Machine Learning

9 – Classification

SS 2018

Gunther Heidemann

Overview

1. Definition of the classification problem
2. Bayesian classification
3. Important properties of classifiers
4. Overview of standard types of classifiers
5. Support vector machine (overview)
6. Random Forest

For details on the SVM, see, e.g.,

Vladimir Cherkassky, Filip Mulier: *Learning from Data*, IEEE Press

Classification

Classification:

- Assigns a discrete class to an object, fact, person, event etc. based on attributes.
- Classification results might be, e.g., {*dry, wet*}, {*good customer, bad customer*}, {*go, stop, left, right*}, {*spoon, fork, knife*}.
- Attributes might be *color, length, age, income, voltage* …
- Technically,
  - attributes are represented by a feature vector $\vec{x} \in \mathbb{R}^d$,
  - output are natural numbers $c(\vec{x}) \in C \subset \mathbb{N}$ assigned to the $|C|$ different classes,
  - alternatively, there are $|C|$ real valued functions $c_1(\vec{x}) \ldots c_{|C|}(\vec{x})$, each of which is the "confidence" that belongs to the class. The c_i are called discriminant functions. The output is

$$c(\vec{x}) = \arg\max_i c_i(\vec{x}), \qquad c(\vec{x}) \in C.$$

Bayes classifier

Idea of the **Bayes classifier**:

Classify an input $\vec{x}$ such that the expected cost is minimized!

That is, choose output class c such that

$$P(c|\vec{x}) = N \cdot P(\vec{x}|c) \cdot P(c),$$

is maximized, where

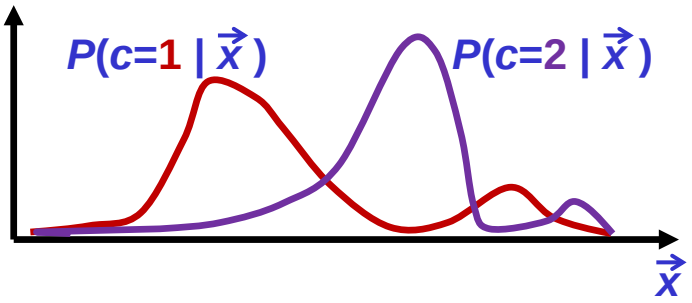
$P(\vec{x}|c)$ is the probability density that class c has features $\vec{x}$,

$P(c)$ is the a priori probability of class c, and

N is the normalization factor.

Representing classes as probabilities,

- overlapping classes can be represented,
- a unique class assignment is not possible.



[H]

Bayes classifier

Problems of the Bayes classifier:

In

$$P(c|\vec{x}) \quad = \quad N \cdot P(\vec{x}|c) \cdot P(c),$$

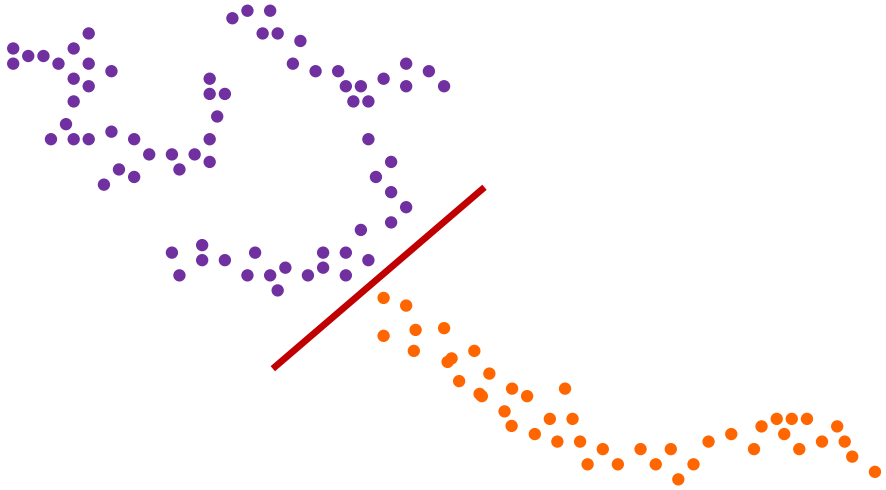
the probability densities $P(\vec{x}|c)$, and sometimes also $P(c)$, are not known explicitly but need to be estimated from the data.

Two strategies:

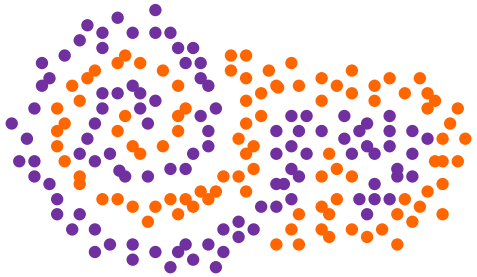
1. Estimate $P(\vec{x}|c)$ and - if necessary - $P(c)$ from the data, then apply original Bayes classifier. May require a lot of effort and more information than necessary to obtain the classifier (because a classifier needs information to derive decision boundaries, not necessarily the complete densities.
2. Construct an approximation to the Bayes classifier directly from the data.

Bayes classifier

Complete knowledge of $P(\vec{x}|c)$ may be unnecessary to find a separatrix



… or necessary:



[H]

Types of classifiers

We will review basic types of classifiers.

These are relevant criteria:

General issues:

- What assumptions are made about the data distribution?
- What is the bias?

Technical issues:

- Representation:
  - How are separatrices represented (implicitly, explicitly)?
  - Type of separatrices?
  - Generalization properties?
  - Sensitivity to errors and noise?

Types of classifiers

- Locality:
  - Is the influence of a single example on the separatrices local or global?
  - How is locality controlled by parameters?
- Parameters:
  - Parameters for knowledge representation, e.g., architecture, flexibility of separatrices, smoothness?
  - Adaptation parameters, e.g., step size?
  - Do the parameters have an intuitive interpretation?
  - How much depends the outcome on the choice of the parameters?
- Speed:
  - Training phase
  - Application phase

Types of classifiers

The following, we compare standard classifiers according to the criteria.

We assume

- the input data have been transformed to zero mean,
- there are only two classes, c^+ and c^-. $\vec{x}^+$, $\vec{x}^-$ denote feature vectors of the classes.

The classifier c is based on a discriminant function $R(\vec{x})$ such that

$$c(\vec{x}) = \begin{cases} c^+ & \text{if } R(\vec{x}) \geq 0, \\ c^- & \text{if } R(\vec{x}) < 0. \end{cases}$$

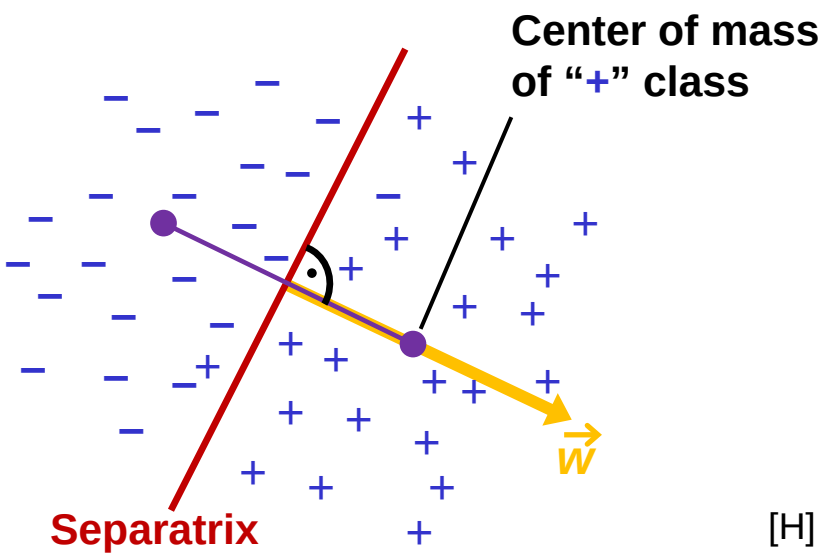
Euclidean classifier

Discriminant is found from the centers of mass of the classes:

$$R(\vec{x}) = \vec{w} \cdot \vec{x}$$

with

$$\vec{w} = \langle \vec{x}^{+} \rangle - \langle \vec{x}^{-} \rangle .$$



[H]

- Linear separatrix, explicit representation.
- Not local.
- Very fast "training".
- No parameters.
- Sensitive to far outliers.

Linear discriminant analysis (LDA)

Assumptions:

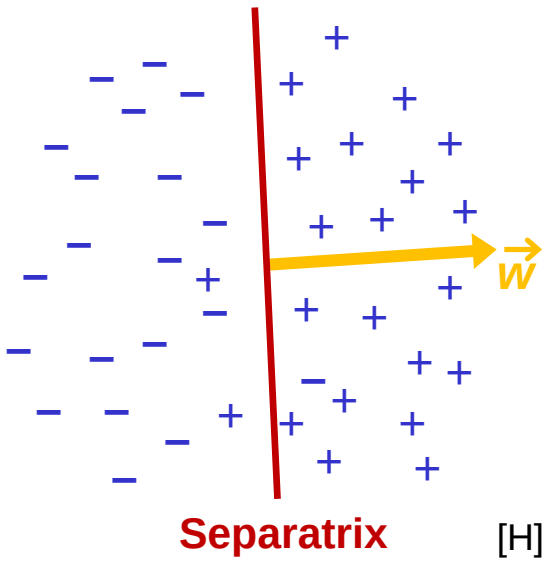
- Classes have identical a priori probability.
- Both classes exhibit a Gaussian distribution with means $\langle\vec{x}^+\rangle$, $\langle\vec{x}^-\rangle$ and covariance matrices Λ^+, Λ^-.
- Covariances are assumed to be identical for both classes $\Lambda = \Lambda^+ = \Lambda^-$.
- Same properties as Euclidean classifier (qualitatively).

$$R(\vec{x}) = \vec{w}\cdot\vec{x}$$

with

$$\vec{w} = \Lambda^{-1}(\langle\vec{x}^+\rangle - \langle\vec{x}^-\rangle),$$

$$\Lambda = \langle\vec{x}\,\vec{x}^T\rangle.$$



[H]

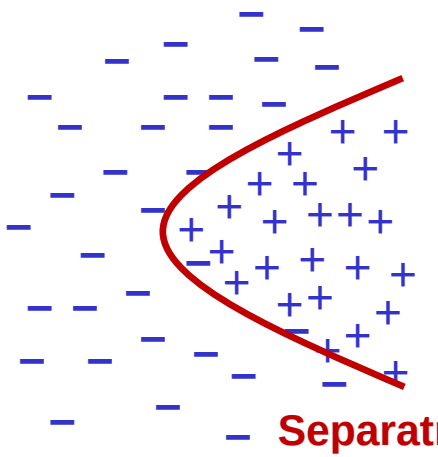
Quadratic classifier

Discriminant is quadratic

$$R(\vec{x}) = \vec{x}^{\top} A \vec{x} + \vec{b}^{\top}\vec{x} + c,$$

$$A \in \mathbb{R}^{d \times d}, \quad \vec{b}, \vec{x} \in \mathbb{R}^{d}, \quad c \in \mathbb{R}.$$

- A, $\vec{b}$, c are found, e.g., by quadratic discriminant analysis (QDA).
- Separatrix is a conic section, i.e., a
  - hyperbola,
  - parabola,
  - ellipsis (special case: circle),
  - line.
- Effect of A, $\vec{b}$, c is not local.
- Fast “training”.
- No parameters.



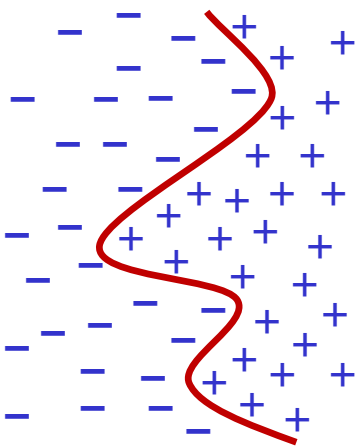
[H]

Polynom classifier

Polynomial discriminant function:

$$R(\vec{x}) \quad = \quad \text{Polynom}(\vec{x}).$$

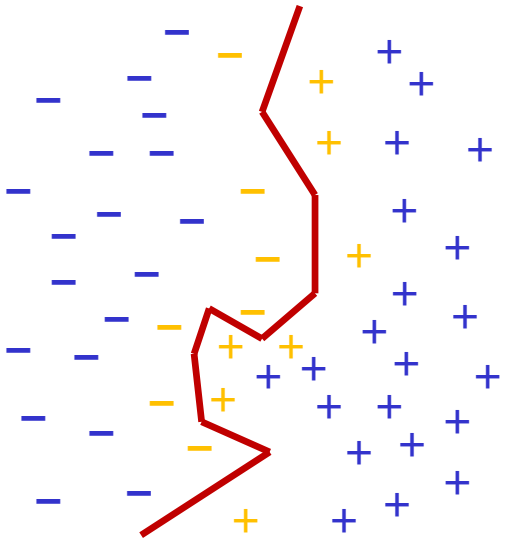
- Separatrix of almost arbitrary complexity (like MLP).
- Generalization can be controlled by degree of polynom,
- … but the effect of the degree as a parameter is difficult to foresee in high dimensions.
- Not local.
- "Training" is relatively efficient.
- Well established technique, e.g., in NLP.



Separatrix [H]

Nearest neighbor classifier

- Separatrix is implicitly defined by neighbors.
- Local.
- No generalization (as far as classification boundary is concerned).
- No parameters.
- No training time but may have considerable memory consumption.
- Application time depends on size.



[H]

Support vector machine

Support vector machine (SVM) is one of the most popular classifiers.

Principle:

1. SVM computes a hyperplane (linear separatrix)
   - based on the examples (*support vectors*) close to the class boundary,
   - such that the *margin* is maximized.
   - *Slack variables* allow to deal with outliers to avoid overfitting.
2. To solve nonlinear problems, the *kernel trick* is used:
   - Project data into a space of *higher* dimension.
   - For sufficiently high dimension, every problem becomes (linearly) separable by a hyperplane.
   - The projection of this hyperplane back to the original data space is a nonlinear separatrix.

Support vector machine

Binary classification problem $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots\}$, $\vec{x}_i \in \mathbb{R}^d$, $y_i \in [-1, 1]$.

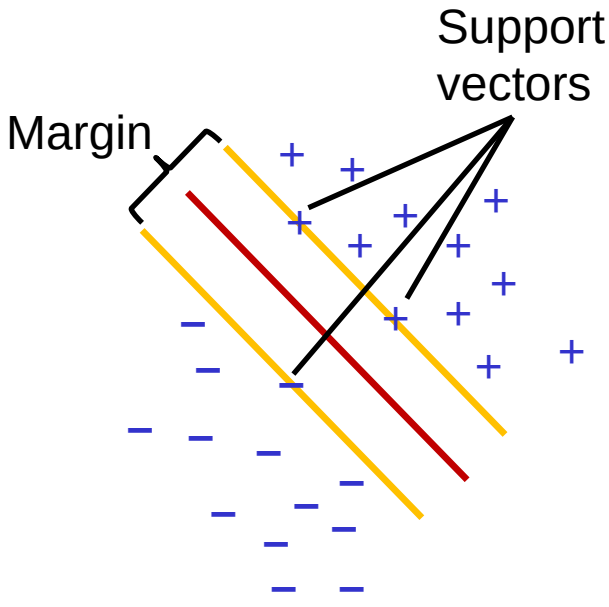
SVM represents a hyperplane

$$\vec{w} \cdot \vec{x} - b = 0,$$

where $\vec{w}$ is the normal to the hyperplane and $b / |\vec{w}|$ is its offset from the origin.

For a linearly separable problem, the SVM selects $\vec{w}$ and b such that the separatrix is in the middle of the margin.

An extension provides a way to deal with outliers that disturb linear separability.

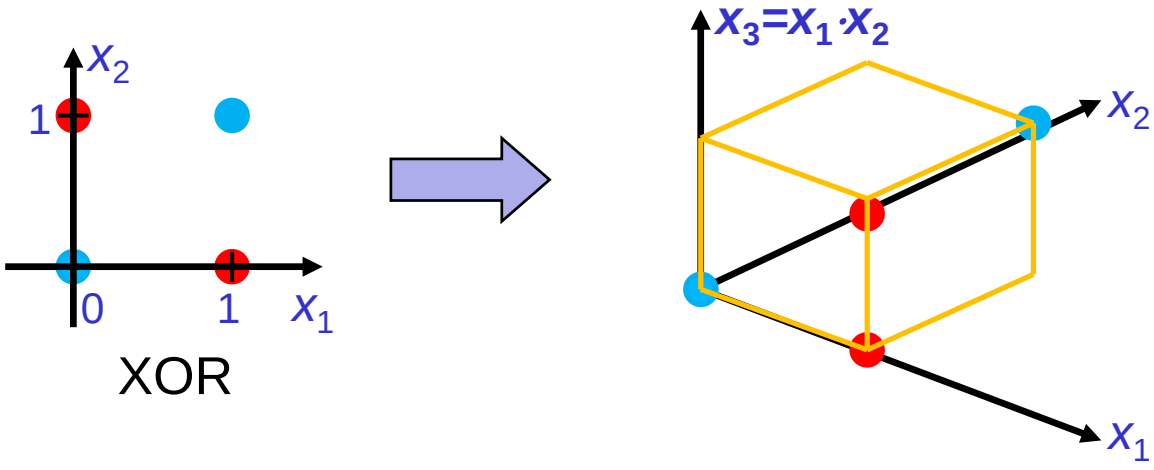


[H]

Support vector machine

To apply the SVM to not linearly separable problems, map the data from the original input space ($\vec{x} \in \mathbb{R}^d$) to a higher dimensional space H where it becomes linearly separable.

Remember solving the XOR problem with a simple perceptron in a space of higher dimension:



[H]

If this is insufficient for a problem, the 2d input vectors $(x_1,x_2)^T$ might be mapped, e.g., using polynomials to

$$(x_1, x_2, x_1^2, x_2^2, x_1 \cdot x_2, x_1^3, x_2^3, x_1^2 \cdot x_2, x_1 \cdot x_2^2)^T \ .$$

Support vector machine

Problem of mapping to a space of higher dimension:

- For more input dimensions, the mapping $\mathbb{R}^d \rightarrow H$ using, e.g., polynomials becomes huge.
- Mapping causes high effort.

Solution by the kernel trick:

- Fortunately, the SVM procedure to find the optimal hyperplane within the margin only requires the inner product of vectors $\vec{x}_a \cdot \vec{x}_b$, not the vectors themselves.
- Thus, only the inner products need to be computed in H.
- This can be done without actually computing the representation of $\vec{x}_a \cdot \vec{x}_b$ in H.
- Instead, the inner product of $\vec{x}_a$ and $\vec{x}_b$ in H can be computed directly in the original input space using a *kernel function* K as $K(\vec{x}_a, \vec{x}_b)$.

Support vector machine

- The kernel trick allows the very efficient implicit computation of a hyperplane in H and thus a nonlinear separatrix in $\mathbb{R}^d$.
- The kernel function should satisfy Mercer's condition, but in practice useful results can be achieved also with other kernel functions.
- Suitable mapping functions with a kernel that satisfies Mercer's condition are, e.g., polynomials / splines, radial basis functions or Fourier expansions.

Random Forests

Motivation:

- Decision trees are easy to build and easy to use.
- Drawback: Overfitting, often due to
  1. ill chosen examples (noise, outliers),
  2. ill chosen features (at least for some of the examples).
- Idea: “Average out” overfitting by
  1. a “*bag of trees*” = “*forest*” approach,
  2. a “*bag of features*” approach.
- Averaging is over the models, not the data.
- Thus the variance of classification is reduced.

Creating a random forest

- A random forest is a (large) collection of uncorrelated decision trees.
- How can we get different, uncorrelated decision trees out of the same set of training data $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots\}$, $\vec{x}_i \in \mathbb{R}^d$, $y_i \in$ N ?
- Create random subsets, e.g.,
  - $D_0 = \{(\vec{x}_4, y_4), (\vec{x}_{22}, y_{22}), \ldots (\vec{x}_{316}, y_{316})\}$,
  - $D_1 = \{(\vec{x}_{17}, y_{17}), (\vec{x}_{22}, y_{22}), \ldots (\vec{x}_{976}, y_{976})\}$,
  - …
- Subsets may overlap.
- From D_0, D_1 …, create the forest of decision trees T_0, T_1 … .
- The forest is *random* insofar as the training sets are random.

Random Forests

- So far: Bagging of trees.
- Improvement: Bagging of features.

Random Forests

Remember how a decision tree is built:

1. *A* ← the “best” decision feature for next *node*, chosen out of all features.
2. Assign *A* as decision attribute for node.
3. For each value of *A*, create new descendant of *node*.
4. Sort training examples to leaf nodes.
5. If the training examples are perfectly classified then STOP
   else iterate over new leaf nodes.

Random Forests

Feature bagging:

1. *A* ← the "best" decision feature for next *node*, chosen out of a random subset of all features.
2. Assign *A* as decision feature for node.
3. For each value of *A*, create new descendant of *node*.
4. Sort training examples to leaf nodes.
5. If the training examples are perfectly classified then STOP
   else iterate over new leaf nodes.

Main advantage of feature bagging:

Influence of features with extreme predictive power is reduced.

Applying a random forest for classificatiton

- Input: Feature vector $\vec{x}$ of unknown class.
- Each tree yields a classification $T_0(\vec{x})$, $T_1(\vec{x})$ … .
- The class the majority of trees votes for is the output of the forest.

Random Forests

Properties:

- Very simple algorithm.
- Very small number of parameters.
- Both training and application are well suited for parallelization.
- Very popular for applications.
- Concerning performance, random forests are said to be the second best to deep learning.

Summary

- Classification is one of the most important ML tasks.
- Classifiers differ with respect to many criteria, the most important of which is the shape, “flexibility” and representation of the separatrices.
- SVMs are popular classifiers, along with (deep) MLPs. Numerous implementations exist.
- Currently, Deep Learning approaches dominate competitions. In practice, Random Forests appear to be superior as they are simple and easy to use.
- Success of classification depends very much on the choice of suitable features from the problem. As a rule of thumb, features are more important than the classifier.

Image sources

[M] Online material available at www.cs.cmu.edu/~tom/mlbook.html for the textbook: Tom M. Mitchell: *Machine Learning*, McGraw-Hill

[H] Gunther Heidemann, 2012.