Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

04 - Perceptron

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Vers. 24.4.2022

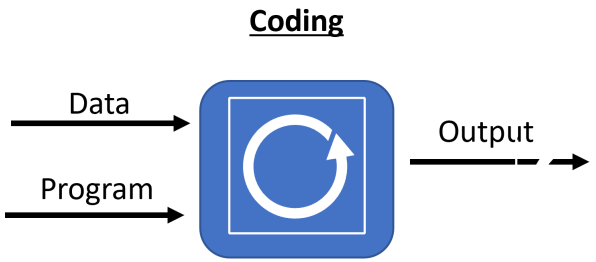
Agenda

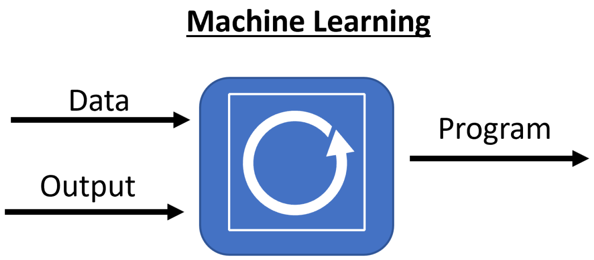
On the menu for today:

- Perceptron
    - What is it and how does it work?
- Nearest Neighbour
    - It is easy to implement!

ML

- **Traditional programming** :
  - Input: DATA (input) + PROGRAM (algorithm)
  - output: OUTPUT(answers)
- **Machine learning** :
  - Input: DATA(Input) + OUTPUT(answers)
  - Output: PROGRAMM(model) - (Evaluation via testing and metrics)

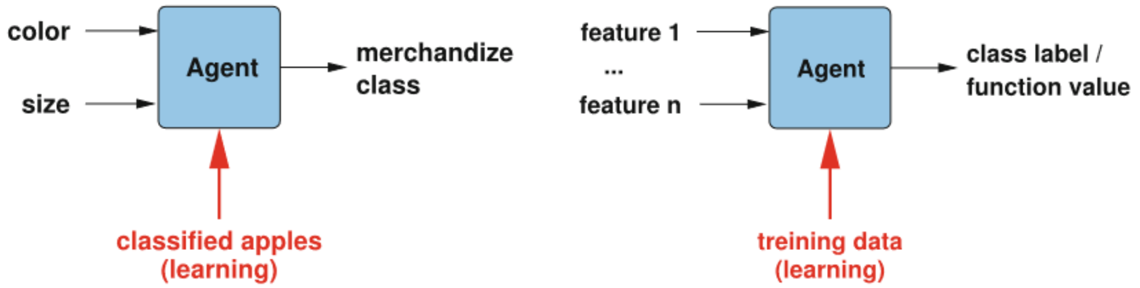




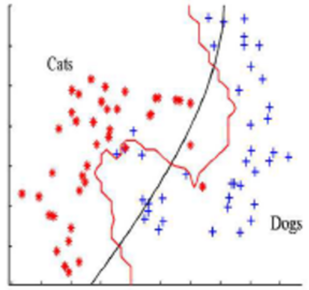
Learning Agent

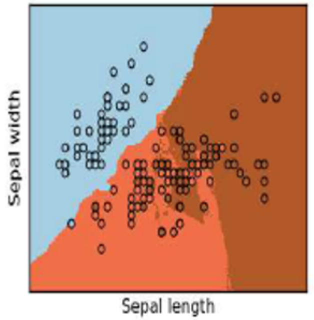
An agent is a learning agent if it improves its performance (measured by a suitable criterion) on new, unknown data over time (after it has seen many training examples).

- We can formally describe a learning agent as a function f which maps a feature vector to a discrete class value or in general to a real number.
- This function is not programmed, rather it comes into existence or changes itself during the learning phase, influenced by the training data.
- During learning, the agent is fed with the already classified data.



Classification





Objective: find separation between input variables x_i based on the class y_i *of observed instances*

The dataset is $D = \{(\vec{x_i}, y_i)\}_{i=1}^{n} \subset X \times Y$ where each data point is a pair of input variables $\vec{x_i} \epsilon\, R^N$ & the corresponding output $y_i \epsilon \{1, \dots, K\}$

The hypothesis space is the set of all functions from the input space *X* to $\{1, \dots, K\}$; i.e.,

F = {f | f : $X \rightarrow \{1, \dots, K\}$}

A common restriction is to just consider the set of linear mappings from X to Y as parametrized by w and b

Linear separable

Two sets $M_1 \in \mathbb{R}^n$ and $M_2 \in \mathbb{R}^n$ are called **linearly separable** if real numbers $a_1, \ldots, a_n, \theta$ exist with

$$\sum_i a_i * x_i > \theta \quad \forall \quad x \in M_1$$

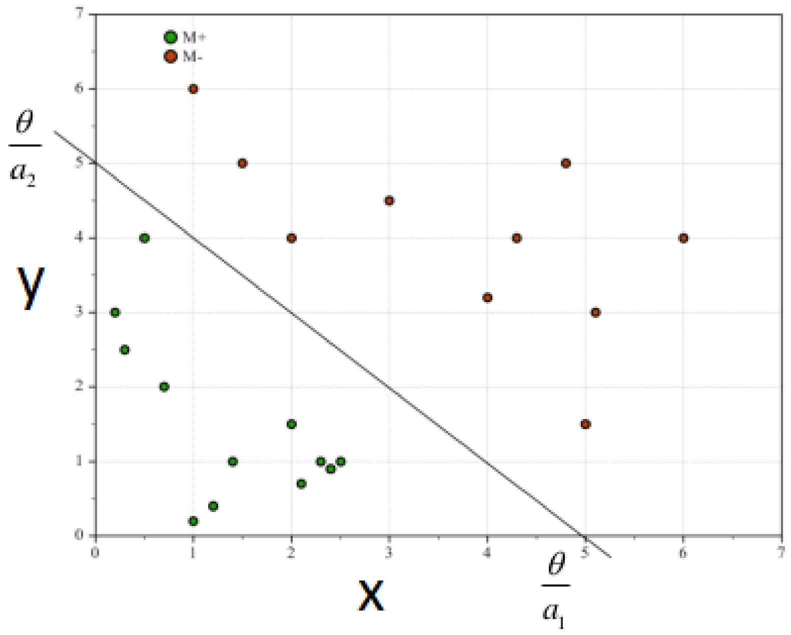
and

$$\sum_i a_i * x_i <= \theta \quad \forall \quad x \in M_2$$

The value θ is denoted the threshold.

Linear separable

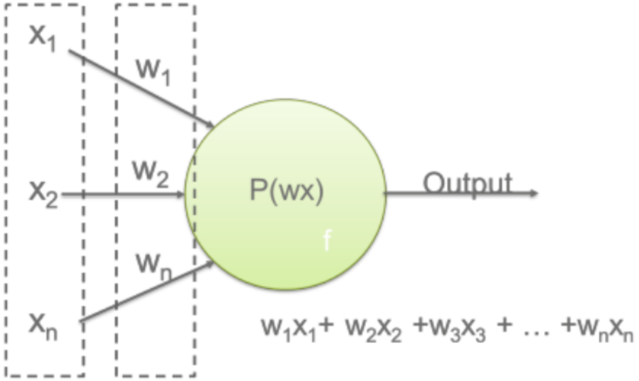
Example: $a_1 x + a_2 y = \theta$



Perceptron

Let $w = (w_1, \dots, w_n) \in \mathbb{R}^n$ be a weight vector and $x \in \mathbb{R}^n$ an input vector. A perceptron represents a function $P : \mathbb{R}^n \to \{0, 1\}$ which corresponds to the following rule:

$$P(x) = \begin{cases} 1 & if\; wx = \sum_{i=0} w_i x_i > 0, \\ 0 & otherwise \end{cases}$$



Perceptron

- The perceptron is a very simple classification algorithm.
- The perceptron as a learning agent, that is, as a mathematical function which maps a feature vector to a function value.
    - Here the input variables x_i are denoted features.
    - As we can see in the formula P(x)=1 definesall points x above the hyperplane are classified as positive and all others as negative
    - The separating hyperplane goes through the origin because $\theta = 0$.
- It is equivalent to a two-layer neural network with activation by a threshold function (next week).

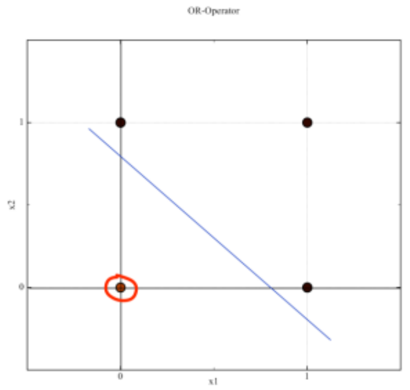
> Note: for example, the boolean function *AND* is **linearly separable** by a perceptron

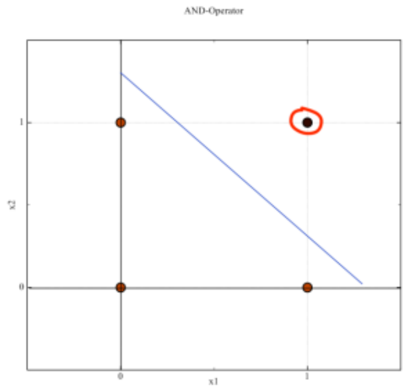
Task

Can you show that the boolean functions AND and OR for two values x and y are linear separable?

Task

Can you show that the boolean functions AND and OR for two values x and y are linear separable?



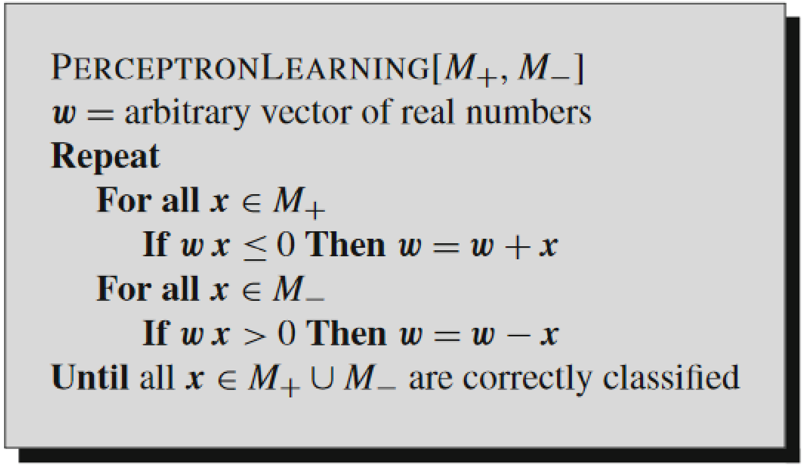


x1	x2	x1+x2	y
0	0	0	**0**
1	0	1	1
0	1	1	1
1	1	2	1

x1	x2	2x1+2x2	y
0	0	0	0
1	0	2	0
0	1	2	0
1	1	4	**1**

Learning rule

With the notation $M+$ and $M-$ for the sets of positive and negative training patterns respectively, the perceptron learning rule reads:



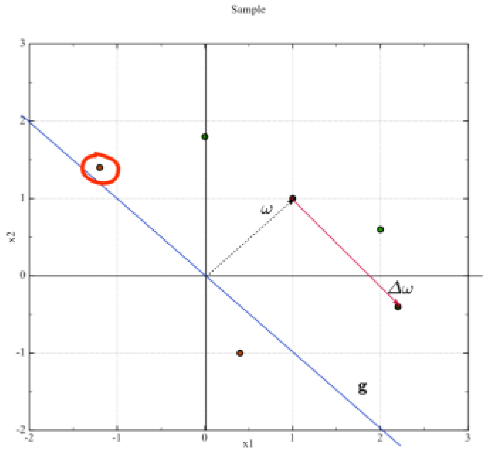
```
PerceptronLearning[$M_+$, $M_-$]
$w$ = arbitrary vector of real numbers
Repeat
    For all $x \in M_+$
        If $w\,x \leq 0$ Then $w = w + x$
    For all $x \in M_-$
        If $w\,x > 0$ Then $w = w - x$
Until all $x \in M_+ \cup M_-$ are correctly classified
```

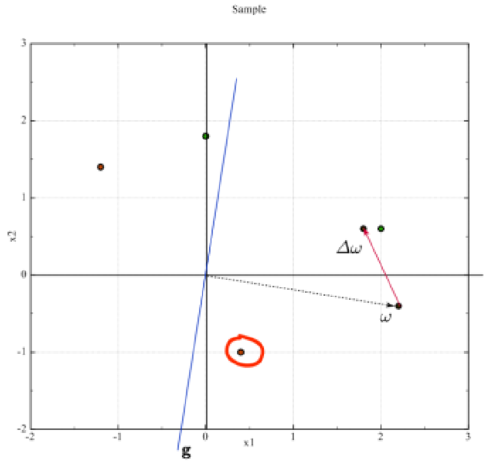
The perceptron should output the value 1 for all $x \in M+$.

Example

$M+=\{(0,1.8),(2,0.6)\}$

$M-=\{(-1.2,1.4),(0.4,-1)\}$





Gewichtsvektor initial: ω=(1,1)

wegen $\dot{\omega}\dot{x}=0$ ist $\dot{w} \perp g$

Beim 1. Durchlauf wird (-1.2,1.4)

falsch klassifiziert, denn

$$(-1.2,1.4)\begin{pmatrix}1\\1\end{pmatrix}=0.2>0\text{ , deshalb}$$

$$w=\begin{pmatrix}1\\1\end{pmatrix}-\begin{pmatrix}-1.2\\1.4\end{pmatrix}=\begin{pmatrix}2.2\\-0.4\end{pmatrix}$$

Gewichtsvektor initial: ω=(2.2, -0.4)

Beim Durchlauf wird (0.4,-1.0)

falsch klassifiziert, denn

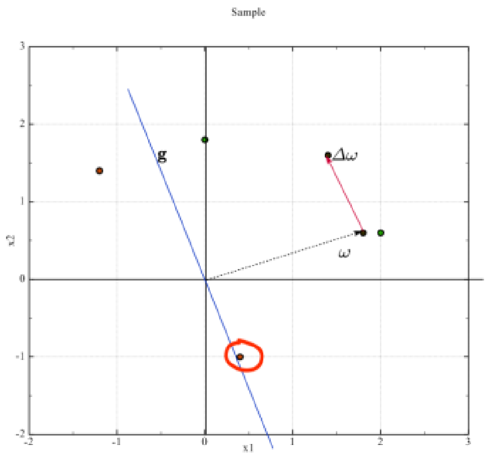
$$(0.4,-1.0)\begin{pmatrix}2.2\\-0.4\end{pmatrix}=1.28>0\text{ , deshalb}$$

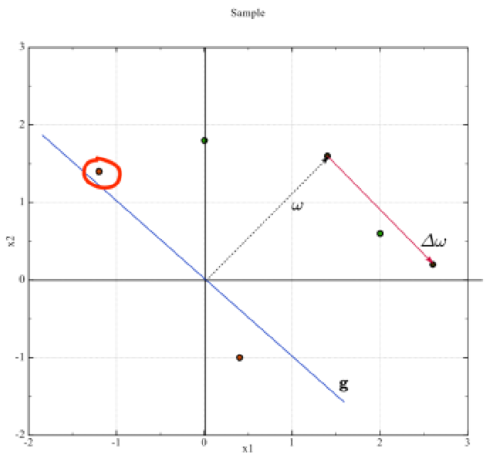
$$w=\begin{pmatrix}2.2\\-0.4\end{pmatrix}-\begin{pmatrix}0.4\\-1.0\end{pmatrix}=\begin{pmatrix}1.8\\0.6\end{pmatrix}$$

Example

$M+=\{(0,1.8),(2,0.6)\}$

$M-=\{(-1.2,1.4),(0.4,-1)\}$





Gewichtsvektor initial: ω=(1.8, 0.6)
Beim Durchlauf wird (0.4,-1.0)
falsch klassifiziert, denn

$(0.4, -1.0)\begin{pmatrix}1.8\\0.6\end{pmatrix}=0.12>0$, deshalb

$w=\begin{pmatrix}1.8\\0.6\end{pmatrix}-\begin{pmatrix}0.4\\-1.0\end{pmatrix}=\begin{pmatrix}1.4\\1.6\end{pmatrix}$

Gewichtsvektor initial: ω=(1.4, 1.6)
Beim Durchlauf wird (-1.2,1.4)
falsch klassifiziert, denn

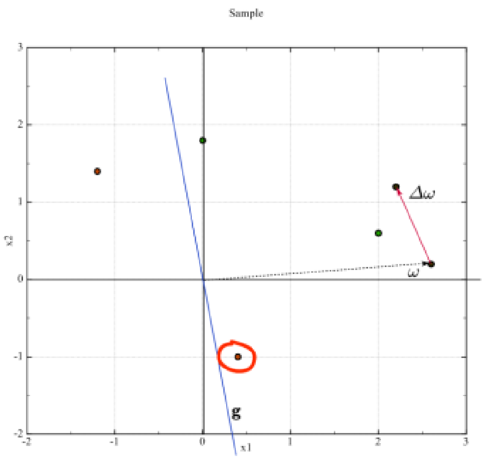
$(-1.2, 1.4)\begin{pmatrix}1.4\\1.6\end{pmatrix}=0.56>0$, deshalb

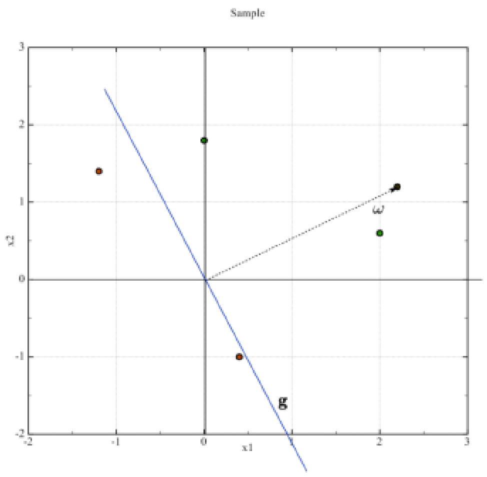
$w=\begin{pmatrix}1.4\\1.6\end{pmatrix}-\begin{pmatrix}-1.2\\1.4\end{pmatrix}=\begin{pmatrix}2.6\\0.2\end{pmatrix}$

Example

$M+ = \{(0,1.8),(2,0.6)\}$

$M- = \{(-1.2,1.4),(0.4,-1)\}$





Gewichtsvektor initial: ω=(2.6, 0.2)
Beim Durchlauf wird (0.4, -1)
falsch klassifiziert, denn

$$(0.4,\ -1.0)\begin{pmatrix}2.6\\0.2\end{pmatrix} = 1.24 > 0 \text{ , deshalb}$$

$$w = \begin{pmatrix}2.6\\0.2\end{pmatrix} - \begin{pmatrix}0.4\\-1.0\end{pmatrix} = \begin{pmatrix}2.2\\1.2\end{pmatrix}$$

Gewichtsvektor initial: ω=(2.2, 1.2)
Alle Punkte werden richtig klassifizier!

Convergence

Let classes M+ and M− be **linearly separable** by a hyperplane $wx > \theta$. Then PERCEPTRONLEARNING converges for every initialization of the vector w.

The perceptron P with the weight vector so calculated divides the classes M+ and M−, that is:

$$P(x) = 1 \Leftrightarrow x \in M+$$

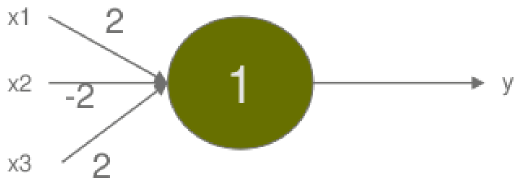
and

$$P(x) = 0 \Leftrightarrow x \in M-$$

Perceptron with Bias

$$P(x) = \begin{cases} 1 & if\, wx = \sum_{i=0}^{n-1} w_i x_i > \theta, \\ 0 & otherwise \end{cases}$$

$$(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$$



x1	x2	x3	2x1-2x2+2x3	Y
0	0	0	0	**0**
1	0	0	2	1
0	1	0	-2	**0**
1	1	0	0	**0**
0	0	1	2	1
1	0	1	4	1
0	1	1	0	**0**
1	1	1	1	1

Task

How could both functions `activation` and `perception` be expressed in python given the following value:

```
weight = [1]
x = [3]
bias=4
```

1. Exporess `activation` as comprehension and express `prediction` as condition based on activation.
2. Convert both into a function `predict`

Task

How could both functions `activation` and `perception` be expressed in python given the following value:

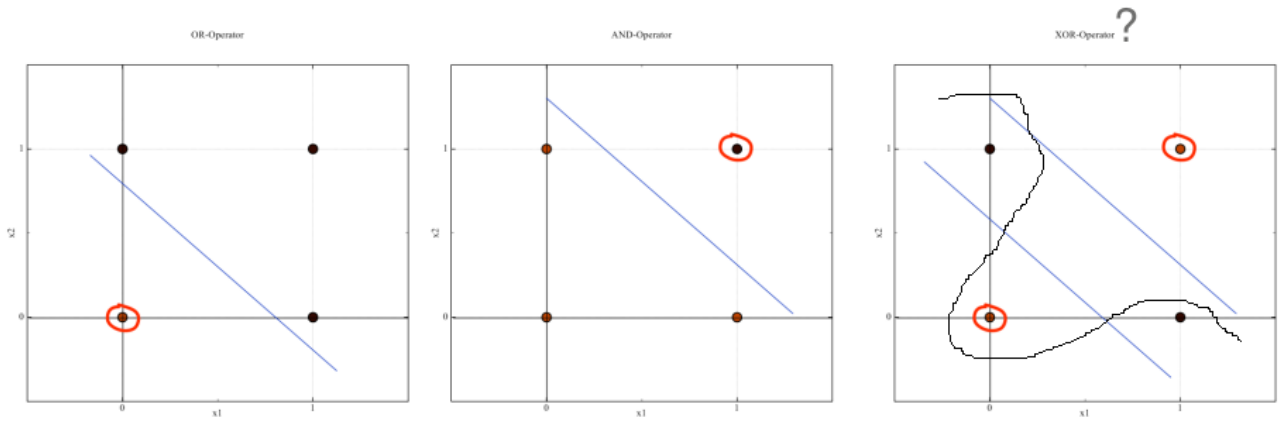
```python
weight = [1]
x = [3]
bias=4

activation = bias
activation += sum([(w * x) for w,x in zip(weight,x)])
prediction = 1.0 if activation >= 0.0 else 0.0

print(prediction)

# Make a prediction with weights
def predict(row, weights, bias):
    # extract bias
    activation = bias
    activation += sum([(w * x)for w,x in zip(weights,row)])
    return 1.0 if activation >= 0.0 else 0.0
```

Task

Can you test your code with the following data:

```python
# Given values; last column is target feature
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]

# first column is bias
weights = [-0.1, 0.20653640140000007, -0.23418117710000003]

for row in dataset:
    prediction = predict(row[:2], weights[1:], weights[0])
    print("Expected=%d, Predicted=%d" % (row[-1], prediction))
```

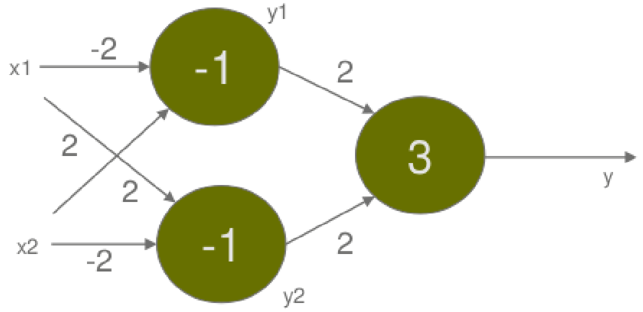
Limits of the perceptron



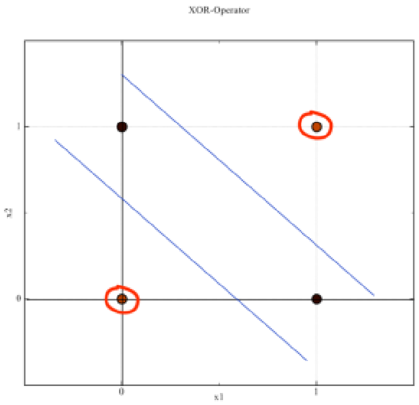
x1	x2	x1+x2	y
0	0	0	**0**
1	0	1	1
0	1	1	1
1	1	2	1

x1	x2	2x1+2x2	y
0	0	0	0
1	0	2	0
0	1	2	0
1	1	4	**1**

2-layer Perceptron



x1	x2	y1	y2	Y
0	0	1	1	1
1	0	0	1	0
0	1	1	0	0
1	1	1	1	1



Perceptron: Optimization

A function $f : \mathbb{R}^n \rightarrow \{0,1\}$ can by represented by a perceptron if and only if the two sets of positive and negative input vectors are linearly separable.

- The perceptron in the form presented converges *very slowly*.
  - It can be accelerated by normalization of the weight-altering vector. The formulas $w = w \pm x$ are replaced by $w = w \pm \frac{x}{|x|}$. Thereby every data point has the same weight during learning, independent of its value.
- The speed of convergence heavily depends on the initialization of the vector w.
  - Ideally it would not need to be changed at all and the algorithm would converge after one iteration. We can get closer to this goal by using the heuristic initialization

$$w_0 = \sum_M x - \sum_M x$$

- The perceptron can be seen as the simplest **neural network model**.

Technische Hochschule Rosenheim

Nearest Neighbour (*NN*)

Motivation

- For a perceptron, knowledge available in the training data is extracted and saved in a compressed form in the weights w_i.
  - As a result, information about the data is lost.
- This is exactly what is desired, however, if the system is supposed to generalize from the training data to new data.
- Generalization in this case is a time-intensive process with the goal of finding a compact representation of data in the form of a function which classifies new data as good as possible.
- For a perceptron learning is extremely simple, but the saved knowledge is not so easily applicable to new, unknown examples.

> The goal: Express similarity of new data in comparsion to known data!

Similarity

We represent the training samples as usual in a multidimensional feature space and define:

The smaller their distance in the feature space, the more two examples are similar.

The distance d(x, y) between two points $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ can for example be measured by the Euclidean distance:

$$d(x, y) = |x - y| = \sqrt{\sum_i (x_i - y_i)^2}$$

- Therefore it is often sensible to scale the features differently by weights w_i. The formula then reads

$$d_w(x, y) = |x - y| = \sqrt{\sum_i w_i (x_i - y_i)^2}$$

NN Algorithm

Technische Hochschule **Rosenheim**

The following simple nearest neighbor classification program searches the training data for the nearest neighbor t to the new example s and then classifies s exactly like t:

```
NEARESTNEIGHBOR[$M_+$, $M_-$, $s$]
    $t = \text{argmin}_{x \in M_+ \cup M_-} \{d(s, x)\}$
    **If** $t \in M_+$   **Then Return** („+”)
                         **Else Return**(„–”)
```

Note: The function argmin and argmax determine, similarly to min and max, the minimum or maximum of a set or function. However, rather than returning the value of the maximum or minimum, they give the position, that is, the argument in which the extremum appears.

Task

Given the following sets Mp and Mm:

```python
from math import sqrt

Mp = [[1,1], [1,1.3], [1.2,2.2]]

Mm = [[2,2], [2.1,3.1], [4.2,3.1]]
```

1. Can you write a function `dist` which takes to points x=[x1,x2] and y=[y1,y2] and calculates the euclidea distance?
2. Write a function `nn` taking three params P and M as sets represented as lists and a point s. The function should return to which set the point s belongs. (Implement the NN algo!)

Task#2

Given the following sets Mp and Mm:

```python
from math import sqrt

Mp = [[1,1], [1,1.3], [1.2,2.2]]

Mm = [[2,2], [2.1,3.1], [4.2,3.1]]

def dist(X,Y):
      return sqrt(sum([(x-y)**2 for x,y in zip(X,Y)]))

def NN(P, M, s):
    V = P.copy()
    V.extend(M)
    l = [dist(s,x) for x in V]
    mindex = min(l)
    t = l.index(mindex)
    if t <= len(P):
        return "+"
    else:
        return "-"
```

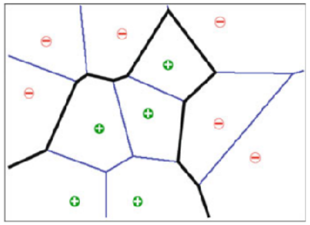
... in case you want to plot it!

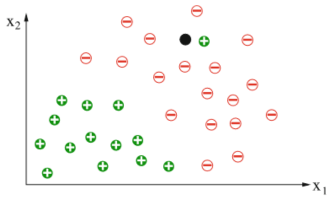
```
from matplotlib import pyplot as plt
import numpy as np

X=np.asarray(Mp)[:,0]
Y=np.asarray(Mp)[:,1]
plt.scatter(X,Y, c="blue")

X=np.asarray(Mm)[:,0]
Y=np.asarray(Mm)[:,1]
plt.scatter(X,Y, c="red")

plt.scatter([0], [1], c="green")
plt.show()
```

NN limitations

Technische Hochschule Rosenheim

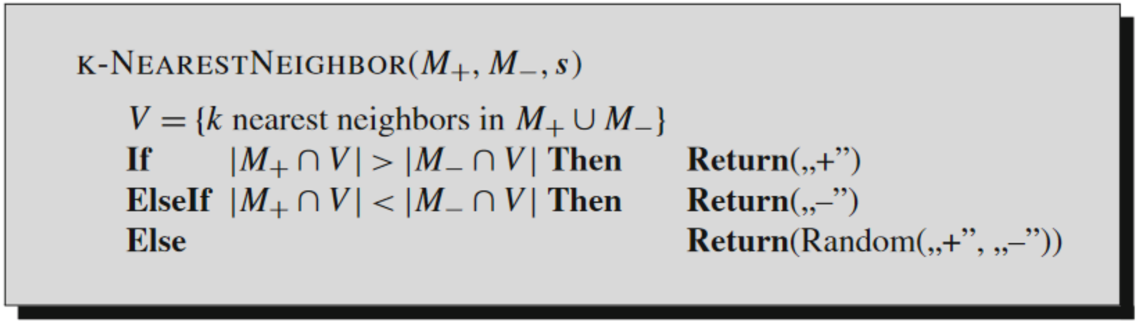
- The nearest neighbor method is significantly more powerful than the perceptron.
- It is capable of correctly realizing arbitrarily complex dividing lines (in general: hyperplanes).
- **BUT**: A single erroneous point can in certain circumstances lead to very bad classification results.
    - The nearest neighbor method may classify this wrong. If the black point is immediately next to a positive point that is an outlier of the positive class, then it will be classified positive rather than negative as would be intended here. An erroneous fitting to random errors (noise) is called *overfitting*.





k-Nearest Neighbour

- To prevent false classifications due to single outliers, it is recommended to smooth out the division surface.
- The K-NEARESTNEIGHBOR algorithm makes a majority decision among the k nearest neighbors:



```
K-NEARESTNEIGHBOR($M_+$, $M_-$, $s$)
    $V = \{k$ nearest neighbors in $M_+ \cup M_-\}$
    If      $|M_+ \cap V| > |M_- \cap V|$ Then    Return(„+”)
    ElseIf  $|M_+ \cap V| < |M_- \cap V|$ Then    Return(„–”)
    Else                                          Return(Random(„+”, „–”))
```

Summary

Lessons learned today:

- Perceptron
- Perceprton Learning
- Nearest Neighbour and k-Nearest Neighbour

Exercise

1. Perceptron

- A bit math … enjoy!

2. Perceptron in Python

- Train your weights based Gradient Descent!

3. k-NN in Action

- Can you write a classifier as k-NN from scratch!
- Compare with what is given by scikit!