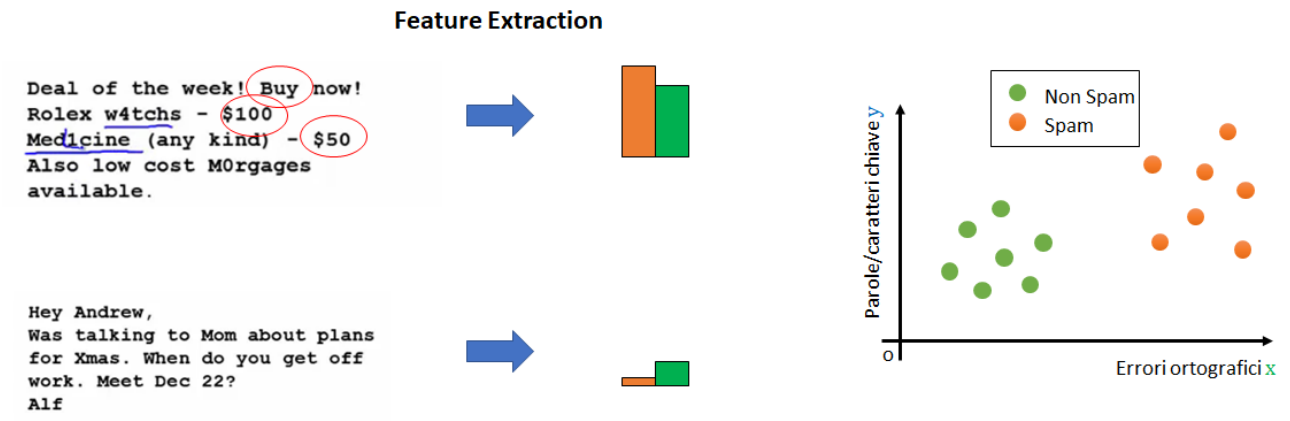
The output of a representation function is hence in general an example $\boldsymbol{x} = (x_1, \dots, x_n)$ comprised of a set of features x_i. **A feature is the specification of an attribute**. It is a measurement which represents aspects of the data which is useful to highlight to solve the considered problem. For example, color may be an attribute. "Color is blue" is a feature *extracted from* an example. We can see attributes as random variables and features as the values assumed by these variables (i.e., features are "data").

The statistical model to be employed is crucially dependent on the choice of the features. Hence it is useful to consider alternative representations of the same measurements (i.e., different features).

The features can be of two main types:

- **Categorical**: A finite number of discrete values. These can be **nominal**, denoting that there is no ordering between the values, such as last names and colors, or **ordinal**, denoting that there is an ordering, such as in an attribute taking on the values low, medium, or high.
- **Continuous**: Commonly, subset of real numbers, where there is a measurable difference between the possible values. Integers are usually treated as continuous in practical problems.

EXAMPLE



Let us consider our example in which we want to distinguish spam vs non-spam e-mails.

The input of the process is email messages, so we need to turn them into feature vectors $\boldsymbol{x} = (x_1, \dots, x_n)$ with a **feature extraction** process. Of course, we expect the extracted features to be useful to solve our task of determining if an email is spam or ham.

We may note that spam e-mails often include orthographic mistakes and words such as 'Buy', 'occasion' and '$100'. Hence, we might decide to represent each e-mail message with two numbers:

- The count of orthographic mistakes.
- The number of times some specific words or patterns appear in the text.

Once the input messages have been converted to feature vectors, they can be seen as **vectors in the 2D space** (see figure). This is convenient as we will see that many machine learning algorithms are based on geometrical observations.

Notes

- We should note that this representation has **thrown away a lot of data**.
- Indeed, if the task was to understand the content of each message, this would not be possible anymore.

- However, to recognize if the message is spam, this representation can be useful.

TYPES OF TASKS

Tasks can be of different types. In the following, we will discuss two main tasks: classification and regression.

We will assume that each machine learning algorithm takes as input examples that have already been **represented with a suitable representation function**.

CLASSIFICATION TASKS

In this type of task, the machine is asked to specify which of a pre-defined set of K categories the input belongs to. Examples of this task are:

- Classifying the Facebook posts as being about politics or something else (politics vs non-politics classification).
- Detecting spam emails (spam vs legitimate email classification).
- Recognizing the object depicted in an image out of 1000 different objects (object recognition).

The learning algorithm is usually provided with a set of **examples** $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}\}$, $\boldsymbol{x}^{(j)} \in \Re^N \; \forall j$ and a set of corresponding **labels** $\{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}, y^{(j)} \in \{1, \ldots, K\} \forall j$ that specify which of the K categories each example belongs to. For instance, if $y^{(j)} = 3$, then $x^{(j)}$ belongs to class "3".

In the case of binary classification (e.g., spam vs non-spam), $y^{(j)} \in \{0,1\}$.

To solve this task, the machine learning algorithm takes the form of a function:

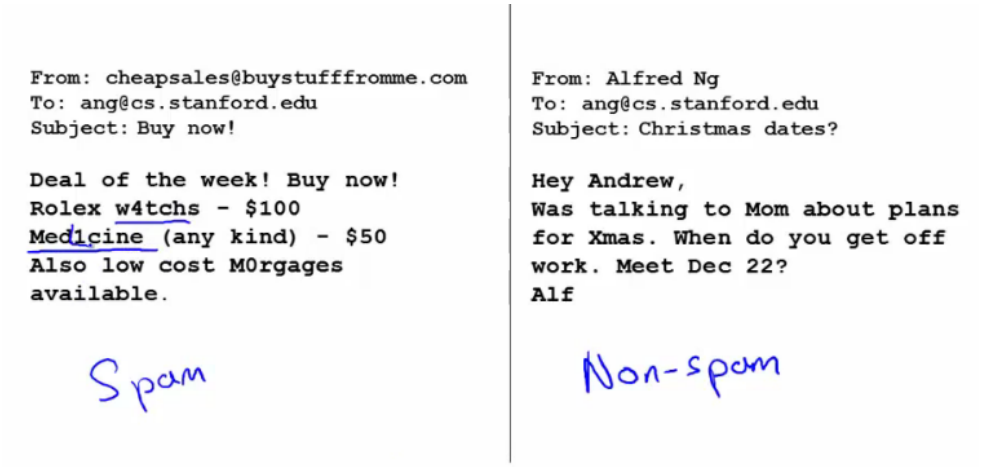
$$\mathbf{h}: \Re^n \rightarrow \{1, \ldots K\}$$

such that

$$\mathrm{y}^{(\mathrm{j})} = \mathbf{h}(\boldsymbol{x}^{(j)})$$

There are other variants of the classification task in which the algorithm produces a probability over all possible classes.

EXAMPLE



- Classification task: given an e-mail, classify it as spam or non-spam.
- Input: n-dimensional examples $\boldsymbol{x} = (x_1, \ldots, x_n)$ containing the features of the email, such as the number of orthographic errors and the occurrence of specific words.

- Output: labels $y \in \{0,1\}$ indicating if the e-mail is legitimate or spam.

REGRESSION TASKS

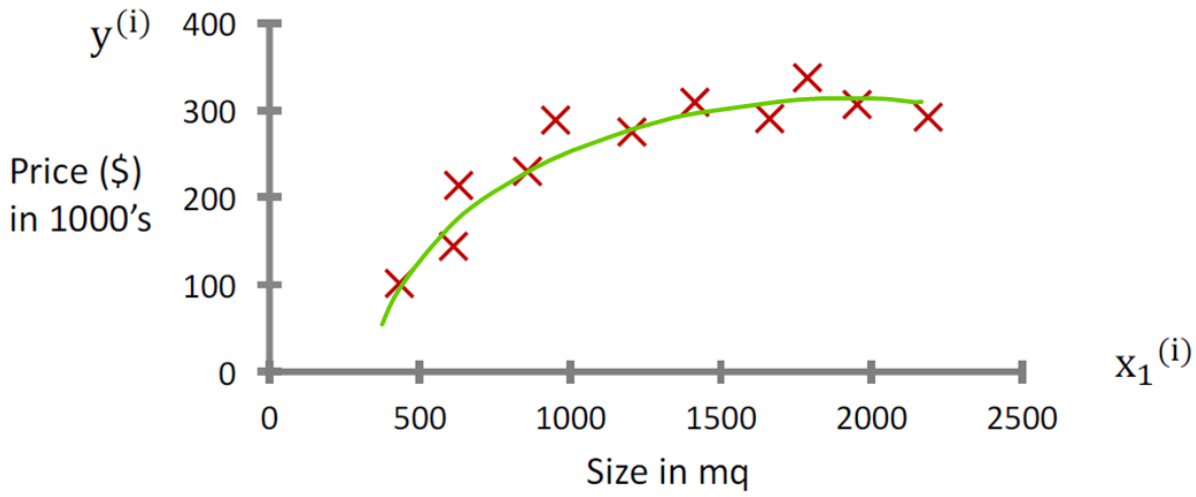
In this type of task, the computer program is asked to predict a numerical value given some input. Examples are:

- Predict the price of houses given some characteristics such as the city, the age, the area, etc.
- Predict the future value of a company stock from the values of other companies or other statistics about the market (stock market prediction).
- Count the number of cars present in an image.

Similar to classification, the algorithm is provided with training examples $\boldsymbol{x} \in \mathfrak{R}^n$ and with desired outputs $y \in \mathfrak{R}$.

The machine learning algorithm takes the form of a function $\mathbf{h}: \mathfrak{R}^n \rightarrow \mathfrak{R}$ such that $y^{(j)} = \mathbf{h}(\boldsymbol{x}^{(j)})$.

EXAMPLE



- Regression task: Predict the price of a house from its size in square meters.
- Input: the size of the house x (a scalar value).
- Output: the price y.

SUPERVISED VS UNSUPERVISED MACHINE LEARNING

Machine learning approaches can be roughly divided into supervised and unsupervised learning approaches.

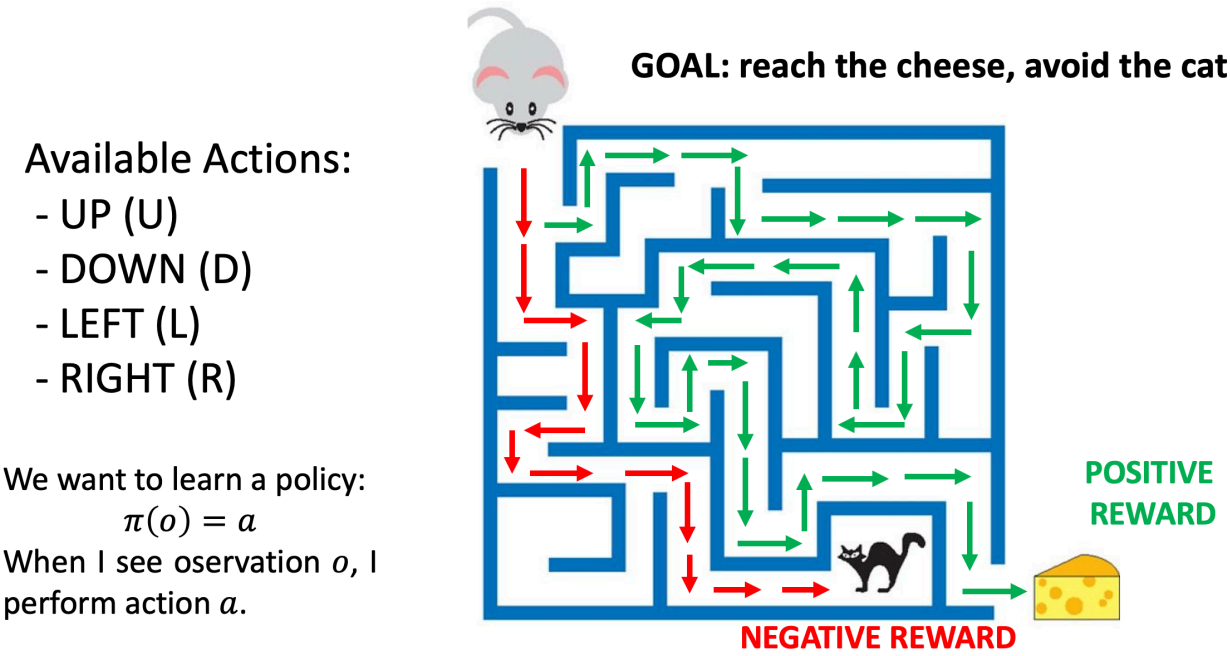
In **supervised** learning, the algorithm is provided with **input examples and desired outputs**. In this case, the task can be solved by **fitting a function** to the data, i.e., creating a function which will return those values. Examples of supervised machine learning tasks are classification and regression.

In **unsupervised** learning, the algorithm is provided only with **input examples** and no **label or desired output is provided** (hence only $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}\}$, $\boldsymbol{x}^{(j)} \in \Re^N \ \forall j$ values are provided). These types of tasks generally aim to model the structure of the data. Clustering is an example of unsupervised learning, in which no additional information is provided apart from the examples.

Supervised approaches are generally easier to deal with, but they require the presence of labels. Obtaining labels is often a time-expensive problem, as it requires people to manually annotate data. For instance, if we need to build a spam-detector using a supervised approach, we need someone to manually flag several emails as 'spam' or 'non-spam'.

REINFORCEMENT LEARNING

Some authors also refer to a third class of machine learning algorithms: reinforcement learning. Reinforcement learning aim to discover the solution to a problem by trial and error, rather than being explicitly instructed on how the task should be solved. This is done by letting the algorithm interact with an environment and receive positive rewards when they take actions which lead to a good outcome (with respect to the problem to solve) and negative rewards when they take actions which lead to a bad outcome. The goal of reinforcement learning algorithms is to learn a policy π which can be used to understand which action a to take when a given observation of the world o is acquired. This process resembles the natural process by which animals learn to solve problems. For instance, think of a mouse which must find the exit to a maze.



THE PERFORMANCE MEASURE: P

To evaluate the abilities of a machine learning algorithm to solve the considered task, we need a quantitative measure of its performance. Usually, this performance measure P is specific to the task T being carried out by the system.

For tasks such as classification, we often measure performance using the **accuracy**, that is the percentage of examples which have been classified correctly by the model. In the case of regression, we might use other measures such as the mean squared error. We will talk more about task-specific performance measures later.

Performance measure is used for two main reasons:

- To understand when a machine learning algorithm is improving on a given task.
- To evaluate the performance of the machine learning algorithm once it has been finalized.

A performance measure can also be seen in terms of error. For instance, an accuracy corresponds to an error rate (the percentage of misclassified examples) computed as 1-accuracy.

EXAMPLE

For example: a spam detector is presented with five emails. The first 3 emails are spam, the rest is legitimate. The algorithm classifies the first two as spam and the rest as legitimate. In this example, the first and last two predictions are correct, whereas the one in the middle is wrong. As a measure of performance, we can report the accuracy, i.e., the percentage of correctly classified examples: $\frac{4}{5} = 0.8$ or alternatively 80%.

THE EXPERIENCE: E

A machine learning algorithm learns from experience to improve a performance measure on a given task.

The experience is a collection of examples $\boldsymbol{x}^{(j)}$ (also known as "data points", as they can be mapped with some representation function to a multi-dimensional space), possibly with the related labels $y^{(i)}$ (depending on the task).

There are two main types of machine learning algorithms:

- Supervised approaches (when we have the paired labels, e.g., classification and regression).
- Unsupervised approaches (without paired labels, e.g., clustering).

The experience takes a different form depending on the type of machine learning approach we are using.

DATASET

Performance measures are generally computed with respect to set of examples, rather than to single examples. A set of examples (possibly with labels) is called **a dataset**. Datasets are generally homogenous, in the sense that the data therein contained has a similar format. For instance, in the Fisher's Iris dataset, all examples have 4 features and one label related to the three classes. In a dataset of images of food, each image is labeled with a class indicating the specific dish.

DESIGN MATRIX

A common way to represent a dataset is by using a **design matrix**. Since each example is a collection of n features, a dataset of m elements can be represented using a matrix $\boldsymbol{X} \in \Re^{m \times n}$ of dimension $[m \times n]$. Each example is a row of the design matrix, whereas each column represents one of the features.

In the case of supervised learning, another matrix $\boldsymbol{Y} \in A^{m \times k}$ is often considered, where k is the dimensionality of the desired outputs. For instance, in the case of classification, $A = \{1, \dots, M\}$ (M number of classes), and k is often equal to 1.

j^{th} feature

	2.2	4.7	3.8	9.2	4.7	3.2	-1.2	8.9	-0.11	4.12
	-0.11	-1.2	-0.11	-1.2	-0.11	4.7	-1.2	4.7	-0.11	9.2
	9.2	-0.11	9.2	4.12	9.2	9.2	-1.2	6.9	-1.2	9.2
	9.2	9.2	-0.11	-1.2	4.12	-8.6	9.2	-0.11	4.7	9.2
X=	-0.15	9.2	-1.2	-0.11	4.7	-0.11	-1.2	8.7	9.2	-87.5
i^{th} example	-0.11	-1.2	4.7	4.7	9.2	4.7	4.12	9.2	9.2	-1.2
	9.2	-0.11	9.2	9.2	-1.2	4.7	4.7	-0.11	-1.2	-0.11
	9.2	-0.11	4.12	-1.2	32.5	-1.2	9.2	9.8	4.12	9.2

$Y =$

2
3
8
2
1
0
8
1

EXAMPLE

Let's suppose we have a dataset with 1000 emails of spam and not spam images. Let's assume that all emails are represented with two features as discussed in the previous examples. The design matrix representing the dataset is a matrix $\boldsymbol{X} \in \Re^{1000\times 2}$. Each element of the matrix is one of the features of one of the examples of the dataset. For instance, $X_{i,1}$ is the *number of orthographic errors* of the i^{th} example, while $X_{j,2}$ is the *number of occurrences of keywords* of the j^{th} example etc. The labels are contained in a vector $Y \in \{0,1\}^{1000}$. For instance, Y_i is the label for the i^{th} example.

LEARNING

A machine learning algorithm uses a dataset of examples to improve its performance on a given task. The process of 'improving the performance' of the machine learning algorithm is called **learning** or **training**.

How does a machine learning algorithm learn? What does the training procedure consist of?

- A machine learning algorithm has some settings called '**parameters**', which can be adjusted to modify its behavior. **The parameters are related to a model (a mathematical function) to be used to solve the task.**
- An algorithm called "training procedure" uses the examples to find values for the parameters.
- Some of the parameters cannot be adjusted automatically by the training procedure. These parameters are called **hyperparameters** and they need to be optimized outside the training procedure, often by trial and error. We will see some examples later.

EXAMPLE

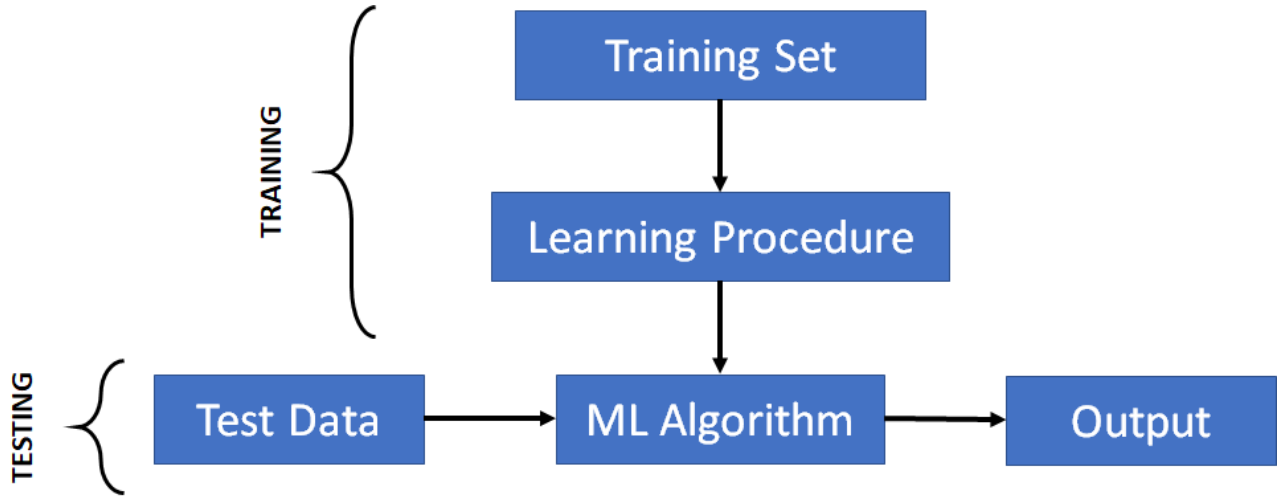
As a simple example, let's consider a spam detector which classifies emails as spam/non-spam based on the number of orthographic errors. The algorithm can be written as:

```
def classify(x):
    if x>θ:
        return 1
    else:
        return 0
```

The algorithm depends on a **single parameter $\boldsymbol{\theta}$**. What value should we set θ to? The training procedure allows to find a suitable value for θ.

A simple training procedure would consist in **trying many values for $\boldsymbol{\theta}$** and recording the performance of the algorithm when a specific value for θ is set. In the end, we can **choose the value of $\boldsymbol{\theta}$ which maximizes the performance** measure P.

THE TRAINING/TEST PARADIGM



A training procedure usually attempts to optimize the performance of the algorithm on the available data. However, in general, there is not guarantee that, after training, the algorithm will perform well on data which has never been seen.

In practice, we generally rely on two sets of data:

- The **training set**, used for training.
- The **test set**, used to evaluate the performance after training.

We hence perform two processes:

- **Training**: we use the learning procedure to optimize the parameters of the machine learning algorithm using the training set. This allows to instantiate a machine learning algorithm, which can be used to process new data.
- **Testing**: we use the trained ML algorithm to process the test data and evaluate the performance of the algorithm on the test data. We also call this process **evaluation**.

TRAINING AND TEST SETS

Ideally, we should train the algorithms on a set of data and evaluate them in the real world. However, this can create some problems, as we need the training and test data to be homogeneous and the evaluation process to be repeatable.

A convenient way to "simulate" the existence of two sets of data consists in **randomly** splitting the dataset into a training and a test set.

It is important to split the dataset **randomly** because we do not want to insert any bias in the training/test mechanism:

- For instance, the data might have been acquired following a specific order (e.g., class by class).
- Since our learning algorithm should work under «in the real world», we want to be independent any kind of order.

TRAINING, VALIDATION AND TEST SETS

As we mentioned, machine learning algorithms also have hyperparameters which are not optimized during training. To tune these parameters, we can divide the dataset into three sets, usually referred to as **training, validation, and test sets**. The split is always performed **randomly**. When tuning many parameters, it is important to have a validation set.

Indeed, we want to make sure that the algorithm works on data **which has never been seen during training**. If we tune our parameters on the test set, we are not sure what would happen when another set of data is provided.

CROSS-VALIDATION

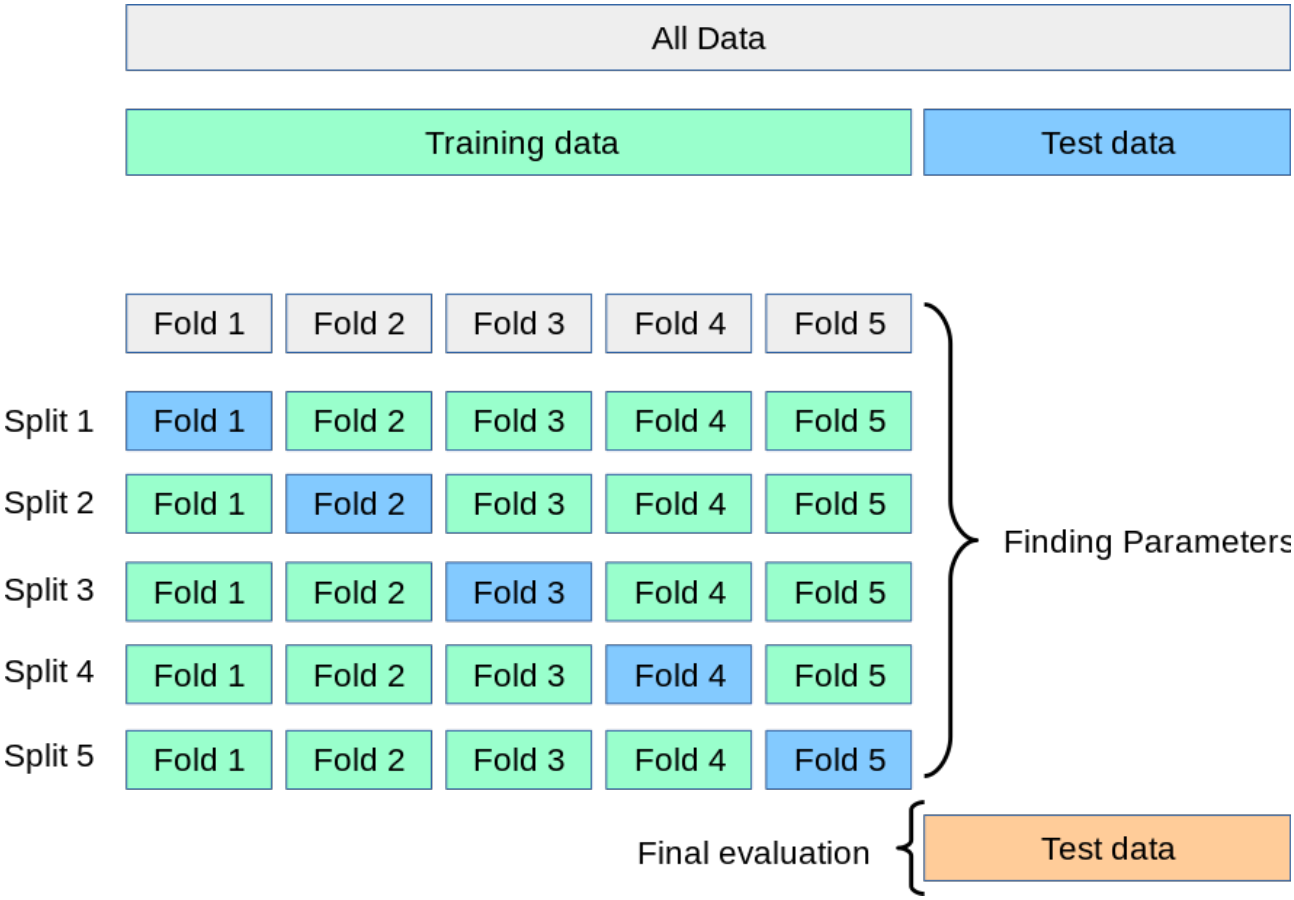


Image from: https://scikit-learn.org/stable/modules/cross_validation.html

When we have a small dataset, splitting it in two or three parts to have a training/validation/test split can greatly reduce the amount of training data.

In these cases, we solve the problem using cross-validation. This consists in:

- Splitting the data into training and test set.
- Further splitting the training set into non-overlapping subsets called 'folds' (e.g., 5).
- Then we will choose one of the folds as the validation fold (e.g., fold 5).
- The rest of the folds will be merged and used for training.
- The process can be repeated for k times (where k is the number of folds).
- Once the best choice of hyperparameters across all folds has been identified, the model can be re-trained on the whole training set and tested on the test data.

This procedure can be very time-consuming, but it works well when the dataset is not too big and provides a good estimate of the model's hyperparameters.

UNDERFITTING AND OVERFITTING

The central challenge in machine learning is that our algorithm must perform well on new, previously unseen inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called **generalization**.

We can use the training and test sets to measure **generalization**. We usually compute two scores:

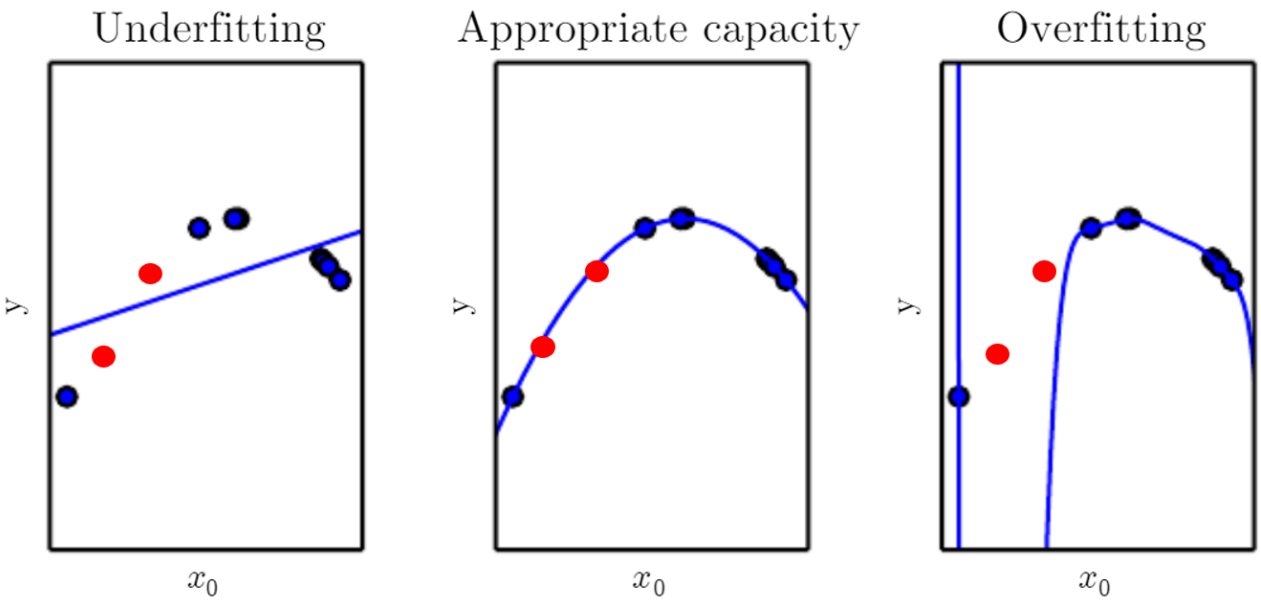
- A **training error**, i.e., the error obtained on the training set-
- A **test error or generalization error**, i.e., the error obtained on the test set.

Typically, we want both the training and test errors to be low. However, two pathological conditions are very common:

- **Underfitting**: the model obtains a large training and test error.
- **Overfitting:** the model obtains a small training error and a large test error.

These two conditions are influenced by the **capacity** of the model. Intuitively, the concept of **capacity** is related to the ability to fit many functions. For instance, a model which can compute only linear functions has a lower capacity than a model which can also model non-linear functions.

- Models with low capacity may struggle to fit the training set.
- Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.



The image shows an example of regression. The blue points are the training data generated using a quadratic function. We train three models on the data:

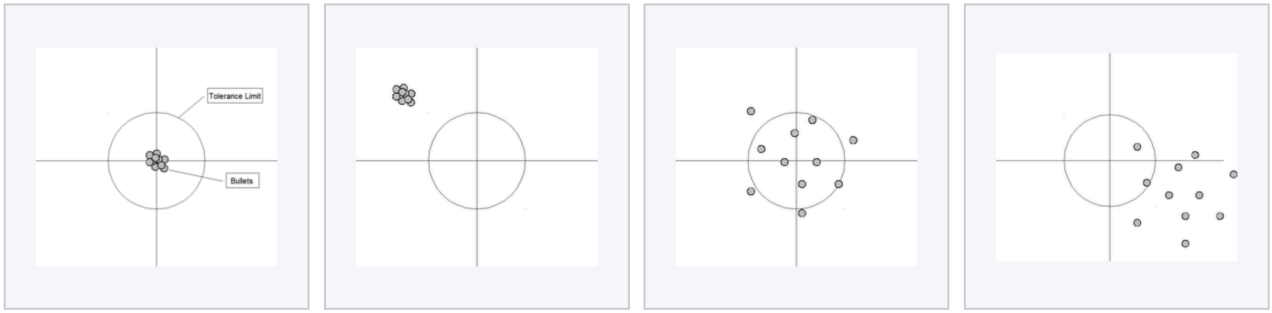
- The first model on the left is linear and has limited capacity. This results in underfitting, obtaining low performance both on the training set and on the test set (the red dots).
- The second model has appropriate capacity and recovers a good model for the data.
- The third model has a larger capacity and overfits the data. The model fits well the training set but does not generalize on the test data (the red dots).

BIAS AND VARIANCE

Closely related to underfitting and overfitting are the concepts of bias and variance. Once a machine learning algorithm is trained, its predictions can suffer from two main problems:

- **Bias Error**: predictions are not correct due to wrong assumptions in the model. High bias can cause the model to miss important relations between features and target outputs, leading to underfitting. For instance, think of a model which tries to approximate a polynomial function with a linear one.
- **Variance Error**: model predictions are highly influenced by small fluctuations in the training set. For instance, if we remove a data point and re-train the algorithm, we obtain very different predictions. This may be due to

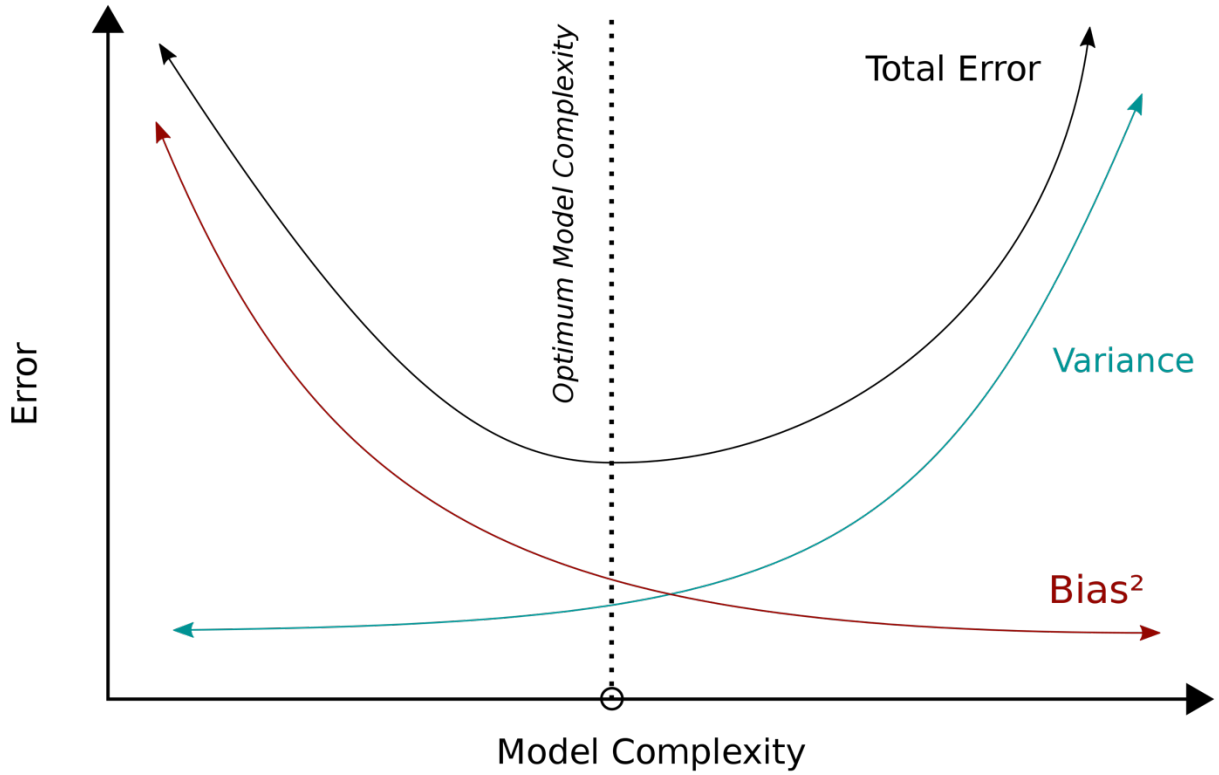
the fact that the algorithm is trying to model the random noise present in the training data, and usually leads to overfitting.



bias low, variance low | bias high, variance low | bias low, variance high | bias high, variance high

From Wikipedia (https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff).

Similarly to overfitting and underfitting, bias and variance are influenced by model capacity (or complexity). Hence a good model is one that achieves a good bias-variance trade-off.



From Wikipedia (https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff).

REFERENCES

- Parts of chapter 1 of [1].
- The relevant parts of chapter 3 of [2].

[1] Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. https://www.deeplearningbook.org/