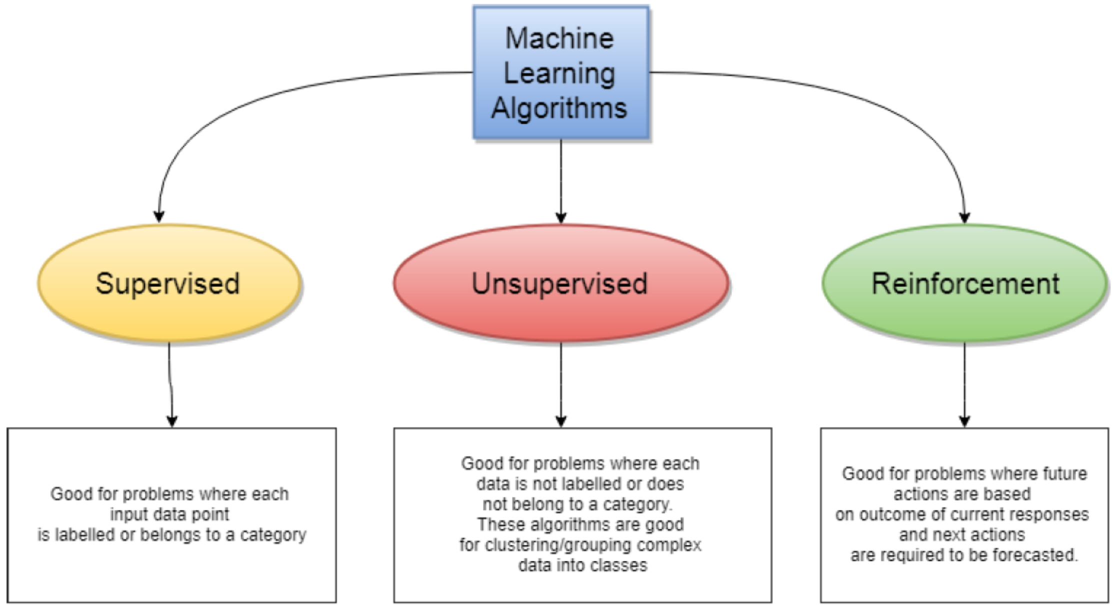
ΕΙΣΑΓΩΓΗ

στην

ΕΠΙΒΛΕΠΟΜΕΝΗ ΜΑΘΗΣΗ

(Supervised Learning)

Types of machine learning algorithms



Machine Learning Algorithms
Supervised
Unsupervised
Reinforcement
Good for problems where each input data point is labelled or belongs to a category
Good for problems where each data is not labelled or does not belong to a category. These algorithms are good for clustering/grouping complex data into classes
Good for problems where future actions are based on outcome of current responses and next actions are required to be forecasted.

https://medium.com/fintechexplained/machine-learning-algorithm-comparison-f14ce372b855

Types of machine learning algorithms

Supervised

Unsupervised

Reinforcement

Labelled data. Learn through examples of which we know the desired output (what we want to predict)

- ❖ Image classification
- ❖ Speech recognition
- ❖ Spam detection
- ❖ Face Recognition
- ❖ Weather forecasting

Supervised learning (training)

- We have a supervision data
  - $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ (N instances)
- Find parameters such that they can predict training instances as correctly as possible
- We assume *generalization*
  - If the parameters predict training instances well, they will work for unseen instances

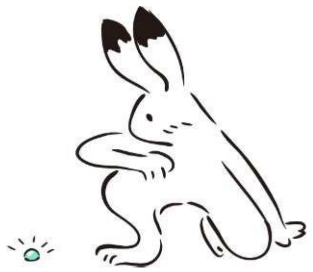
Supervised learning for single-layer NNs

- For simplicity, we include a bias term b in $\boldsymbol{w}$ hereafter
  - Redefine $\boldsymbol{x}^{(\mathrm{new})} = (x_1, x_2, \ldots, x_d, 1)^\intercal$, $\boldsymbol{w}^{(\mathrm{new})} = (w_1, w_2, \ldots, w_\mathrm{d}, b)^\intercal$
  - Then, $\boldsymbol{w}^{(\mathrm{new})} \cdot \boldsymbol{x}^{(\mathrm{new})} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$ (original form)

- We introduce a new notation to distinguish a computed output $\hat{y}$ from the gold output y in the supervision data
  - $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ (N instances)
  - We distinguish two kinds of outputs hereafter
    - $\hat{y}$: the output computed (predicted) by the model (perceptron) for the input
    - y: the true (gold) output for the input in the supervision data

- Training: find $\boldsymbol{w}$ such that,

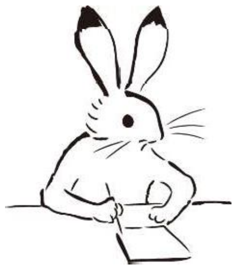
$$\forall n \in \{1, \ldots, N\}: g(\boldsymbol{w} \cdot \boldsymbol{x}_n) = y_n$$



```
1.  $\boldsymbol{w} = 0$
2.  Repeat:
3.      $(\boldsymbol{x}_n, y_n) \longleftarrow$ a random sample from $D$
4.      $\hat{y} \longleftarrow g(\boldsymbol{w} \cdot \boldsymbol{x}_{\boldsymbol{n}})$
5.      if $\hat{y} \neq y_n$ then:
6.          if $y_n = 1$ then:
7.              $\boldsymbol{w} \longleftarrow \boldsymbol{w} + \eta \boldsymbol{x}_n$
8.          else:
9.              $\boldsymbol{w} \longleftarrow \boldsymbol{w} - \eta \boldsymbol{x}_n$
10. Until no instance updates $\boldsymbol{w}$
```

η $(0 < \eta)$ is the learning rate

Exercise: Train an SLP to realize OR



- Convert the truth table into training data

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$\Rightarrow \quad D = \left\{ \begin{array}{l} \left((0\ 0\ 1)^\top, 0\right), \\ \left((0\ 1\ 1)^\top, 1\right), \\ \left((1\ 0\ 1)^\top, 1\right), \\ \left((1\ 1\ 1)^\top, 1\right) \end{array} \right\}$$

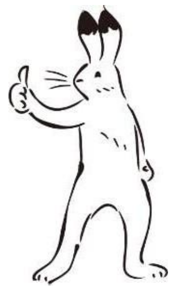
- Initialize the weight vector $\boldsymbol{w} = 0$
- Apply the perceptron algorithm to find $\boldsymbol{w}$
  - Fix $\eta = 1$ in the exercise

Updating weights for OR

- Data: $D = \{((0\ 0\ 1)^\intercal, 0), ((0\ 1\ 1)^\intercal, 1), ((1\ 0\ 1)^\intercal, 1), ((1\ 1\ 1)^\intercal, 1)\}$
- Initialization: $\boldsymbol{w} = (0\ 0\ 0)^\intercal$
- Iteration #1: choose $(\boldsymbol{x}_4, y_4) = ((1\ 1\ 1)^\intercal, 1)$
  - Classification: $\hat{y} = g(\boldsymbol{w} \cdot \boldsymbol{x}_4) = g(0) = 0 \neq y_4$
  - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}_4 = (1 1 1)^\intercal$
- Iteration #2: choose $(\boldsymbol{x}_1, y_1) = ((0\ 0\ 1)^\intercal, 0)$
  - Classification: $\hat{y} = g(\boldsymbol{w} \cdot \boldsymbol{x}_1) = g(1) = 1 \neq y_1$
  - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{x}_1 = (1 1 0)^\intercal$

We chose the instances in the order that minimizes the required number of updates

- Terminate (the weight $\boldsymbol{w}$ classifies all instances correctly)
  - $\boldsymbol{x} = (0\ 0\ 1)^\intercal$: $y = g((1 1 0)(0 0 1)^\intercal) = 0$
  - $\boldsymbol{x} = (0\ 1\ 1)^\intercal$: $y = g((1 1 0)(0 1 1)^\intercal) = 1$
  - $\boldsymbol{x} = (1\ 0\ 1)^\intercal$: $y = g((1 1 0)(1 0 1)^\intercal) = 1$
  - $\boldsymbol{x} = (1\ 1\ 1)^\intercal$: $y = g((1 1 0)(1 1 1)^\intercal) = 1$

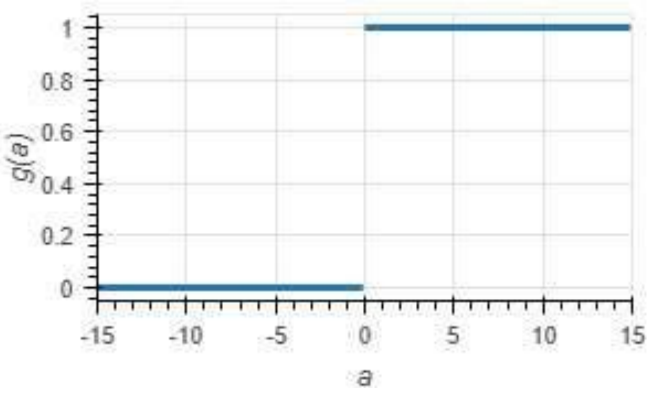


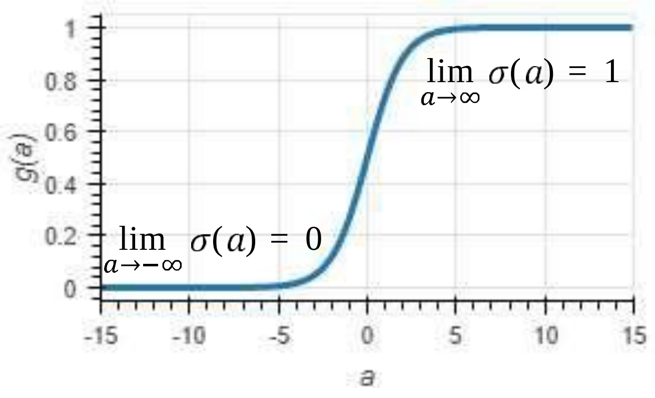
Why perceptron algorithm learns

- Suppose the parameter $\boldsymbol{w}$ misclassifies $(\boldsymbol{x}_n, y_n)$
  - If $y_n = 1$ then:
    - Update the weight vector $\boldsymbol{w}' \longleftarrow \boldsymbol{w} + \boldsymbol{x}_n$
    - If we classify $\boldsymbol{x}_n$ again with the updated weights $\boldsymbol{w}'$:
      - $\boldsymbol{w}' \cdot \boldsymbol{x}_n = (\boldsymbol{w} + \boldsymbol{x}_n) \cdot \boldsymbol{x}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n + \boldsymbol{x}_n \cdot \boldsymbol{x}_n \geq \boldsymbol{w} \cdot \boldsymbol{x}_n$
      - The dot product was increased (more likely to be classified as 1)
  - If $y_n = 0$ then:
    - Update the weight vector $\boldsymbol{w}' \longleftarrow \boldsymbol{w} - \boldsymbol{x}_n$
    - If we classify $\boldsymbol{x}_n$ again with the updated weights $\boldsymbol{w}'$:
      - $\boldsymbol{w}' \cdot \boldsymbol{x}_n = (\boldsymbol{w} - \boldsymbol{x}_n) \cdot \boldsymbol{x}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n - \boldsymbol{x}_n \cdot \boldsymbol{x}_n \leq \boldsymbol{w} \cdot \boldsymbol{x}_n$
      - The dot product was decreased (more likely to be classified as 0)

- The algorithm updates the parameter $\boldsymbol{w}$ to the direction where it will classify $(\boldsymbol{x}_n, y_n)$ more correctly

Activation function: step → sigmoid





Step function: $\mathbb{R} \to \{0,1\}$

$$g(a) = \begin{cases} 1 & (\text{if } a > 0) \\ 0 & (\text{otherwise}) \end{cases}$$

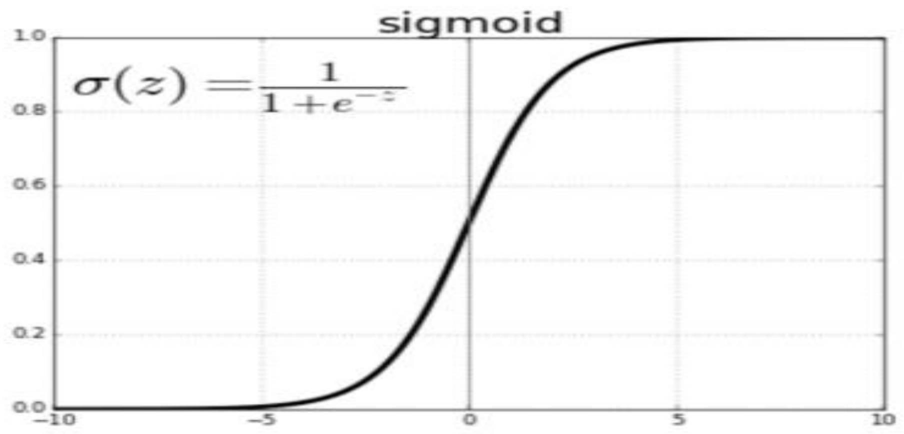
- Yields binary outputs
  - Unusable for multi-class classification
- Indifferentiable at zero
- With zero gradients

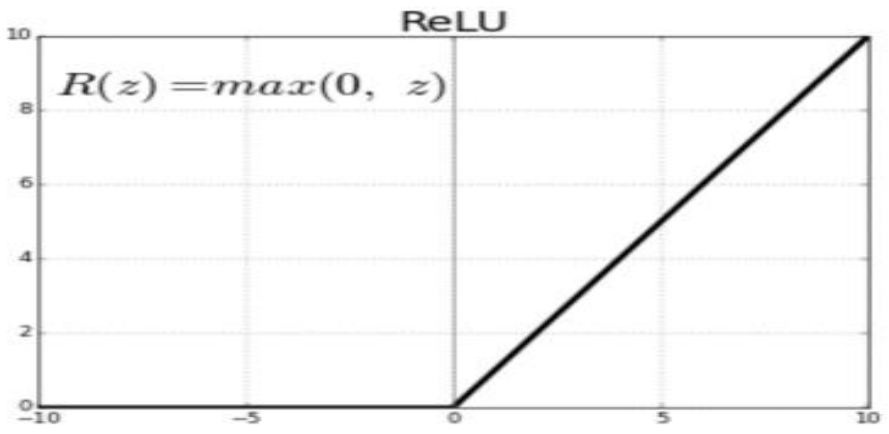
Sigmoid function: $\mathbb{R} \to (0,1)$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- Yields continuous scores
  - Usable for multi-class classification
- Differentiable at all points
- With mostly non-zero gradients
  - Useful for gradient descent

Sigmoid or Logistic vs Rectified Linear Unit (ReLU) Activation Functions





- Squashes real numbers between 0 and 1
- They have nice derivatives, which make learning easy.
- Currently not used as much because result in gradients too close to 0 stopping learning.

- Most popular because it is simple to compute and very robust to noisy inputs.

General form with sigmoid

- Single layer NN with sigmoid function

$$\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w} \cdot \boldsymbol{x}}}$$

  - Given an input $\boldsymbol{x} \in \mathbb{R}^d$, it computes an output $\hat{y} \in (0,1)$ by using the parameter $\boldsymbol{w} \in \mathbb{R}^d$

- This is also known as *logistic regression*
  - We can interpret $\hat{y}$ as the conditional probability $P(1|\boldsymbol{x})$ where an input is classified to 1 (positive category)
  - Rule to classify an input to 1:

$$\hat{y} > 0.5 \iff \frac{1}{1 + e^{-\boldsymbol{w} \cdot \boldsymbol{x}}} > \frac{1}{2} \iff \boldsymbol{w} \cdot \boldsymbol{x} > 0$$

    - The classification rule is the same as the linear models

Example: logical AND

- The same parameter in the previous example

$$\hat{y} = \sigma(a), a = x_1 + x_2 - 1.5$$

x_1	x_2	$y = x_1 \wedge x_2$	a	$\hat{y} = \sigma(a)$
0	0	0	-1.5	0.182
0	1	0	-0.5	0.378
1	0	0	-0.5	0.378
1	1	1	0.5	0.622

- The outputs are acceptable, but
  - $P(x_1 \wedge x_2 = 1 | x_1 = 1, x_2 = 1)$ is not so high (62.2%)
  - Room for improving $\boldsymbol{w}$ so that it yields $y \rightarrow 1$ (100%) for positives (true) and $y \rightarrow 1$ (0%) for negatives (false)

Instance-wise likelihood

- We introduce *instance-wise likelihood*, to measure how well the parameters reproduce $(\boldsymbol{x}_n, y_n)$

$$p_n = \begin{cases} \hat{y}_n & (\text{if } y_n = 1) \\ (1 - \hat{y}_n) & (\text{otherwise}) \end{cases}$$

x_1	x_2	$y = x_1 \wedge x_2$	a	$\hat{y} = \sigma(a)$	p	
0	0	0	-1.5	0.182	$1 - \hat{y} = 0.818$	$\Rightarrow$ 1
0	1	0	-0.5	0.378	$1 - \hat{y} = 0.622$	$\Rightarrow$ 1
1	0	0	-0.5	0.378	$1 - \hat{y} = 0.622$	$\Rightarrow$ 1
1	1	1	0.5	0.622	$\hat{y} = 0.622$	$\Rightarrow$ 1

Parameters of AND: $\hat{y} = \sigma(a), a = x_1 + x_2 - 1.5$

- Likelihood is a probability representing the 'fitness' of the parameters to the training data
  - We want to increase the likelihood by changing $\boldsymbol{w}$

Likelihood on the training data

- We assume that all instances in the training data are i.i.d. (independent and identically distributed)
- We define *likelihood* as a joint probability on data,

$$L_D(\boldsymbol{w}) = \prod_{n=1}^{N} p_n$$

  - When the training data $D = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$ is fixed, likelihood is a function of the parameters $\boldsymbol{w}$

- Let us maximize $L_D(\boldsymbol{w})$ by changing $\boldsymbol{w}$
  - This is called *Maximum Likelihood Estimation (MLE)*
  - The maximizer $\boldsymbol{w}^*$ reproduces the training data well

Training as a minimization problem

- Products of $(0,1)$ values often cause underflow
- Use *log-likelihood*, the logarithm of the likelihood, instead

$$LL_D(\boldsymbol{w}) = \log L_D(\boldsymbol{w}) = \log \prod_{n=1}^{N} p_n = \sum_{n=1}^{N} \log p_n$$

- In mathematical optimization, we usually consider a minimization problem instead of maximization
- We define an objective function $E_D(\boldsymbol{w})$ by using the negative of the log-likelihood

$$E_D(\boldsymbol{w}) = -LL_D(\boldsymbol{w}) = -\sum_{n=1}^{N} \log p_n$$

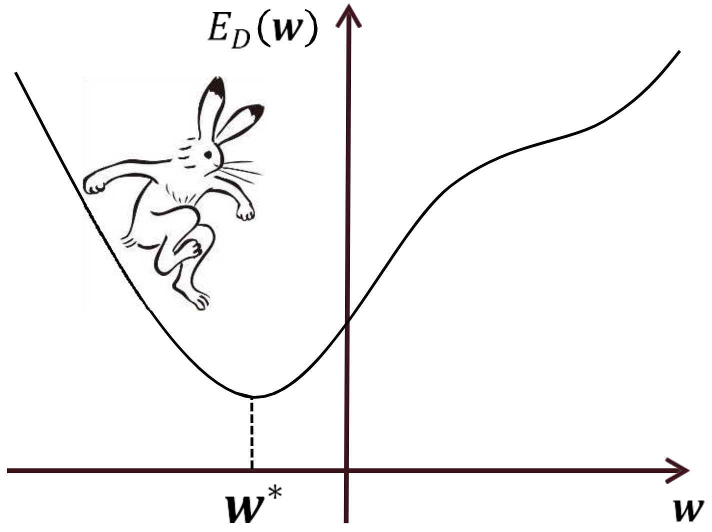
  - $E_D(\boldsymbol{w})$ is called a *loss function* or *error function*

Training as a minimization problem

- Given the training data $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, find $\boldsymbol{w}^*$ as the minimization problem,

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} E_D(\boldsymbol{w}) = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{n=1}^{N} l_n \,,$$

$$l_n = -\log p_n = \begin{Bmatrix} -\log \hat{y}_n & (\text{if } y_n = 1) \\ -\log(1 - \hat{y}_n) & (\text{otherwise}) \end{Bmatrix} = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$$



Stochastic Gradient Descent (SGD)

- The objective function is the sum of losses of instances,

$$E_D(\boldsymbol{w}) = \sum_{n=1}^{N} l_n$$

- We can use Stochastic Gradient Descent (SGD) and its variants (e.g., Adam) for minimizing $E_D(\boldsymbol{w})$
- SGD Algorithm (T is the number of updates)

```
1.  Initialize $\boldsymbol{w}$ with random values
2.  for $t \leftarrow 1$ to $T$:
3.      $\eta_t \leftarrow 1/t$
4.      $(\boldsymbol{x}_n, y_n) \leftarrow$ a random sample from $D$
5.      $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \frac{\partial l_n}{\partial \boldsymbol{w}}$
```

Dataset Preparation

Training Dataset

Training Dataset:

The actual dataset that we use to train the model.

The model *sees* and *learns* from this data.

Validation Dataset

***Validation Dataset*:**

The validation set is used to evaluate a given model.

We use this data to fine-tune the model hyperparameters.
Hence the model occasionally *sees* this data, but it never does "*Learn*" from this.

The validation set is also known as the Development set. This makes sense since this dataset helps during the "development" stage of the model.

Testing Dataset

Testing Dataset:

The sample of data used to provide an evaluation of the final model fit on the training dataset.

It is only used once a model is completely trained (using the train and validation sets).

The test set is generally well curated. It contains carefully sampled data that span the various classes that the model would face, when used in the real world.

Main Problems during training

- Insufficient quantity of training data
- Non-representative training data
- Poor-quality data
- Irrelevant features
- Overfitting the training data
- Underfitting the training data

Overfitting Problem (I)

- task is to predict if an image shows a balloon or not
- train a model using a dataset containing many blue coloured balloons (and other irrelevant objects)
- test the model on the original dataset: it gives 99% accuracy!
- test the model on a new ("unseen") dataset containing yellow coloured balloons: it gives 20% accuracy!

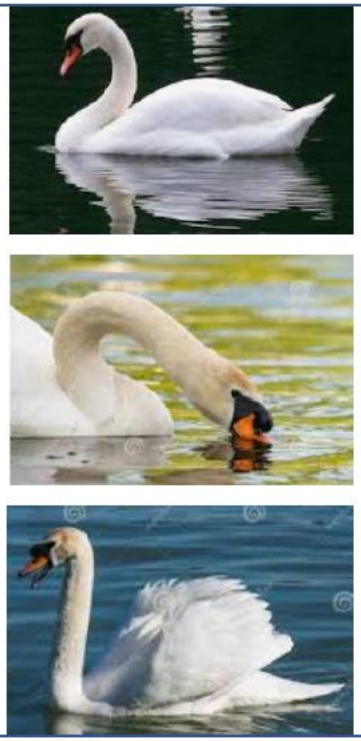
Our model doesn't *generalise* well from our training data to unseen data. This is known as overfitting.

Overfitting Problem (II)

A model that has learned the noise instead of the signal is considered “overfit” because it fits the training dataset but has poor fit with new datasets.

Overfitting happens when the ANN is said to be over-trained so that the model captures the exact relationship between the specific input-output used during training phase.

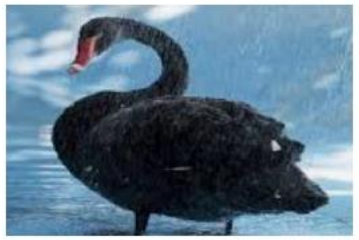
Overfitting Example



learned features via model

Have wings
Mouth is long
Neck is long
Shape looks like ‘2’
little larger than duck
White color

Not swan



Not swan

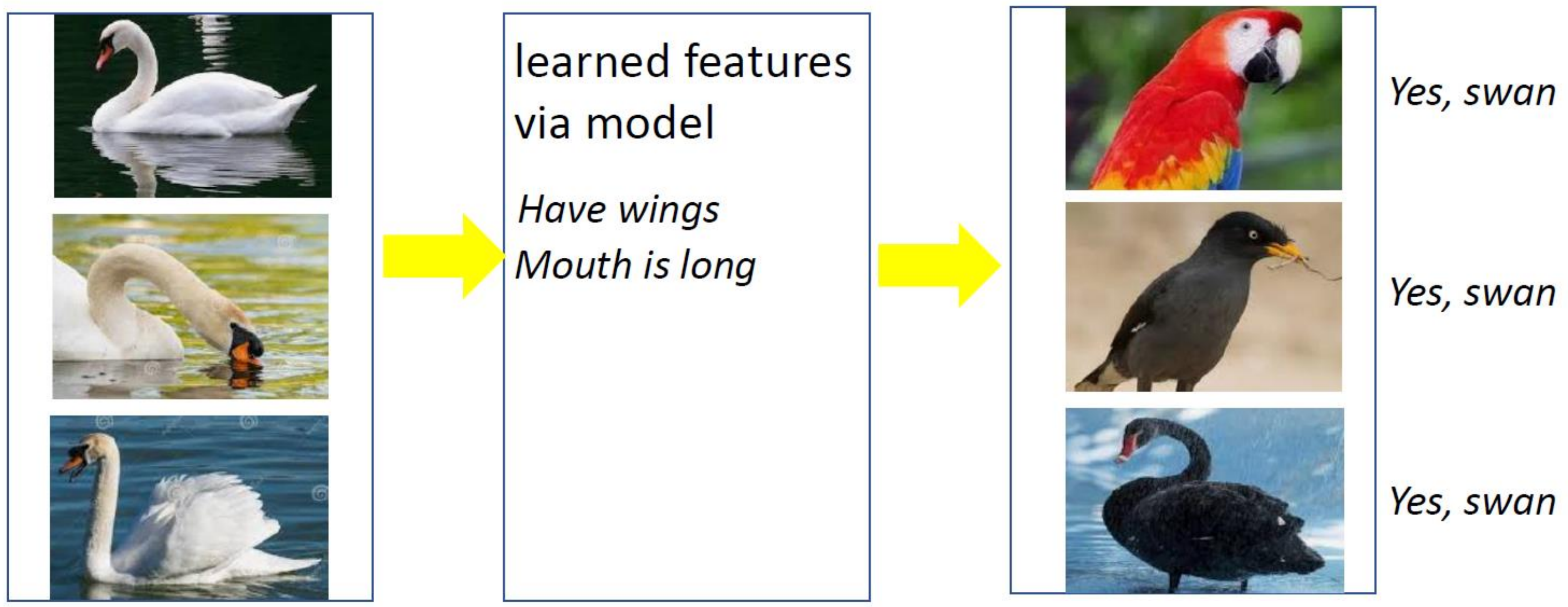
The model has the ability to distinguish swan and other birds. Due to the fact that all training images are white swans, the model learns that all swans are white, hence it cannot predict accurately swans of other colours.

Underfitting Problem

Underfitting happens when a machine learning model is not complex enough to accurately capture relationships between the input features and the target variable.

Underfitting appears when the network is not able to capture the underlying function mapping input – output data, either due to the small size of the training dataset or the poor architecture of the model.

Underfitting Example



The learned features of the swan are too few, so the criterion for distinguishing whether the images are swan or not is not clear. It is difficult to predict accurately.

More on generalization ability (1)

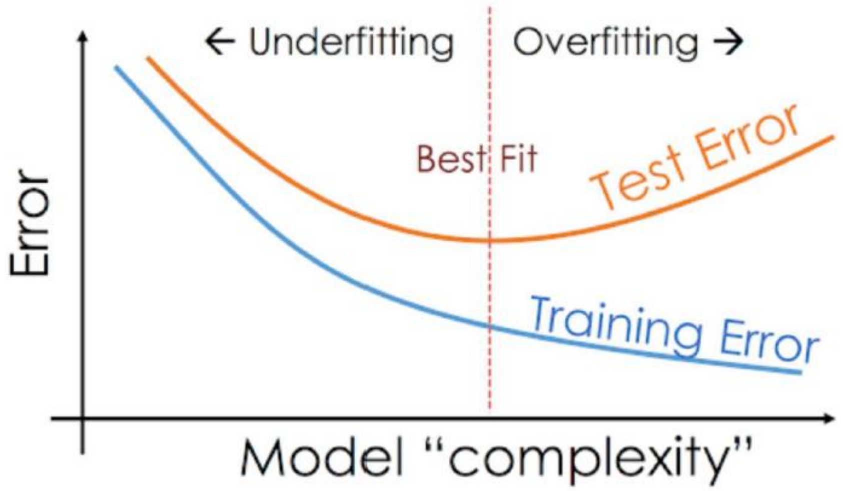
Definition

- Overfitting

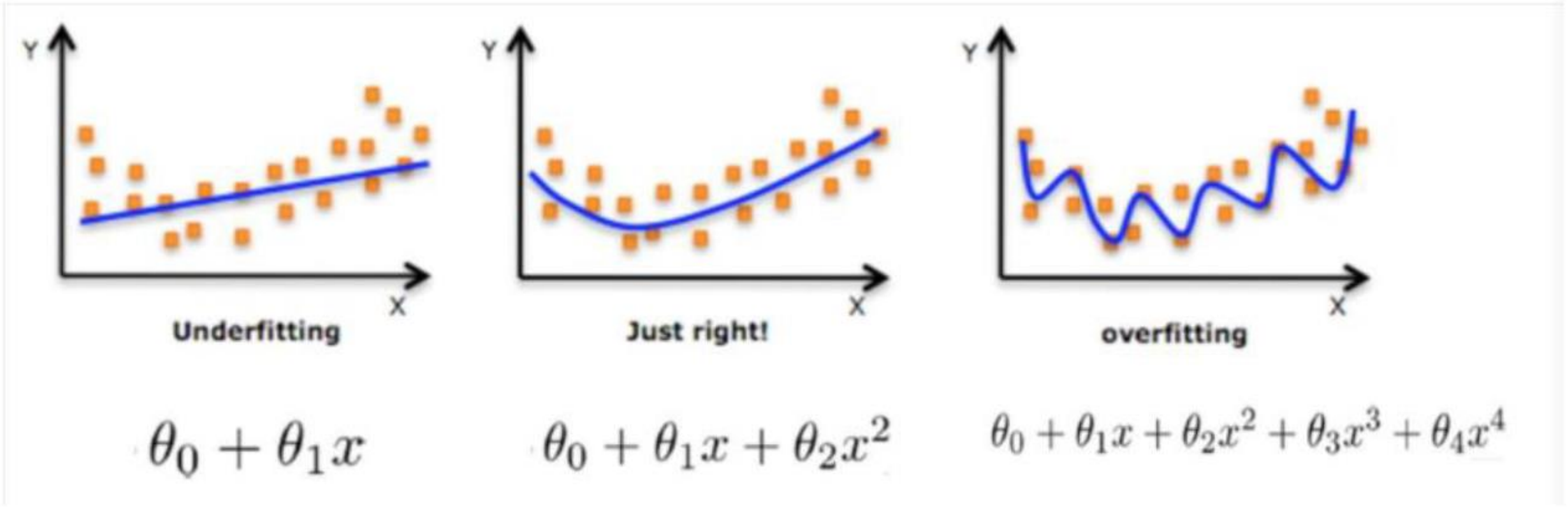
Good performance on the training data, poor performance on the test data (model is too complex)

- Underfitting

Poor performance on the training data and poor performance on the test data (model is too simple)

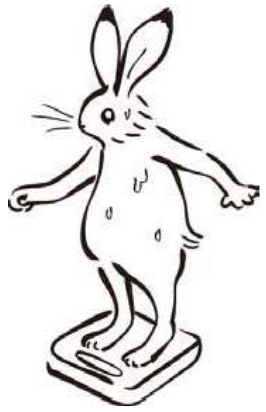


More on generalization ability (2)



https://www.programmersought.com/article/58683868977/

Regularization



- MLE often causes over-fitting
  - When the training data is linearly separable

$$|\boldsymbol{w}| \to \infty \text{ as} \sum_{n=1}^{N} l_n \to 0$$

  - Subject to be affected by noises in the training data

- We use regularization (MAP estimation)
  - We introduce a penalty term when $\boldsymbol{w}$ becomes large
  - The loss function with an L2 regularization term:

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} l_n + C|\boldsymbol{w}|^2$$

    - C is the hyper parameter to control the trade-off between over/under fitting

Cross-validation (1)

Cross-validation (or rotation estimation)

This is a procedure for finding how well a predictor performs on an unknown dataset, i.e., how well it generalizes.

A general and simple approach in training a predictor is:

- Divide the dataset into two (or three) non-overlapping subsets, one called the **test set** and the other the **training set**. In ANN applications, some researchers use the test set for periodical testing of the network, and a third set called **validation set** , for the predictor evaluation.
- Build the model using the training set.
- Check its predictions by using the test set (or the validation set).

Cross-validation (2)

This procedure is repeated in multiple iterations each time selecting a different non-overlapping test/training/validation dataset.

Strategies in cross-validation:

HOLD-OUT:

Randomly pick a part of the dataset, usually $\frac{2}{3}$ of the data, to be used as the training set and the remaining $\frac{1}{3}$ as the test set.

***k*-FOLD:**

Divide the data into *k* partitions. Use one partition as the test set and the remaining *k*-1 partitions for the training.

LEAVE-ONE-OUT:

This is a special case of *k*-fold.

Beyond accuracy: Confusion matrix

Confusion matrix or Error matrix (1)

	Cases predicted as **not having** the disease **(NO)**	Cases predicted as **having** the disease **(YES)**
Actual cases **without** the disease **(NO)**	**50**	**15**
Actual cases **with** the disease **(YES)**	**5**	**130**

We see that 180 out of the 200 cases were correctly classified, while the remaining 20 were misclassified.

A simple appraisal would have been to say that there is classification accuracy of 180/200 = 90%.

However, it would have been a very serious, even fatal, misclassification of the **5** cases that were predicted as not having the disease while actually they had it, or **15** cases that were predicted with the disease, while actually they didn’t have it.

Confusion matrix or Error matrix (2)

	Cases predicted as **not having** the disease **(NO)**	Cases predicted as **having** the disease **(YES)**
Actual cases **without** the disease **(NO)**	**50**	**15**
Actual cases **with** the disease **(YES)**	**5**	**130**

Also, in highly unbalanced data the accuracy will easily be biased towards high values.

Thus we need a system of better performance appraisal.

This leads us to some important new definitions.

Confusion matrix or Error matrix (3)

	Cases predicted as not having the disease (NO)	Cases predicted as having the disease (YES)
Actual cases **without** the disease (NO)	**TN = 50**	**15**
Actual cases **with** the disease (YES)	**5**	**TP = 130**

DEFINITIONS:

TRUE POSITIVE (TP):

The number of cases that were **correctly predicted** as **YES**.

In our example, TP = 130.

TRUE NEGATIVE (TN):

The number of cases that were **correctly predicted** as NO.

In our example, TN = 50.

Confusion matrix or Error matrix (4)

	Cases predicted as **not having** the disease **(NO)**	Cases predicted as **having** the disease **(YES)**
Actual cases **without** the disease **(NO)**	**TN = 50**	**FP =15**
Actual cases **with** the disease **(YES)**	**FN = 5**	**TP = 130**

DEFINITIONS:

FALSE POSITIVE (FP) (or Type I error, or False alarm):

The number of cases that were **wrongly predicted as having the disease** (YES) but **actually do not have the disease**.

In our example, FP = 15.

Confusion matrix or Error matrix (5)

TN = 50	FP =15
FN = 5	TP = 130

ACCURACY (AC):

(or **correctness** or **recognition rate**)

The rate of **correct classifications**.

$$AC \equiv \frac{TP+TN}{TOTAL} = \frac{TP+TN}{P+N} = \frac{TP+TN}{TP+TN+FN+FP} = \frac{130+50}{200} = 0.90 = 90\%.$$

Confusion matrix or Error matrix (6)

TN = 50	FP =15
FN = 5	TP = 130

POSITIVE PREDICTIVE VALUE (PPV)

(or Precision):

$$\text{PPV} \equiv \frac{\text{TP}}{\text{TP+FP}} = \frac{130}{130+15} = 89.6\%.$$

Confusion matrix or Error matrix (7)

TN = 50	FP =15
FN = 5	TP = 130

TRUE (REAL) POSITIVE RATE (TPR)

(or Sensitivity, or Recall, or Hit rate):

$$TPR \equiv \frac{TP}{P} = \frac{TP}{TP+FN} = \frac{130}{130+5} = 96.3\%.$$

TRUE (REAL) NEGATIVE RATE (TNR) (or Specificity):

$$TNR \equiv \frac{TN}{N} = \frac{TN}{TN+FP} = \frac{50}{50+15} = 76.9\%.$$

FALSE NEGATIVE RATE (FNR) (or Miss rate):

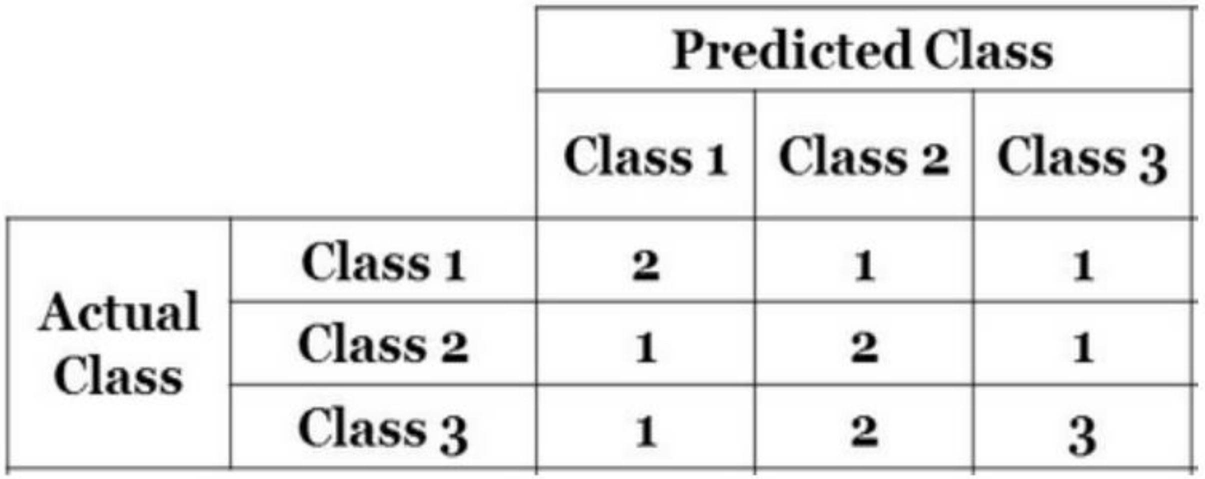
$$FNR \equiv \frac{FN}{P} = \frac{FN}{FN+TP} = \frac{5}{5+130} = 3.7\% = 1 - TPR$$

FALSE POSITIVE RATE (FPR) (or Fall out):

$$FPR \equiv \frac{FP}{N} = \frac{FP}{FP+TN} = \frac{5}{5+130} = 3.7\% = 1 - TNR$$

Confusion matrix or Error matrix (8)

The ideas of binary confusion matrix are easily extended to multiple classes by proper transformations.



		Predicted Class		
		Class 1	Class 2	Class 3
Actual Class	Class 1	2	1	1
	Class 2	1	2	1
	Class 3	1	2	3

F1 score (Twice the Inverse of sum of Inverses of precision and recall)

- F1 score penalises extremely low precision and recall

Practical prediction

Precision: 0.500

Recall: 0.750

Mean: 0.625

F1 score: 0.60

Predicted	Actual	Performance
Benign	Benign	Correct
Benign	**Malignant**	Wrong
Benign	Benign	Correct
Malignant	Benign	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	**Malignant**	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	Benign	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	Benign	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	**Malignant**	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	**Malignant**	Correct

Negative-based

Precision:1.000

Recall: 0.250

Mean: 0.625

F1 score: 0.40

Predicted	Actual	Performance
Benign	Benign	Correct
Benign	**Malignant**	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	**Malignant**	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	**Malignant**	Wrong
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Benign	Benign	Correct
Malignant	**Malignant**	Correct

Receiver Operating Characteristic (ROC) Graph

TPR/FPR Reminder

TRUE (REAL) POSITIVE RATE (TPR)

(or Sensitivity, or Recall, or Hit rate):

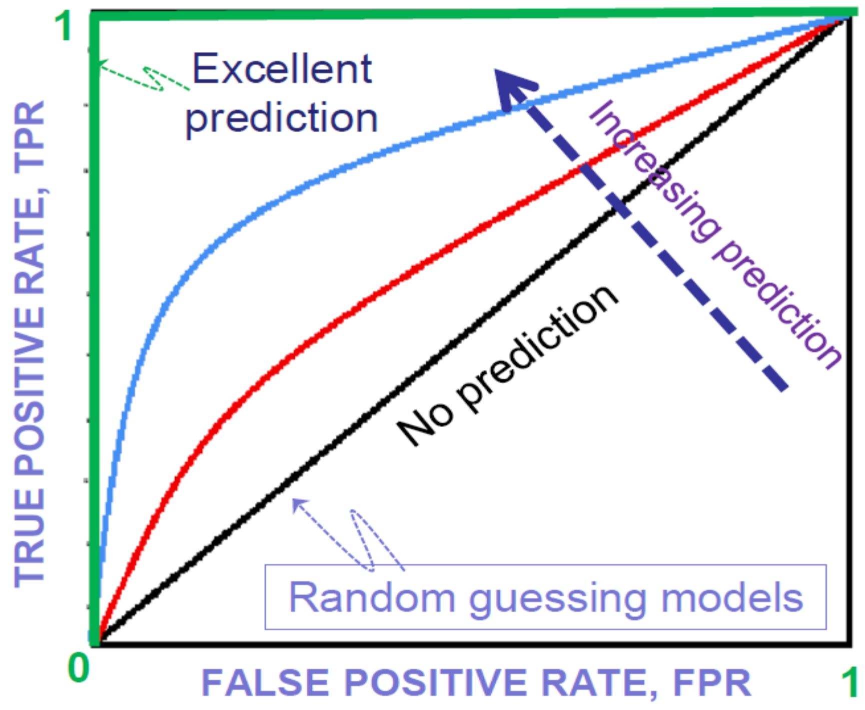
$$\text{TPR} \equiv \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP+FN}} = \frac{130}{130+5} = 96.3\%.$$

FALSE POSITIVE RATE (FPR) (or Fall out):

$$\text{FPR} \equiv \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP+TN}} = \frac{5}{5+130} = 3.7\% = 1 - \text{TNR}$$

Receiver Operating Characteristic ROC Graph (1)

This is a plot of the true positive rate vs false positive rate that is generated **as we vary the threshold** for assigning observations to a given class.



Receiver Operating Characteristic ROC Graph (2)

This is an alternative way to the confusion matrices, to examine the **performance of supervised learning** classifiers.

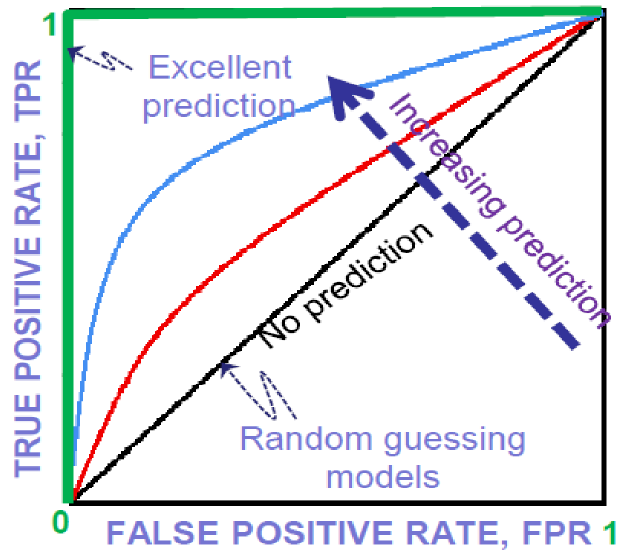
It is popular in CI biological classifiers and medical diagnostic systems .

It is a visual way to examine the tradeoff between the ability of a classifier to correctly identify positive cases and the number of negative cases that are incorrectly classified.

It encapsulates all the information of the confusion matrix, since FN is complement to TP and TN is complement to FP.

Receiver Operating Characteristic ROC Graph (3)

The point (0,1) is the **perfect classifier** because it classifies all positive cases and negative cases correctly (the false positive rate is 0 (none), and the true positive rate is 1 (all)).



The point (0,0) represents a classifier that **predicts all cases to be negative**, while the point (1,1) corresponds to a classifier that predicts every case to be positive.

Point (1,0) is the classifier that is incorrect for all classifications.

ROC Area

The **area under the ROC curve (AUC, or ROCA)** is also a good indicator of classifier performance.

This performance metric has the advantage that it is more robust than the accuracy, especially in problems of unbalanced classes.

This metric has been well accepted, but it has some problems when the ROCS for different models cross.