Westsächsische Hochschule Zwickau
University of Applied Sciences

PTI90070 Artificial Intelligence

Prof. Dr.-Ing. Sven Hellbach

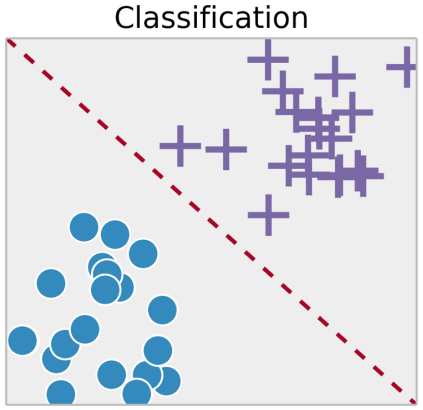
Classification

classification

To which class does an unknown data point belong?

Given:

- For course data the division into classes is already known.
- Classes are discreet
- There must be no order of classes
- for example: $Y = \{\text{red}, \text{green}, \text{blue}, \text{yellow}\}$



Wanted:

- The function that best predicts the given classes of data.
- Issue Y is *qualitative*

classification problem

data

model

prediction

	fatigue	weight	blood pressure	hair color	diabetes
Record 1	Frequently	120	high	blonde	Yes
Data record 2	Rare	120	medium	blonde	No
Record 3	Occasionally	70	low	darkness	No
Record 4	Occasionally	100	high	darkness	Yes
	x1	***x2***	***x3***	***x4***	***y***

Existing examples

$y \in \{Ja, Nein\}$

Training

$y = f(x1, x2, x3, x4)$

Application/Test

attributes
features
input data

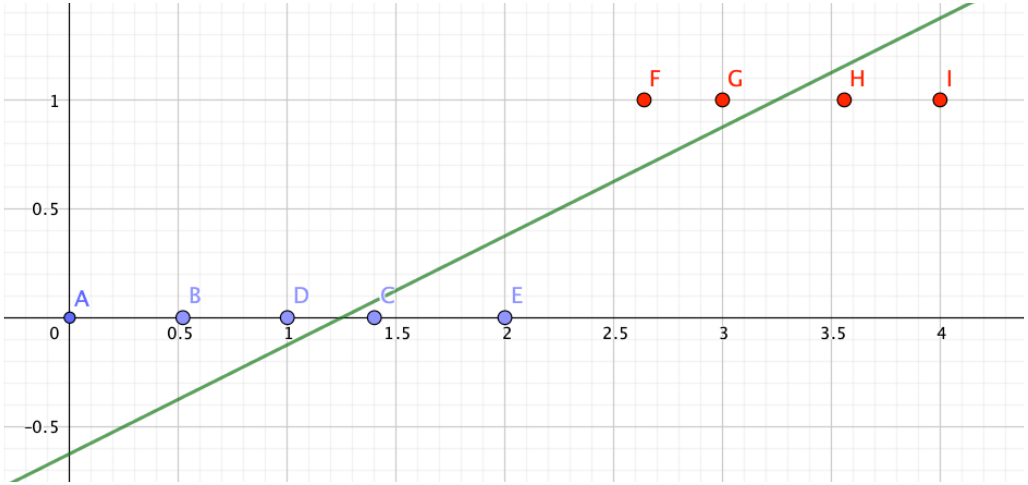
	fatigue	weight	blood pressure	hair color	diabetes
Unknown	Frequently	130	high	brunette	???

Solution attempt by means of regression

Given:

- Data points A,B,C,D,E in class 0
- Data points F,G,H,I in class 1

- Can we perhaps solve the new problem with regression?
- Linear model also provides values unequal to 0 & 1 in estimation

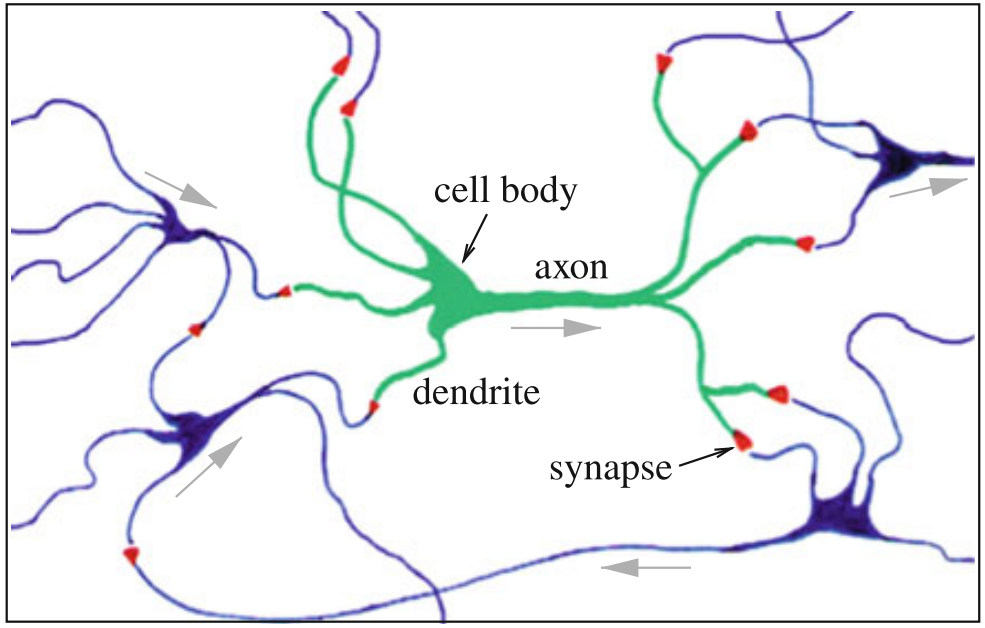


$$f(x) = w_1 \cdot x + w_0$$

Perceptron

Biological model Perzeptron

- *Perceptron* and *adaptive linear neurons:*
  one of the first classification algorithms described in the literature
- 1943 Warren McCulloch and Walter Pitts publish the first concept of a simplified brain cell => MCP neuron (McCulloch-Pitts neuron)



McCulloch-Pitts-Neuron, W.S. McCulloch und W. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, The bulletin of mathematical biophysics, 5(4):115-133, 1943

Definition Perceptron

- 1957 Frank Rosenblatt the first concept of a Perzeptron learning rule
- Task of a neuron: binary classification

Definition 8.3 Let $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ be a weight vector and $\boldsymbol{x} \in \mathbb{R}^n$ an input vector. A *perceptron* represents a function $P: \mathbb{R}^n \rightarrow \{0, 1\}$ which corresponds to the following rule:

$$P(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}\, x = \sum_{i=1}^{n} w_i x_i > 0, \\ 0 & \text{else.} \end{cases}$$

F. Rosenblatt, The Perceptron, a Perceiving and Recognizing Automaton, Cornell Aeronautical Laboratory, 1957

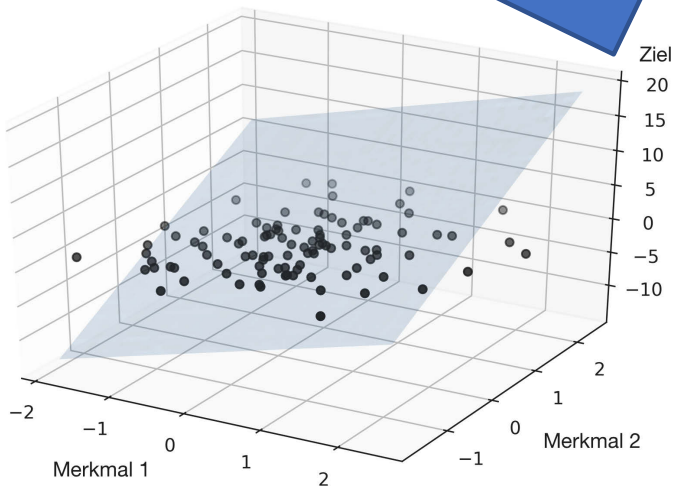
Multiple Linear Regression

As a reminder

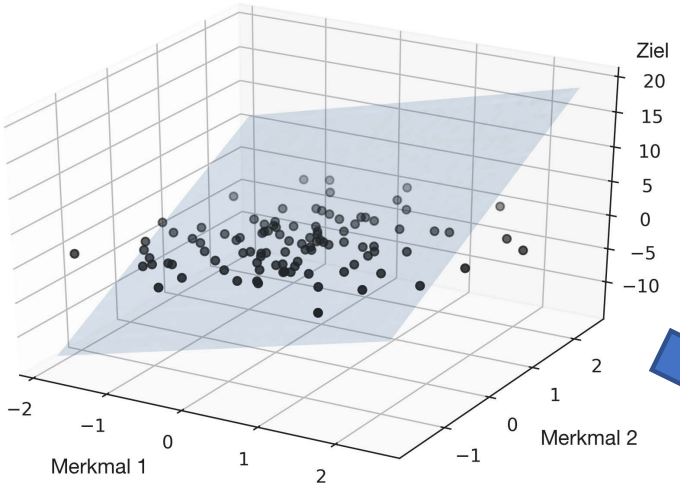
- Linear regression with multiple variables
- Extension of the idea to several characteristics (multivariate case)

$$
\begin{aligned}
y &= w_0 + w_1 \cdot x_1 + \ldots + w_m \cdot x_m \\
&= w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_m \cdot x_m \text{ mit } x_0 = 1 \\
&= \sum_{i=0}^{m} w_i \cdot x_i \\
&= \mathbf{w}^T \mathbf{x}
\end{aligned}
$$

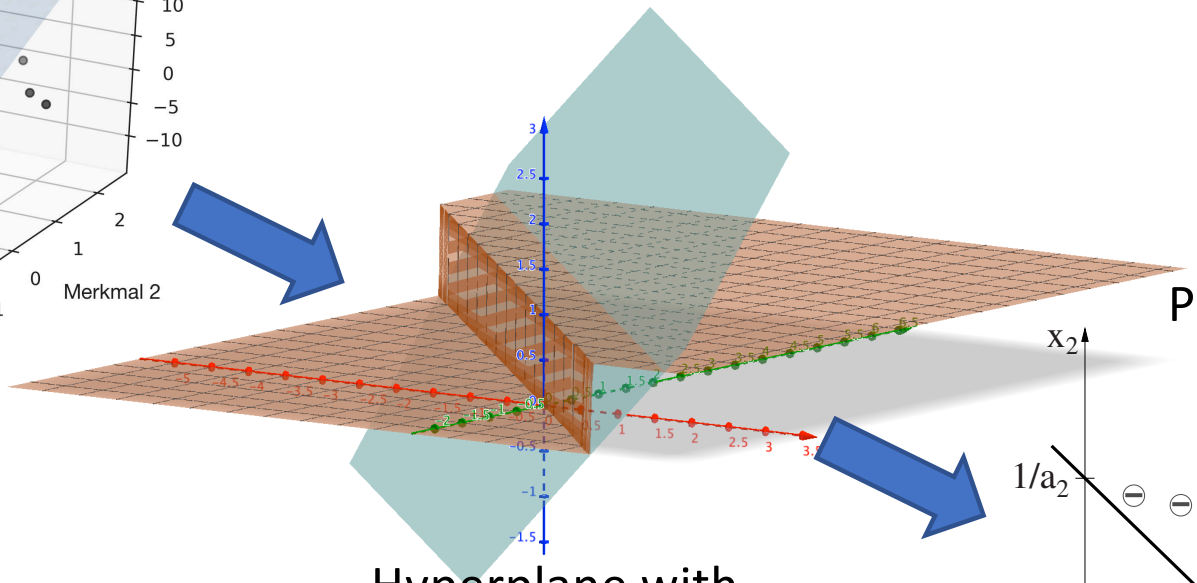
- Estimation by means of plane or hyperplane



Threshold value and hyperplane

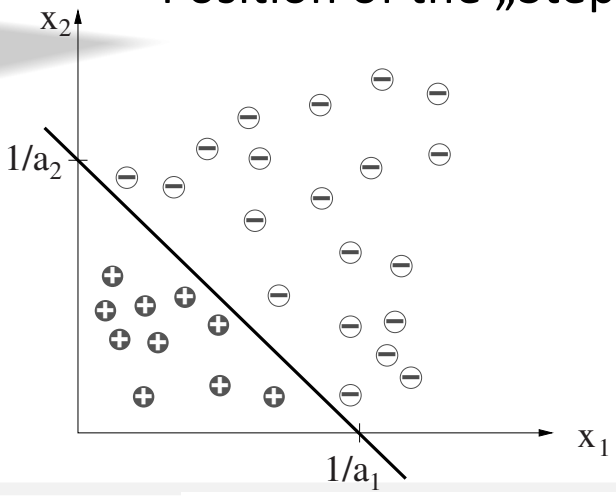


hyperplane



Hyperplane with step function

Position of the „Step“



Linear separability

Definition 8.2 Two sets $M_1 \subset \mathbb{R}^n$ and $M_2 \subset \mathbb{R}^n$ are called *linearly separable* if real numbers $a_1, \ldots, a_n, \theta$ exist with

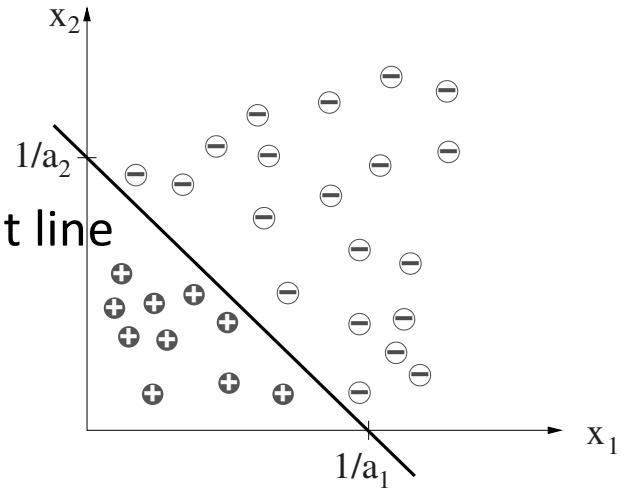
$$\sum_{i=1}^{n} a_i x_i > \theta \quad \text{for all } \boldsymbol{x} \in M_1 \quad \text{and} \quad \sum_{i=1}^{n} a_i x_i \leq \theta \quad \text{for all } \boldsymbol{x} \in M_2.$$

The value θ is denoted the threshold.

- In the simplest case: $a_1 x_1 + a_2 x_2 > 1$

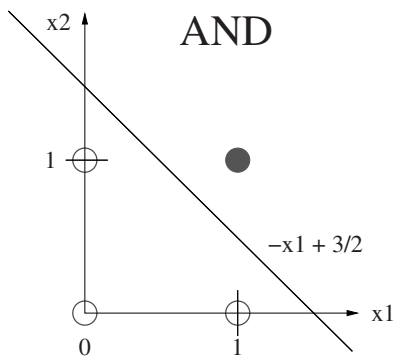
  $a_1 x_1 + a_2 x_2 \leq 1$

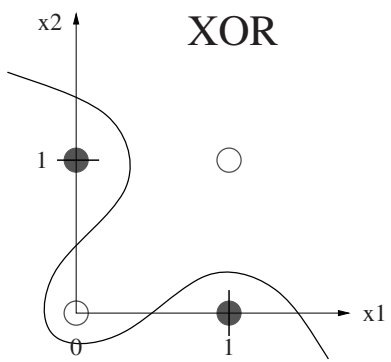
  Two-dimensional data can be separated by a straight line
- In n dimensions, the separation is performed by a Hyperplane required

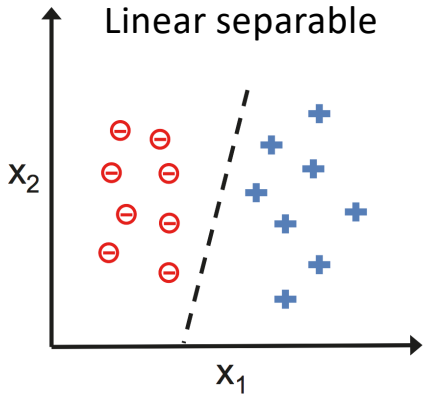


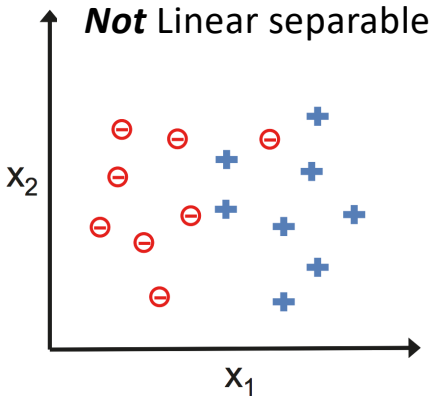
Linear separability

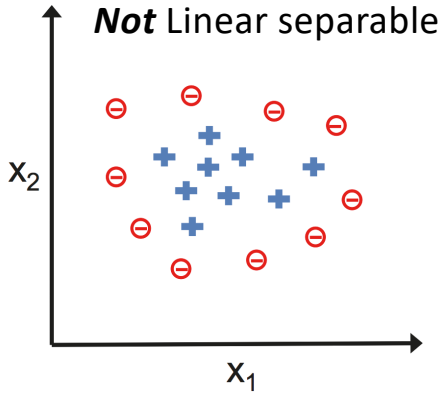
The Boolean function AND is linearly separable, XOR is not.
(• = true, ◦ = false).











Perzeptron learning rule

- $M+$ and M- are the quantities of positive and negative training patterns respectively

  Für $x \in M_+$ soll $P(x) = 1$ und
  für $x \in M_-$ soll $P(x) = 0$ liefern.

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

- Converts for each initialization of w if the problem is linearly separable.

Implementation with Scikit-Learn

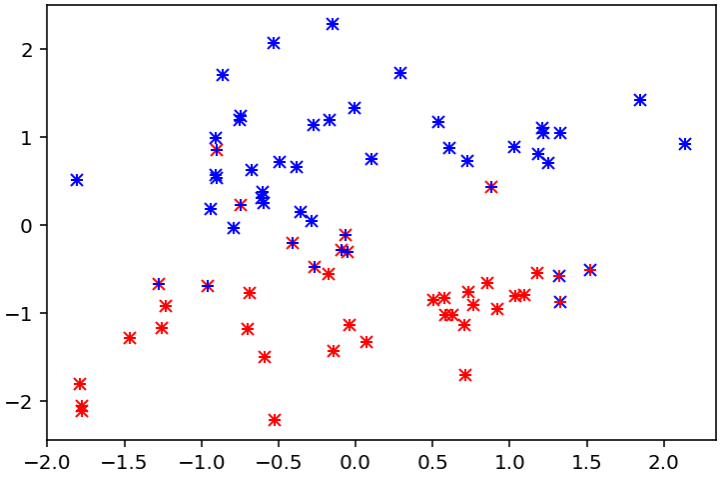
Training phase:

```
from sklearn.linear_model import Perceptron

model = Perceptron()
model.fit(X_train, y_train)

print("Trainingsdaten: ",model.score(X_train, y_train))
print("Testdaten: ",model.score(X_test, y_test))

print(model.coef_)
print(model.intercept_)
```

```
Trainingsdaten:  0.8378378378378378
Testdaten:  0.8266666666666667
[[-0.80556685  2.46648391]]
[1.]
```

Application phase:

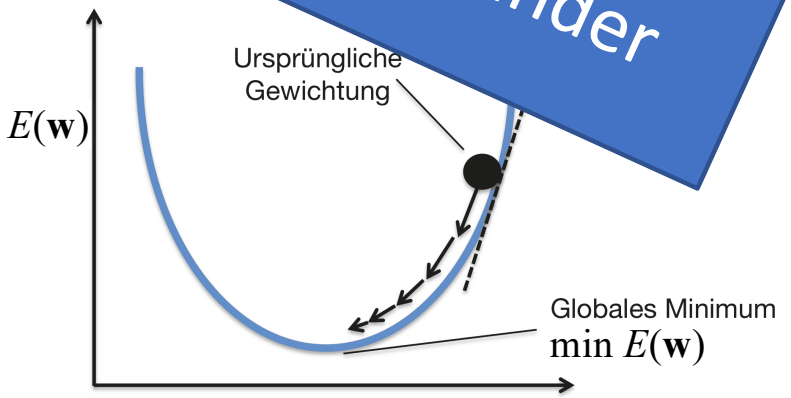
```
y_predicted = model.predict(X_test)

plt.plot(X_test[y_test==0,0], X_test[y_test==0,1],'rx')
plt.plot(X_test[y_test==1,0], X_test[y_test==1,1],'bx')
plt.plot(X_test[y_predicted==0,0], X_test[y_predicted==0,1],'r+')
plt.plot(X_test[y_predicted==1,0], X_test[y_predicted==1,1],'b+')

plt.show()
```



Determination of parameters (I)

- Minimization by means of gradient descent (gradient descent)
  - Objective function must be differentiable
  - Convergence to the global optimum requires Target function must be convex.

As a reminder



$$\mathbf{w}_{neu} := \mathbf{w}_{alt} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = -\eta\nabla E(\mathbf{w})$$

learning rate

Multidimensional gradient

Problems with the perceptron are caused by the differentiability of the jump function.

Adaptive linear neurons

ADALINE

- ADAptive LInear NEuron (Adaline)
- Bernard Widrow and his doctoral student Tedd Hoff formulate a differentiable error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=0}^{m}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{2}\sum_{i=0}^{m}\left(y_i - \phi(\mathbf{w}^T\mathbf{x}_i)\right)^2$$

B. Widrow et al. Adaptive "Adaline" neuron using chemical "memistors". Number Technical Report 1553-2. Stanford Electron. Labs. Stanford, CA, Oktober 1960

- *Updating of* weights is based on a linear **activation function,** the identity function

$$\phi(\mathbf{w}^T\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i$$

- The step function is only used for *prediction.*
- Thus E(w) can be differentiated again

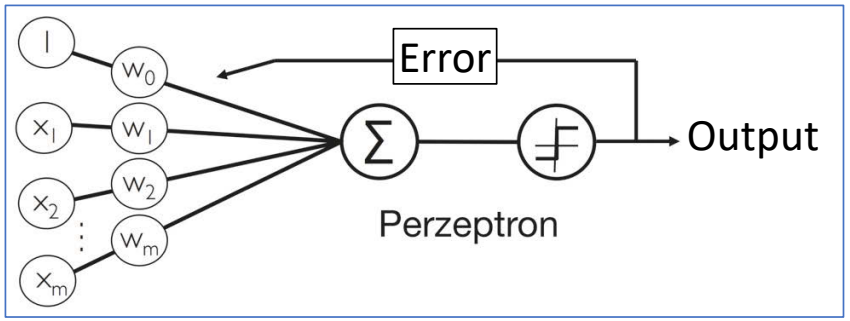
  => Numerical optimization e.g. by gradient descent

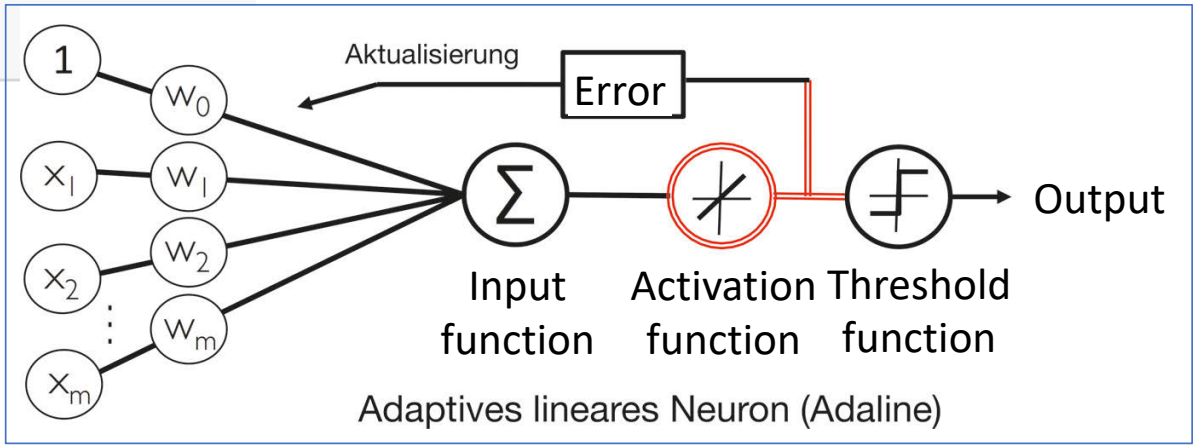
Adaline implementation

training phase

```
from sklearn.linear_model import SGDClassifier

model = SGDClassifier(loss="perceptron", max_iter=5000)
model.fit(X_train, y_train)

print("Trainingsdaten: ",model.score(X_train, y_train))
print("Testdaten: ",model.score(X_test, y_test))

print(model.coef_)
print(model.intercept_)
```

```
Trainingsdaten:  0.8513513513513513
Testdaten:  0.8666666666666667
[[-0.00385616  0.02738256]]
[0.00903053]
```





Logistic Regression

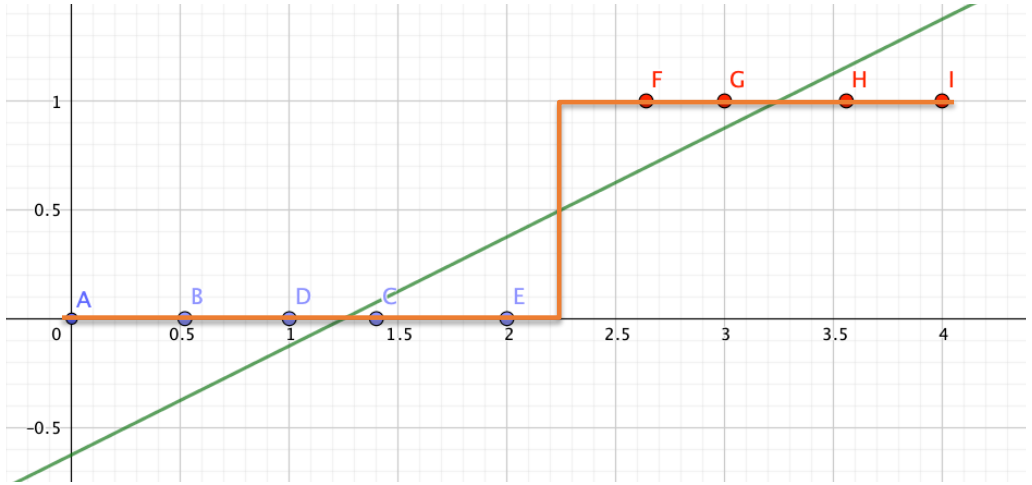
Changed activation function

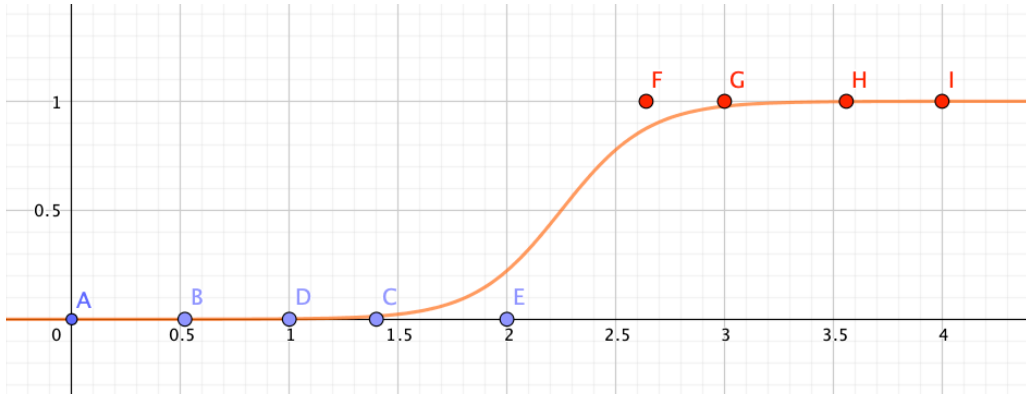
- Threshold function provides very hard class boundary
- Threshold function cannot be differentiated
- Idea: Finding a continuously differentiable function that is similar to a threshold value
- Sigmoidal functions, e.g:
  - Fermi function (logistic function):

$$f(x) = \frac{1}{1 + e^{-ax+b}}$$

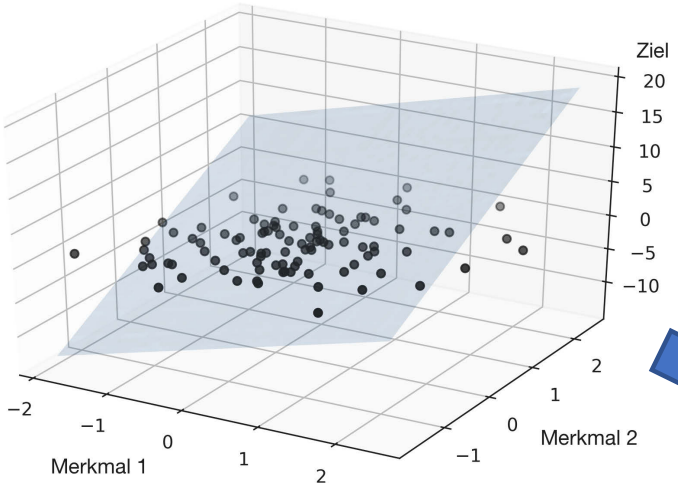
  - hyperbolic tangent

$$f(x) = tanh(x)$$

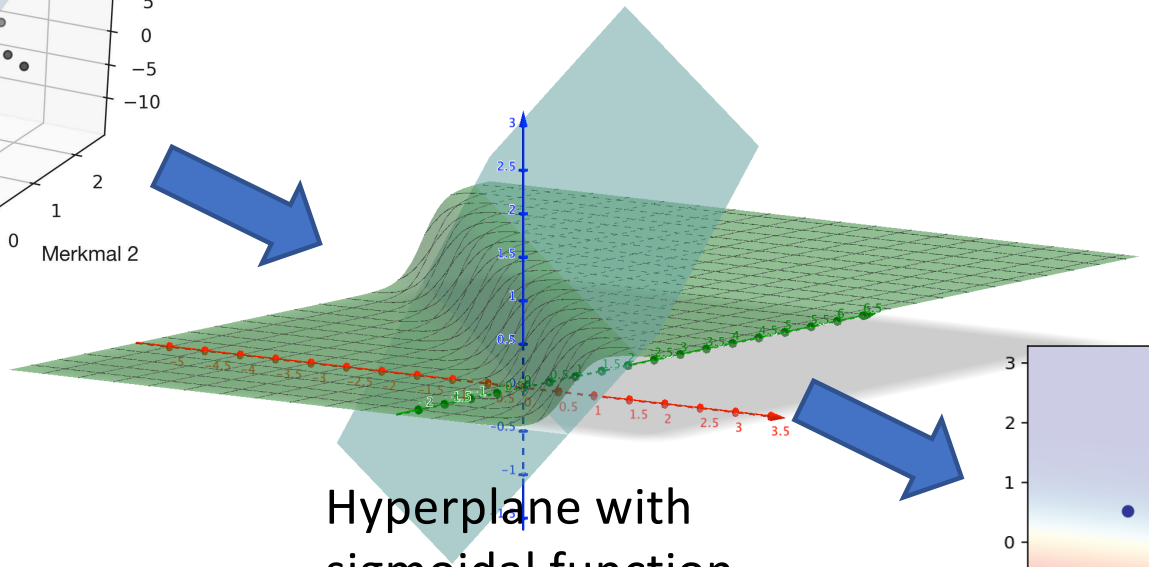




Sigmoidal function and hyperplane

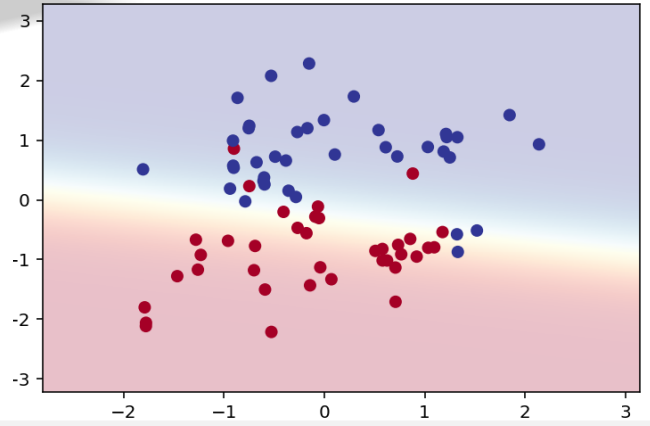


hyperplane



Hyperplane with sigmoidal function

Gradual gradient between classes



Implementation in Scikit-Learn

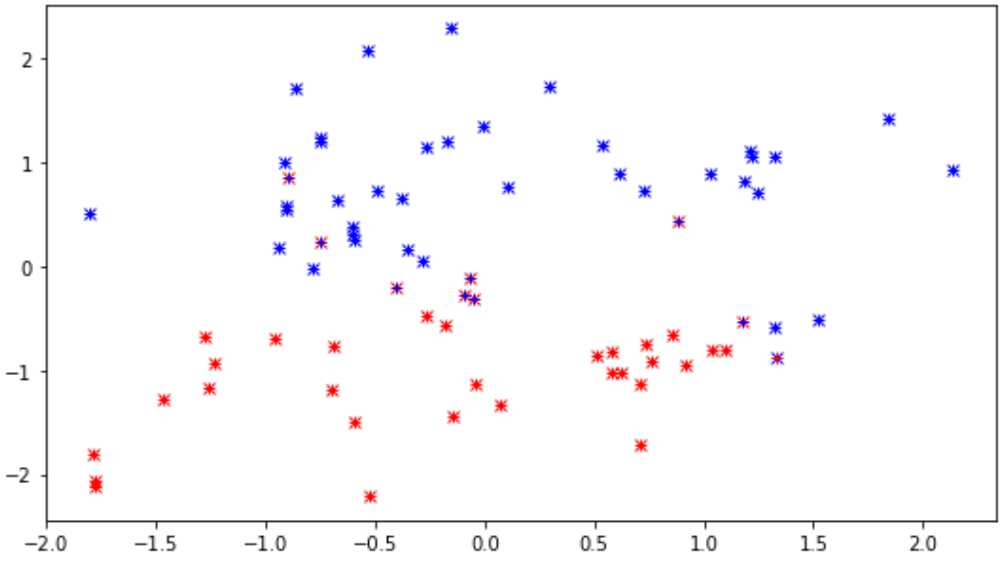
training phase

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)

print("Trainingsdaten: ",model.score(X_train, y_train))
print("Testdaten: ",model.score(X_test, y_test))

print(model.coef_)
print(model.intercept_)
```

```
Trainingsdaten:  0.8783783783783784
Testdaten:  0.88
[[0.53963126 2.9823613 ]]
[1.05441661]
```

Application phase:

```
y_predicted = model.predict(X_test)

plt.plot(X_test[y_test==0,0], X_test[y_test==0,1],'rx')
plt.plot(X_test[y_test==1,0], X_test[y_test==1,1],'bx')
plt.plot(X_test[y_predicted==0,0], X_test[y_predicted==0,1],'r+')
plt.plot(X_test[y_predicted==1,0], X_test[y_predicted==1,1],'b+')

plt.show()
```



Implementation in Scikit-Learn

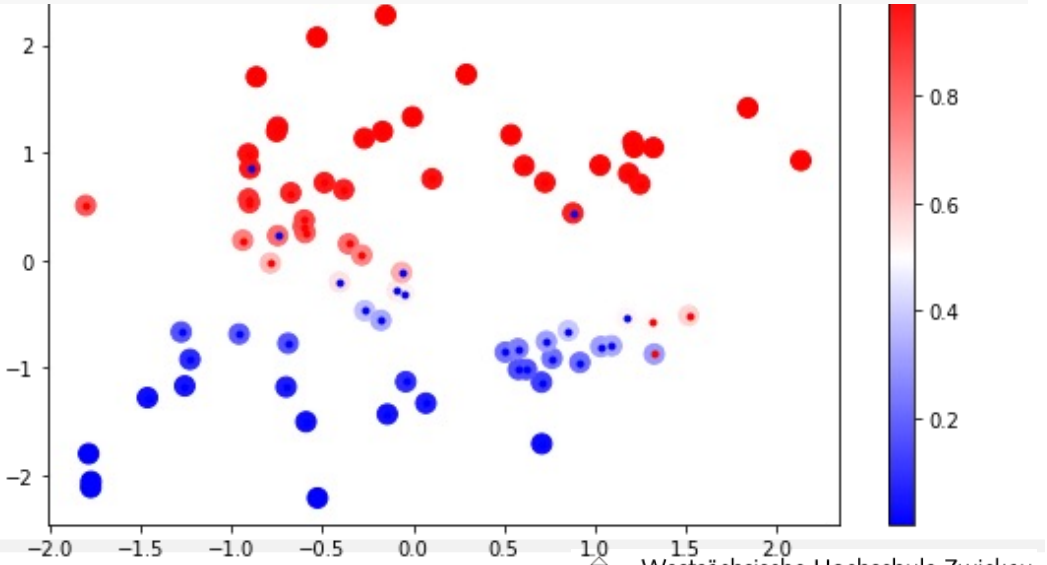
training phase

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)

print("Trainingsdaten: ",model.score(X_train, y_train))
print("Testdaten: ",model.score(X_test, y_test))

print(model.coef_)
print(model.intercept_)
```

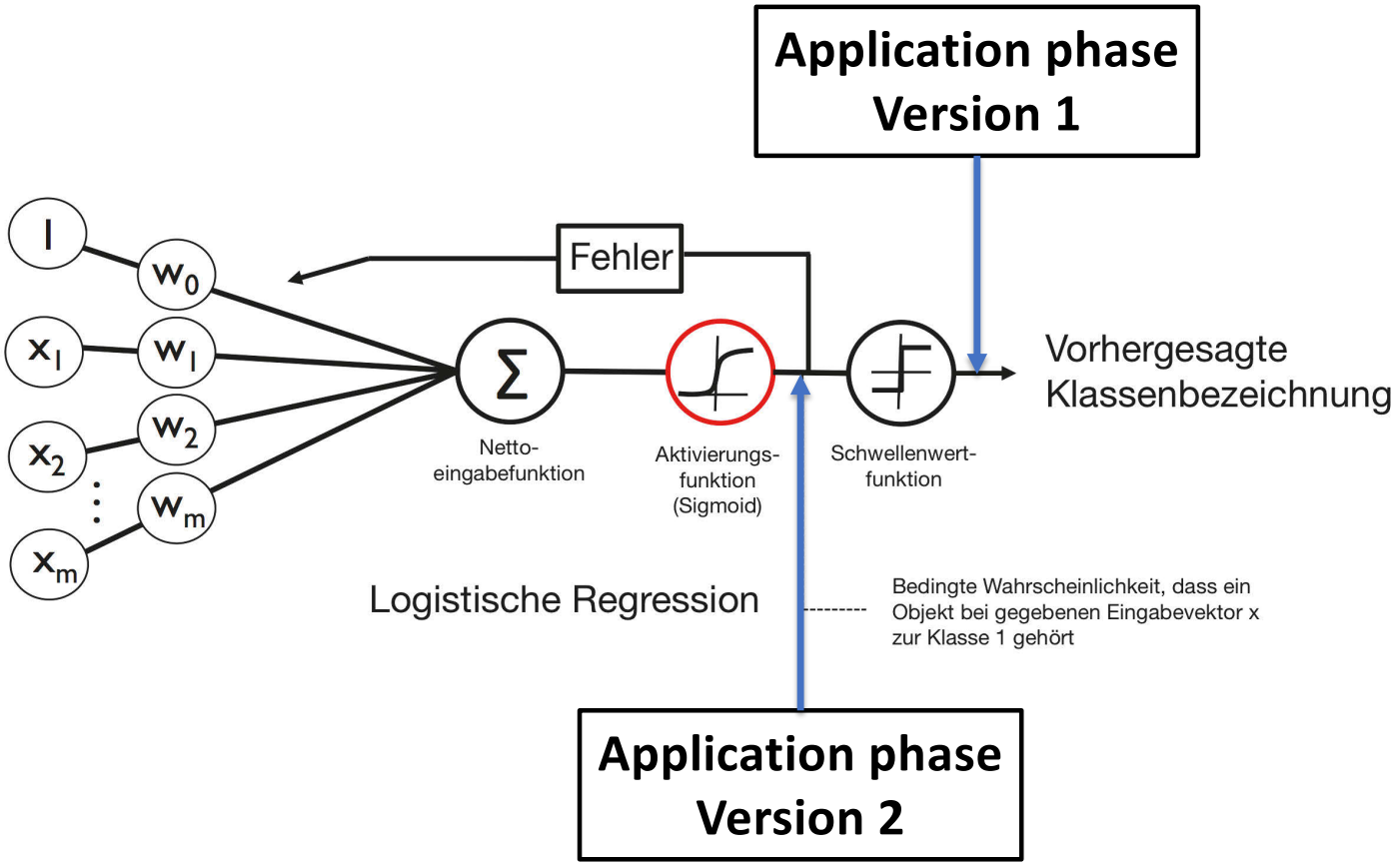
```
Trainingsdaten:  0.8783783783783784
Testdaten:  0.88
[[0.53963126 2.9823613 ]]
[1.05441661]
```

Application phase - Version 2:

```
y_predicted = model.predict_proba(X_test)[:,1]


plt.scatter(X_test[:,0], X_test[:,1],
            c=y_predicted, s=100,cmap='bwr')
plt.plot(X_test[y_test==0,0], X_test[y_test==0,1],'b.')
plt.plot(X_test[y_test==1,0], X_test[y_test==1,1],'r.')
plt.colorbar()
```



Implementation in Scikit-Learn



data preprocessing

- In many applications, the use of features with very different value ranges is unavoidable, e.g. in image processing.
  - Gray values: 0...255,
  - fractal dimension: 2.0...3.0
  - etc.
- Problems with distance calculation (e.g. Euclidean distance): Characteristics with a large range of values dominate the distance calculation considerably.
- Necessity to
  - Reduction of the influence of features with a wide range of values
  - Increasing the influence of features with a low value range
- If necessary, remove correlation of features => *dimension reduction*

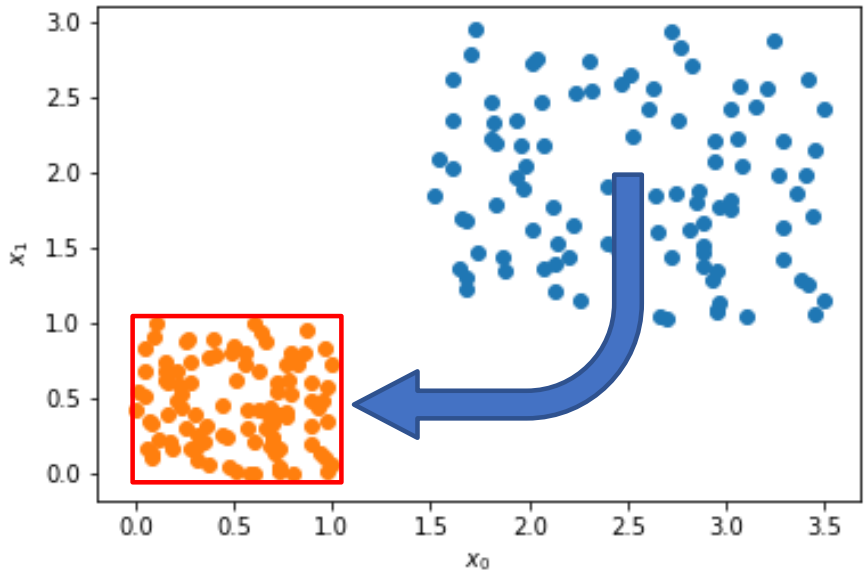
feature scaling

- **Objective: To** transform the value ranges of all characteristics into a meaningful interval:

$$\forall x_i : 0 \leq x_i \leq 1 \text{ d.h. } x_i \in [0, 1]$$
$$\forall x_i : -1 \leq x_i \leq 1 \text{ d.h. } x_i \in [-1, 1]$$

- **Disadvantage**: "Outliers" retain influence on the arrangement of the scaled feature vectors in the feature space, because "nonoutliers" are pushed into a small area of the feature space after scaling.

```
from sklearn.preprocessing import minmax_scale
xn = minmax_scale(x)
```



$$\tilde{x_i} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

mit $x_{min} = \min_i x_i$

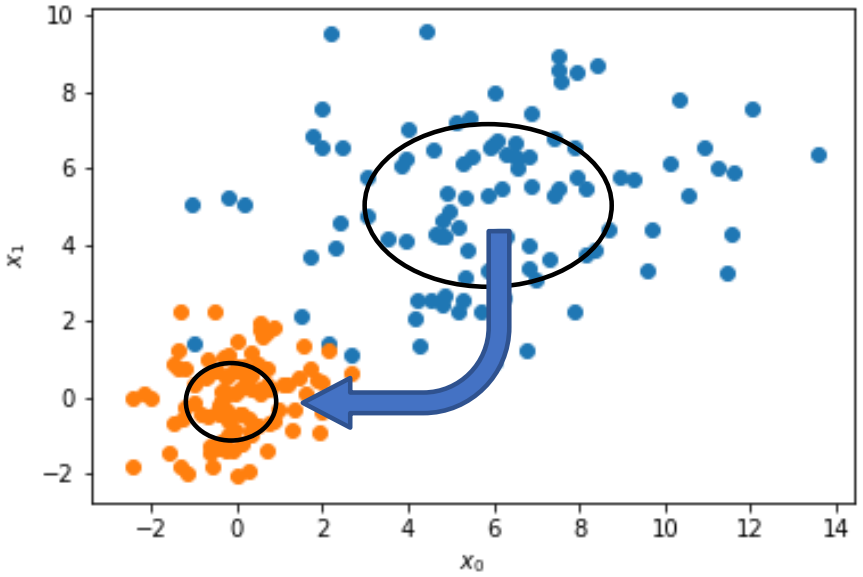
und $x_{max} = \max_i x_i$

characteristic normalization

- **Objective**: Generation
  - average-free feature vectors
    => mean value exemption
  - with standardized standard deviation (σ = 1.0)
    => Standardization

- "Outliers" - with little influence on the arrangement of the standardized data units in the characteristic space.
- Range of values of the different characteristics approximated to each other
- Problems with variant-based procedures

```
from sklearn.preprocessing import scale
xs = scale(x)
```



$$\tilde{x_i} = \frac{x_i - \mu}{\sigma}$$

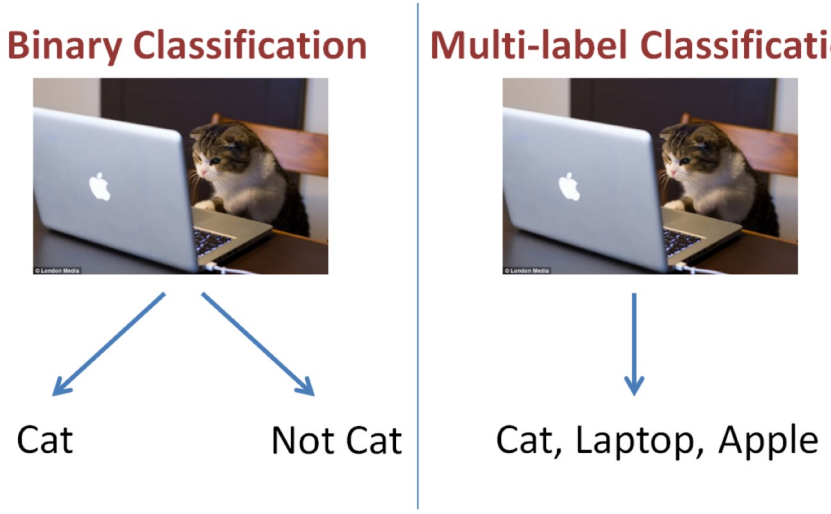
mit $\mu = \frac{1}{n}\sum_i x_i$

und $\sigma^2 = \frac{1}{n}\sum_i (x_i - \mu)^2$

multi-class problems

multi-class problems

- Some classifiers can only distinguish between two classes:
  for example: Sick vs. healthy
- Many problems with more than two classes:
  for example: Cold vs. bronchitis vs. flu



Terms:

- Multiclass Learning:
  Each training example belongs to one category
  $y_i \in \{1, \ldots, L\}$
- Multi-label problems:
  Each training example can belong to several categories

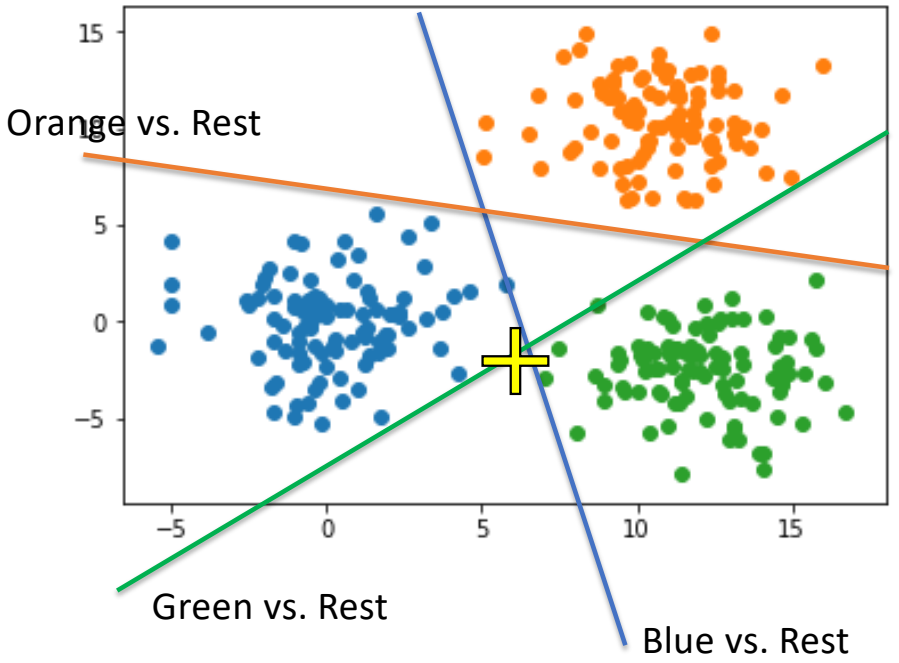
$$y_i \in \{0,1\}^L \text{ (bzw. } Y_i \subseteq \{1, \ldots, L\})$$

Possible solutions:

- Use of a suitable classifier
- One vs. All Approach
- One vs. one approach

One vs. All / One vs. Rest

- Train one classifier per class => k models
- Each classifier delimits each class from all other data from



- Final decision e.g. about Winner Takes All Strategy:
  - Green vs. rest: 95% Green
  - Orange vs. balance: 10% Orange
  - Blue vs. balance:7% Blue

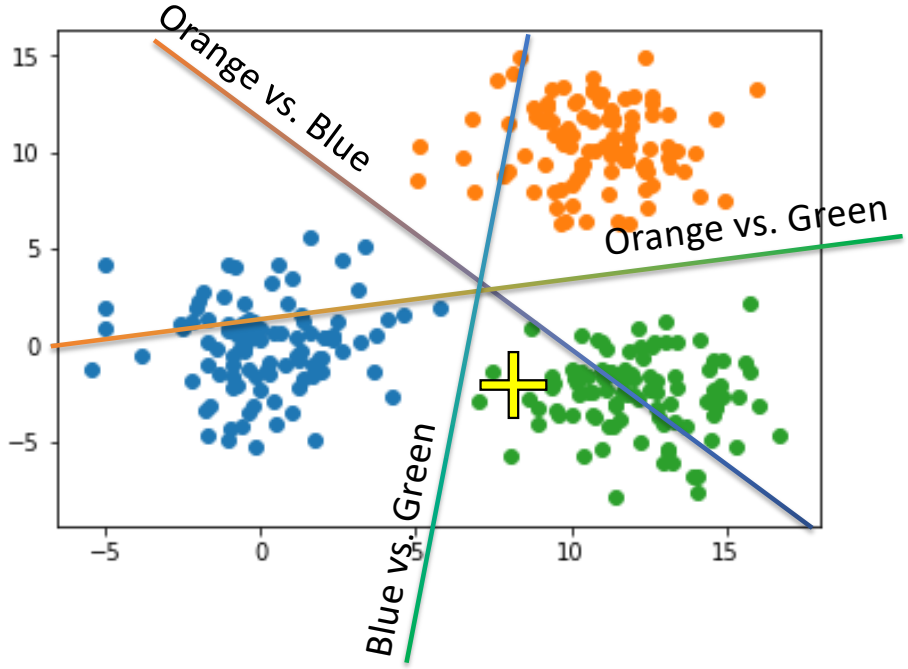
- **Disadvantage**: Unbalanced data approaches per classifier
- `sklearn.linear_model.LogisticRegression` uses 1-vs.-all

One vs. One

- Training of one classifier per possible combination of 2 classes each
- Each classifier delimits these two class pairs from each other, and
- Ignores remaining data

- Final decision e.g. about Majority Voting:
  - Green vs. Blue: **Green**
  - Orange vs. Blue: Blue
  - Blue vs. Green: **Green**

- Less training data per model



$$\frac{k \cdot (k-1)}{2} \text{ Modelle}$$

Which is better?

Hsu, Chih-Wei and Chih-Jen Lin. “A comparison of methods for multiclass support vector machines.” *IEEE transactions on neural networks* 13 2 (2002): 415-25 .

- Depends on data set!

TABLE V.2

A COMPARISON USING THE RBF KERNEL (BEST RATES BOLD-FACED)

	One-against-one		DAG		One-against-all		[25], [27]		C&S	
Problem	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate
iris	$(2^{12},2^{-9})$	**97.333**	$(2^{12},2^{-8})$	96.667	$(2^{9},2^{-3})$	96.667	$(2^{12},2^{-8})$	**97.333**	$(2^{10},2^{-7})$	**97.333**
wine	$(2^{7},2^{-10})$	**99.438**	$(2^{6},2^{-9})$	98.876	$(2^{7},2^{-6})$	98.876	$(2^{0},2^{-2})$	98.876	$(2^{1},2^{-3})$	98.876
glass	$(2^{11},2^{-2})$	71.495	$(2^{12},2^{-3})$	**73.832**	$(2^{11},2^{-2})$	71.963	$(2^{9},2^{-4})$	71.028	$(2^{4},2^{1})$	71.963
vowel	$(2^{4},2^{0})$	**99.053**	$(2^{2},2^{2})$	98.674	$(2^{4},2^{1})$	98.485	$(2^{3},2^{0})$	98.485	$(2^{1},2^{3})$	98.674
vehicle	$(2^{9},2^{-3})$	86.643	$(2^{11},2^{-5})$	86.052	$(2^{11},2^{-4})$	**87.470**	$(2^{10},2^{-4})$	86.998	$(2^{9},2^{-4})$	86.761
segment	$(2^{6},2^{0})$	97.403	$(2^{11},2^{-3})$	97.359	$(2^{7},2^{0})$	97.532	$(2^{5},2^{0})$	**97.576**	$(2^{0},2^{3})$	97.316
dna	$(2^{3},2^{-6})$	95.447	$(2^{3},2^{-6})$	95.447	$(2^{2},2^{-6})$	95.784	$(2^{4},2^{-6})$	95.616	$(2^{1},2^{-6})$	**95.869**
satimage	$(2^{4},2^{0})$	91.3	$(2^{4},2^{0})$	91.25	$(2^{2},2^{1})$	91.7	$(2^{3},2^{0})$	91.25	$(2^{2},2^{2})$	**92.35**
letter	$(2^{4},2^{2})$	**97.98**	$(2^{4},2^{2})$	**97.98**	$(2^{2},2^{2})$	97.88	$(2^{1},2^{2})$	97.76	$(2^{3},2^{2})$	97.68
shuttle	$(2^{11},2^{3})$	99.924	$(2^{11},2^{3})$	99.924	$(2^{9},2^{4})$	99.910	$(2^{9},2^{4})$	99.910	$(2^{12},2^{4})$	**99.938**

Multinomial Logistic Regression

- One vs. all strategy provides probabilities for each individual classification action problem:
  - Green vs. rest: 95% Green
  - Orange vs. balance: 10% Orange
  - Blue vs. balance:7% Blue
- The result cannot therefore be interpreted as a probability of the class.

- Model is defined using softmax function:

$$p(y = i|z) = \text{softmax}(z_i)$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \text{ mit } \mathbf{W} \in \mathbb{R}^{L \times N} \text{ und } \mathbf{x} \in \mathbb{R}^{N}$$

evaluation measures

Selection of a suitable classifier

- Different algorithms have certain peculiarities or are based on certain assumptions
- "No Free Lunch theorem by David H. Wolpert:
  There is no classifier that would be suitable for all conceivable scenarios.
- Ergo: Comparison of potentially suitable algorithms unavoidable in practice
- Preselection depends on:
  - Number of characteristics
  - signal-to-noise ratio
  - Linear separability of data
  - computational power

The Lack of A Priori Distinctions Between Learning Algorithms, Wolpert, David H., Neural Computation (1996): 1341–1390

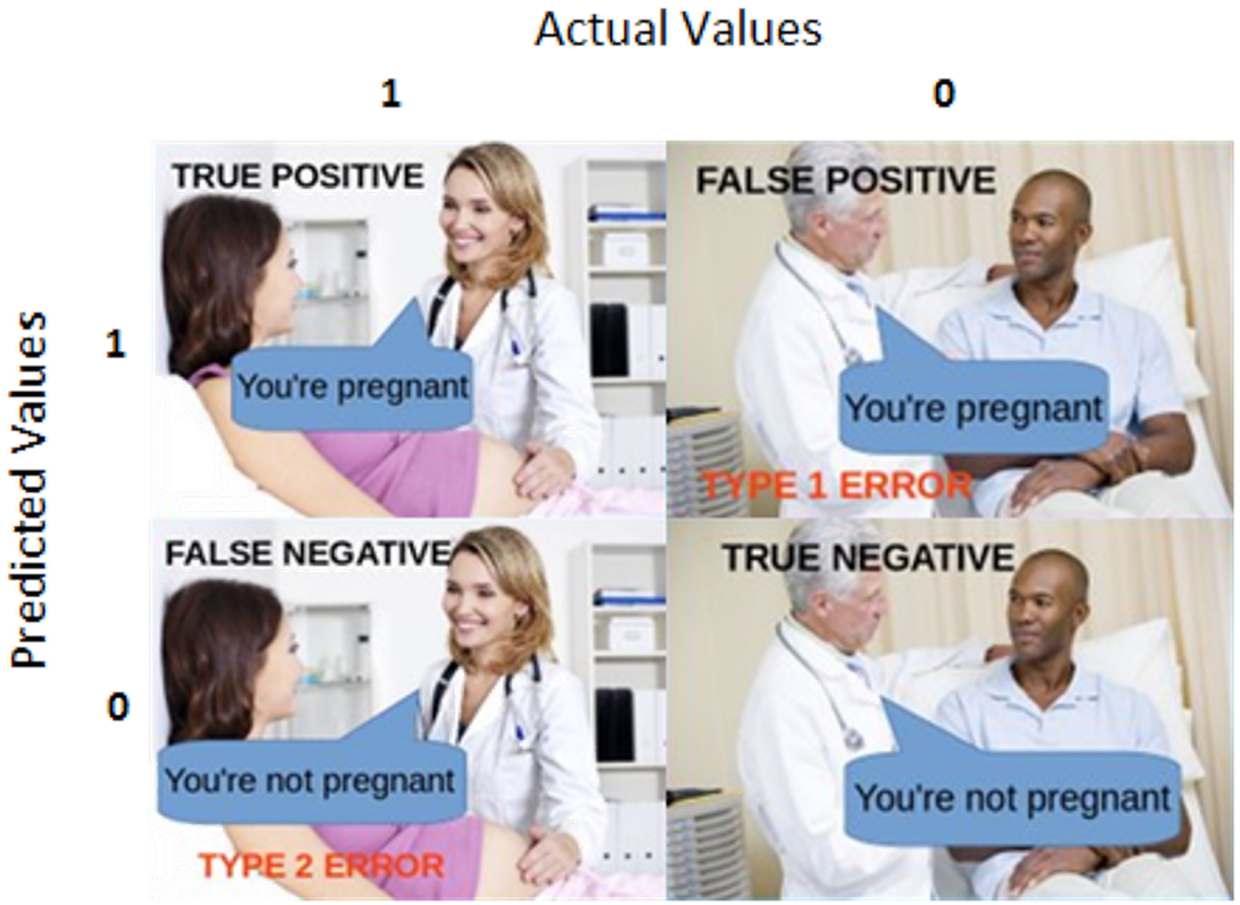
Confusion Matrix

- Confusion matrix for binary classification problems

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

- **True Positive:** Prediction is positive and that is correct
- **True Negative:** Prediction is negative and that is correct
- **False positives:** Prediction is (erroneously) positive, but that is false.
- **False Negative:** Prediction is (erroneously) negative, but this is wrong.

Example


Actual Values
1
0
Predicted Values
1
0
TRUE POSITIVE
You're pregnant
FALSE POSITIVE
You're pregnant
TYPE 1 ERROR
FALSE NEGATIVE
You're not pregnant
TYPE 2 ERROR
TRUE NEGATIVE
You're not pregnant

Measures for evaluating a classifier

- Correct classification rate (probability of confidence) (accuracy)
  - Quota of correctly classified data points

$$ACC = \frac{TN + TP}{TN + FP + FN + TP} = \frac{TN + TP}{Total}$$

  - Implementierung in ScikitLearn:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

  - Or

```
model.score(X_test_std, y_test)
```

- Metric not always useful (but still often used!):
  - Given: 1000 examples (995 negatives, 5 positives)
  - A classifier that basically classifies all inputs as *negative* has an accuracy on the example data of 99.5%.

Alternative dimensions

- Accuracy (Precision)
  The correctly classified from all classes

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

- Hit rate (Recall, Sensitivity, true positve rate)
  The correctly classified of all positive examples => As high as possible

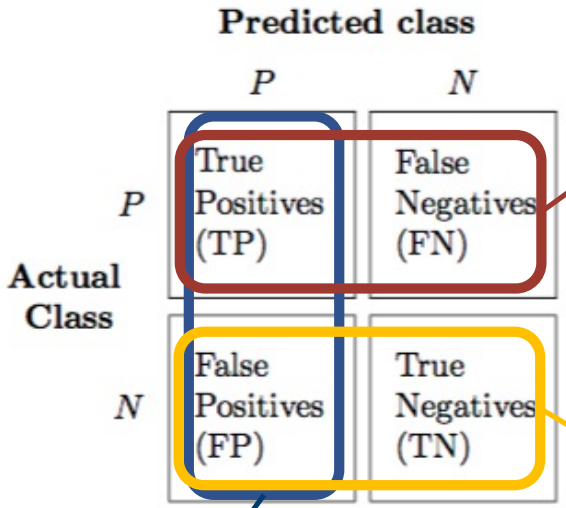
$$recall = TPR = \frac{TP}{FN + TP}$$

- False positive rate
  The misclassified of all negative examples

$$FPR = \frac{FP}{TN + FP}$$

Dimensions in the Confusion Matrix

$$recall = TPR = \frac{TP}{FN + TP}$$



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

$$FPR = \frac{FP}{TN + FP}$$

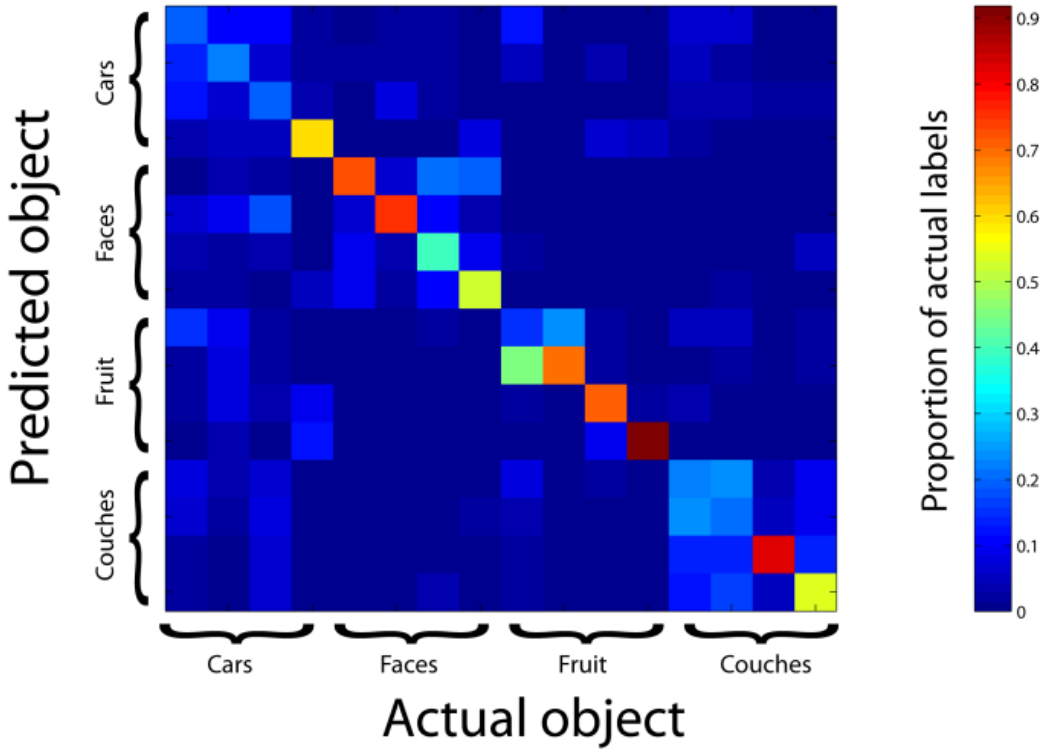
There are many more...

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

multi-class problems

- Confusion-Matrix can also be applied to multiclass problems
  - Similarities between the classes are recognizable



Ying Zhang, Ethan M. Meyers, Narcisse P. Bichot, Thomas Serre, Tomaso A. Poggio, Robert Desimone: Object decoding with attention in inferior temporal cortex. Proceedings of the National Academy of Sciences May 2011, 108 (21) 8850-8855; DOI:10.1073/pnas.1100999108

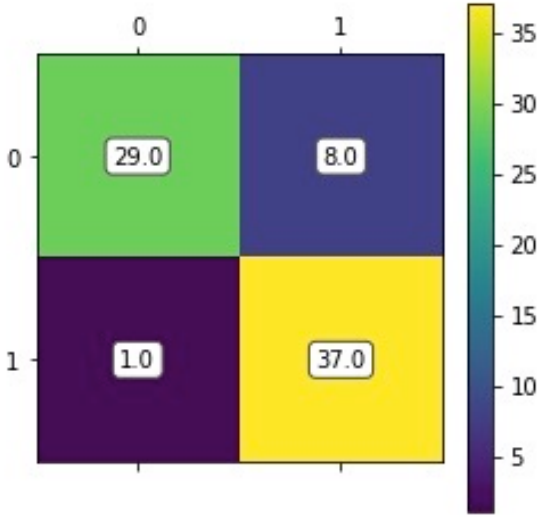
Confusion Matrix in Scikit-Learn

```
y_test_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_test_pred)
print(cm)
```

```
[[29  8]
 [ 1 37]]
```

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

plt.matshow(cm)

for (i, j), z in np.ndenumerate(cm):
    plt.text(j, i, '{:0.1f}'.format(z),
             ha='center', va='center',
             bbox=dict(boxstyle='round',
                       facecolor='white',
                       edgecolor='0.3'))

plt.colorbar()
```



ROC curves

Introductory example

- **Given:** Classification model already trained on problem

- How to understand the predictive probabilities of our classifier
- Derive different classifiers by varying the decision threshold:

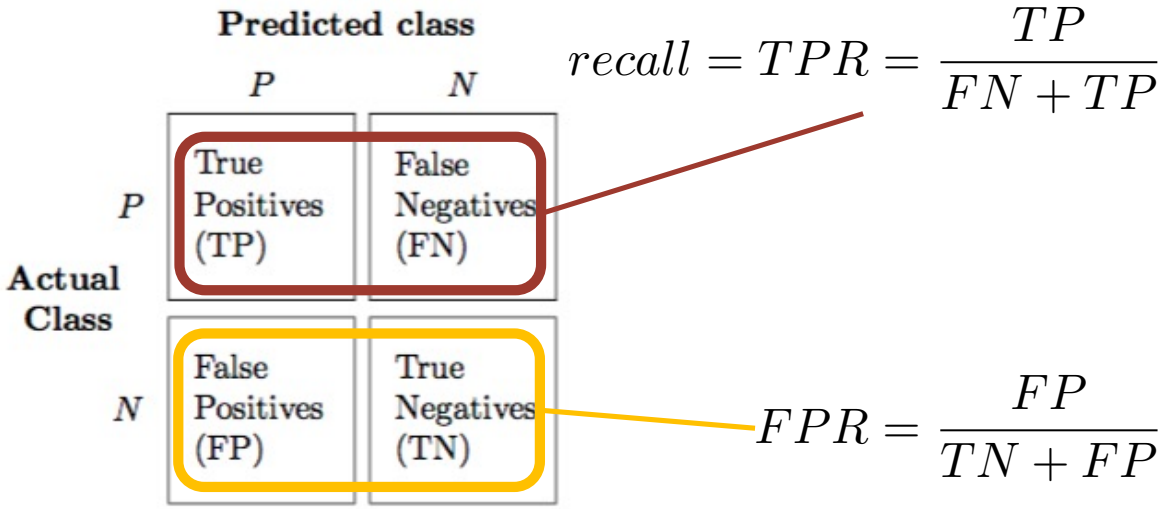
	Actual class	P(sick)	Always "sick"	Threshold at 5%	Threshold at 50%	Threshold at 80%	Always "healthy"
Patient 1	Sick	90%					
Patient 2	Sick	52%					
Patient 3	Healthy	10%					
Patient 4	Healthy	1%					

Play it safe and treat healthy patients as well

In the case of high-risk treatment, the classification must be reliable.

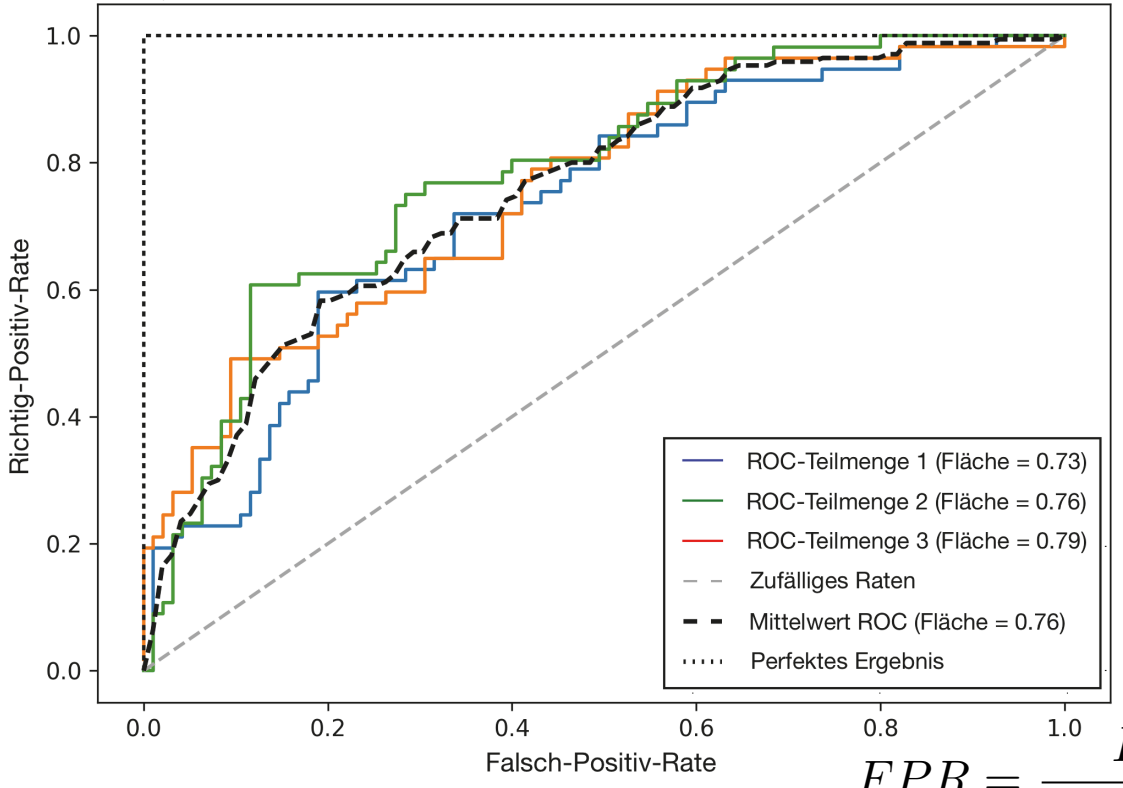
We take with us

- Classifier variants not dependent on the course
- Depending on the application, different properties can be important.
- **Objective:** We are looking for a way to present as much information as possible about a classifier.
- Does the Confusion Matrix help us?
  - In principle, but too much. Information at once
  - Break down to two values, that fully cover the matrix



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

- Receiver Operator Characteristics
- TPR & FNR Calculate for Different Decision Limits
- Perfect classifier: Upper left corner
- Summarizing the statement of a ROC curve: Area under ROC curve (AUC)



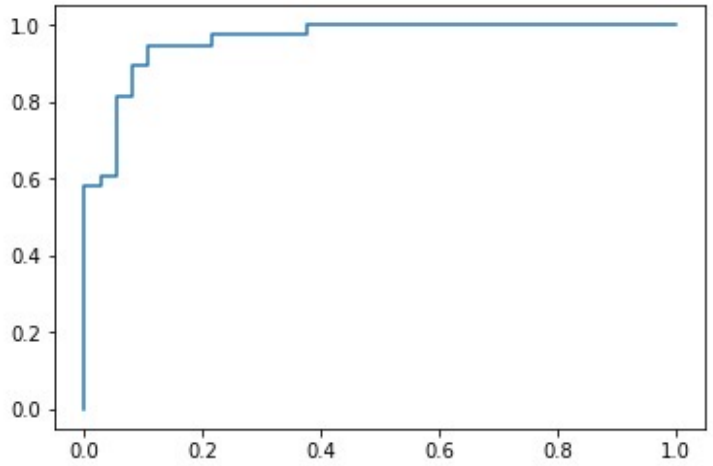
$$FPR = \frac{FP}{TN + FP}$$

Scikit-Learn implementation

```
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, y_test_proba)
```

```
y_test_proba = model.predict_proba(X_test)[:, 1]
```

```
import matplotlib.pyplot as plt

plt.plot(fpr, tpr)
plt.show()
```



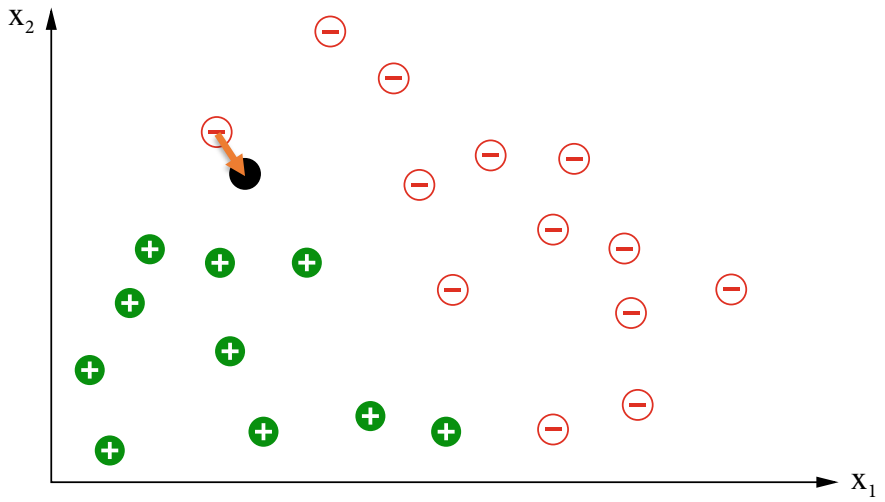

```
roc_auc_score(y_test, y_test_proba)
```

```
0.9601706970128023
```

K-Nearest Neighbor

basic principle

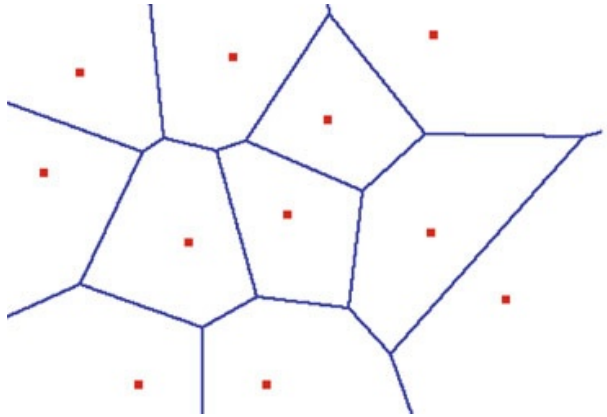
- Basic idea comparable with diagnosis of diseases:
  - Diagnose a new case and trace it back to similar known cases
  - Simplest form: Using the most similar case (*Nearest Neighbor*)
  - issue
    - How is similarity defined?
      => Use a suitable
      Distance dimension: The smaller the distance, the more so similar
    - How similar do two cases have to be in order to to justify the diagnosis?



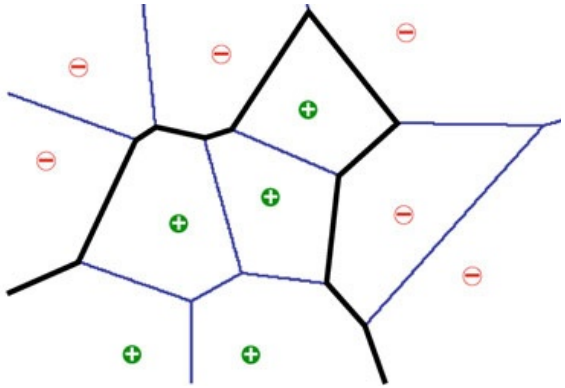
- **Lazy-Learning**: Training examples are not processed

Voronoi regions

- In this way, the example data divides the characteristic space: **Voronoi regions**
- Done on the basis of the distance dimension
- *Voronoi region of an example data point includes all points of the space for which it is the nearest neighbor*



- Class boundary is defined by neighborhood relationship:
  - Border runs between points of different class
  - Each along the regional border

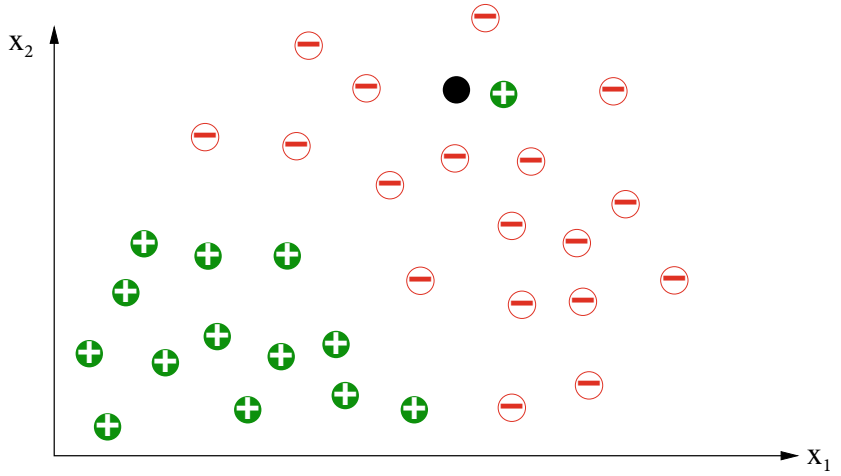


- Enables a non-linear class boundary

Problem with Nearest Neighbor approach

- An incorrect example data point leads to misclassifications.
- Leads to overfitting (overfitting)

- Solution: Expand the approach:
  - Include multiple neighbors:
    k-Nearest Neighbor
  - Class selection via majority decision



k-NEARESTNEIGHBOR(M_+, M_-, s)

$V = \{k \text{ nearest neighbors in } M_+ \cup M_-\}$
If $|M_+ \cap V| > |M_- \cap V|$ **Then** **Return**(„+")
ElseIf $|M_+ \cap V| < |M_- \cap V|$ **Then** **Return**(„–")
Else **Return**(Random(„+", „–"))

frontiers

- Approach easily extendable to any number of classes
- Classifier adapts immediately to new data

- Problem with a lot of training data: High computing time (O(n))
  (Often accompanied by a large number of classes)

- *k* getting bigger*:* typically more neighbours with a large distance than those with a small distance => dominance of distant neighbours
- Solution: Weighting the influence in the decision

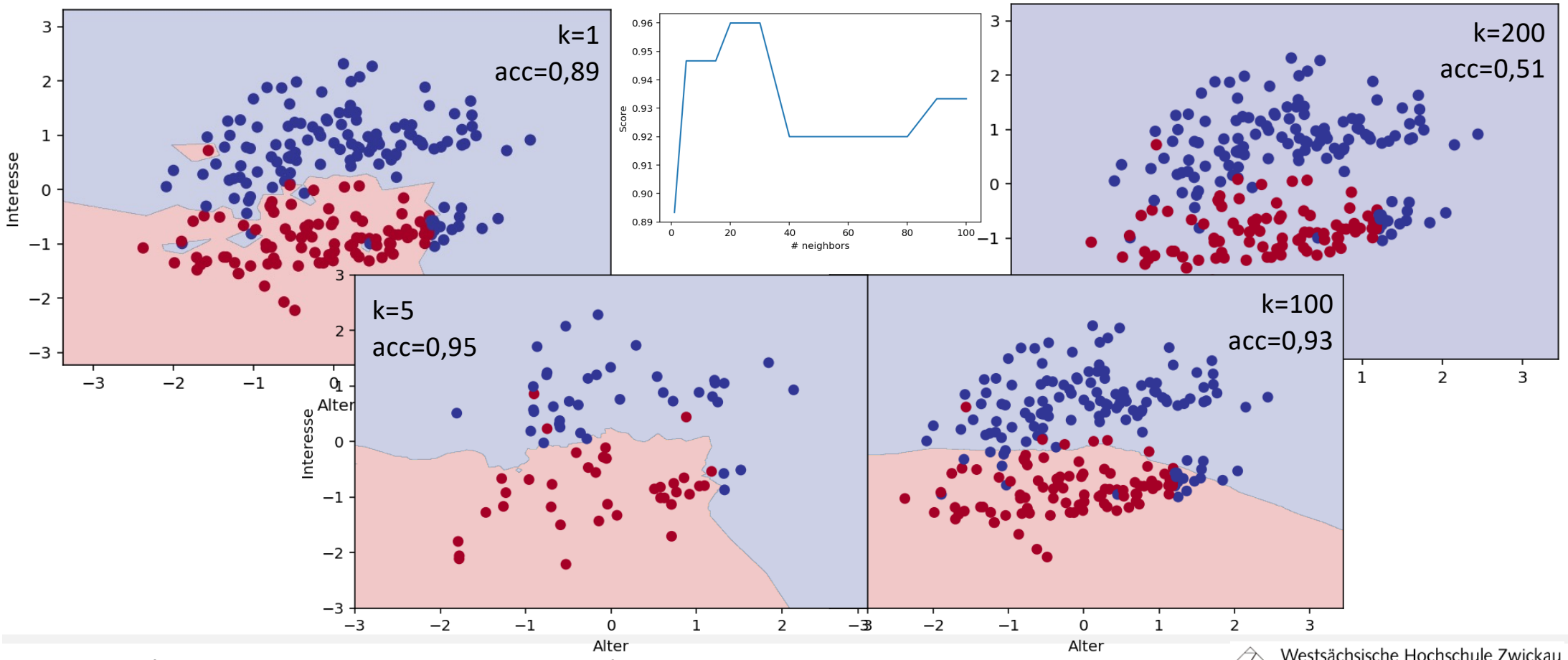
$$w_i = \frac{1}{1 + \alpha d(\boldsymbol{x}, \boldsymbol{x}_i)^2}$$

- Expandable to regression problems by weighted average of the given function values of the training examples:

$$\hat{f}(\boldsymbol{x}) = \frac{\sum_{i=1}^{k} w_i f(\boldsymbol{x}_i)}{\sum_{i=1}^{k} w_i}.$$

Choice of number of neighbours

- Over-adjustment if too few neighbours Under-adjustment if too many neighbours



Implementation with Scikit-Learn

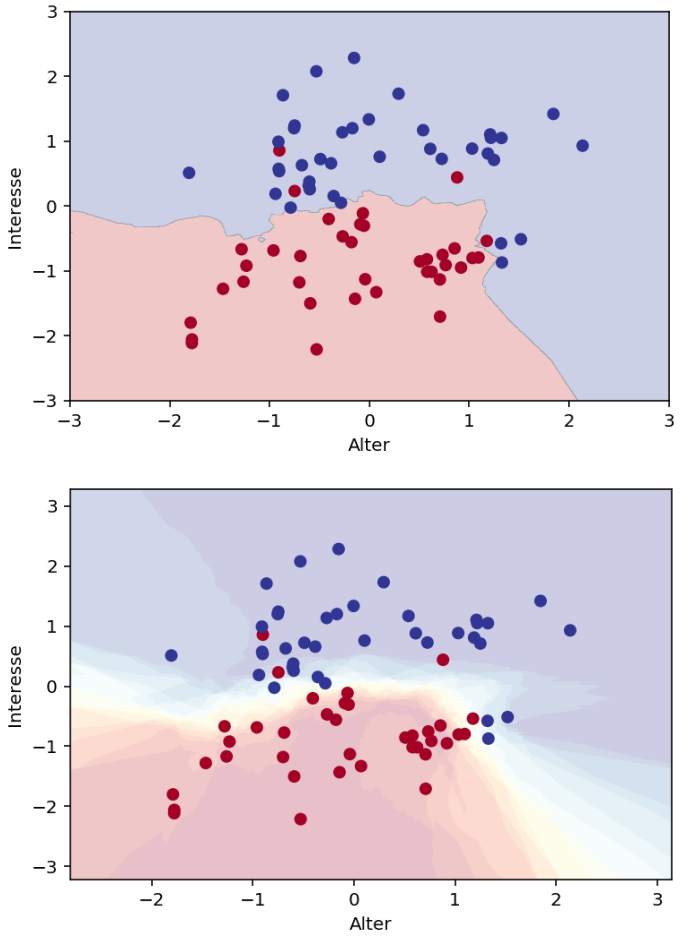
"Training."

```
1  from sklearn.neighbors import KNeighborsClassifier
2
3  model = KNeighborsClassifier(n_neighbors = 15,
4                               weights = 'uniform',
5                               metric = 'minkowski', p=2)
6  model.fit(X_train, y_train)
7
8  print(model.score(X_test, y_test))
```

$$d\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sqrt[p]{\sum_k \left|x_k^{(i)} - x_k^{(j)}\right|^p}$$

Application

```
1  predicted = model.predict(X_test)
2  predicted_proba = model.predict_proba(X_test)
```



Curse of Dimensionality

- Curse of Dimensionality

- High-dimensional feature space occupies thinner space than low-dimensional feature space (with same training data quantity)
- Influence of individual dimensions decreases with distance calculation

- K-Nearest Neighbor therefore very susceptible to over-fitting:
  - even the closest neighbours are too far apart to be able to make a good assessment

- Solution:
  - Feature Selection
  - dimension reduction

Support Vector Machine

decision trees

entropy

Random Forests