Universität Stuttgart

Pattern Recognition Chapter 10: Nonprobabilistic Discriminative Classifiers

Prof. Dr.-Ing. Uwe Sörgel
soergel@ifp.uni-stuttgart.de

ifp

Non-probabilistic discriminative classifiers

- Goal: Definition of a function $f(\mathbf{x})$ that predicts the class label C from the data $\mathbf{x}$, i.e. $C = f(\mathbf{x})$
- Probabilities are not considered directly in this context.
- No assumptions about the distribution of the data!
- Focus on decision boundaries

  → Good results with a relatively low amount of training data.

- Posterior probabilities can usually be derived in post-processing
  - Required for further processing in a probabilistic context

Non-probabilistic discriminative classifiers: Overview

- Different principles:
  - Decision Trees: Hierarchical classification of feature space
  - Random Forests: Combination of decision trees
  - Boosting: Combination of weak classifiers
  - Support Vector Machines: Find decision boundary having a maximum distance from the training samples.
  - Neural Networks: Motivated by a model of information flow in neuron (cells of the nervous system)
  - etc.

Contents

Decision trees

- Many problems in everyday life are analyzed by going through and answering a series of questions.

- Example: “Is the grass wet?“

  - 1. Question 1: “Does it rain“?

    If yes, the grass is probably wet
    If not:

    → 2. Question 2: “Is the lawn sprinkler on?“

    If yes, the grass is probably wet
    If not, the grass is probably not wet

Decision trees

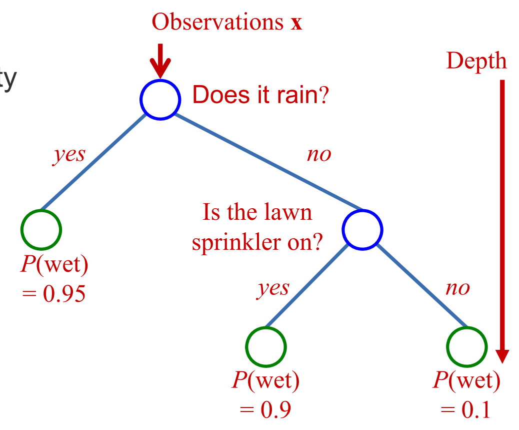
- This sequence of queries can be represented by a binary decision tree.

- Node ○: Queries / Decisions

- Leafs ○: Either a result or a probability

- Binary Tree: Every node that is no leaf has two child nodes.

- Depth of the tree



Example from remote sensing

- Features:
  - x_1: nDOM (normalized OM)
  - x_2: NDVI

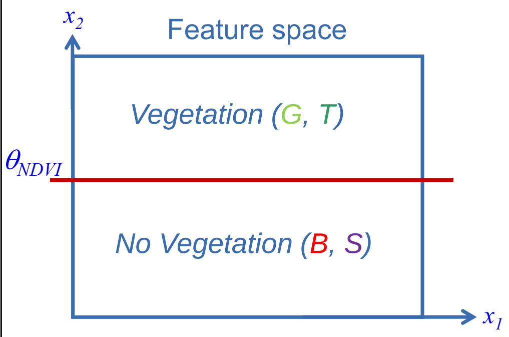
nDOM

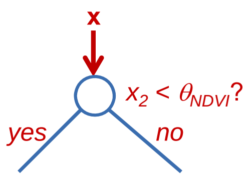
NDVI

- Classes: Buildings (B), Street (S), Grass (G), Tree (T)

Example from remote sensing

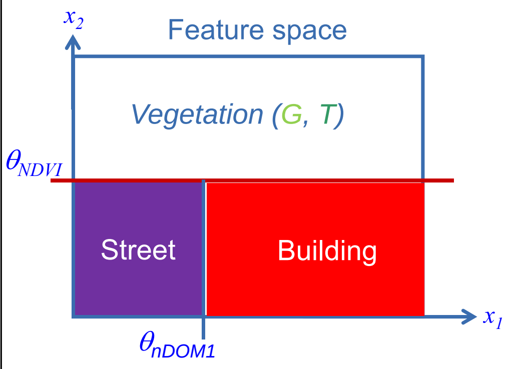
- Each decision splits the feature space up into sub-regions.
- Feature vector $\mathbf{x} = (x_1, x_2)^T$ is presented to the root node
  1. Decision: Is x_2 (NDVI) smaller than a threshold value θ_{NDVI}?

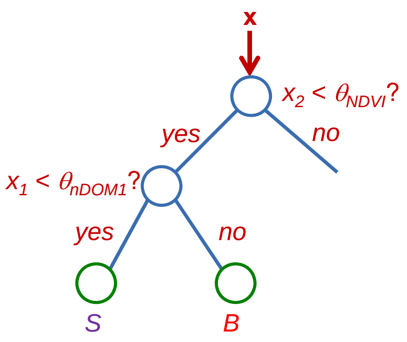




Example from remote sensing

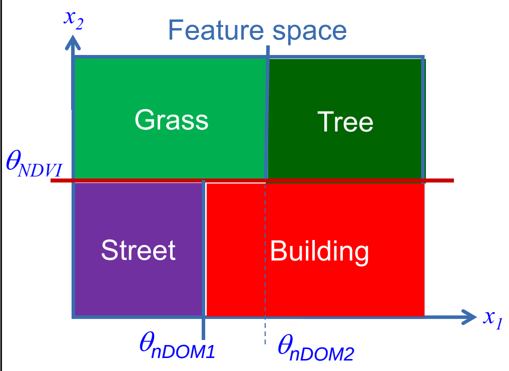
- Each decision splits the feature space up into sub-regions.
- Feature vector $\mathbf{x} = (x_1, x_2)^T$ is presented to the root node
  1. Decision: Is x_2 (NDVI) smaller than a threshold value θ_{NDVI}?
  2. Decision 2 for *no vegetation*: Is x_1 smaller than as θ_{nDOM1}?

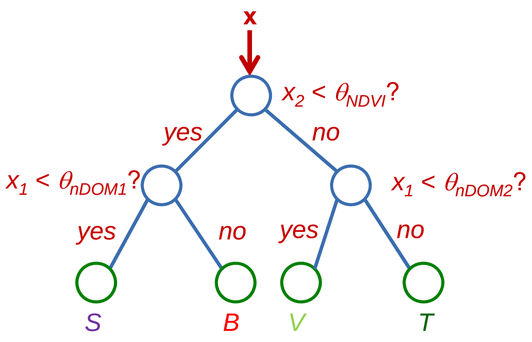




Example from remote sensing

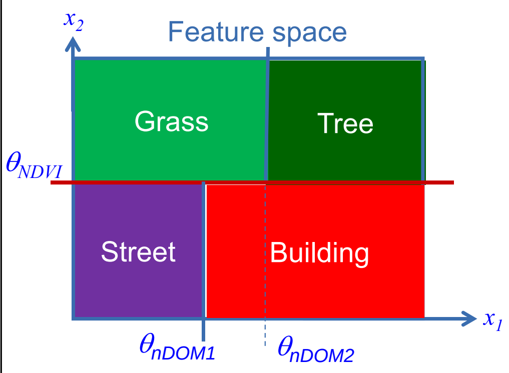
- Each decision splits the feature space up into sub-regions.
- Feature vector $\mathbf{x} = (x_1, x_2)^T$ is presented to the root node
  1. Decision: Is x_2 (NDVI) smaller than a threshold value θ_{NDVI}?
  2. Decision 2 for *no vegetation*: Is x_1 smaller than θ_{nDOM1}?
  3. Decision 2 for *vegetation*: Is x_1 smaller than θ_{nDOM2}?

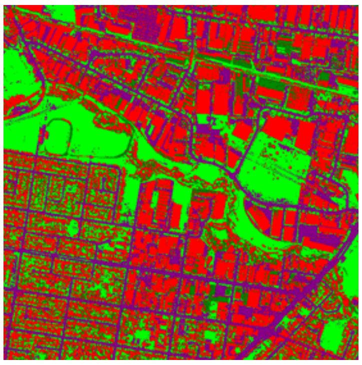




Example from remote sensing

- The feature space is hierarchically split into disjunct regions.
- Using different values for the three parameters (θ_{NDVI}, θ_{nDOM1}, θ_{nDOM2}) lead to different results.





Decision trees: Discussion

- Very simple and "clear" design → very popular
- Can be adapted by the user easily (choice of thresholds).
- This partitioning of the feature space does not adapt very well to the shapes of the clusters in feature space.
- Result depends on the choice of the threshold values.
- Different possibilities for the construction of the tree.
  - Can these trees be learned for the training data?
  - Is there a better way to adapt the decision boundaries than interactive trial-and-error?

CART (Classification and Regression Trees)

- General method for learning of binary trees
- Applicable for classification, regression and clustering
- There are different versions of CART
- What is to be determined during training?
  - How to split the data in each node?
  - How to decide whether a node corresponds to a leaf or not?
  - How to determine which class corresponds to a leaf?

CART: Splitting of data

- A test is carried out in each node.
- Up to now: Each test is based on the comparison of a single feature with a threshold value.
- More general type of test: Split the feature space with a linear decision boundary (a hyperplane):
  - Simultaneous consideration of several features.
  - Allows for decision boundaries in feature space that are **not** parallel to coordinate axes
- The type of the tests (threshold vs. hyperplane) must be defined in advance.
-
- The parameters of the tests can be learned.

CART: Learning of the tests

- Learning the tests only requires a part (e.g. 1/3) of the training data.
- In each node of the tree i:
  - Randomly select n_i features.
  - Randomly generate r different separating hyperplanes operating on the selected features.
  - Each hyperplane is examined according to how well it can separate the data
    → For example, information gain criterion
  - The best hyperplane is retained for the node.

CART: Selection of the separating hyperplane

- The number of features for the test has to be specified by the user; good value for D-dimensional feature vectors:

$$n_i = \sqrt{D}$$

- **Random** choice of n_i features.
- Separating hyperplane $\mathbf{w}^T \cdot \mathbf{x} + w_0 = 0$
  - → For n_i selected features: weight vector $\mathbf{w}$ built from n_i random numbers in $[-1, 1]$
  - → Bias w_0 is a random number between $[min\,(\mathbf{w}^T \cdot \mathbf{x}),\ max\,(\mathbf{w}^T \cdot \mathbf{x})]$
- The hyperplane splits the training data in two parts M_1, M_2:

$$M_1: \quad \mathbf{w}^T \cdot \mathbf{x} + w_0 \le 0$$
$$M_2: \quad \mathbf{w}^T \cdot \mathbf{x} + w_0 > 0$$

- M_1 and M_2 correspond to the branches leaving the node.

CART: Information gain criterion

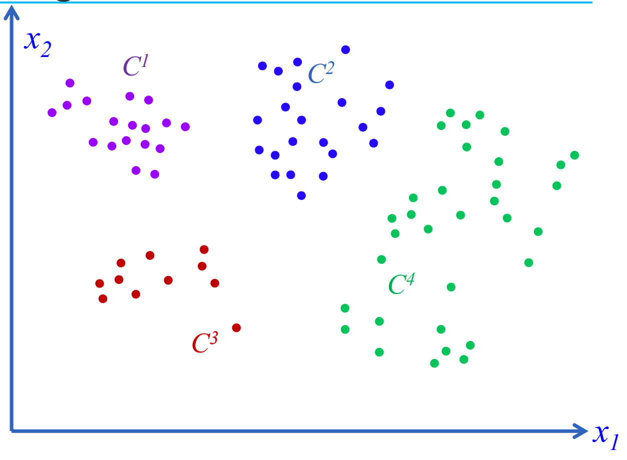
- For each of the subsets (M_1, M_2) generated by the test, a histogram of the class labels can be determined.
- The histogram entries for M_j are interpreted as $P_j(C^k)$.
- Possible criterion for the quality of the separation: **information gain ΔE**:

$$\Delta E = \frac{N_1}{N_1+N_2}\cdot\sum_k P_1\left(C^k\right)\cdot\log_2\left[P_1\left(C^k\right)\right]+\frac{N_2}{N_1+N_2}\cdot\sum_k P_2\left(C^k\right)\cdot\log_2\left[P_2\left(C^k\right)\right]$$

  - N_1, N_2 are the number of training samples in M_1 and M_2, respectively.
  - Each of the sums (except of sign) is the entropy E of the histogram.
  - The bigger ΔE, the better a hyperplane separates the data.

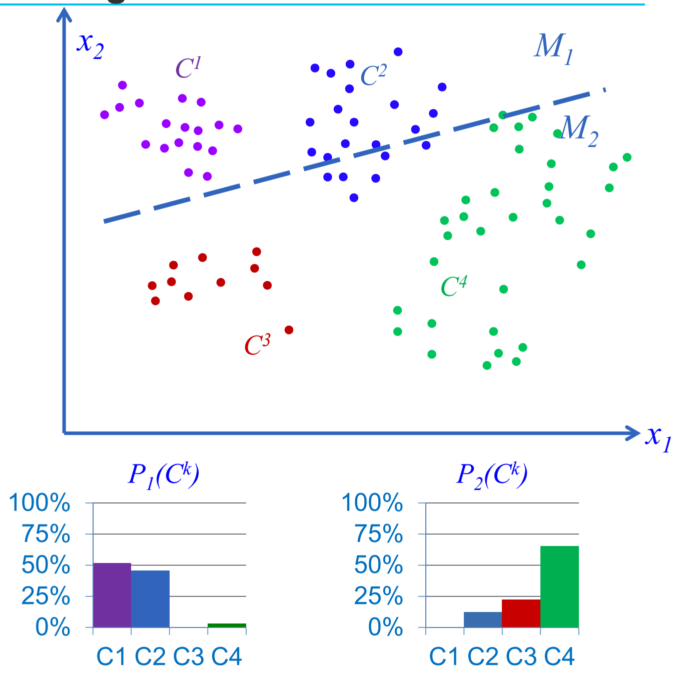
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:



CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -1.215

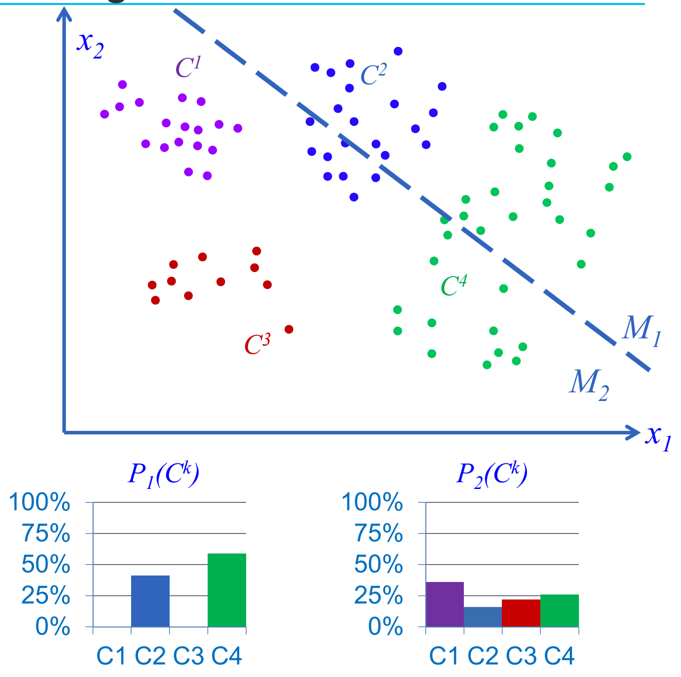


Universität Stuttgart

ifp

CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -1.550

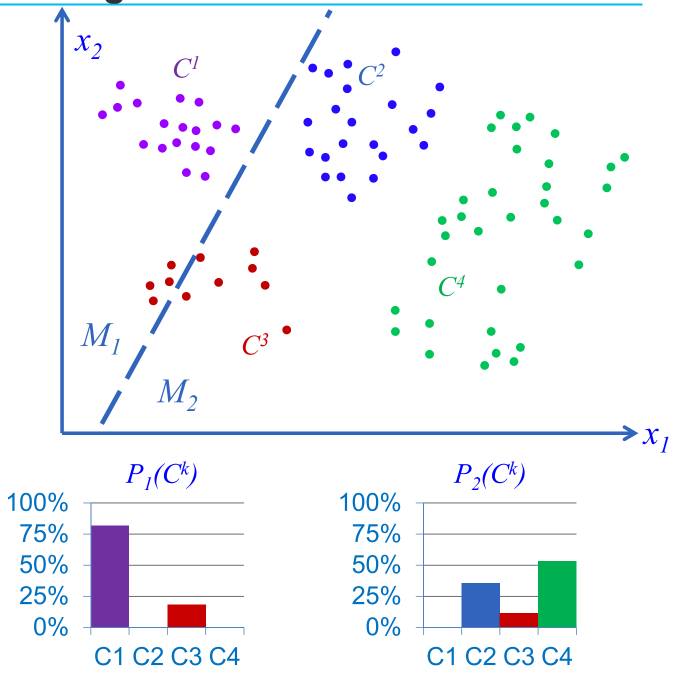


Universität Stuttgart

ifp

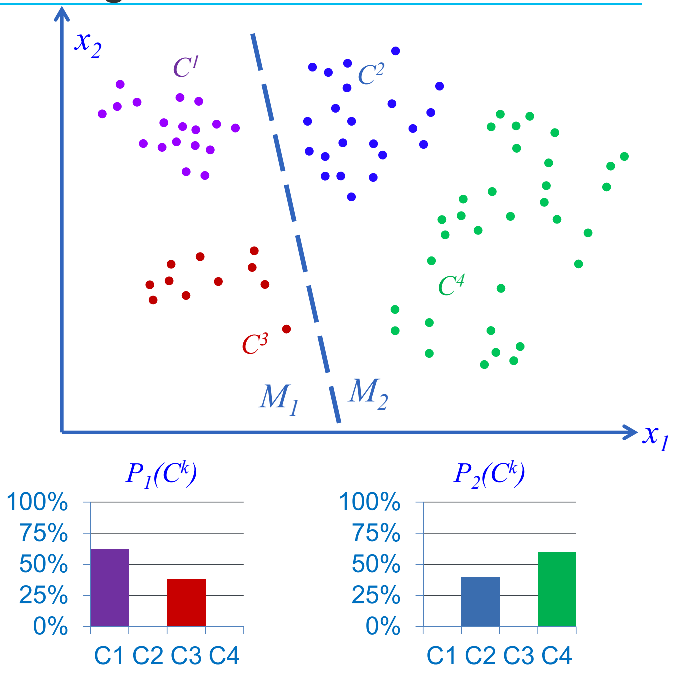
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -1.190



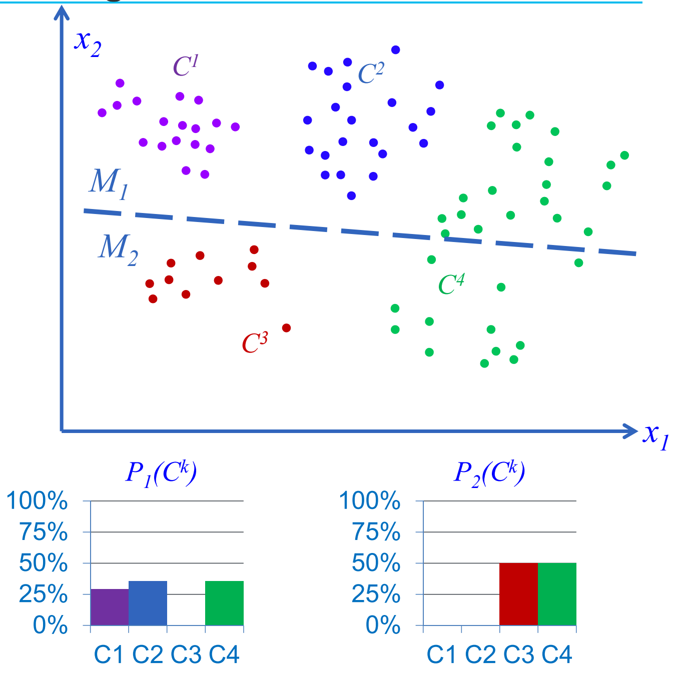
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -0.966



CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -1.427

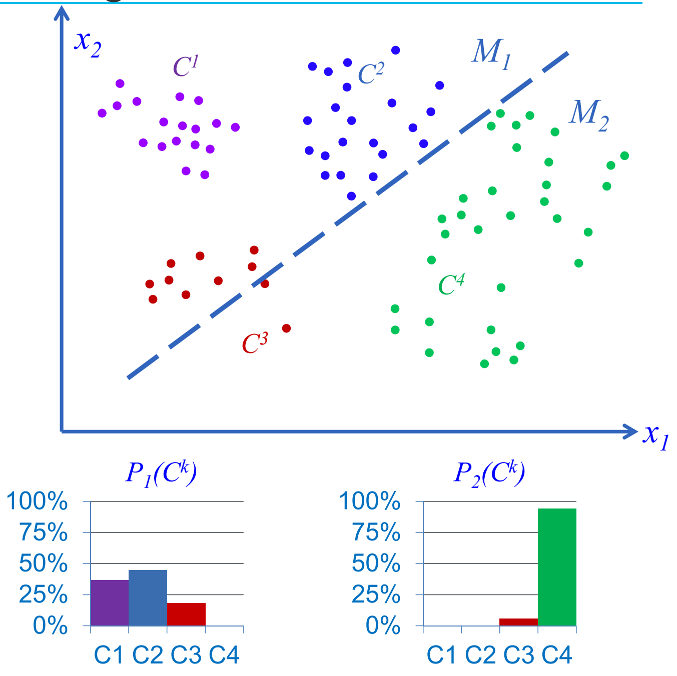


Universität Stuttgart

ifp

CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- From that ΔE = -1.006



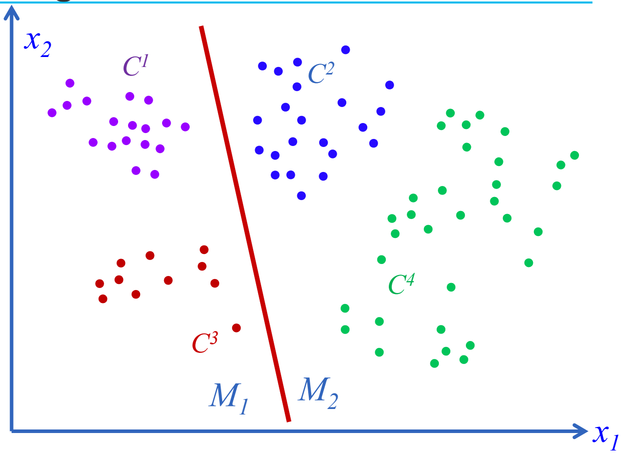
Universität Stuttgart

ifp

CART: Example for learning

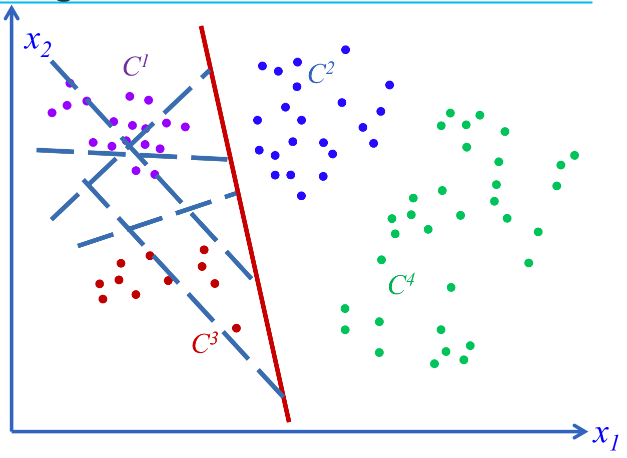
- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms
- Selection of the hyper-plane with maximum:
  ΔE = -0.966

- Repeat recursively for M_1 and M_2



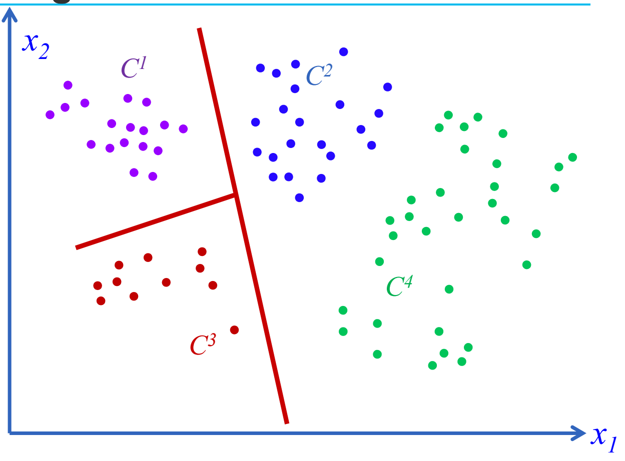
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms and ΔE



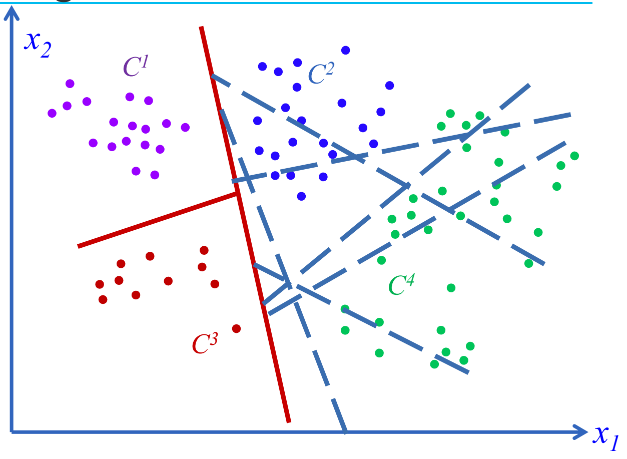
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms and ΔE
- Selection of the best separating hyperplane
- Recursion...



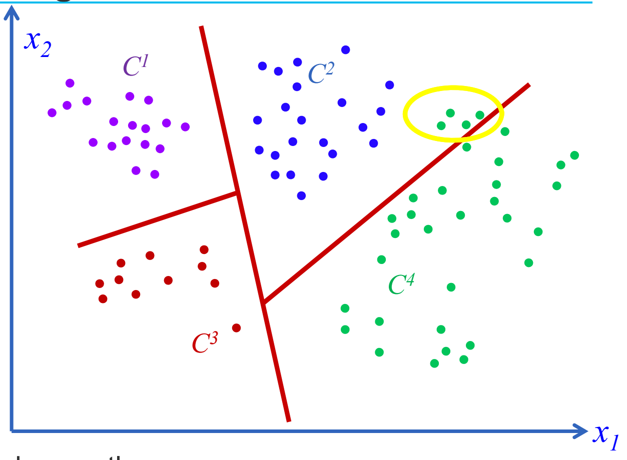
CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms and ΔE



CART: Example for learning

- Example with four classes , two features x_1, x_2 [Shotton et al., 2009]:
- Random selection of a separating hyperplane
- Determination of the histograms and ΔE
- Selection of the best separating hyperplane
- The classes are not yet separated correctly...



- When to stop the recursion?

CART: Stopping Criteria for Training

- For a unique assignment of a leaf to class: recursion is finished if only training samples of a single class are available in the leaf.
- This may lead to overfitting and very deep trees
  → Finish the recursion if
  - very few training samples fall into one node,
  - the information gain is very small, or
  - a specified maximum depth is reached.
- If one of the termination criteria is met, a node is declared to a leaf.
- As soon as each path through the tree ends in a leaf, the training of the test is finished.

CART: Assignment of leafs to classes

- Remember: The learning of the tests only requires a part (e.g. 1/3) of the training data.
- The remaining training samples $\mathbf{x}$ are presented to the tree and passed through the tree until they end up in a leaf.
- In every leaf b the normalized histogram of class labels $P_b(C^k)$ is determined on the basis of the training samples arriving at the leaf.
- Interpretation of histogram as **posterior**: $p_b\left(C^k\right) = p\left(C^k \mid \mathbf{x}\right)$
- The leaf is assigned to the class k for which: $p_b\left(C^k\right) = p\left(C^k \mid \mathbf{x}\right) \rightarrow \max$
- The posterior can be stored in the leaf if a probabilistic output is required.

CART: Pruning

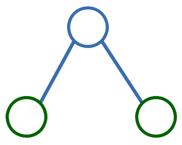
- Problems of the CART-algorithm:
  - Overfitting
  - Generation of trees that are too deep.
  - Generation of trees that have too many leaves.

→**Pruning:** Check whether the training error or a different criterion will change significantly if a node that is not a leaf is declared a leaf;
if not → branches emanating from that node are deleted.

- Variants of decision trees
  - ID3: Multipath splits, termination if all leaves are "pure“
  - C4.5 [Quinlan, 1993]: Based on ID3, includes pruning

CART: Discussion

- How many features or tests should one try?
  - Only one → „Extremely randomized tree“
  - Few → Fast training, may lead to underfitting.
  - Many → slower training, may lead to overfitting.
- „Decision Stump“: The simplest conceivable tree consisting of the root and two leafs only
  - Used in combination with other methods (e.g., Boosting).



CART: Discussion

- CART are still quite popular
- Requires good choice of the parameters for learning:
  - Type of the tests to be carried out in each node
  - Number of features per test
  - Number r of attempts to find the optimal boundary in a node
  - Minimum number of training points per node
  - Maximum depth
- Very fast both in training as well as in classification.
- CART have a tendency to overfit
- Small changes in the training data can lead to major changes in the decision boundaries.

Contents

Bootstrapping

- How to determine an accuracy measure for a classifier if only (relatively) few training data are available?
- **Generation of bootstrap data sets**:
  - Out of N samples $\mathbf{x}_n$ of the training data set, $N_I < N$ samples are selected randomly.
  - This is done with replacement, i.e., a point $\mathbf{x}_n$ may be included several times in the bootstrap data set.
  - A certain number B of these bootstrap data sets is generated. These data sets are considered to be independent.
  - The classifier is trained using each data set.
  - Uncertainty measures can be derived from the variation of the results.

Application of the bootstrap principle

- Using of bootstrap data sets for classification - Bagging (Bootstrap AGGregatING):
  - Training:
    - Generate B bootstrap data sets.
    - Each data set b is used to train a classifier $f_b(\mathbf{x})$.
  - Classification:
    - For each classifier f_b: Determine result $C_b = f_b(\mathbf{x})$
    - Result: The class that gets the highest number of “votes“.
- Bagging can be applied to any classifier (… but for some it might not make much sense).
- However, the classifiers f_b usually are of the same type.

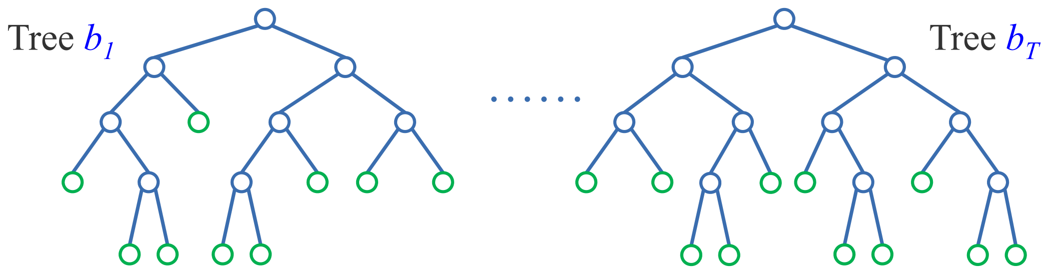
Application of the bootstrap principle

- Bagging can improve results of unstable classifiers.
- CART are "unstable" in the sense that small changes in the input data may lead to major changes in the class boundaries .
- Bootstrapping for CART → Random Forests

Contents

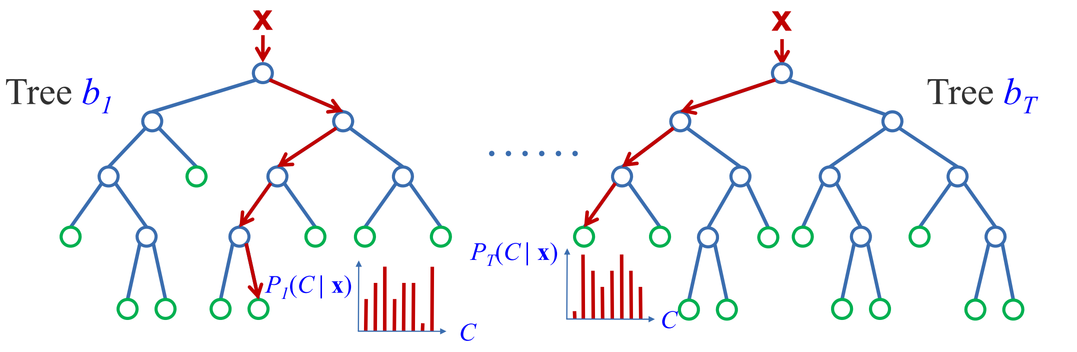
Random Forests

- A Random Forest [Breiman, 2001] consists of T decision trees (CART).
- Classification: A feature vector $\mathbf{x}$ is classified by each tree.



Random Forests

- A Random Forest [Breiman, 2001] consists of T decision trees (CART).
- Classification: A feature vector $\mathbf{x}$ is classified by each tree.



In every tree t: posterior distribution $P_t(C\,|\,\mathbf{x})$ according to tree t

- Posterior: average probability:

$$P(C\,|\,\mathbf{x}) = \frac{1}{T}\cdot\sum_{t=1}^{T} P_t(C\,|\,\mathbf{x})$$

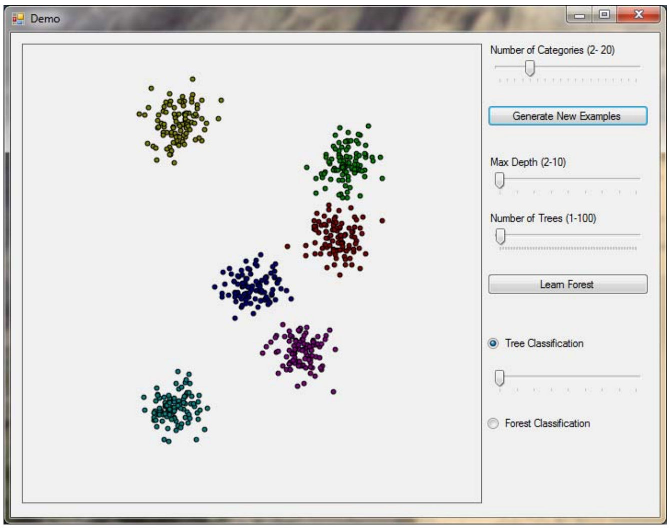
Training for random forest

- Generation of T bootstrap data sets (e.g., $T = 50$).
- Using each data set t one tree b_t is trained (cf. CART training).
- Important: Independent and random selection of the bootstrap data sets.

- Due to the combination of several trees:
  - Better generalization
  - Better stability

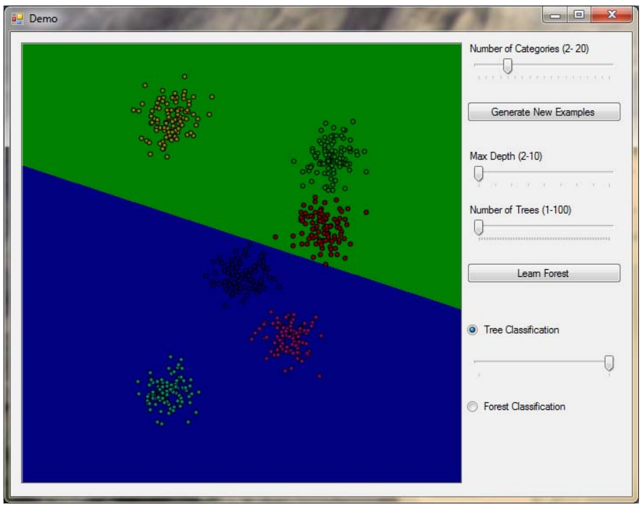
- Can be parallelized easily due to the independence of the bootstrap data sets and the trees.

Example [Shotton et al., 2009]



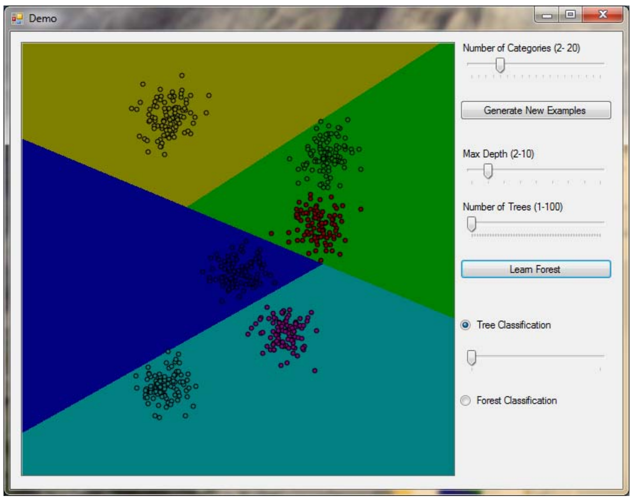
6 classes in a 2D feature space
Functions for separating the classes: straight lines in feature space

Example [Shotton et al., 2009]



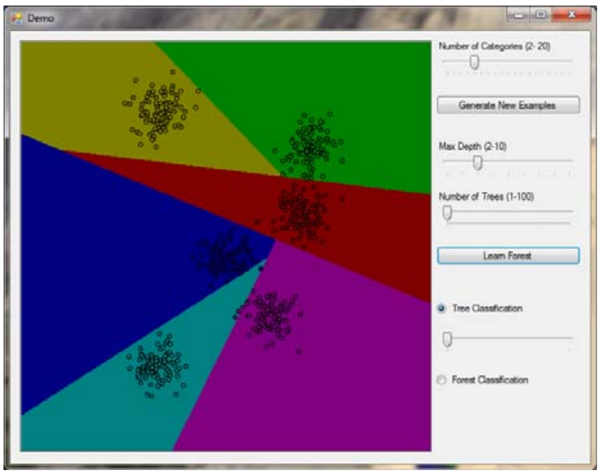
CART of depth 2 (inclusive root): Not all classes can be separated.

Example [Shotton et al., 2009]



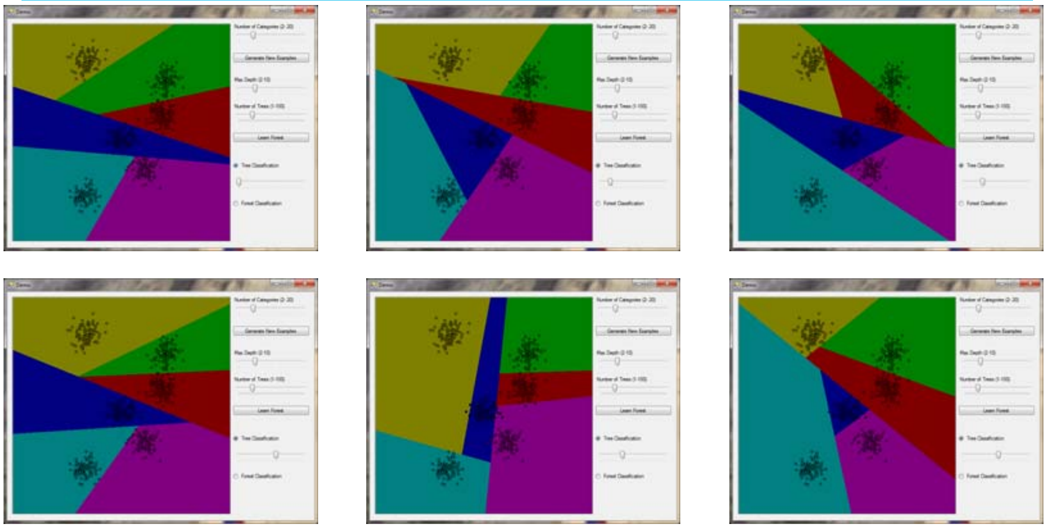
With a depth 3 tree, you are doing better, but still cannot separate all six classes.

Example [Shotton et al., 2009]



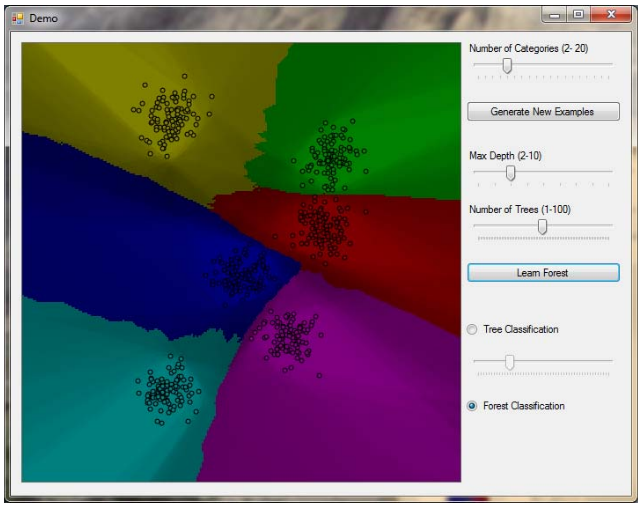
With a depth 4 tree, you now have at least as many leaf nodes as classes, and so are able to classify most examples correctly.

Example [Shotton et al., 2009]



Different trees in the Random Forest → Individual trees have different decision boundaries; none of them is very good at a first glance.

Example [Shotton et al., 2009]



Combining many trees (here: 50) → very good decision boundaries, similar to SVM, but: algorithm is much faster!
(Shading: entropy; the darker, the more contradicting votes)

Random Forests: Discussion

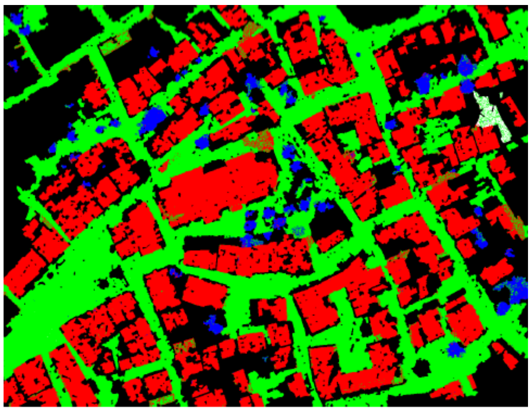
- Random forests are considered to be one of the best local classifiers today.
- Similar quality to the results as SVM, but much faster.
- Training and classification can be parallelized easily.
- Can be applied to multi-class problems by design.
- Probabilistic interpretation of the outputs → can be used in subsequent processing steps!
- Random forests may consume a lot of memory.
- There are implementations in Matlab, openCV, …

Random Forests: Application examples

- Classification of aerial images and DSM [Schindler, 2012]
- Matching of SIFT features by classification [Lepetit et al., 2006]
- Classification of laser scanner data [Niemeyer et al., 2013]
  - Features: Height above ground, local distribution of points, full-waveform features
  - Classification of each individual point with RF

Random Forests: Application examples

- Classification in urban areas [Chehata et al., 2009]:
  - Classification of points from airborne laser scanning data
  - Geometric und full-waveform features
  - Classes:
    Buildings, Vegetation,
    Artificial ground,
    Natural ground
  - Overall accuracy: 95%



Contents

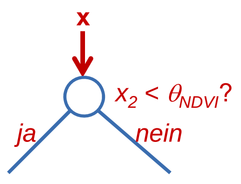
- Decision Trees
- Bootstrapping
- Random Forests
- Boosting
- Support Vector Machines
- Neural Networks

Boosting: Principle

- Boosting: Combination of "weak classifiers $f_b(\mathbf{x})$ to a strong combined classifier.
- Prerequisite: The weak classifiers must provide results that are (at least slightly) better than chance
  - This means, an **error rate** < 50% is sufficient in the two-class case.
- Consequently, very simple classifiers can be combined → Speed!
- There are different variants of boosting
- Here: AdaBoost (Adaptible Boosting)

Weak classifiers

- Two-class problem: class $C \in \{-1, +1\}$
- “Weak Classifier“:
  - A function $h(\mathbf{x})$ that predicts the class C for the feature vector $\mathbf{x}$, i.e., $h(\mathbf{x}) = \pm 1$.
  - **Success rate** must be > 50% (“better than chance“)
  - The better the success rate, the faster the boosting.
  - However, with good classifiers, boosting does not make sense.
- Examples for weak classifiers :
  - „Decision Stumps“
  - Other types of trees



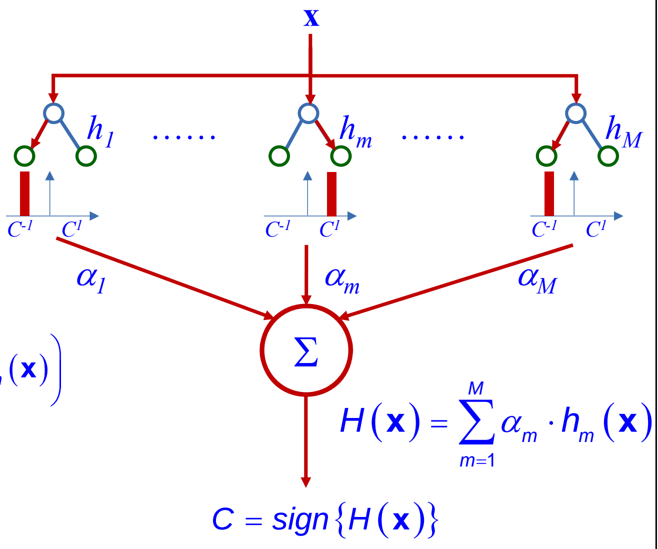
Decision Stump

Combination of weak classifiers I

- AdaBoost links M weak binary classifiers $h_m(\mathbf{x})$
- Each classifier returns a decision for the class C.
- Calculate the weighted mean $H(\mathbf{x})$, whose sign determines class C:

$$C = sign\{H(\mathbf{x})\} = sign\left(\sum_{m=1}^{M} \alpha_m \cdot h_m(\mathbf{x})\right)$$

- $H(\mathbf{x})$ is a strong classifier.
- Each classifier $h_m(\mathbf{x})$ has a weight α_m that is also determined in training.

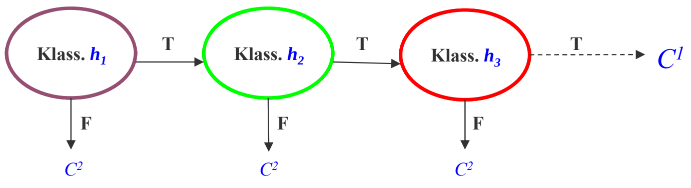


$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \cdot h_m(\mathbf{x})$$

$$C = sign\{H(\mathbf{x})\}$$

Combination of weak classifiers II

- Difference to **bagging:** Classifier $h_m(\mathbf{x})$ depends on the classifiers $h_1(\mathbf{x}), \dots, h_{m-1}(\mathbf{x})$ learned previously.
- This is achieved by considering a weight w_n^m for each training sample $\mathbf{x}_n$ when learning $h_m(\mathbf{x})$:
  - w_n^m large: $\mathbf{x}_n$ was classified incorrectly by $h_{m-1}(\mathbf{x})$
  - w_n^m small: $\mathbf{x}_n$ was classified correctly by $h_{m-1}(\mathbf{x})$
- Thus, AdaBoost focuses on "difficult" (incorrectly classified) training samples: if a sample is mis-classified by several weak classifiers in a row, its weight will become larger and larger!
- The training error can be arbitrarily small by increasing M (adding weak classifiers).



AdaBoost: Training

- Given:
  - N feature vectors $\mathbf{x}_n$ with known class labels C_n
  - Number M of the classifiers to be learned
  - Type of classifiers to be learned
- Wanted:
  - M weak classifiers $h_m(\mathbf{x})$
  - Weights α_m for the combination of these classifiers

AdaBoost: Training

- Initialize the weights: $w_n^1 = 1 / N$
- For $m = 1, ...M$:
  1. Train the classifier $h_m(\mathbf{x})$ by minimizing the weighted training error J_m:

$$J_m = \sum_{n=1}^{N} w_n^m \cdot \delta\left(h_m\left(\mathbf{x}_n\right) \neq C_n\right) \text{ with } \delta(a) = \begin{cases} 1 & \dots \quad a = \mathit{true} \\ 0 & \dots \quad a = \mathit{false} \end{cases}$$

  2. Calculate $\varepsilon_m = \dfrac{J_m}{\sum_{n=1}^{N} w_n^m}$ and accordingly update weight of h_m: $\alpha_m = \ln\left(\dfrac{1-\varepsilon_m}{\varepsilon_m}\right)$
  3. New weights: $w_n^{m+1} = w_n^m \cdot \exp\left\{\alpha_m \cdot \delta\left(h_m\left(\mathbf{x}_n\right) \neq C_n\right)\right\}$
  4. Normalize the weights w_n^{m+1} such that their sum is 1

AdaBoost Training: Discussion I

- Iteration 1: “normal“ learning of $h_1(\mathbf{x})$ (identical weights).
- Starting from iteration 2 the training samples have different weights and therefore different influence on the result.
- Each weak classifier is trained so that it is more likely to correctly classify a training sample having a high weight than a sample having a low weight.
- The quantity ε_m is the normalized weighted training error J_m of the classifier $h_m(\mathbf{x})$, $\varepsilon_m \in [0, 1]$.
- The smaller ε_m, the bigger the weight of α_m of $h_m(\mathbf{x})$

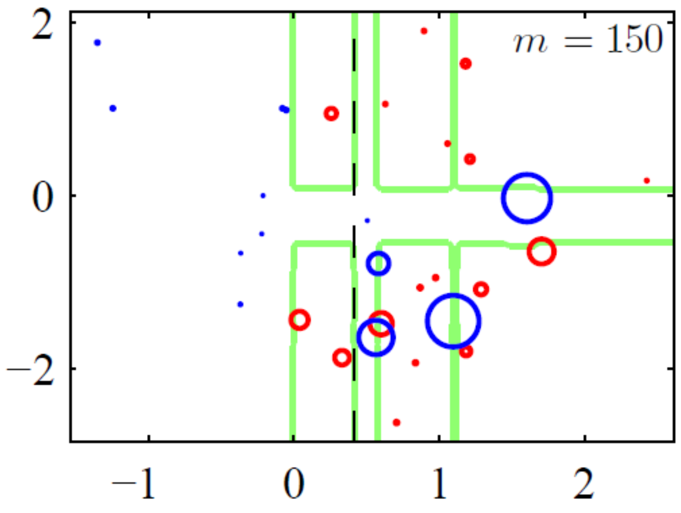
  → Classifiers with small training errors have a larger influence on the result

AdaBoost Training: Discussion II

- Update of the weights: $w_n^{m+1} = w_n^m \cdot \exp\left\{\alpha_m \cdot \delta\left(h_m\left(\mathbf{x}_n\right) \neq C_n\right)\right\}$
- Results of the classifier $h_m(\mathbf{x})$ for the training sample $\mathbf{x}_n$:
  - Correct → Weight of the sample is not changed: $w_n^{m+1} = w_n^m$
  - Not correct → Weight of the sample increases by a factor $\exp(a_m)$: $w_n^{m+1} = w_n^m \cdot e^{\alpha_m}$
- The training procedure can be derived mathematically by the sequential minimization of the exponential error function (cf. [Bishop, 2006]):

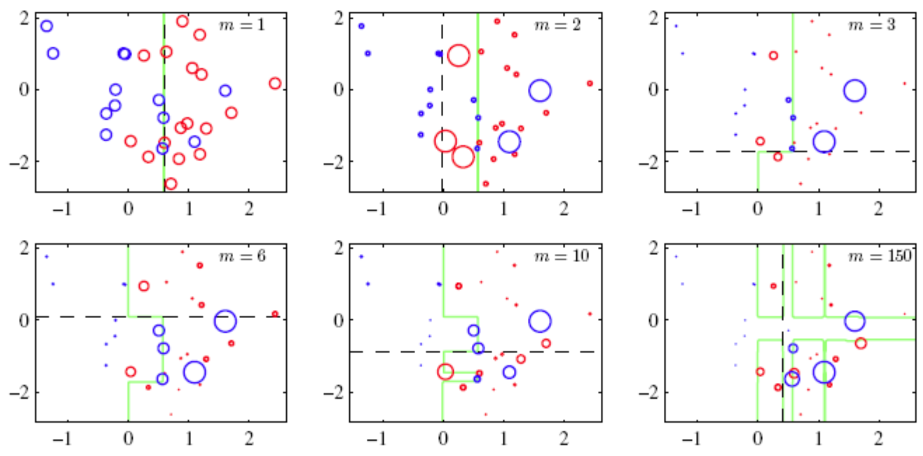
$$E = \sum_{n=1}^{N} \exp\left\{\frac{1}{2} \cdot C_n \cdot f_m\left(\mathbf{x}_n\right)\right\} = \sum_{n=1}^{N} \exp\left\{\frac{1}{2} \cdot C_n \cdot \sum_{k=1}^{m} \alpha_l \cdot h_l\left(\mathbf{x}_n\right)\right\}$$

AdaBoost Training: Beispiel [Bishop, 2006]



- Green line: The current overall classifier
- Dashed line: current weak classifier $h_m(\mathbf{x})$
- Radius of a circle is proportional to the weight w_n^m
  - Those training samples carry high weight, which have been misclassified before.

AdaBoost Training: Beispiel [Bishop, 2006]



- Green line: The current overall classifier
- Dashed line: current weak classifier $h_m(\mathbf{x})$
- Radius of a circle is proportional to the weight w_n^m
  - Those training samples carry high weight, which have been misclassified before.

AdaBoost: Multi-Class Case

- AdaBoost can be directly applied to the multi-class case
  - Definition of the weak classifiers $h_m(\mathbf{x})$ must be modified so that their output directly becomes the class label C → AdaBoost.M1

- Problem:
  - The individual classifiers must still be better than 50% (otherwise α_m becomes negative),
  - This is actually much better rather than "slightly better than chance" (error rate better than $1 / N_c$ with N_c classes).

- Possible solution: Split problem into several two-class problems

  - „One against all"
  - „One against one"

  $\}$ see SVM

AdaBoost: Multi-Class Case

- Solution 2: SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function)

  - Similar to AdaBoost.M1

  - Difference: Consider the number of classes N_c for computing the weights in iteration m:

$$\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right) + \ln[N_c - 1]$$

  - In case $\varepsilon_m < 0.5$, this modification ensures that $\alpha_m < 0.$

  - For number of classes $N_c = 2$ is this identical to AdaBoost.

Probabilities

- AdaBoost does not deliver probabilities.
- For the two-class case one can show that

$$C = sign\{H(\mathbf{x})\} = sign\left(\sum_{i=1}^{M} \alpha_m \cdot h_m(\mathbf{x})\right)$$

leads to the following interpretation of $H(\mathbf{x})$:

$$H(\mathbf{x}) = \frac{1}{2} \cdot \ln\left\{\frac{P(C=1|\mathbf{x})}{P(C=-1|\mathbf{x})}\right\}$$

- Therefore, using the logistic Sigmoid function σ

$$P(C=1|\mathbf{x}) = \frac{1}{1+e^{-2 \cdot H(\mathbf{x})}} = \sigma(2 \cdot H(\mathbf{x}))$$

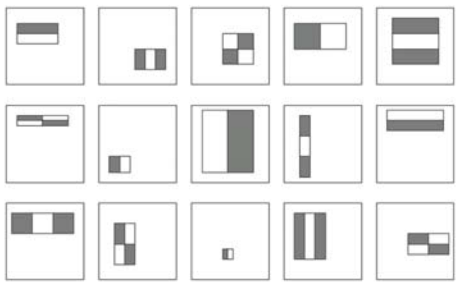
AdaBoost: Discussion

- AdaBoost can improve the quality of the results significantly above those of the weak classifiers.
- AdaBoost is regarded as a very good and fast classifier that is easy to apply.
- AdaBoost actually describes a family of classifiers (“meta classifier“).
- There are theoretical connections, e.g., to SVM.
- AdaBoost is a bit slower than random forests.
- AdaBoost can have problems with noisy training data.
- Derivation of probabilities is not as clear as with RF.

Contents

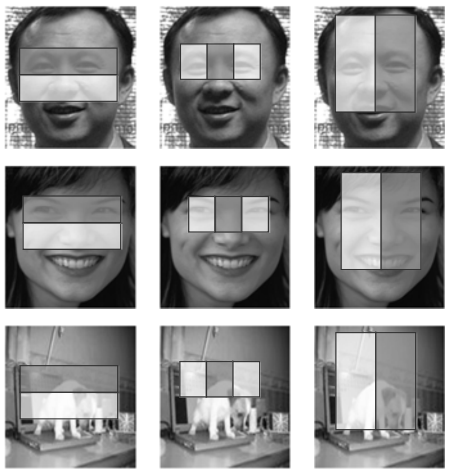
Application: Face detection – Features

- Classical application: Face detection in real time [Viola & Jones, 2004]
- A set of rectangular so-called Haar-Wavelets as features is slid over the image.



Examples of Haar-Wavelets

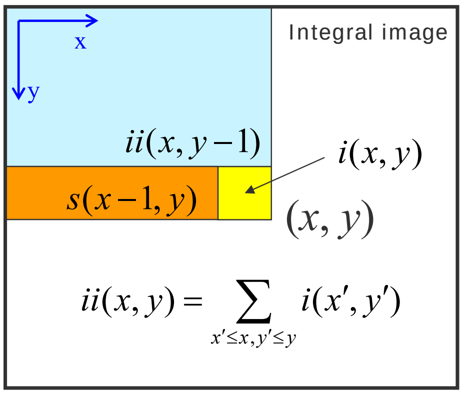
Calculation:
- Sum of gray values in white area – Sum of gray values in dark area



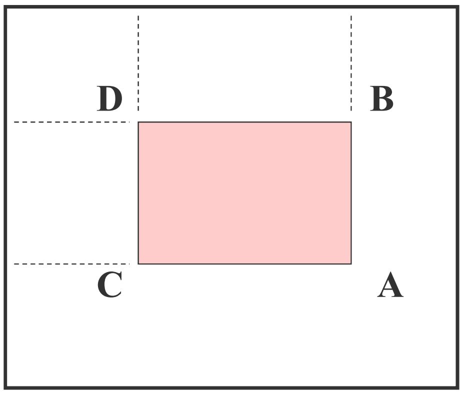
After [Viola & Jones, 2004]

Application: Face detection – Integral image

- These features can be calculated very efficient using a so-called integral image, which is derived in pre-processing.
- For every pixel $i(x,y)$ we sum-up the gray values $ii(x,y)$ of all pixels $i(x',y')$ in the image to the left and above.
- At runtime the calculation of the sum of gray values inside a rectangle of arbitrary size requires only **4 storage access steps and 3 additions!**



$$ii(x,y) = \sum_{x'\le x, y'\le y} i(x',y')$$



$$s(x,y) = s(x-1,y) + i(x,y)$$
$$ii(x,y) = ii(x,y-1) + s(x,y)$$

$$Sum = ii_A - ii_B - ii_C + ii_D$$

Application: Face detection – Concept I

- The approach is tailored for real-time applications
  - For example, camera, smartphone: Search for faces to optimize focus there.
- Further applications
  - Go through some image data base to find faces.
- Competing objectives:
  - High detection rate (DR):
    - Find as many faces as possible, for instance, 90%.
  - Low false alarm rate (FAR):
    - Return as few as possible image patches without, for instance, 1:1.000.000…
  - And on top: This shall work in real-time!
  - How can this be achieved??

Application: Face detection – Concept II

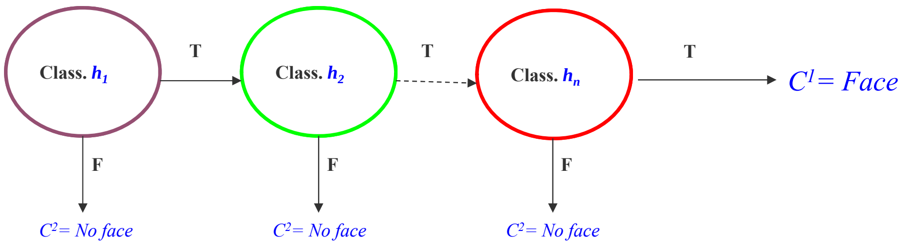
- Faces are comparable seldom in typical images.

  - Therefore, it is reasonable to exclude in early stages as many regions as possible of which the algorithm is „quite sure that they contain no face“ in order to focus on the rest:

    ➔ The **best** simple features h_1 should be applied first

    ➔ Then the next best h_2 an so on. …

    ➔ Due to this significant reduction of data by filtering we can later afford to apply more complex features h_n to analyze the rest.



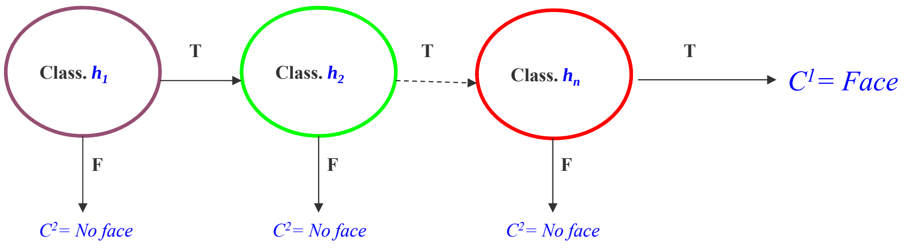
Application: Face detection – Concept III

- This means we apply a cascade of classifiers.

- We yield for the ensemble classifier detection rate DR as a function of the individual detection rates DR_i :

$$DR = \prod_i DR_i$$

- This means, in case all $DR_i = 0.99$, we end up after 10 stages of cascade with an ensemble detection rate $DR \approx 0.90$!

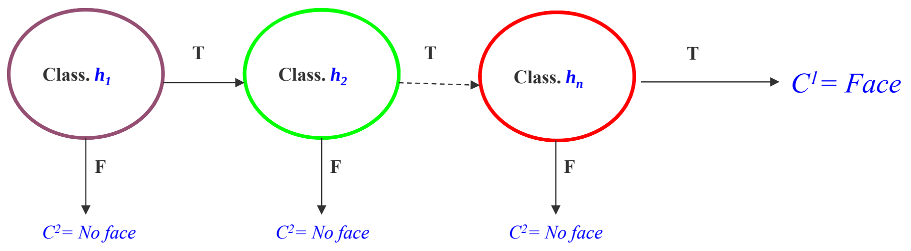


Application: Face detection – Concept IV

- What about false alarm rate *FAR*?
- We deal with the same multiplicative relationship:

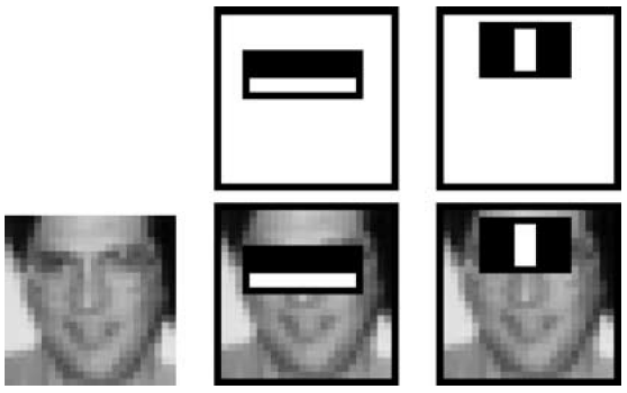
$$FAR = \prod_i FAR_i$$

- This means, in case all $FAR_i = 0.3$, we end up after 10 stages of cascade with an ensemble $FAR \approx 10^{-6}$!



Application: Face detection – Best features

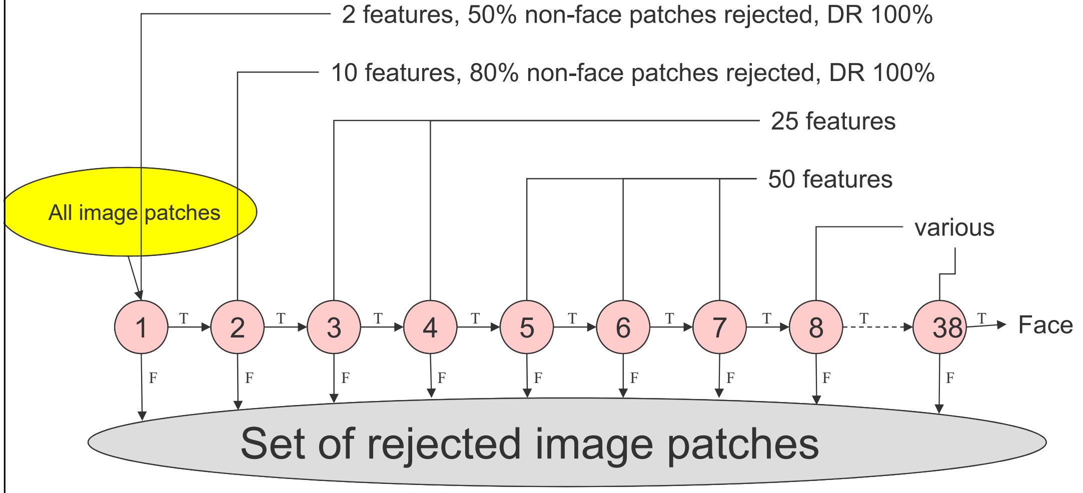
- The two strongest features are:



- Almost 100% detection rate *DR* and 50% false alarm rate *FAR*
  - The high detection rate is more important here, false alarms can be sorted out at later stages.

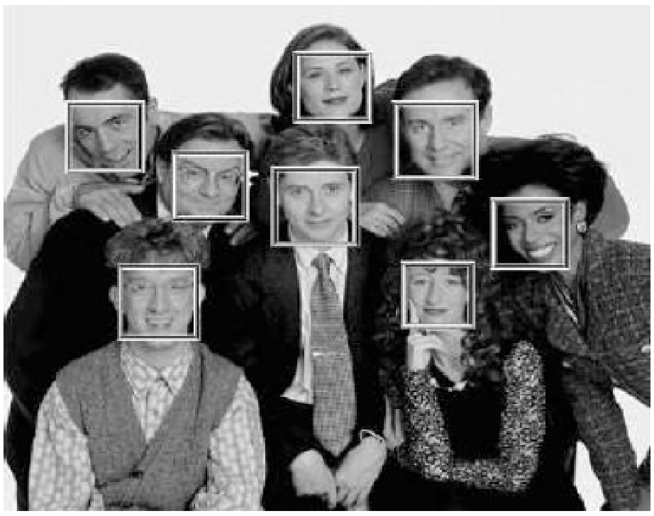
Application: Face detection – The cascade

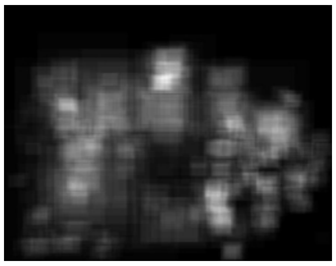
- 38 subsequent stages.
- Overall 6060 features
- The later the the stage, the more complex the approach.


2 features, 50% non-face patches rejected, DR 100%
10 features, 80% non-face patches rejected, DR 100%
25 features
50 features
various
All image patches
1
2
3
4
5
6
7
8
38
T
F
Face
Set of rejected image patches

Application: Face detection – Example

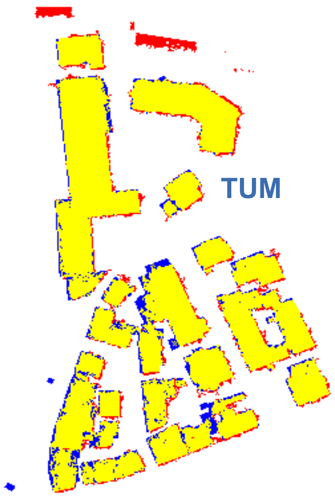
- Finally local non-maximum-suppression applied


Result after non-maximum-suppression


Result as gray value
(the brighter, the higher
confidence in presence of face)

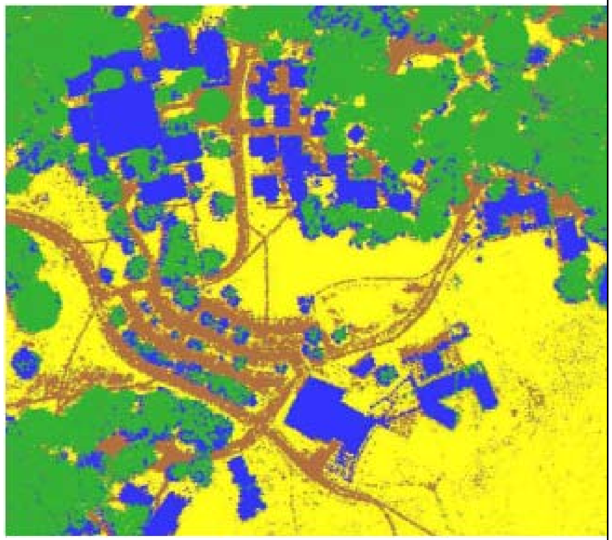
AdaBoost: Application building detection

- Application in remote sensing: e.g. contribution of TU Munich for ISPRS-Test on urban object detection
  - Multispectral image, DSM (8 cm GSD)
  - Local means / variances of bands
  - Among the best classifiers
  - Evaluation
    - yellow: correct
    - blue: missing building
    - red: false positive



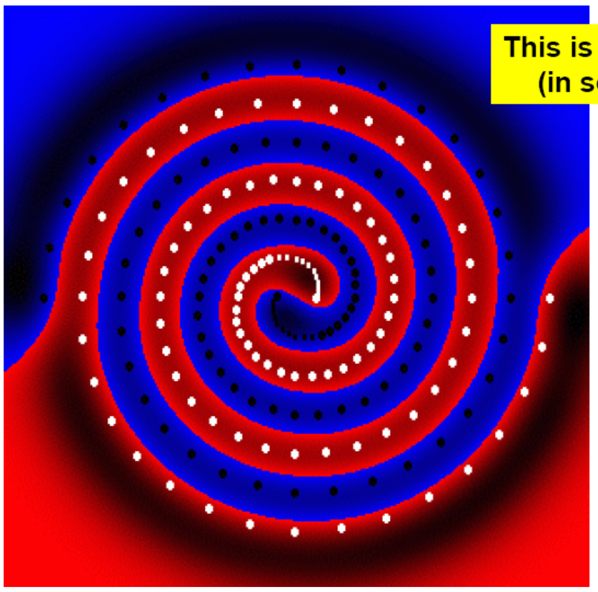
AdaBoost: Application land cover classification

- Classification of urban areas [Lodha et al., 2007]:
  - DSM from laser scanner data + aerial image (intensity only)
  - Classes: Street, Grass, Building, Tree
  - Overall accuracy: 92%



Contents

Support Vector Machine (SVM)



This is a hyperplane!
(in some space)

Support Vector Machine (SVM)

- Basic idea
- Linearly separable data
- Non-linear case
  - Separate in higher dimensional feature space
- Accept (a few) misclassification errors
- Choice of hyper-parameters / multi-class problem

SVM separates two classes

- Binary classification: $C \in \{-1, +1\}$
- Search for hyperplane ε in feature space that separates the classes in the training data.
  - We use the notation common in SVM literature with factor b (Bias) instead of w_0:

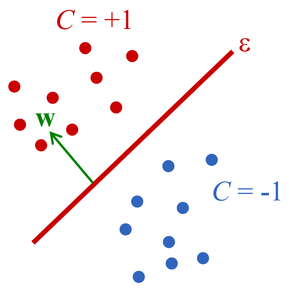
$$\varepsilon : \mathbf{w}^T \cdot \mathbf{x}_n + b = 0$$

Given: set of samples

$$(\mathbf{x}_1, C_1), \dots, (\mathbf{x}_m, C_m) \quad \text{with}$$
$$\mathbf{x}_i \in R^n \quad \text{and} \quad C_i \in \{-1, +1\}$$

Training: determine $\mathbf{w} \in R^n, b \in R$

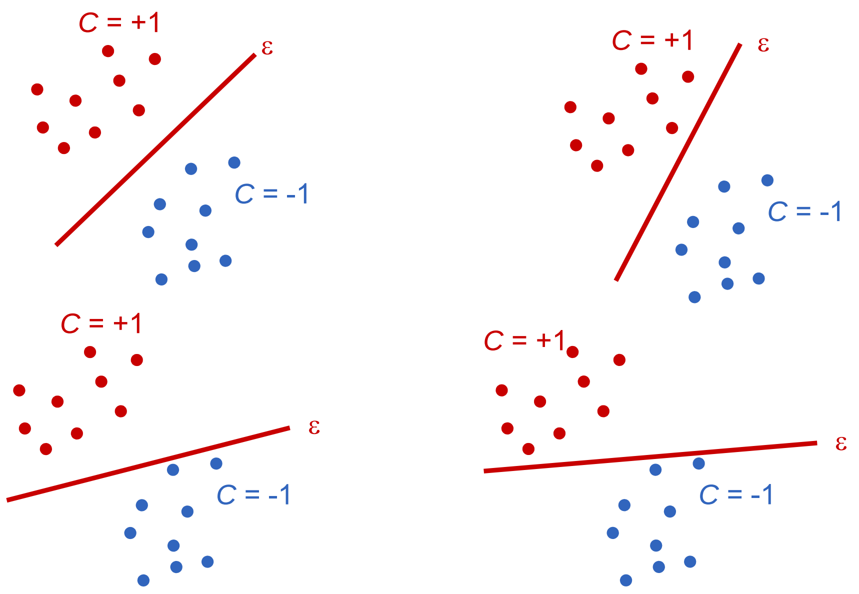
Classification: $C_n = sign\left(\mathbf{w}^T \cdot \mathbf{x}_n + b\right)$



But: Which is the best hyperplane for separation of the given training data?

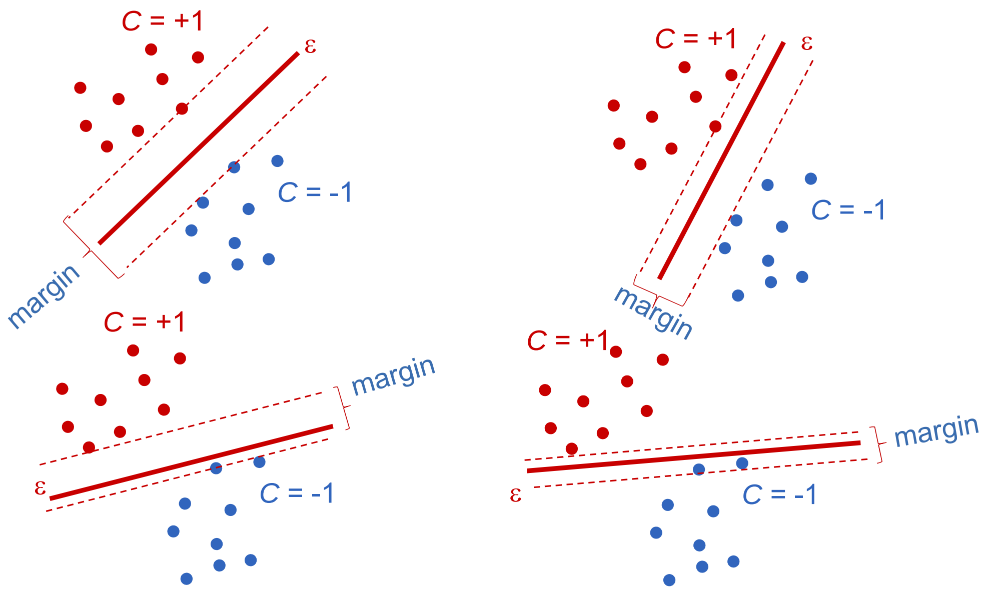
Support Vector Machines: Principle

There are many possible hyperplanes…



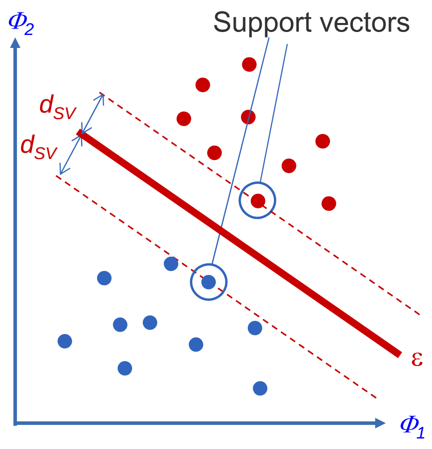
Support Vector Machines: Principle

Margin: Region near the hyperplane without training data



Support Vector Machines: Principle

- maximum margin principle: Determine ε so that the distance d_{SV} of ε to the nearest training sample is maximized [Vapnik, 1998].
- The points with distance d_{SV} of ε are called **Support Vectors** (SV)
- Result depends on SVs only



- But: before training one does not know which samples are SVs
- Feature Space Mapping $\mathbf{\Phi(x)}$ if the classes are not linearly separable.

Support Vector Machine (SVM)

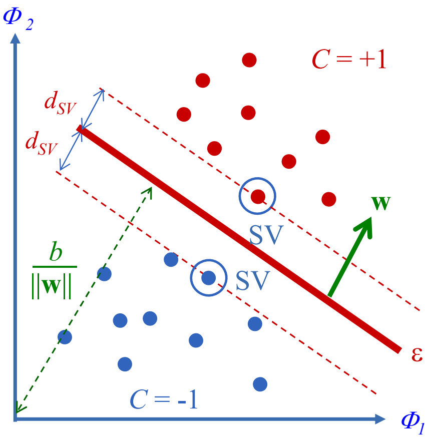
- Basic idea
- Linearly separable data
- Non-linear case
  - Separate in higher dimensional feature space
- Accept (a few) misclassification errors
- Choice of hyper-parameters / multi-class problem

Support Vector Machines: Hyperplane

- Binary classification: class $C \in \{-1, +1\}$
- Hyperplane in the transformed feature space: $\varepsilon : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b = 0$
- Distance d_n of a point $\boldsymbol{\Phi}(\mathbf{x}_n)$ from ε:

$$d = \frac{1}{\|\mathbf{w}\|}\left(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b\right)$$

- Distance of the plane from the origin: $b/||\mathbf{w}||$
- The length of $\mathbf{w}$ is undefined → How to scale $\mathbf{w}$?



Support Vector Machines: Margin

- Scaling of $\mathbf{w}$ so that

  $\mathbf{w}^T \cdot \boldsymbol{\Phi}(\mathbf{x}_n) + b = \pm 1$ for SVs

- Margin is limited by two planes ε_1, ε_2 which are parallel to ε (same normal $\mathbf{w}$)

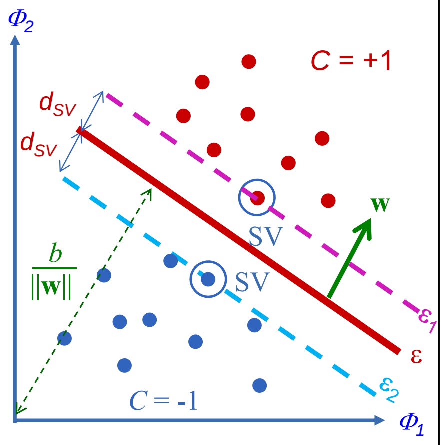
  $\varepsilon_1 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b = +1$

  $\varepsilon_2 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b = -1$

  or

  $\varepsilon_1 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b - 1 = 0$

  $\varepsilon_2 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b + 1 = 0$

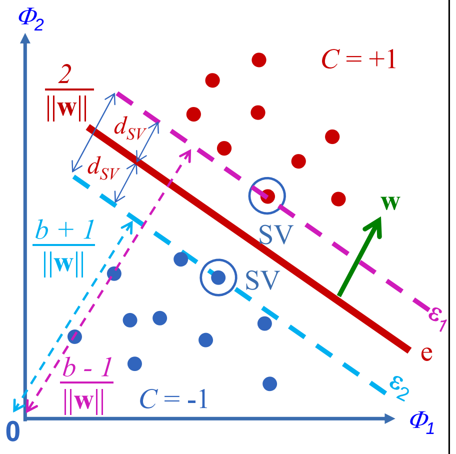


Support Vector Machines: Margin Width

- Distances of the planes ε_1, ε_2 from the origin $\mathbf{0}$:
  - $\varepsilon_1 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b - 1 = 0$

    $$d(\varepsilon_1, \mathbf{0}) = \frac{b-1}{\|\mathbf{w}\|}$$
  - $\varepsilon_2 : \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b + 1 = 0$

    $$d(\varepsilon_2, \mathbf{0}) = \frac{b+1}{\|\mathbf{w}\|}$$



- Distance between the two planes: width of the margin $2 \cdot d_{SV}$

$$2 \cdot d_{SV} = d(\varepsilon_2, \mathbf{0}) - d(\varepsilon_1, \mathbf{0}) = \frac{b+1}{\|\mathbf{w}\|} - \frac{b-1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

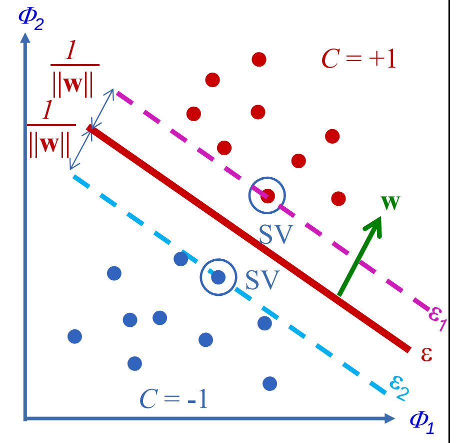
Support Vector Machines: Maximum Margin Criterion

- Result: If we scale $\mathbf{w}$ as defined on the previous slides, maximising the margin is equivalent to

$$\frac{1}{\|\mathbf{w}\|} \rightarrow \max$$

- Without considering the training data, this would result in $\mathbf{w} = \mathbf{0}$

- To obtain a meaningful solution, we have to introduce constraints for the training data!

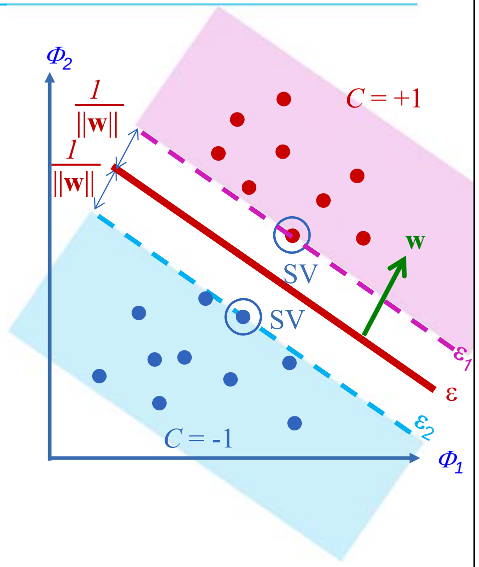


Support Vector Machines: Constraints

- Constraints for feature vectors $\mathbf{x}_n$ with class $C_n = +1$:
  - SVs are on plane ε_1

    $\Rightarrow \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b = 1$
  - All other points have to be on the side of ε_1 indicated by the direction of $\mathbf{w}$ (because they have to be classified correctly!)

    $\Rightarrow \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b > 1$

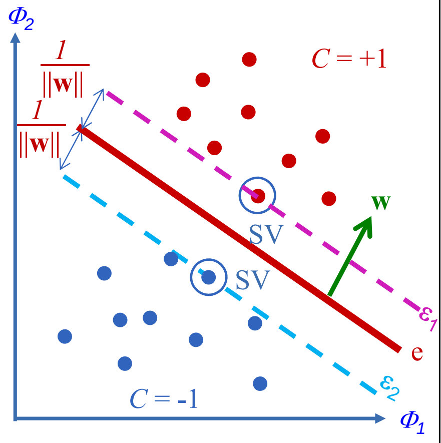


- Consequently: $\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \geq 1$ for $C_n = +1$
- Similarly: $\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \leq -1$ for $C_n = -1$

Support Vector Machines: Constraints

- Constraints:

$$\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \geq 1 \quad \text{for } C_n = +1$$
$$\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \leq -1 \quad \text{for } C_n = -1$$



- Multiplication of these inequalities by C_n yields a uniform representation for the constraints:

$$\boxed{C_n \cdot \left[\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b\right] \geq 1 \quad \forall \mathbf{x}_n}$$

- The identity applies to the support vectors.

Training: Maximize margin M

Aim 1: All training vectors $\mathbf{x}_n$ shall be classified correctly:

$$\left.\begin{array}{ll}\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \geq 1 & \text{for} \quad C_n = +1 \\ \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b \leq -1 & \text{for} \quad C_n = -1\end{array}\right\}$$

$$C_n \cdot \left[\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b\right] \geq 1 \quad \forall n$$

Aim 2: we want to maximize margin M :

$$M = \frac{2}{\|\mathbf{w}\|} \rightarrow \max \qquad \text{equivalent to:} \qquad \frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} \rightarrow \min$$

We yield a quadratic optimization problem with inequalities as constraints:

$$\frac{1}{2}\|\mathbf{w}\|^2 \rightarrow \min \qquad C_n \cdot \left[\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b\right] \geq 1 \quad \forall n$$

Training: Solve the optimization problem I

- New objective function with Lagrange multipliers $\alpha_n \geq 0$ (one for each training sample):

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} - \sum_n \alpha_n \left(C_n \left[\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b\right] - 1\right) \rightarrow \min$$

- Derivatives: $\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum \alpha_n \cdot C_n \cdot \Phi(\mathbf{x}_n)$ $\qquad$ $\frac{\partial L}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum \alpha_n \cdot C_n \cdot \Phi(\mathbf{x}_n)$

$$\frac{\partial L}{\partial b} = -\sum \alpha_n \cdot C_n \qquad \frac{\partial L}{\partial b} = 0 \rightarrow \sum \alpha_n \cdot C_n = 0$$

- Substituting this result in L leads to a new objective function:

$$W(\boldsymbol{\alpha}) = \sum_n \alpha_n - \frac{1}{2}\sum_n \sum_m \alpha_n \alpha_m C_n C_m \left(\Phi(\mathbf{x}_n^T) \cdot \Phi(\mathbf{x}_m)\right) \rightarrow \max$$

Subject to: $\sum_n C_n \alpha_n = 0 \quad \text{and} \quad \alpha_n \geq 0 \quad \forall \text{n}$

Training: Solve the optimization problem II

- Finally, we get this solution:

$$\mathbf{w} = \sum_{i=n}^{m} \alpha_n C_n \Phi\left(\mathbf{x}_n\right) \quad \text{and} \quad b = C_k - \mathbf{w}_k^T \cdot \Phi\left(\mathbf{x}_k\right) \quad \text{for} \quad k \in \{1,\ldots,m\}, \alpha_k \neq 0$$

- And this classification function $f(\mathbf{x})$ for an unknown vector $\mathbf{x}$:

Only vectors with an $\alpha \neq 0$ are relevant, those are exactly the **support vectors**!

$$f(\mathbf{x}) = sign\left(\sum_{n=1}^{m} \alpha_n C_n \Phi\left(\mathbf{x}_n\right)^T \cdot \Phi\left(\mathbf{x}\right) + b \right)$$

The feature vectors $\mathbf{x}$ or their transform $\Phi(\mathbf{x})$ occur just in a scalar product!

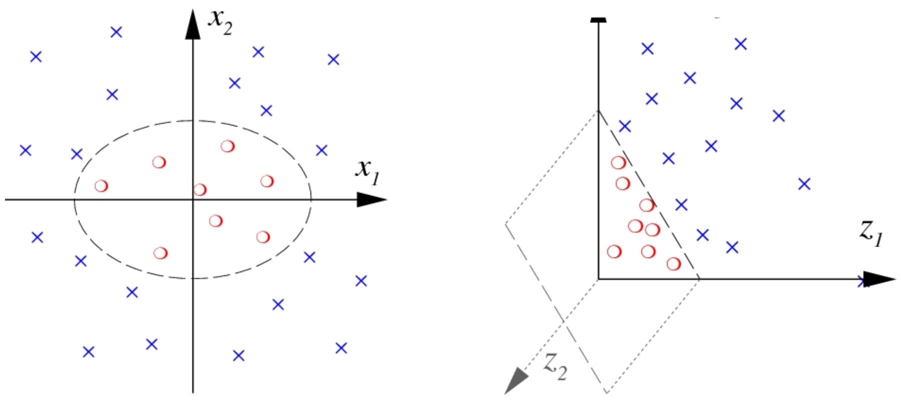
Support Vector Machine (SVM)

- Basic idea
- Linearly separable data
- Non-linear case
  - Separate in higher-dimensional feature space
- Accept (a few) misclassification errors
- Choice of hyper-parameters / multi-class problem

Feature space mapping

- Remember: The classes can more easily be separated in a higher-dimensional space (cf. logistic regression).

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\,x_1 x_2, x_2^2)$$



Feature space mapping

- As discussed, in $f(\mathbf{x})$ the $\Phi(\mathbf{x})$ occur in a scalar product

$$f(\mathbf{x}) = sign\left(\sum_{n=1}^{m} \alpha_n C_n \Phi(\mathbf{x}_n)^T \cdot \Phi(\mathbf{x}) + b\right)$$

- In order to benefit from better separability in higher-dimensional space H, in principle we are required to:

  1. Apply Φ for all SV $\mathbf{x}_n$ and vector $\mathbf{x}$ to be classified: $\mathbf{x}_n \to \Phi(\mathbf{x}_n)$

  2. Calculate scalar products in H: $\Phi^T(\mathbf{x}_n) \cdot \Phi(\mathbf{x})$

- Hard / impossible for high dimension of H

The „Kernel Trick“ I

- Theorem of Mercer
  - We have a (low9 vector space $\mathbf{X}$ (set of our data $\mathbf{x}_n$) and
  - $\mathbf{F}$ is some higher-dimensional feature (vector) space.

  - A positive semidefinite mapping $K : X \times X \rightarrow R$
  - is called Kernel

  - if in $\mathbf{F}$ the scalar product is defined and

  - a function exists $\Phi : X \rightarrow F$ with:

$$\boxed{K(\mathbf{x}_n, \mathbf{x}_m) = \Phi^T(\mathbf{x}_n) \cdot \Phi(\mathbf{x}_m) \quad \forall \mathbf{x}_n, \mathbf{x}_m}$$

The „Kernel Trick“ II

- The trick:
  - In case we know the kernel function $K(x_i, x_j)$, there is no need to perform a Feature Space Mapping $\Phi(\mathbf{x})$ into high-dimensional space.

  - This transition occurs only implicitly (inside kernel function), all calculations are carried out in original space.

- The classification function is $f(\mathbf{x}) = sign\left(\sum_{n=1}^{m} \alpha_n C_n K\left(\mathbf{x}_n^T, \mathbf{x}\right) + b\right)$

- Important kernel functions

  - Polynomial kernel: $K_{poly}(\mathbf{x}_1, \mathbf{x}_2) = \left(s\mathbf{x}_1^T \cdot \mathbf{x}_2 + r\right)^d$

  - “Radial Basis Function“, RBF: $K_{RBF}\left(\mathbf{x}_1, \mathbf{x}_2\right) = e^{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}}$

Example

$$\mathbf{x} = (x_1, x_2),\ \mathbf{x}' = (x_1', x_2')$$

$$\Phi : R^2 \to R^3 := (x_1, x_2) \to \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = \begin{pmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{pmatrix} \cdot \begin{pmatrix} x_1'^2 \\ \sqrt{2}x_1'x_2' \\ x_2'^2 \end{pmatrix} =$$

$$= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 =$$

$$= (x_1x_1' + x_2x_2')^2 = \left(\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)^2 = (\mathbf{x} \cdot \mathbf{x}')^2$$

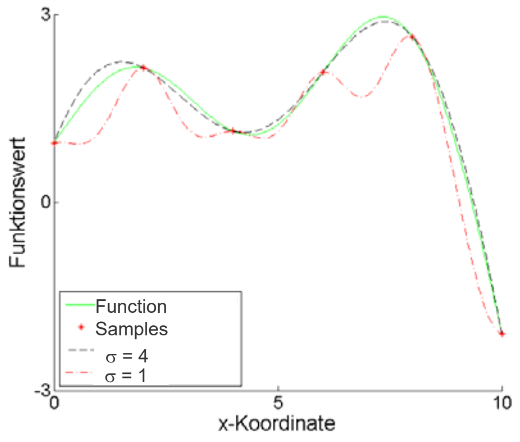
$$:= K_{Poly}(s\mathbf{x} \cdot \mathbf{x}' + r)^d \quad \text{mit} \quad r = 0, s = 1, d = 2$$

It is sufficient to calculate the scalar product and to square the result in R²!

Gaussian radial basis function

- Often used
- Parameter σ needs to be adjusted in training
- Example: $f(x)$ shall be approximated by sum of Gaussians of same σ
  (cf. kernel density estimator in chapter Segmentation and Bayes classifier)

$$f(x) = \sum_{i=1}^{5} \lambda_i e^{-\frac{\|x - c_i\|^2}{2\sigma^2}}$$



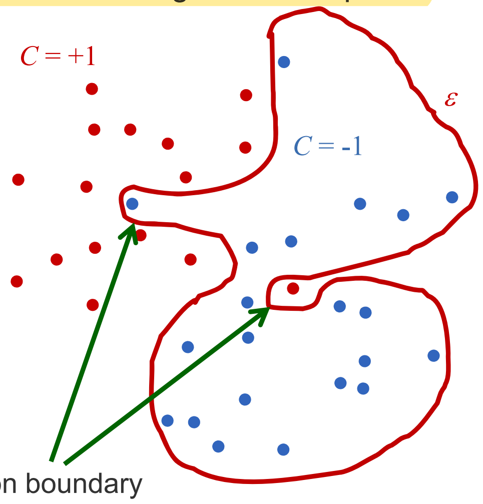
Support Vector Machine (SVM)

- Basic idea
- Linearly separable data
- Non-linear case
  - Separate in higher dimensional feature space
- Accept (a few) misclassification errors
- Choice of hyper-parameters / multi-class problem

Support Vector Machines: Overfitting

- SVM can potentially separate all possible configurations of points
- **Danger:** Overfitting
  -
  - **Complex models requiring too many parameters (→ many SVs)**

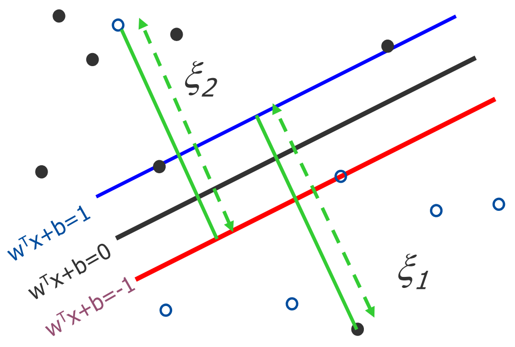
→ Expansion of the model!



Very unlikely shape of the decision boundary
→ Stronger generalization is desired

SVM with errors in training data (Soft Margin Classification)

- We allow (a few) errors in the training data to yield simpler hyperplane.
- Erroneous samples: use of slack variables $\xi_i \geq 0$.
- The related vectors become support vectors.
- P is a weight punishing use of too many erroneous samples (regularization!).



$$\frac{1}{2}\|\mathbf{w}\|^2 + P\left(\sum_{n=1}^{m}\xi_n\right) \to \min$$

$$C_n(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n) + b) \geq 1 - \xi_n \quad \forall n$$

Universität Stuttgart

ifp

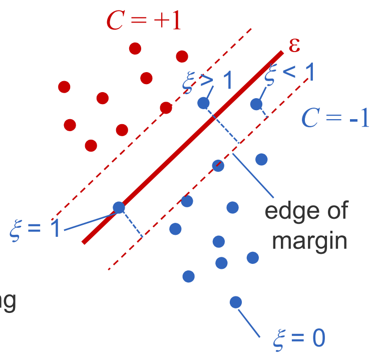
SVM with errors in training data

- Determine values of slack variables $\xi_n \geq 0$:

  - $\xi_n = 0$ Samples at the edge of the margin or in the region assigned to y_n.
  - $\xi_n = 1$ Samples on ε
  - $\xi_n < 1$ Sample in the margin but on the correct side of ε.
  - $\xi_n > 1$ Samples on the wrong side of ε, i.e. training samples having a wrong class label.

  - For points inside the margin or on the wrong side of ε, this definition implies: $\xi_n = |C_n - \mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}_n) + b|$.

- The objective function needs to be extended accordingly, apart from that the training and classification procedure is the same.



Universität Stuttgart

ifp

Support Vector Machine (SVM)

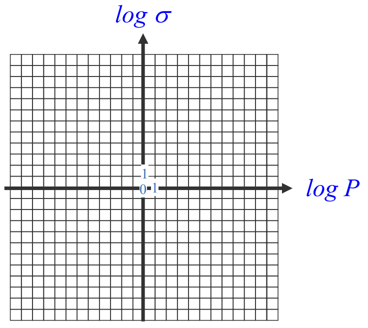
- Basic idea
- Linearly separable data
- Non-linear case
  - Separate in higher dimensional feature space
- Accept (a few) misclassification errors
- Choice of hyper-parameters / multi-class problem

Choice of parameters σ and C

- The possible range of parameter is quite large and suitable values for task at hand are required to be optimized in preprocessing.
- Possible way of doing so (LIBSVM):
  - Grid search combined with cross-validation:
    1. Systematical scan of parameter space
    2. For each pair of parameters under consideration train SVM on part of training data
    3. Finally, pick best pair
  - Coarse to fine search possible
  - Test can be conducted in parallel

$$\sigma = \frac{1}{2\sigma} = 2^{-15}, 2^{-13} \ldots 2^{3}$$

$$P = 2^{-5}, 2^{-3} \ldots 2^{15}$$



Choice of parameters σ and C

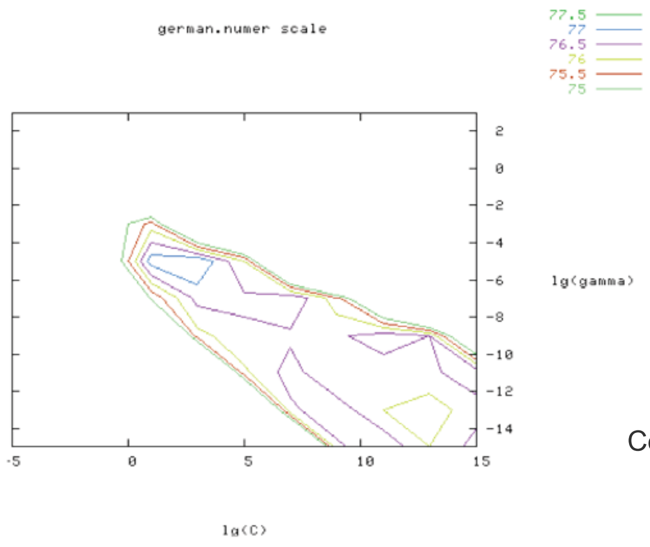
- Cross-validation

- Example: 10-times cross-validation :
  1. Split data into 10 bootstrap blocks.
  2. One block used for test, the other nine for training.
  3. Repeat 2 and 3, until each block was used once for testing.

- Cross-validation conducted for all pairs of σ and P.

➔ Choice of pair with best performance.

Choice of parameters σ and C: Example

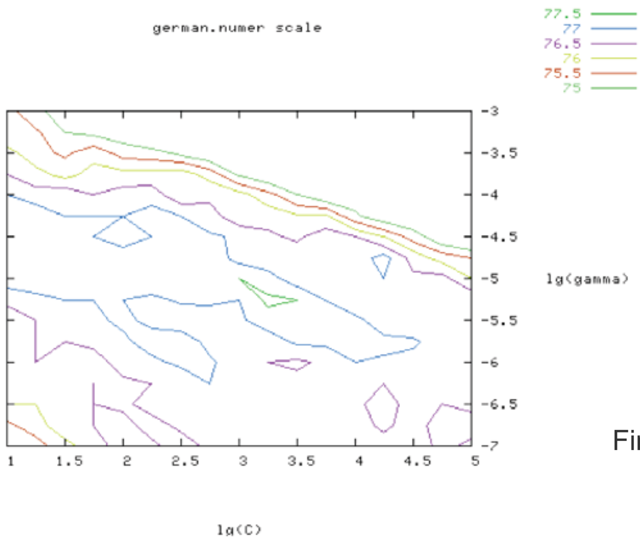


Coarse search

Figure 2: Loose grid search on $P = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ an $\sigma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$.

LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Choice of parameters σ and C: Example



Fine search

Figure 3: Fine grid-search on $P = 2^1, 2^{1.25}, \ldots, 2^5$ anc $\sigma = 2^{-7}, 2^{-6.75}, \ldots, 2^{-3}$.

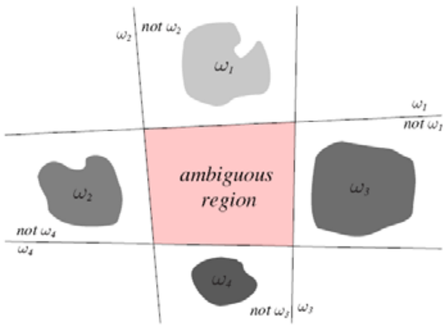
LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

SVM: Expansion to more than two classes

Two methods in use:

1. **"one against the rest"**
   n classes → *n* binary classifiers
   - Problem: areas of ambiguity

2. **"one against one"**
   → *0.5 n(n-1)* pairwise binary classifiers
   - Run all pairwise classifiers
   - Chose class with most "votes".
   - Still areas of ambiguity, but usually smaller overall area affected.



Probabilities

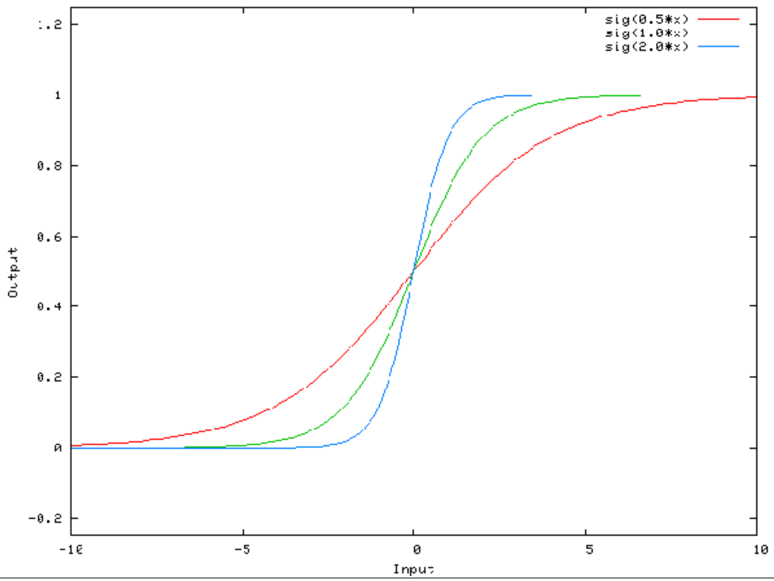
- SVM provides no probabilities.
- Classification for SVM:

$$f(\mathbf{x}) = sign\left(\sum_{n=1}^{m} \alpha_n C_n K(\mathbf{x}_n, \mathbf{x}) + b\right) = sign(f(\mathbf{x}))$$

- The function $f(\mathbf{x})$ depends on the distance d of $\Phi(\mathbf{x})$ from the decision boundary: $f(\mathbf{x}) = d \cdot ||\mathbf{w}||$
- Note that d is a signed distance: $C = sign(d)$
- Remember: For logistic regression, the posterior probability was determined as $\sigma(d \cdot ||\mathbf{w}||)$
  with σ: logistic Sigmoid function

Logistic regression: Geometrical interpretation

- Interpretation of $|| w ||$: The larger $|| w ||$, the steeper the sigmoid function.



Probabilities

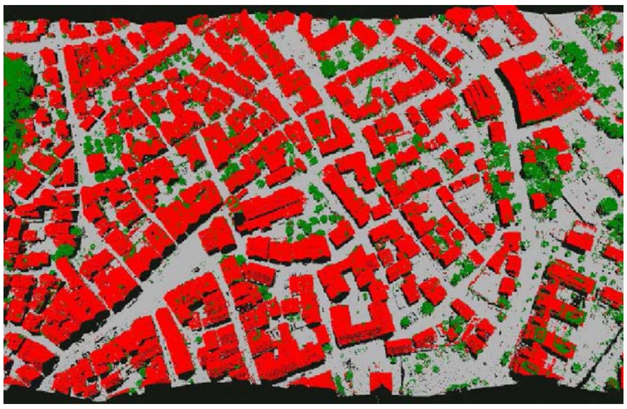
- $f(\mathbf{x})$ is a new feature and becomes an argument for the sigmoid function.
- But we have to perform a linear transformation (because $||\mathbf{w}||$ is not known).
- Thus: $P(C=1|\mathbf{x}) = \sigma(A \cdot f(\mathbf{x}) + B) = \dfrac{1}{1+e^{-(A \cdot f(\mathbf{x})+B)}}$
- The parameters A, B are learned from training data.
- For that purpose, one must use training samples that have **not** been used to train the SV
  → Again, the training data are divided into two groups
- Training: see logistic regression.

Support Vector Machines: Discussion

- SVM with Gauss-Kernel and slack variables provide good results in many applications
- SVM often serve as a baseline for comparison with other procedures.
- Parameters of the kernel function and P must be determined.
- Both of these parameters affect the smoothing of the decision boundary.
- Problems of SVM:
  - The transition to more than two classes not obvious
  - Derivation of a quality indicator for the result
  - SVM is slow compared to Random Forests, especially during training.

Support Vector Machines: Applications

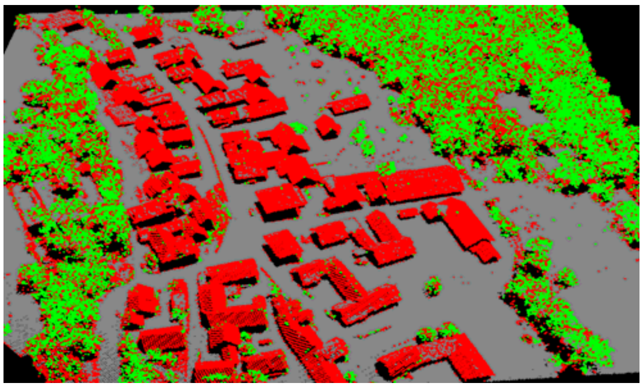
- Classification of laser scanning data in urban areas [Mallet et al., 2011]
  - Classification of 3D points
  - Classes: Buildings, Vegetation, Ground
  - Different features, incl. full waveform
  - Overall accuracy: 95%



Support Vector Machines: Applications

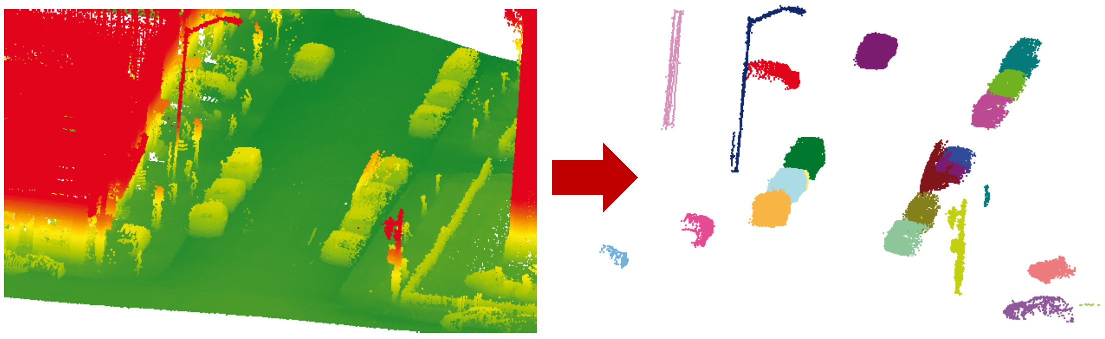
- Classification of laser scanning data in urban areas [Niemeyer et al., 2011]
  - Classification of 3D points
  - Classes: Buildings, Vegetation, Ground
  - Different features, incl. full waveform
  - Overall accuracy: 80.3%
  - Problem: very dense tree crowns

    Improvement by consideration of context possible!



Support Vector Machines: Applications

- Classification of street furniture [Golovinskiy et al., 2009]
  - Combination aerial / terrestrial LiDAR
  - Separation foreground / background (Graph cuts)
  - Classification of the segments in the foreground using SVM
  - Best results: Short pole and lantern post (70% - 90% completeness / correctness).
  - Limiting factor: Number of training samples



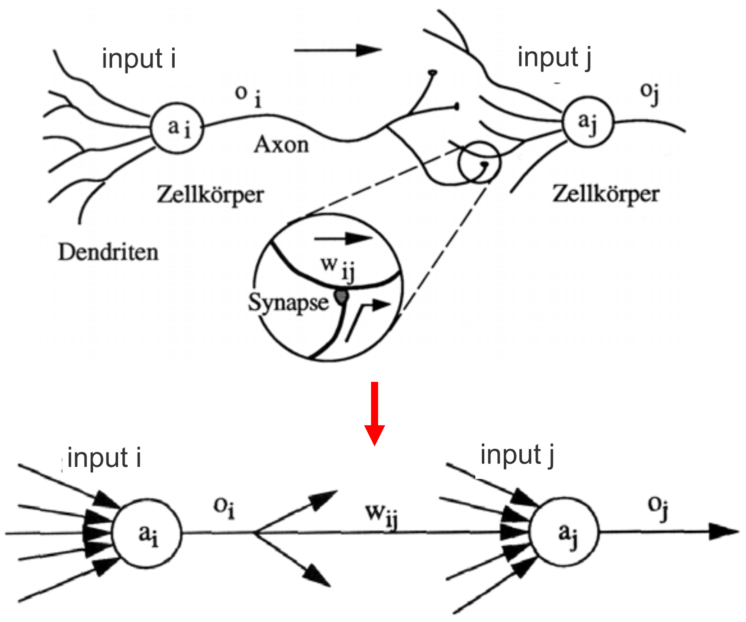
Contents

- Decision Trees
- Bootstrapping
- Random Forests
- Boosting
- Support Vector Machines
- Neural Networks

Neural Networks: Motivation

- The human brain is very good at interpreting scenes.
- The human brain consists of relatively simple nerve cells (neurons), but these are strongly interconnected.
- Assumption: The performance of the brain is related to this strong connectedness.
- Attempt to simulate these network structures in pattern recognition
  $\Rightarrow$ neural networks
- Research on neural networks started in the 1940s! Since the 1960s, they have gone in and out of fashion several times.
- Most recent development: Convolutional Neural Networks (CNN), “Deep Learning”.

Simulation of neurons

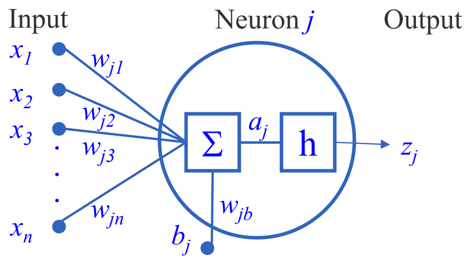


Quelle: Zell (1997), Simulation neuronaler Netze

Artificial model of a neuron I

- We deal again with a linear discriminant function.
- Input variables x_i: Components of the feature vector $\mathbf{x}$
- Each input variable is multiplied with a weight; Determine weighted sum:

$$a_j = \sum_i w_{ij} \cdot x_i + b_j = \mathbf{w}_j^T \cdot \mathbf{x} + b_j$$



Artificial model of a neuron II

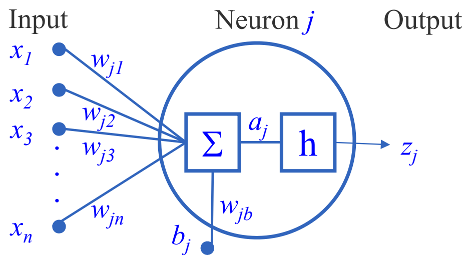
$$a_j = \sum_i w_{ij} \cdot x_i + b_j = \mathbf{w}_j^T \cdot \mathbf{x} + b_j$$

- b_j: „Bias“, considered to be a component of each feature vector; Simplified notation :

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix}^T \text{ und } \quad \mathbf{w}_j = \begin{bmatrix} \mathbf{w}_j^T & b_j \end{bmatrix}^T \Rightarrow a_j = \mathbf{w}_j^T \cdot \mathbf{x}$$

- The output of the neuron z_j is derived from a_j subject to a non-linear activation function $h(a_j)$:

$$z_j = h\left(a_j\right) = h\left(\mathbf{w}_j^T \cdot \mathbf{x}\right)$$

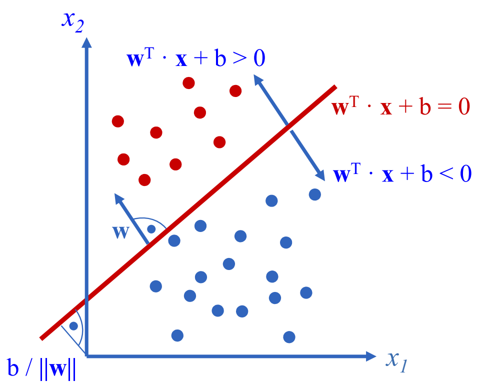


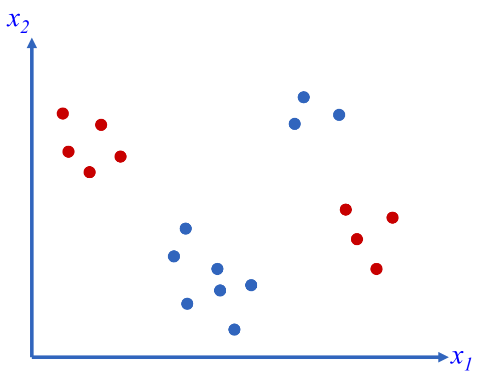
The Perceptron

- The Perceptron is the simplest possible neural network, consisting of one neuron.
- Proposed by Frank Rosenblatt already 1962!
  - The feature vector **x** is mapped by fixed non-linear transform Φ: $\mathbf{x} \to \Phi(\mathbf{x})$
  - Output: $z(\mathbf{x}) = h(a) = f\left(\mathbf{w}^T \cdot \Phi(\mathbf{x})\right)$
- Simplest activation function: $h(a) = \begin{cases} -1 & \text{for} \quad a < 0 \\ +1 & \text{for} \quad a \geq 0 \end{cases}$
- Binary classification, i.e. $C \in \{-1, +1\}$
- Class $C = sign\left(h(\mathbf{x})\right)$
- The reasons for the choice of $h(z)$ and $C \in \{-1, +1\}$ are discussed later in the context of learning of weight vector **w**.

Geometrical interpretation of the perceptron

- The perceptron delivers a hyperplane as decision boundary.
- A single layer perceptron only works if the classes are linearly separable in feature space (i.e., are separable by a linear hyperplane).


x_2
$\mathbf{w}^T \cdot \mathbf{x} + b > 0$
$\mathbf{w}^T \cdot \mathbf{x} + b = 0$
$\mathbf{w}^T \cdot \mathbf{x} + b < 0$
$\mathbf{w}$
$b / \|\mathbf{w}\|$
x_1
linear separable



not linear separable

How to deal with non-linear case?

- Networks consisting of several layers of "neurons“.
- In this way more complex decision boundaries between classes become possible ➔ “Multilayer Perceptron (MLP)”

- Example for two layers and “feed forward“ - architecture:
  - Input: Features $\mathbf{x}_i$
  - Hidden layer with neurons z_j:

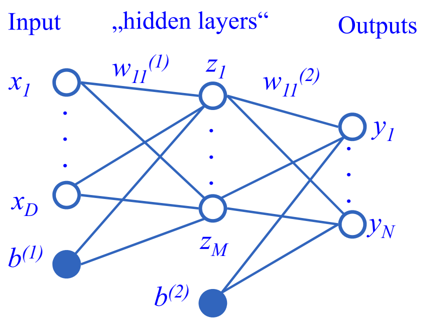
$$z_j = h\left(\sum w_{ij}^{(1)} \cdot x_i\right)$$

  - Output: Degree of membership to class C_k

$$y_k = h\left(\sum w_{ij}^{(2)} \cdot z_i\right)$$

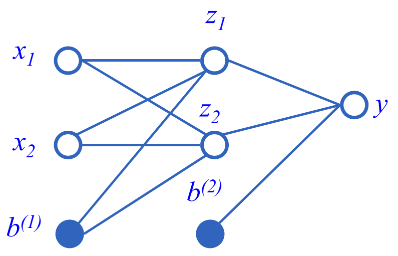
- Different choices for activation functions h

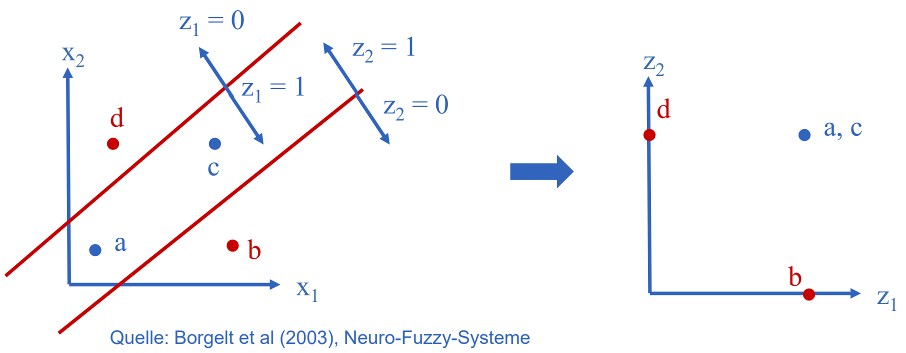
- Extension to more "hidden layers“ possible.



Geometrical interpretation of multi-layered networks

- Hidden layers act as feature space space mapping with **adaptive functions** Φ



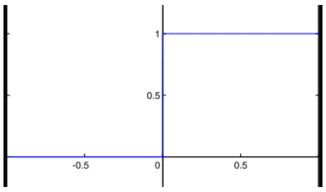


Quelle: Borgelt et al (2003), Neuro-Fuzzy-Systeme

Activation functions

- Step function:

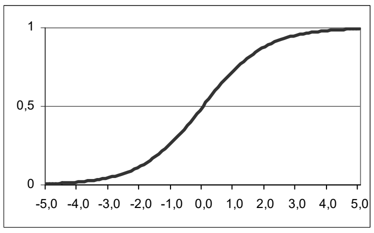
$$h(a)=\begin{cases}0 & \text{for} \quad a<0\\ 1 & \text{for} \quad a\geq 0\end{cases}$$



- Logistic sigmoid function:

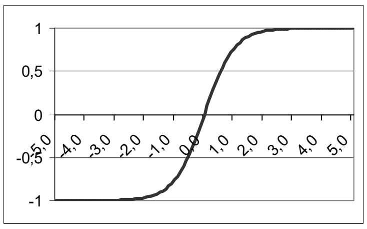
$$h(a)=\sigma(a)=\frac{1}{1+e^{-a}}$$

with $h'(a)=h(a)(1-h(a))$



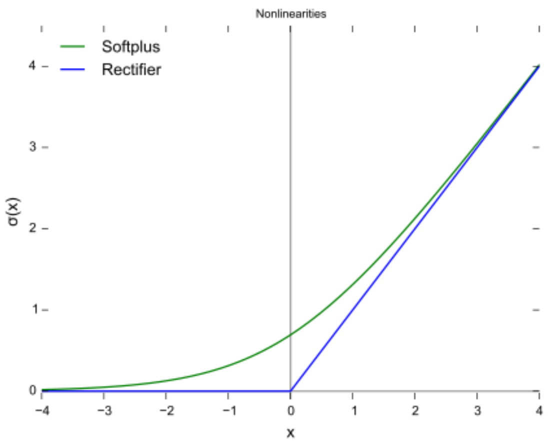
- Hyperbolic tangent (tanh):

$$h(a)=\tanh(a)=\frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$$

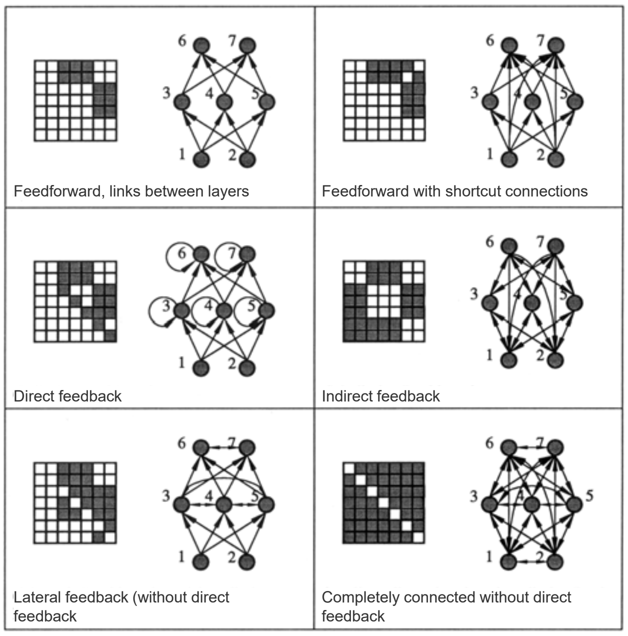


Further activation functions: ReLU and Softplus

- In particular often applied for CNN:
  - „Rectified linear unit" (ReLU) $h(a)=\max(0,a)$
  - A differentiable approximation (the derivation is just the sigmoid function) of ReLU results in so-called softplus function: $h(a)=\ln(1+e^{a})$



Examples of network topologies



Feedforward, links between layers
Feedforward with shortcut connections
Direct feedback
Indirect feedback
Lateral feedback (without direct feedback
Completely connected without direct feedback

Source: Zell (1997), Simulation neuronaler Netze

Supervised Learning: Perceptron – Ansatz

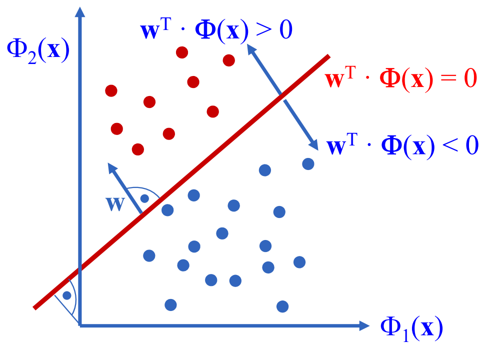
- Given:
  - N feature vectors $\mathbf{x}_n$ with class labels $C_n \in \{-1,+1\}$
  - Activation function: $$h(a) = \begin{cases} -1 & \text{für} \quad a < 0 \\ +1 & \text{für} \quad a \geq 0 \end{cases}$$
  - Desired classifier: $$C_n = sign\left(z\left(\mathbf{x}_n\right)\right) = sign\left(h\left(\mathbf{w}^T \cdot \Phi\left(\mathbf{x}_n\right)\right)\right)$$
- Wanted: Weights $\mathbf{w}$ of perceptron
  - One could try to determine $\mathbf{w}$ by minimizing the number of training samples that are assigned to the wrong class
    - Problem: the activation function is a step function, therefore, the gradient is always zero except of step edges.

      → Gradient descent is impossible!
  - Better: *Perceptron Criterion*-Method according to Rosenblatt

Supervised Learning: Perceptron – Perceptron Criterion I

- We encountered this criterion already in similar manner for SVM
  - Our aim is that all feature vectors $\mathbf{x}_n$ are sorted to the correct side of the hyperplane.
  - Again, we benefit from a smart choice of class labels as $C_n \in \{-1,+1\}$

$$\left.\begin{array}{l} \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) > 0 \text{ for } C_n = +1 \\ \mathbf{w}^T \cdot \Phi(\mathbf{x}_n) < 0 \text{ for } C_n = -1 \end{array}\right\} \rightarrow C_n\left(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n)\right) > 0$$

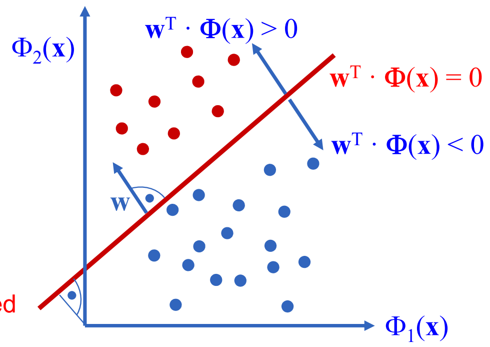

$\Phi_2(\mathbf{x})$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) > 0$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) = 0$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) < 0$
$\mathbf{w}$
$\Phi_1(\mathbf{x})$

Supervised Learning: Perceptron – Perceptron Criterion II

- The Perceptron Criterion is: $$E_P(\mathbf{w}) = -\sum_{n \in M} \left(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n)\right) \cdot C_n$$

- Interpretation:
  - $\mathbf{M}$ is the set of wrongly classified $\mathbf{x}_n$.
  - Correctly classified features $\mathbf{x}_n$ are not considered in error function.
  - Wrong classification:
    The error $\left(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n)\right) \cdot C_n$ is a *linear function* of $\mathbf{w}$.
  - Therefore, the error function is piecewise linear.

➔ Gradient descent methods can be applied


$\Phi_2(\mathbf{x})$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) > 0$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) = 0$
$\mathbf{w}^T \cdot \Phi(\mathbf{x}) < 0$
$\mathbf{w}$
$\Phi_1(\mathbf{x})$

Supervised Learning: Perceptron – Algorithm

- Given: N feature vectors $\mathbf{x}_n$ of classes $C_n \in \{-1,+1\}$
- Wanted: Weights $\mathbf{w}$ of perceptron
- Minimize the error function : $$E_P(\mathbf{w}) = -\sum_{n\in M}\left(\mathbf{w}^T \cdot \Phi(\mathbf{x}_n)\right)\cdot C_n$$
- Algorithm
  1. Initialize the weights with random values: $\mathbf{w}^{(0)}$
  2. As long as the minimum of $E_p(\mathbf{w})$ is not found, cycle through training data:
     1. Select a training sample $\mathbf{x}_n$ with class C_n
     2. Classify $\mathbf{x}_n$ using the current values of $\mathbf{w}$ → class C'_n
     3. If $C'_n \neq C_n$: Determination of new weights, e.g., by stochastic gradient descent:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\cdot\nabla E_P\left(\mathbf{w}^{(\tau)}\right) = \mathbf{w}^{(\tau)} + \eta\cdot\Phi(\mathbf{x}_n)\cdot C_n$$

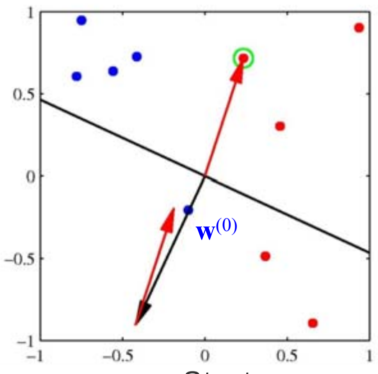
with η … Learning rate

Universität Stuttgart ifp

Supervised learning: Perceptron – Example

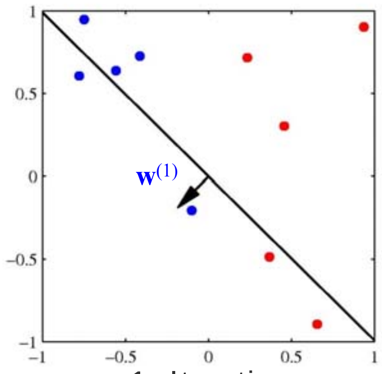
- Image left: Initial vector $\mathbf{w}^{(0)}$ (black), randomly selected training sample $\mathbf{x}_n$ assigned to the wrong class (green circle).
- Red vector: Error vector of the misclassified sample (here η =1), it is added to $\mathbf{w}^{(0)}$ to obtain $\mathbf{w}^{(1)}$ in iteration 1 (right image).

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\cdot\nabla E_P\left(\mathbf{w}^{(\tau)}\right) = \mathbf{w}^{(\tau)} + \eta\cdot\Phi(\mathbf{x}_n)\cdot C_n$$



Start

Nach Bishop, 2006



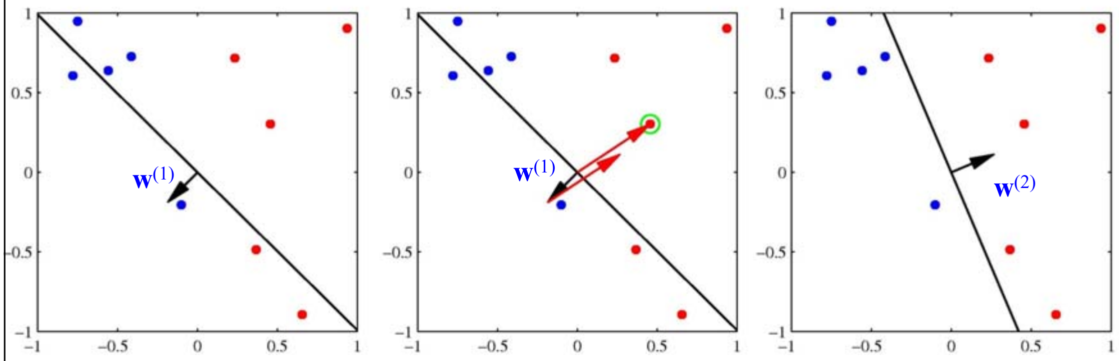
1. Iteration

Universität Stuttgart ifp

Supervised learning: Perceptron – Example

- Center image: Start with $\mathbf{w}^{(1)}$ (black vector) and another misclassified sample $\mathbf{x}_n$ (green circle).
- The vector of the misclassified sample (red) it is added to $\mathbf{w}^{(1)}$ to obtain $\mathbf{w}^{(2)}$ shown in the left image.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \cdot \nabla E_P\left(\mathbf{w}^{(\tau)}\right) = \mathbf{w}^{(\tau)} + \eta \cdot \Phi\left(\mathbf{x}_n\right) \cdot C_n$$



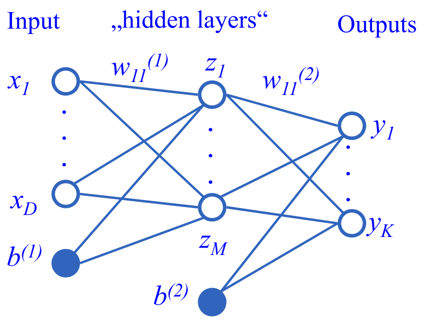
1. Iteration

2. Iteration

Nach Bishop, 2006

Multilayer networks: Backpropagation – Ansatz I

- In case of multilayer networks the supervised learning is more complicated.
- The standard method is called backpropagation, which subdivides the problem into parts for which the problem can be solved easier:
  - Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. doi:10.1038/323533a0.
- We discuss this method in the following slides.



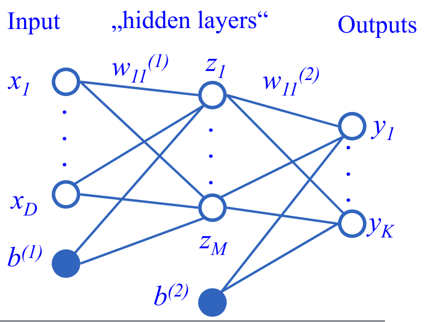
Multilayer networks: Backpropagation – Ansatz II

- The method works in two phases.
- **Phase 1:** Propagation of pattern forward and error backwards

1. Propagation of some $\mathbf{x}_n$ forward through the network to generate the output value.
2. Compare the network output with desired output. The difference is considered as **cost** that needs to be minimized.
3. Propagation of the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

- **Phase 2:** Weight update

1. The weight's output delta and input activation are multiplied to find the gradient of the weight.
2. A ratio (percentage) of the weight's gradient is subtracted from the weight.



Diskussion

- Neural networks had gone out of fashion compared to procedures such as SVM or random forests:
  - Networks with few layers: not adaptable enough
  - Networks with many neurons: numerical problems in the determination of the parameters
- Neural networks have come back in the context of “Deep Learning“ (“Google Brain“ project)
  - Networks with many layers (“deep" networks), many neurons.
  - Innovation: improved initialization of the weights by successive unsupervised learning of the individual layers

    → Back propagation for the final optimization of weights
  - Needs a lot of training data
  - Discussed in more detail in next chapter

Literature

- Barsi, A.,Thematic Classification of Landsat TM Imagery by a Neuro-Fuzzy Method. In: IntArchPhRS XXXII (7), 1998, S. 323-327.
- Barsi, A., Heipke, C., Artificial neural networks for the detection of road junctions in aerial images. In: IntArchPhRS XXXIV (3/W8) 2003, S. 113-118.
- Bishop, C. : Pattern Recognition and Machine Learning. 1st edition, Springer, New York, NY, 2006.
- Borgelt C., Klawonn F., Kruse R., Nauck D., Neuro-Fuzzy-Systeme, Vieweg Verlag, 2003.
- Breiman, L., Random Forrests. Journal of Machine Learning, pp. 5-32, 2001.
- Breiman, L., Cutler, A.: Random Forests web site. http://www.stat.berkeley.edu/~breiman/RandomForests, zuletzt besucht am 7.11.2012.
- Chehata, N., Guo, L., Mallet, C.: Airborne LiDAR feature selection for urban classification using random forests," International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences 38(3/W8):207-211, 2009.
- Duda, R. O., Hart, P. E., Stork, D. G.: Pattern Classification. 2nd edition, Wiley & Sons, New York, USA, 2001.

Literature

- Farabet, C., Couprie, C., Najman, L., LeCun, Y.: Learning hierarchical features for scene labeling. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(9):1915-1929, 2013.
- Golovinskiy, A., Kim, V.G., Funkhouser, T.: Shape-based recognition of 3D point clouds in urban environments," Proc. Int'l Conference on Computer Vision (on DVD), 2009.
- Goodfellow, I., Bengio, Y., Courville, A., Deep Learning. MIT Press, 2016. (online https://github.com/janishar/mit-deep-learning-book-pdf)
- Helmholz, P.: Verifikation von Ackerland- und Grünlandobjekten eines topographischen Datensatzes mit monotemporalen Bildern. In: Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover, ISSN 0174-1454, Nr. 299, Dissertation, Hannover, 2012.
- Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
- Lepetit, V., Fua, P.:, Keypoint Recognition using Randomized Trees. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(9):1465-1479, 2006.
- Lodha, S.K., Fitzpatrick, D.M., Helmbold, D.P.: Aerial LiDAR data classification using AdaBoost," Proc. 6th Int'l Conference on 3-D Digital Imaging and Modeling (3DIM '07), pp. 435-442, 2007.

Literature

- Mallet, C., Bretar, F., Roux, M., Soergel, U., Heipke, C.: Relevance assessment of full-waveform LiDAR data for urban area classification," ISPRS Journal Photogrammetry & Remote Sensing 66(6):S71–S84, 2011.
- Niemeyer, J., Wegner, J.-D., Mallet, C., Rottensteiner, F., Soergel, U.: Conditional Random Fields for urban scene classification with full waveform LiDAR Data, Stilla, et al. (Eds.): Photogrammetric Image Analysis, Lecture Notes in Computer Science, vol. 6952, Springer, Heidelberg, Dordrecht, London, New York, pp. 233-244, 2011.
- Niemeyer, J., Rottensteiner, F., Soergel, U.: Classification of urban LiDAR data using Conditional Random Field and Random Forests. Joint Urban Remote Sensing Event (JURSE), Sao Paulo, Brasilien, 2013.
- Platt, J. C.: Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Smola, A. J., Bartlett, P., Schölkopf, B., Schuurmans, D. (Hrsg.): Advances in Large Margin Classifiers, MIT Press, 1999.
- Quinlan, J. R.: C4.5: Programs for machine learning. Morgan Kaufmann, San Francisco, CA, USA, 1993.
- Ritter H., Martinez T., Schulten K.: Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke, Addison-Wesley, 1990.

Literature

- Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. doi:10.1038/323533a0.
- Rojas R.: Theorie der neuronalen Netze, Eine systematische Einführung, Springer Verlag, 1996.
- Schindler, K.: An overview and comparison of smooth labeling methods for land-cover classification. IEEE Transactions on Geoscience and Remote Sensing 99, zugänglich via http://www.igp.ethz.ch/photogrammetry/publications/pdf_folder/tgrs-2col.pdf (zuletzt besucht am 6.11.2012).
- Shotton, J., Kim, T.-K., Stenger, B.: Boosting & randomized forests for visual recognition, Tutorial at the International Conference on Computer Vision (ICCV), Kyoto, Japan, 2009; http://mi.eng.cam.ac.uk/~tkk22/iccv09_tutorial
- Vapnik, V. N.: Statistical learning theory. Wiley, 1998.
- Viola, P., Jones, M.: Robust real-time object detection, International Journal of Computer Vision 57(2):137-154, 2004.
- Zell A.: Simulation neuronaler Netze, Oldenbourg Verlag, 1994.
- Ziems, M., Heipke, C., Rottensteiner, F.: SVM-based road verification with partly non-representative training data. In: Joint IEEE-GRSS/ISPRS Workshop on Remote Sensing and Data Fusion over Urban Areas, S. 37-40, München, April 2011.