Lecture 9: Machine Learning Algorithms

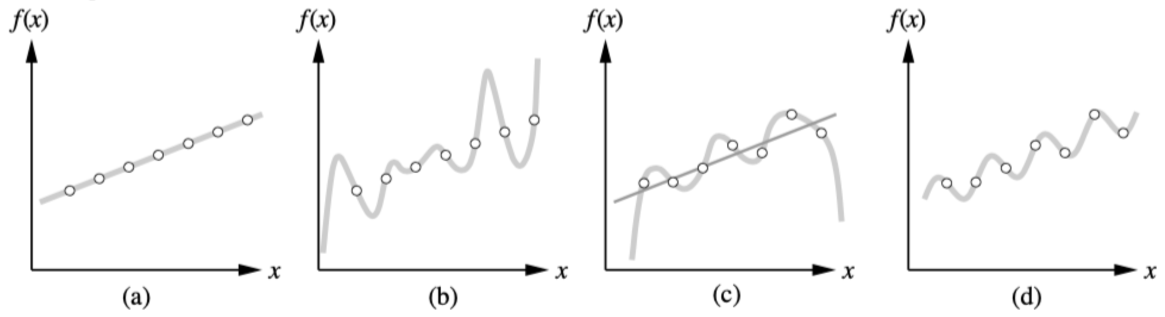
Machine learning is a sub-field of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Machine learning is not about learning in general (like humans), but about learning a specific/specific task in a targeted manner (e.g., classify nationality, or predict stock price etc.). This task is intended to improve the system over time.

A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. (Mitchell, 1997)

Hypothesis space:
A hypothesis that fits a set of training examples, is called the consistent hypothesis.

Examples:



Set of polynomials as hypothesis space for (a) and (b) and two consistent hypotheses.
How do we choose from multiple consistent hypotheses?

In the example above, consider case (c). It would be better to find a simple straight line that isn't exactly consistent but that allows for meaningful predictions.

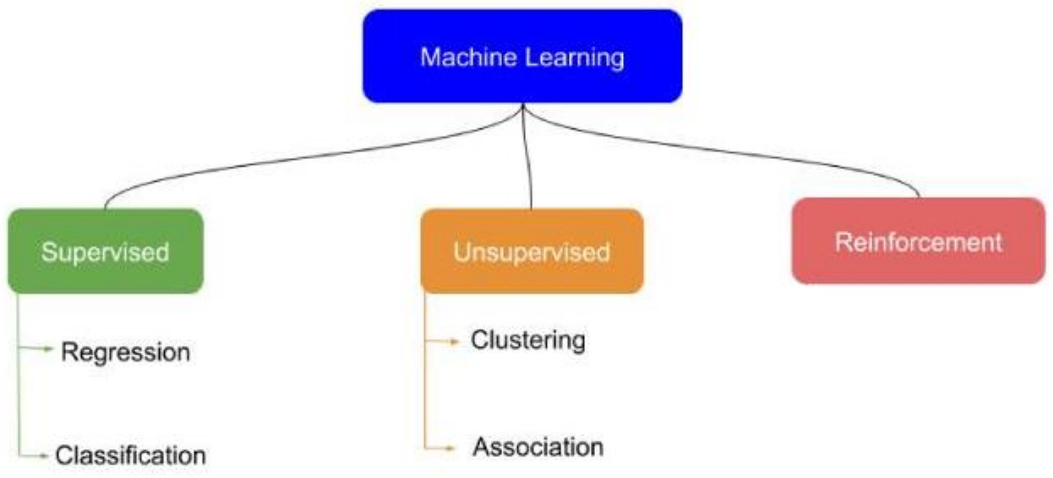
For non-deterministic functions there is an unavoidable trade-off between the complexity of the hypothesis and its degree the agreement with the data.

Example (d) shows that the data from (c) are consistent with a simple function of the form a*x + b + c * sin x.

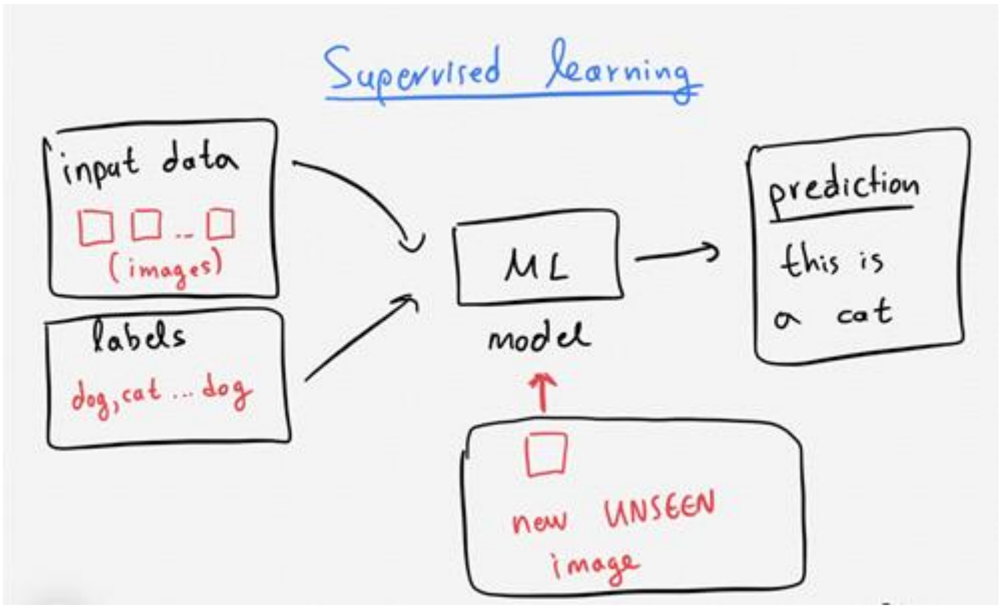
⇨ Importance of choosing the hypothesis space

Types of Machine Learning

Machine Learning is a very vast subject and every individual field in ML is an area of research in itself. The subject is expanding at a rapid rate due to new areas of studies constantly coming forward. In this section, we will be looking at three popular types of ML.



Supervised Learning:



For this family of models, the research needs to have at hand a dataset with some observations and the labels/classes of the observations. For example, the observations could be images of animals and the labels the name of the animal (e.g. cat, dog etc).

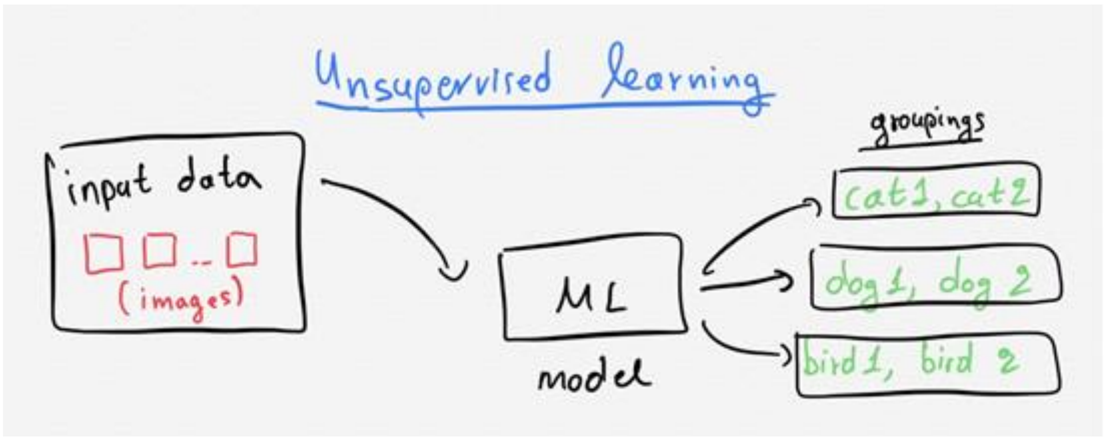
These models learn from the labeled dataset and then are used to predict future events. For the training procedure, the input is a known training data set with its corresponding labels, and the learning algorithm produces an inferred function to finally make predictions about some new unseen observations that one can give to the model. The model is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct intended output (ground truth label) and find errors in order to modify itself accordingly (e.g. via back-propagation).

Supervised models can be further grouped into regression and classification cases:

- **Classification**: A classification problem is when the output variable is a category, e.g., "disease" / "no disease".
- **Regression**: A regression problem is when the output variable is a real continuous value, e.g., stock price prediction

Some examples of models that belong to this family are the following: SVC, LDA, SVR, regression, random forests etc.

Unsupervised Learning:



For this family of models, the research needs to have at hand a dataset with some observations without the need of having also the labels/classes of the observations.

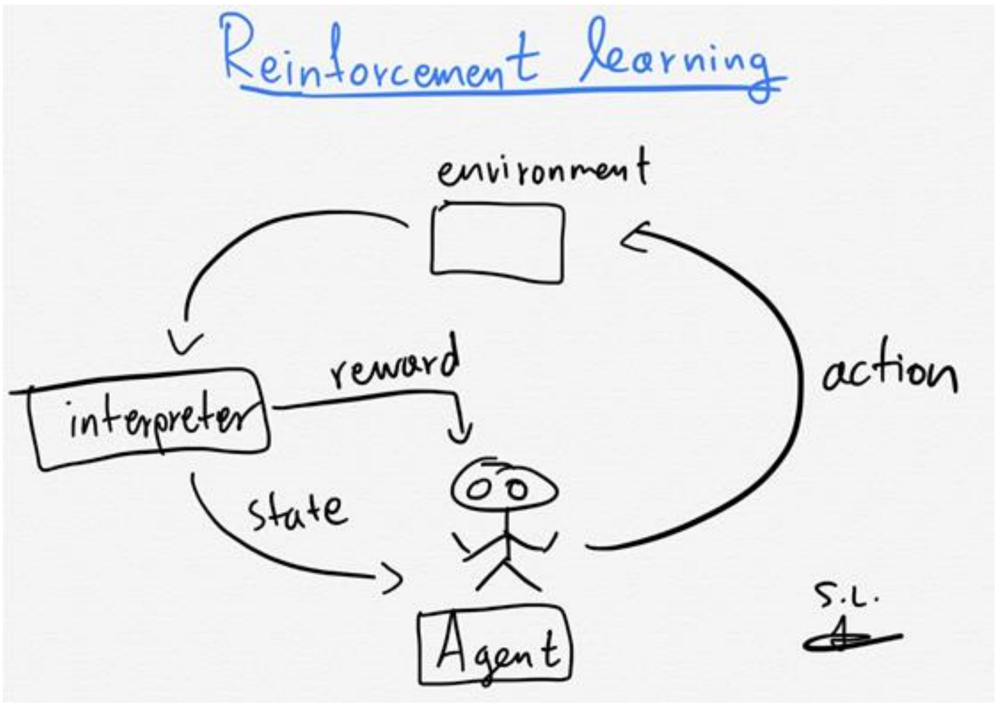
Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't predict the right output, but instead, it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Unsupervised models can be further grouped into clustering and associationcases.

- **Clustering**: A clustering problem is where you want to unveil the inherentgroupings in the data, such as grouping animals based on some characteristics/features e.g. number of legs.
- **Association**: An association rule learning is where you want to discover association rules such as people that buy X also tend to buy Y.

Some examples of models that belong to this family are the following: PCA, K-means, DBSCAN, mixture models etc.

Reinforcement Learning:



This family of models consists of algorithms that use the estimated errors as rewards or penalties. If the error is big, then the penalty is high and the reward low. If the error is small, then the penalty is low and the reward high.

Trial error search and delayed reward are the most relevant characteristics of reinforcement learning. This family of models allows the automatic determination of the ideal behavior within a specific context in order to maximize the desired performance.

Reward feedback is required for the model to learn which action is best and this is known as "the reinforcement signal".

Some examples of models that belong to this family is the Q-learning.

Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the *scale* of the numbers could cause problems when you attempt to combine the values as features during modeling.

Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

This module offers several options for transforming numeric data:

- You can change all values to a 0-1 scale, or transform the values by representing them as percentile ranks rather than absolute values.
- You can apply normalization to a single column, or to multiple columns in the same dataset.

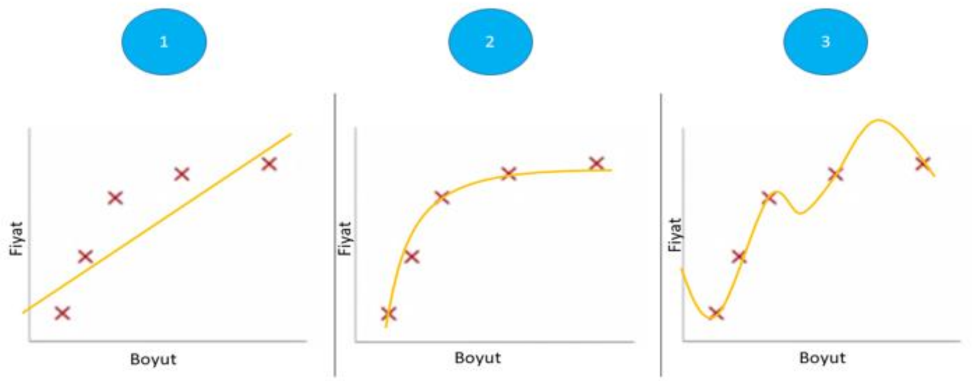
- If you need to repeat the experiment, or apply the same normalization steps to other data, you can save the steps as a normalization transform, and apply it to other datasets that have the same schema.

Overfitting and Underfitting

If the model has been underfitting, it has failed to grasp the underlying logic of the data. The model does not know what to do with this data and gives inaccurate results. In the other case, that is, if the model is overfitted, it overfits the dataset and misses the actual case.

Let us illustrate these two concepts in regression and classification problems.

Consider a regression problem that finds the price of houses according to their size.



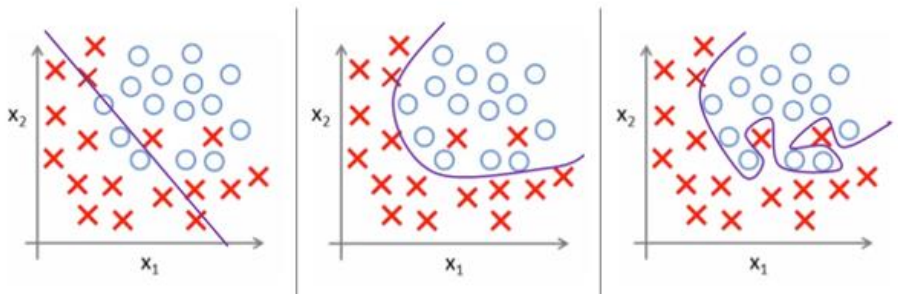
In the first graph, we see a linear pattern. The model predicts that the price will always increase as the size increases. But when we look at the dots on the chart, we see that the price can stay stable as the size increases. In this case the model has been underfitting and the error is quite high.

In the second graph, the model is the quadratic function. We can see the fit of this function with the data. This model can be successful in training the data.

In the final chart, the model appears to fit very well with the data. But there are so many ups and downs. We can think of this as if the model forced itself to fit the data. But this model is failing and overfitting because what we want is that the model doesn't just work well on this dataset.

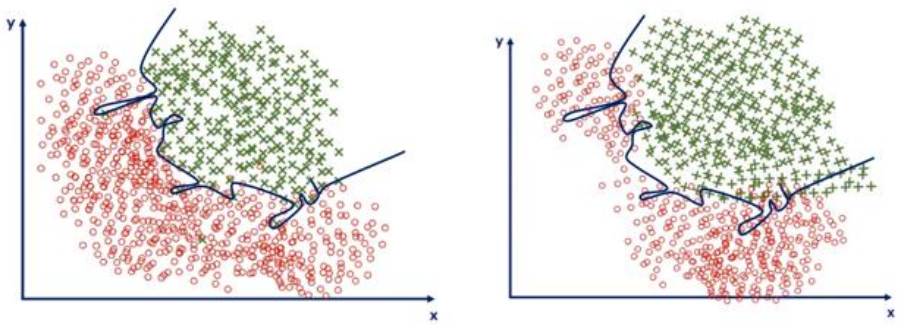
The point that makes overfitting important is that the error is very close to zero and sometimes even zero. This is the point that misleads us. We think the model is very good, but actually it has been overfitted.

Let's examine the case of overfitting for the classification problem.



In the first case the model has been underfitting. The second case is desirable, although it contains some error. However, the third case was overfitting. It just works fine with this data layout.

The graphs below show what kind of a problem an overfitted model will cause for different datasets.

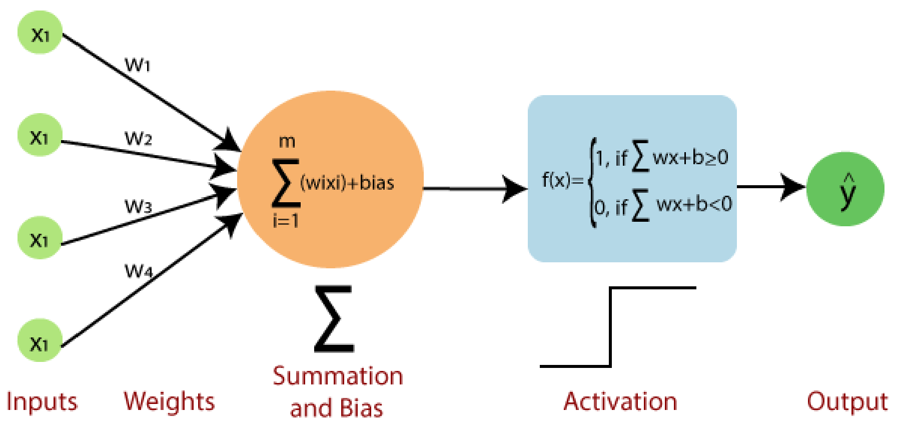


Artificial Neural Networks

The single layer perceptron consists of 4 parts.

- **One input layer:** The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
- **Weights and Bias:**
  Weight: It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
  Bias: It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
- **Net sum:** It calculates the total sum.
- **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.

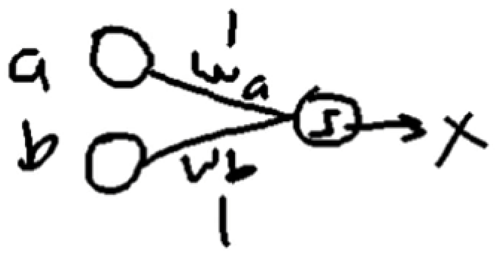
A regular single perceptron neural network looks like this:


X1
X1
X1
X1
W1
W2
W3
W4
$\sum_{i=1}^{m}$(wixi)+bias
$f(x)=\begin{cases}1, \text{if} \sum wx+b\geq 0\\ 0, \text{if} \sum wx+b<0\end{cases}$
ŷ
Inputs
Weights
Summation and Bias
Activation
Output

AND Example with Single Layer Perceptron:

For simplicity reasons, we do not consider the bias factor.

a	**b**	**x**
0	0	0
0	1	0
1	0	0
1	1	1



Step function:

1 if $a*w_a + b*w_b > t$

0 otherwise

0*1 + 0*1 > 5 ➔ 0

0*1 + 1*1 > 5 ➔ 0

1*1 + 0*1 > 5 ➔ 0

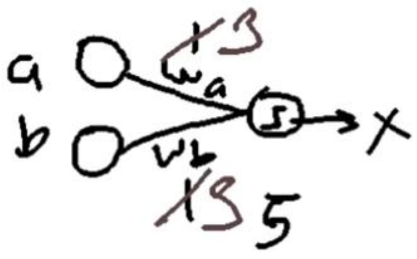
1*1 + 1*1 > 5 ➔ 0 -> is not consistent with the training data. We need a 1 here !

$$\textit{Our expected value is } 1$$

$$\textit{The actual output value} \rightarrow 0 => \ \Delta = 1$$

$$\textit{The weights must be changed according to learning rate } \alpha, e.g.\ \alpha = 2$$

$$\alpha * \ \Delta = 2 * 1 = 2 \ (\textit{this value will be added to the previos weight})$$



0*3 + 0*3 > 5 ➔ 0

0*3 + 1*3 > 5 ➔ 0

1*3 + 0*3 > 5 ➔ 0

1*3 + 1*3 > 5 ➔ 1 -> Now, it is consistent with the training data!

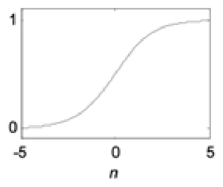
What would happen if we would select $\alpha = 1$?

For the example, we used a simple threshold value (step function) with value = 5. Numbers smaller than 5 give 0, numbers greater than 5 gives 1.

Examples for further activation functions:

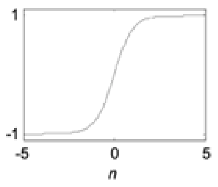
Sigmoid

$$y = \frac{1}{1+e^{-net}}$$



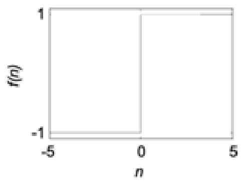
Hiperbolik tanjant

$$y = \frac{1-e^{-2net}}{1+e^{2net}}$$

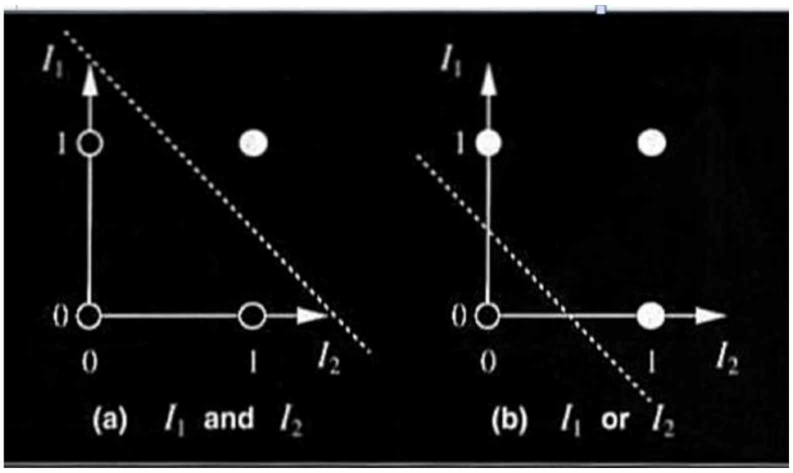


Adım basamak

$$y = \begin{cases} 1 & net >= 0 \\ -1 & net < 0 \end{cases}$$



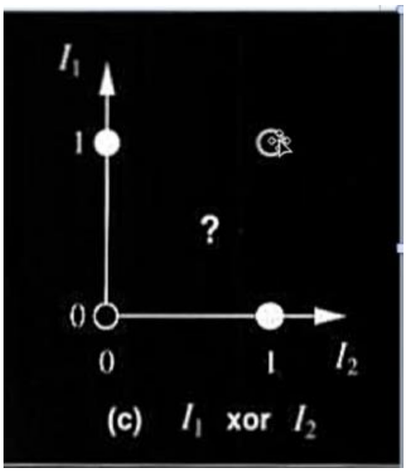
Linearly separable problems: AND, OR etc. Values can be distinguished from each other by a linear line in a two-dimensional chart. More than one line can also distinguish these values from each other.



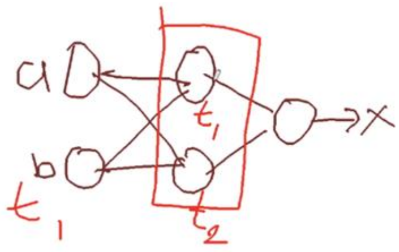
XOR Example with Multilayer Perceptron:

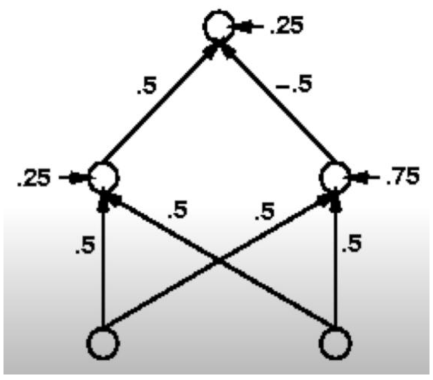
Multilayer perception is most widely recognized type of ANNs and consists of one input and output layers along with one or more hidden layers. The ANN models work by creating a nonlinear relationship between dependent and independent variables depending on a set of experimental data.

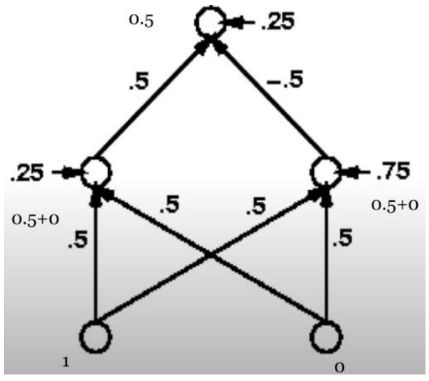
a	b	x
0	0	0
0	1	1
1	0	1
1	1	0

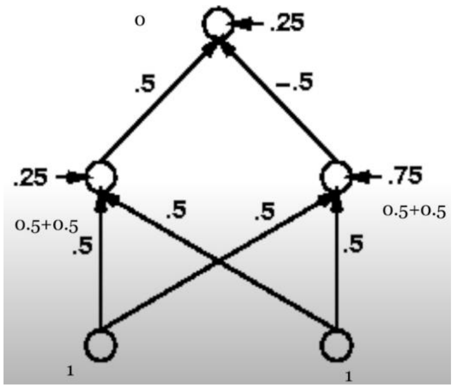


XOR is not a linearly separable problem. In such cases, we need **<u>hidden layers</u>**.



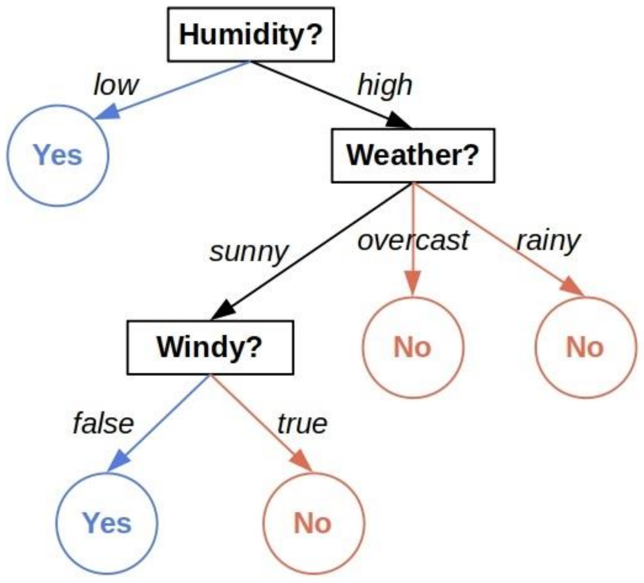






Decision Trees

A decision tree is a tree-shaped plan of checks we perform on the features of an object before making a prediction. For example, here's a tree for predicting if the day is good for playing outside:



Each internal node inspects a feature and directs us to one of its sub-trees depending on the feature's value and the leaves output decisions. Every leaf contains the subset of training objects that pass the checks on the path from the root to the leaf. Upon visiting it, we output the majority class or the average value of the leaf's set.

However, trees are unstable. Slight changes to the training set, such as the omission of a handful of instances, can result in totally different trees after fitting. Further, trees can be inaccurate and perform worse than other machine-learning models on many datasets.

Random Forests

A random forest is a collection of trees, all of which are trained independently and on different subsets of instances and features. The rationale is that although a single tree may be inaccurate, the collective decisions of a bunch of trees are likely to be right most of the time.

For example, let's imagine that our training set $\mathcal{D}$ consists of 200 instances with four features: A, B, C, and D. To train a tree, we randomly draw a sample $\mathcal{S}$ of instances from $\mathcal{D}$. Then, we randomly draw a sample of features. For instance, A and C. Once we do that, we fit a tree to $\mathcal{S}$ using only those two features. After that, we repeat the process choosing different samples of data and features each time until we train the desired number of trees.

Forests are more robust and typically more accurate than a single tree. But, they are harder to interpret since each classification decision or regression output has not one but multiple decision paths. Also, training a group of m trees will take m times longer than fitting only one. However, we can train the trees in parallel since they are by construction mutually independent.

Gradient boosting trees

Gradient boosting is a widely used technique in machine learning. Applied to decision trees, it also creates ensembles. However, the core difference between the classical forests lies in the training process of gradient boosting trees. Let's illustrate it with a regression example (the x_i are the training instances, whose features we omit for brevity):

<u>The first tree:</u>

First, we train a decision tree (f_1) using all the data and features. Then, we calculate its predictions $f_1(\cdot)$ and compare them to the ground truth y :

x	y	$f_1(x)$	$y - f_1(x)$
x_1	10	9	1
x_2	11	13	-2
x_3	13	15	-2
x_4	20	25	-5
x_5	22	31	-9

The second tree:

As we see, the first tree seems off. How can we improve it? Well, an intuitive strategy is to fit another regressor $\boldsymbol{f_2}$ on the residuals $\boldsymbol{y - f(1)}$. If it's accurate, then $f_1(x) + f_2(x) \approx f_1(x) + y - f_1(x) = y$, and we'll get a precise model. To evaluate the adequacy of $f_1 + f_2$, we calculate the new residuals $y - f_1(x) - f_2(x)$:

x	y	$f_1(x)$	$f_2(x)$	$y - f_1(x) - f_2(x)$
x_1	10	9	0.5	0.5
x_2	11	13	1	-3
x_3	13	15	-1	-1
x_4	20	25	-2	-3
x_5	22	31	-4	-5

If were satisfied, we stop here and use the sequence of trees f_1 and f_2 as our model However, if the residuals $y - f_1(x) - f_2(x)$ indicate that the sequence $f_1 + f_2$ has a too high error, we fit another tree f_3 to predict $y - f_1(x) - f_2(x)$ and use the sequence $f_1 + f_2 + f_3$ as our model.

We repeat the process until we reduce the errors to an acceptable level or fit the maximum number of trees.

Gradient boosting trees can be more accurate than random forests. Because we train them to correct each other's errors, they're capable of capturing complex patterns in the data. However, if the data are noisy, the boosted trees may overfit and start modeling the noise.

The main differences between Gradient Boosting Trees and Random Forests

There are two main differences between the gradient boosting trees and the random forests.

- We train the former sequentially, one tree at a time, each to correct the errors of the previous ones. In contrast, we construct the trees in a random forest independently. Because of this, we can train a forest in parallel but not the gradient-boosting trees.

Cl

⇨ The other principal difference is in how they output decisions. Since the trees in a random forest are independent, they can determine their outputs in any order. Then, we aggregate the individual predictions into a collective one: the majority class in classification problems or the average value in regression. On the other hand, the gradient boosting trees run in a fixed order, and that sequence cannot change. For that reason, they admit only sequential evaluation.

Performance metrics for classification models

Classification errors are calculated using accuracy, precision, recall, and F1-score metrics, which are given below.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Accuracy value is calculated by the ratio of the classifications that we predict correctly in the model to the total data set. Model accuracy alone is not sufficient, especially in unbiased datasets that are not evenly distributed. For example, let's say we have a dataset of 100 patients with and without cancer. Only 10 of all patients were diagnosed with cancer. In such a case, we do not want to have patients with cancer that cannot be diagnosed (False Negative). Therefore, we should evaluate the results of other metrics together.

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

Precision (or Specifity), on the other hand, shows how many of the values we estimated as Positive are actually Positive.

The precision value is especially important when the cost of False Positive estimation is high. For example, if your model marks the mails that should arrive in your mailbox as spam (FP), then you will not be able to see the important mails you need to receive and you will be in a loss-making situation for you. In this case, having a high Precision value is an important criterion for model selection for us.

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

Recall (or Sensitivity) is a metric that shows how much of the operations we need to estimate as Positive, we estimate as Positive.

The recall value is also a metric that will help us in situations where the cost of estimating as False Negative is high. It should be as high as possible. For example, if a model we created for Fraud Detection marks a transaction that is Fraud as non-fraud, the consequences of such a situation will pose a problem for a bank.

$$F1 - \text{Score} = 2 \times \frac{PrecisionxRecall}{Precision + Recall}$$

F1 Score value shows us the harmonic mean of Precision and Recall values. The reason why it is a harmonic mean instead of a simple mean is that we should not ignore the extreme cases. If it were a simple average calculation, a model with a Precision value of 1 and a Recall value of 0 would have an F1 Score of 0.5, which would mislead us.

The main reason for using F1 Score value instead of Accuracy is not to make an incorrect model selection in unevenly distributed data sets. In addition, F1 Score is very important to us as we need a measurement metric that will include not only False Negative or False Positive but also all error costs.

Performance metrics for regression models

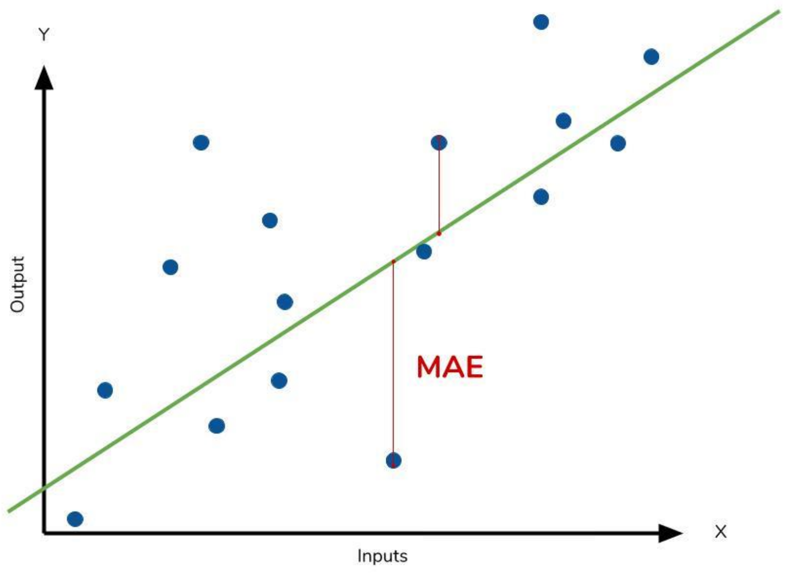
The MAE, MSE, and MAPE metrics are mainly used to evaluate the prediction error rates and model performance in regression analysis.

Mean absolute error

The mean absolute error (MAE) is the simplest regression error metric to understand. We’ll calculate the residual for every data point, taking only the absolute value of each so that negative and positive residuals do not cancel out. We then take the average of all these residuals. Effectively, MAE describes the *typical* magnitude of the residuals.

$$MAE = \overset{\text{Divide by the total number of data points}}{\boxed{\frac{1}{n}}} \underset{\text{Sum of}}{\sum} \underbrace{\left| \overset{\text{Actual output value}}{\boxed{y}} - \overset{\text{Predicted output value}}{\boxed{\hat{y}}} \right|}_{\text{The absolute value of the residual}}$$

The picture below is a graphical description of the MAE. The green line represents our model’s predictions, and the blue points represent our data.



The MAE is also the most intuitive of the metrics since we're just looking at the absolute difference between the data and the model's predictions. Because we use the absolute value of the residual, the MAE does not indicate underperformance or overperformance of the model (whether or not the model under or overshoots actual data). Each residual contributes proportionally to the total amount of error, meaning that larger errors will contribute linearly to the overall error. Like we've said above, a small MAE suggests the model is great at prediction, while a large MAE suggests that your model may have trouble in certain areas. A MAE of 0 means that your model is a perfect predictor of the outputs (but this will almost never happen).

While the MAE is easily interpretable, using the absolute value of the residual often is not as desirable as squaring this difference. Depending on how you want your model to treat outliers, or extreme values, in your data, you may want to bring more attention to these outliers or downplay them. The issue of outliers can play a major role in which error metric you use.

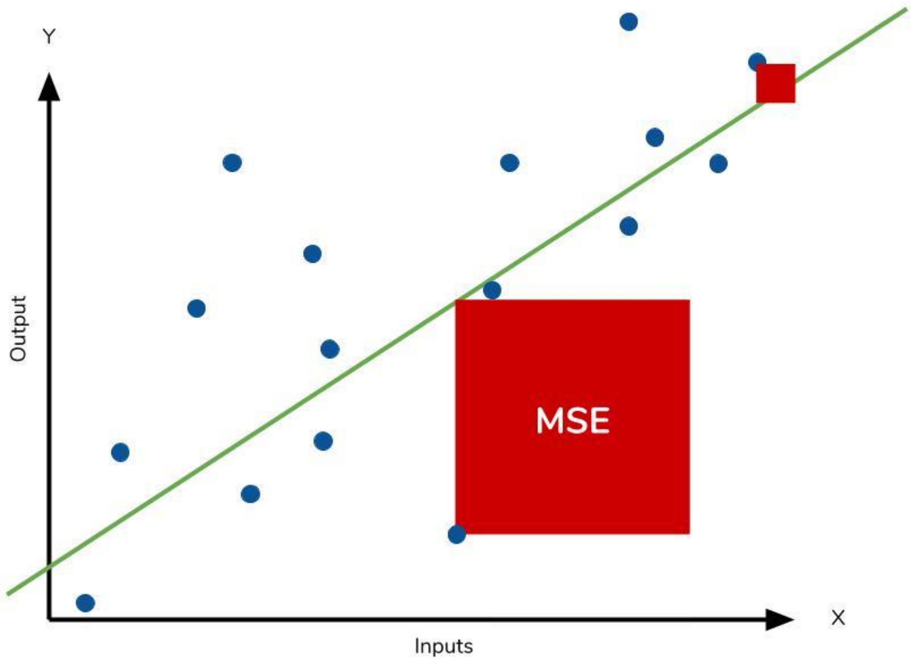
Mean square error

The mean square error (MSE) is just like the MAE, but *squares* the difference before summing them all instead of using the absolute value. We can see this difference in the equation below.

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Consequences of the Square Term

Because we are squaring the difference, the MSE will almost always be bigger than the MAE. For this reason, we cannot directly compare the MAE to the MSE. We can only compare our model's error metrics to those of a competing model. The effect of the square term in the MSE equation is most apparent with the presence of outliers in our data. While each residual in MAE contributes proportionally to the total error, the error grows quadratically in MSE. This ultimately means that outliers in our data will contribute to much higher total error in the MSE than they would the MAE. Similarly, our model will be penalized more for making predictions that differ greatly from the corresponding actual value. This is to say that large differences between actual and predicted are punished more in MSE than in MAE. The following picture graphically demonstrates what an individual residual in the MSE might look like.



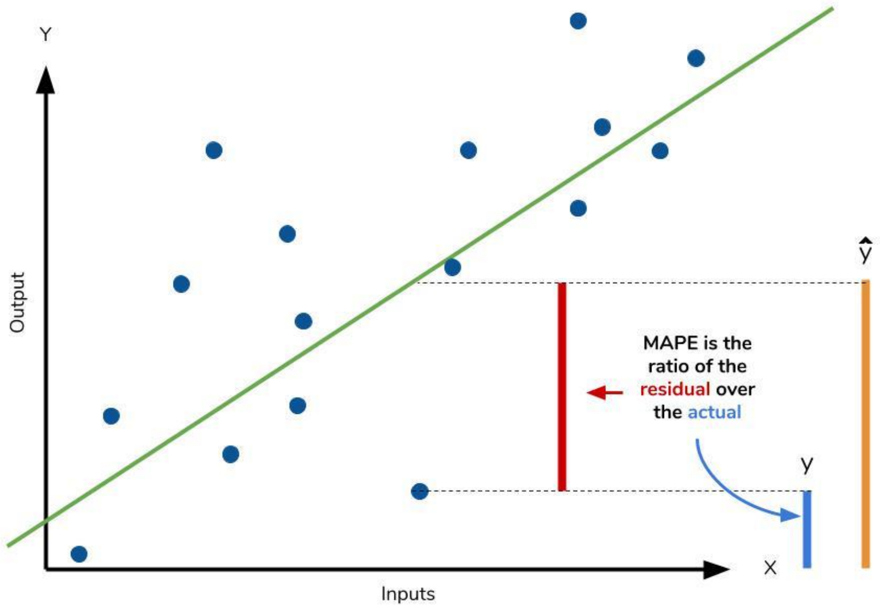
Outliers will produce these exponentially larger differences, and it is our job to judge how we should approach them.

Mean absolute percentage error

The mean absolute percentage error (MAPE) is the percentage equivalent of MAE. The equation looks just like that of MAE, but with adjustments to convert everything into percentages.

$$MAPE = \frac{\overset{\text{Multiplying by 100\% converts to percentage}}{\boxed{100\%}}}{n} \sum \left| \frac{\overbrace{y-\hat{y}}^{\text{The residual}}}{\underset{\text{Each residual is scaled against the actual value}}{\boxed{y}}} \right|$$

Just as MAE is the average magnitude of error produced by your model, the MAPE is how far the model's predictions are off from their corresponding outputs on average. Like MAE, MAPE also has a clear interpretation since percentages are easier for people to conceptualize. Both MAPE and MAE are robust to the effects of outliers thanks to the use of absolute value.


Y
Output
Inputs
X
ŷ
y
MAPE is the ratio of the residual over the actual