Agenda

1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification

   4.2 Regression

5. Advanced Topics
Additional Literature for this Chapter

Introduction: Example

▶ Training data

<table>
<thead>
<tr>
<th>age</th>
<th>car_type</th>
<th>max_speed</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>family</td>
<td>180</td>
<td>high</td>
</tr>
<tr>
<td>17</td>
<td>sportive</td>
<td>240</td>
<td>high</td>
</tr>
<tr>
<td>43</td>
<td>sportive</td>
<td>246</td>
<td>high</td>
</tr>
<tr>
<td>68</td>
<td>family</td>
<td>183</td>
<td>low</td>
</tr>
<tr>
<td>32</td>
<td>truck</td>
<td>110</td>
<td>low</td>
</tr>
</tbody>
</table>

▶ Simple classifier

\[
\begin{align*}
\text{if } & \text{ age } > 50 \text{ then risk } = \text{ low} \\
\text{if } & \text{ age } \leq 50 \text{ and car type } = \text{ truck then risk } = \text{ low} \\
\text{if } & \text{ age } \leq 50 \text{ and car type } \neq \text{ truck then risk } = \text{ high}
\end{align*}
\]
## Classification: Training Phase (Model Construction)

### Table

<table>
<thead>
<tr>
<th>ID</th>
<th>age</th>
<th>car type</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23</td>
<td>family</td>
<td>high</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>sportive</td>
<td>high</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>sportive</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>68</td>
<td>family</td>
<td>low</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>truck</td>
<td>low</td>
</tr>
</tbody>
</table>

### Diagram

1. **Training Data**
   - Input: (age=60, family)
   - Rule: if age > 50 then risk = low;
       - if age ≤ 50 and car type = truck then risk = low;
       - if age ≤ 50 and car type ≠ truck then risk = high

2. **Classifier**
3. **Class Label**
Classification: Prediction Phase (Application)

<table>
<thead>
<tr>
<th>ID</th>
<th>age</th>
<th>car type</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23</td>
<td>family</td>
<td>high</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>sportive</td>
<td>high</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>sportive</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>68</td>
<td>family</td>
<td>low</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>truck</td>
<td>low</td>
</tr>
</tbody>
</table>

Training data:

unknown data → classifier

(age=60, family)

if age > 50 then risk = low;
if age ≤ 50 and car type = truck then risk = low;
if age ≤ 50 and car type ≠ truck then risk = high

class label

risk = low
Classification

The systematic assignment of new observations to known categories according to criteria learned from a training set.

Formal Setup

- A classifier $K$ for a model $M(\theta)$ is a function $K_{M(\theta)} : D \rightarrow Y$, where
  - $D$: data space
    - Often $d$-dim. space with attributes $a_1, \ldots, a_d$ (not necessarily a vector space)
    - Some other space, e.g. metric space
  - $Y = \{y_1, \ldots, y_k\}$: set of $k$ distinct class labels
  - $O \subseteq D$: set of training objects $o$ with known class labels $y \in Y$

- Classification: Application of classifier $K$ on objects from $D \setminus O$

- Model $M(\theta)$ is the "type" of the classifier, and $\theta$ are the model parameters

- Supervised learning: find/learn optimal parameters $\theta$ for $M(\theta)$ given $O$
Supervised vs. Unsupervised Learning

Unsupervised Learning (clustering)

- The class labels of training data are unknown
- Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data
  - Classes (=clusters) are to be determined

Supervised Learning (classification)

- Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - Classes are known in advance (a priori)
- New data is classified based on information extracted from the training set
Numerical Prediction

▶ Related problem to classification: numerical prediction
  ▶ Determine the numerical value of an object
  ▶ Method: e.g., regression analysis
  ▶ Example: Prediction of flight delays

▶ Numerical prediction is different from classification
  ▶ Classification refers to predict categorical class label
  ▶ Numerical prediction models continuous-valued functions

▶ Numerical prediction is similar to classification
  ▶ First, construct a model
  ▶ Second, use model to predict unknown value
  ▶ Major method for numerical prediction is regression:
    ▶ Linear and multiple regression
    ▶ Non-linear regression
Goals for this Section

1. Introduction of different classification models
2. Learning techniques for these models

<table>
<thead>
<tr>
<th>age</th>
<th>car_type</th>
<th>max_speed</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>family</td>
<td>180</td>
<td>high</td>
</tr>
<tr>
<td>17</td>
<td>sportive</td>
<td>240</td>
<td>high</td>
</tr>
<tr>
<td>43</td>
<td>sportive</td>
<td>246</td>
<td>high</td>
</tr>
<tr>
<td>68</td>
<td>family</td>
<td>183</td>
<td>low</td>
</tr>
<tr>
<td>32</td>
<td>truck</td>
<td>110</td>
<td>low</td>
</tr>
</tbody>
</table>

Bayes classifier  Linear classifier  Decision tree  k-NN classifier
Quality Measures for Classifiers

- Classification accuracy or classification error (complementary)
- Compactness of the model
  - Decision tree size, number of decision rules, ...
- Interpretability of the model
  - Insights and understanding of the data provided by the model
- Efficiency
  - Time to generate the model (training time)
  - Time to apply the model (prediction time)
- Scalability for large databases
  - Efficiency in disk-resident databases
- Robustness
  - Robust against noise or missing values
Evaluation of Classifiers: Notions

- Using training data to build a classifier and to estimate the model’s accuracy may result in misleading and overoptimistic estimates
  - Overspecialization of the learning model to the training data
- **Train-and-Test**: Decomposition of labeled data set \( O \) into two partitions
  - Training data is used to train the classifier
    - Construction of the model by using information about the class labels
  - Test data is used to evaluate the classifier
    - Temporarily hide class labels, predict them anew and compare with original class labels
- Train-and-Test is not applicable if the set of objects for which the class label is known is very small
Evaluation of Classifiers: Cross Validation

$m$-fold Cross Validation

- Decompose data set evenly into $m$ subsets of (nearly) equal size
- Iteratively use $(m - 1)$ partitions for training data and the remaining single partition as test data
- Combine the $m$ classification accuracy values to an overall classification accuracy

Leave-one-out: Special case of cross validation ($m = n$)

- For each of the objects $o$ in the data set $O$:
  - Use set $O \setminus \{o\}$ as training set
  - Use the singleton set $\{o\}$ as test set
  - Compute classification accuracy by dividing the number of correct predictions through the database size $|O|$  
- Particularly well applicable to nearest-neighbor classifiers
Let $K$ be a classifier

Let $C(o)$ denote the correct class label of an object $o$

Measure the quality of $K$:

- Predict the class label for each object $o$ from a data set $T \subseteq O$
- Determine the fraction of correctly predicted class labels

**Classification Accuracy of $K$**

$$G_T(K) = \frac{|\{o \in T \mid K(o) = C(o)\}|}{|T|}$$

**Classification Error of $K$**

$$F_T(K) = \frac{|\{o \in T \mid K(o) \neq C(o)\}|}{|T|} = 1 - G_T(K)$$
Quality Measures: Accuracy and Error

- Let $K$ be a classifier
- Let $TR \subseteq O$ be the training set: Used to build the classifier
- Let $TE \subseteq O$ be the test set: Used to test the classifier

**Resubstitution Error of $K$**

$$F_{TR}(K) = \frac{|\{o \in TR \mid K(o) \neq C(o)\}|}{|TR|}$$

**True Classification Error of $K$**

$$F_{TE}(K) = \frac{|\{o \in TE \mid K(o) \neq C(o)\}|}{|TE|}$$
Quality Measures: Confusion Matrix

▶ Results on the test set: Confusion matrix

<table>
<thead>
<tr>
<th>correct label</th>
<th>class 1</th>
<th>class 2</th>
<th>class 3</th>
<th>class 4</th>
<th>class 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>class 1</td>
<td>35</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>class 2</td>
<td>0</td>
<td>31</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>class 3</td>
<td>3</td>
<td>1</td>
<td>50</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>class 4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>class 5</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>16</td>
<td>13</td>
</tr>
</tbody>
</table>

(correctly classified in green)

▶ Based on the confusion matrix, we can compute several accuracy measures, including:
  ▶ Classification Accuracy/Error
  ▶ Precision and Recall
Quality Measures: Precision and Recall

Recall

Fraction of test objects of class $i$, which have been identified correctly.

\[
Recall_{TE}(K, i) = \frac{|\{o \in C_i \mid K(o) = C(o)\}|}{|C_i|}
\]

Precision

Fraction of test objects assigned to class $i$, which have been identified correctly.

\[
Precision_{TE}(K, i) = \frac{|\{o \in C_i \mid K_i(o) = C(o)\}|}{|K_i|}
\]
Overfitting

Characterization of Overfitting

The classifier adapts too closely to the training dataset and may therefore fail to accurately predict class labels for test objects unseen during training.

Example: Decision Tree

Generalization $\leftarrow$ \textit{classifier} $\rightarrow$ \textit{specialization}

“overfitting”
Overfitting

- Occurs when the classifier is too optimized to the (noisy) training data
- As a result, the classifier yields worse results on the test data set
- Potential reasons:
  - Bad quality of training data (noise, missing values, wrong values)
  - Different statistical characteristics of training data and test data

Overfitting Avoidance

- Removal of *noisy/erroneous/contradicting* training data
- Choice of an appropriate *size* of the training set
  - Not too small, not too large
- Choice of appropriate sample
  - Sample should describe all aspects of the domain and not only parts of it
Occurs when the classifiers model is too simple, e.g. trying to separate classes linearly that can only be separated by a quadratic surface

Happens seldomly

\textit{Trade-off}: Usually one has to find a good balance between over- and underfitting.
4. Supervised Methods

4.1 Classification

4.1.1 Bayesian Classifiers
4.1.2 Linear Discriminant Functions
4.1.3 Support Vector Machines
4.1.4 Kernel Methods
4.1.5 Decision Tree Classifiers
4.1.6 Nearest Neighbor Classifiers
4.1.7 Ensemble Classification

4.2 Regression

5. Advanced Topics
Bayes Classification

- Probability based classification
  - Based on likelihood of observed data, estimate explicit probabilities for classes
  - Classify objects depending on costs for possible decisions and the probabilities for the classes
- Incremental
  - Likelihood functions built up from classified data
  - Each training example can incrementally increase/decrease the probability that a hypothesis (class) is correct
  - Prior knowledge can be combined with observed data.
- Good classification results in many applications
Bayes’ Theorem

Probability Theory

- Conditional probability: \( P(A \mid B) = \frac{P(A \wedge B)}{P(B)} \) ("prob. of A given B")
- Product Rule: \( P(A \wedge B) = P(A \mid B) \cdot P(B) \)

Bayes’ Theorem

- \( P(A \wedge B) = P(A \mid B) \cdot P(B) \)
- \( P(B \wedge A) = P(B \mid A) \cdot P(A) \)
- Since \( P(A \wedge B) = P(B \wedge A) \), \( P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A) \), and thus

\[
P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}
\]
Bayes Classifier

- Bayes’ rule: \( P(c_j | x) = \frac{P(x|c_j) \cdot P(c_j)}{p(x)} \) for object \( x \) and class \( c_j \in C \).
- We are interested in maximizing this, i.e.

\[
\operatorname{argmax}_{c_j \in C} (P(c_j | x)) = \operatorname{argmax}_{c_j \in C} \left( \frac{P(x | c_j) \cdot P(c_j)}{p(x)} \right) \overset{(*)}{=} \operatorname{argmax}_{c_j \in C} (P(x | c_j) \cdot P(c_j))
\]

where \((*)\) assumes the value of \( p(x) \) is constant and hence does not change the result.
- Final decision rule:

\[
K(x) = c_{\text{max}} = \operatorname{argmax}_{c_j \in C} (P(x | c_j) \cdot P(c_j))
\]

- But how to obtain \( P(c_j) \) and \( P(x | c_j) \).
Bayes Classifier: Density Estimation

A-Priori Class Probabilities

Estimate the a-priori probabilities $P(c_j)$ of classes $c_j \in \mathcal{C}$ by using the observed relative frequency of the individual class labels $c_j$ in the training set, i.e.,

$$P(c_j) = \frac{N_{c_j}}{N}$$

Conditional Probabilities

- **Non-parametric** methods: Kernel methods Parzen’s window, Gaussian kernels, etc.
- **Parametric** methods, e.g.
  - Single Gaussian distribution: Computed by maximum likelihood estimators (MLE)
  - *Mixture* models: e.g. Gaussian Mixture Model computed by EM algorithm
Bayes Classifier: Density Estimation

Problem
Curse of dimensionality often lead to problems in high dimensional data \( \leadsto \) Density functions become too uninformative

Solution
- Dimensionality reduction
- Usage of statistical independence of single attributes (extreme case: naïve Bayes)
Naïve Bayes Classifier

Assumptions

▶ Objects are given as $d$-dimensional vectors, $x = (x_1, \ldots, x_d)$
▶ For any given class $c_j$ the attribute values $x_i$ are conditionally independent, i.e.

$$P(x_1, \ldots, x_d \mid c_j) = \prod_{i=1}^{d} P(x_i \mid c_j) = P(x_1 \mid c_j) \cdot \ldots \cdot P(x_d \mid c_j)$$

Decision Rule

$$K_{\text{naïve}}(x) = \arg\max_{c_j \in C} \left( P(c_j) \cdot \prod_{i=1}^{d} P(x_i \mid c_j) \right)$$
### Naïve Bayes Classifier

**Categorical Attribute \( x_i \)**

\[ P(x_i \mid c_j) \] can be estimated as the relative frequency of samples having value \( v_i \) as the \( i \)th attribute in class \( c_j \) in the training set.

**Continuous Attribute \( x_i \)**

\[ P(x_i \mid c_j) \] can, for example, be estimated through a Gaussian distribution determined by \( \mu_{ij}, \sigma_{ij} \).

\[ \rightsquigarrow \text{Computationally easy in both cases.} \]
Naïve Bayes Classifier: Example

<table>
<thead>
<tr>
<th>age</th>
<th>car_type</th>
<th>max_speed</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>family</td>
<td>180</td>
<td>high</td>
</tr>
<tr>
<td>17</td>
<td>sportive</td>
<td>240</td>
<td>high</td>
</tr>
<tr>
<td>43</td>
<td>sportive</td>
<td>246</td>
<td>high</td>
</tr>
<tr>
<td>68</td>
<td>family</td>
<td>183</td>
<td>low</td>
</tr>
<tr>
<td>32</td>
<td>truck</td>
<td>110</td>
<td>low</td>
</tr>
</tbody>
</table>

Model Setup

- \( \text{Age} \sim N(\mu, \sigma^2) \) (normal distribution)
- \( \text{car\_type} \sim \text{relative frequencies} \)
- \( \text{max\_speed} \sim N(\mu, \sigma^2) \) (normal distribution)
Naive Bayes Classifier: Example (cont’d)

Query

\( q = (\text{age} = 60; \text{car\_type} = \text{family}; \text{max\_speed} = 190) \)

Example

We have:

- \( P(\text{high}) = \frac{3}{5} \)
- \( \mu_{\text{age,high}} = \frac{83}{3}, \sigma^2_{\text{age,high}} = \frac{1112}{3} \implies P(\text{age} = 60 \mid \text{high}) \approx 0.00506 \)
- \( P(\text{car\_type} = \text{family} \mid \text{high}) = \frac{1}{3} \)
- \( \mu_{\text{max\_speed,high}} = 222, \sigma^2_{\text{max\_speed,high}} = 2664 \implies P(\text{max\_speed} = 190 \mid \text{high}) \approx 0.00638 \)

and hence

\[
P(\text{high})P(q \mid \text{high}) = P(\text{high})P(\text{age} = 60 \mid \text{high})P(\text{car\_type} = \text{family} \mid \text{high})P(\text{max\_speed} = 190 \mid \text{high}) \\
\approx 6.45166 \cdot 10^{-6}
\]

Analogously, we obtain \( P(\text{low})P(q \mid \text{low}) = 15.72290 \cdot 10^{-6} \implies K_{\text{naive}}(q) = \text{low}. \)
Bayesian Classifier

- Assuming dimensions of $x = (x_1, \ldots, x_d)$ are not independent
- Assume multivariate normal distribution (i.e. Gaussian)

$$P(x \mid C_j) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_j)}} \exp \left( -\frac{1}{2} (x - \mu_j)\Sigma_j^{-1}(x - \mu_j)^T \right)$$

with

- $\mu_j$: mean vector of class $C_j$
- $\Sigma_j$ is the $d \times d$ covariance matrix
- $\det(\Sigma_j)$ is the determinant of $\Sigma_j$, and $\Sigma_j^{-1}$ its inverse
Example: Interpretation of Raster Images

- Scenario: Automated interpretation of raster images
  - Take an image from a certain region (in \(d\) different frequency bands, e.g., infrared, etc.)
  - Represent each pixel by \(d\) values: \((x_1, \ldots, x_d)\)
- Basic assumption: different surface properties of the earth ("landuse") follow a characteristic reflection and emission pattern

![Diagram of surface of the earth and feature-space with values and bands](image-url)
Example: Interpretation of Raster Images

Application of the Bayes classifier:

▶ Estimation of the $P(x \mid c)$ without assumption of conditional independence

▶ Assumption of $d$-dimensional normal (= Gaussian) distributions for the value vectors of a class
### Method

Estimate the following measures from training data

- $\mu_j$: $d$-dimensional mean vector of all feature vectors of class $C_j$
- $\Sigma_j$: $d \times d$ covariance matrix of class $C_j$

### Problems

- if likelihood of respective class is very low
- if several classes share the same likelihood

Mitigate e.g. by applying some minimum likelihood threshold; do not classify regions below.
## Bayesian Classifiers: Discussion

### Pro

- High classification accuracy for many applications if density function defined properly
- Incremental computation: many models can be adopted to new training objects by updating densities
  - For Gaussian: store count, sum, squared sum to derive mean, variance
  - For histogram: store count to derive relative frequencies
- Incorporation of expert knowledge about the application in the prior $P(C_i)$

### Contra

- Limited applicability: often, required conditional probabilities are not available
- Lack of efficient computation: in case of a high number of attributes (particularly for Bayesian belief networks)
The Independence Hypothesis

The Independence Hypothesis . . .

- . . . makes efficient computation possible
- . . . yields optimal classifiers when satisfied
- . . . but is seldom satisfied in practice, as attributes (variables) are often correlated.

Attempts to overcome this limitation

- *Bayesian networks*, that combine Bayesian reasoning with causal relationships between attributes
- *Decision trees*, that reason on one attribute at the time, considering most important attributes first
1. Introduction
2. Basics
3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification

4.2 Regression

5. Advanced Topics
Linear Discriminant Function Classifier

Idea

Separate points of two classes by a hyperplane

▶ I.e., classification model is a hyperplane
▶ Points of one class in one half space, points of second class in the other half space

Questions

▶ How to formalize the classifier?
▶ How to find optimal parameters of the model?
Basic Notions

Recall some general algebraic notions for a vector space \( V \):

- \( \langle x, y \rangle \) denotes an inner product of two vectors \( x, y \in V \)
  - E.g., the scalar product \( \langle x, y \rangle = x^T y = \sum_{i=1}^{d} x_i y_i \)
  - \( H(w, w_0) \) denotes a hyperplane with normal vector \( w \) and constant term \( w_0 \):
    \[
    x \in H \iff \langle x, w \rangle + w_0 = 0
    \]
  - The normal vector \( w \) may be normalized to \( w' \):
    \[
    w' = \frac{1}{\sqrt{\langle w, w \rangle}} w \implies \langle w', w' \rangle = 1
    \]
  - Distance of a point \( x \) to the hyperplane \( H(w', w_0) \):
    \[
    dist(x, H(w', w_0)) = |\langle w', x \rangle + w_0|
    \]
Consider a two-class example (generalizations later on):

- $D$: $d$-dimensional vector space with attributes $a_1, \ldots, a_d$
- $Y = \{-1, 1\}$ set of 2 distinct class labels $y_j$
- $O \subseteq D$: Set of objects $o = (o_1, \ldots, o_d)$ with known class labels $y \in Y$ and cardinality $|O| = N$
- A hyperplane $H(w, w_0)$ with normal vector $w$ and constant term $w_0$

$$x \in H \iff w^T x + w_0 = 0$$

**Classification Rule**

$$K_{H(w,w_0)}(x) = \text{sign}(w^T x + w_0)$$
Optimal Parameter Estimation

How to estimate optimal parameters $w, w_0$?

1. Define an objective/loss function $L(\cdot)$ that assigns a value (e.g., the error on the training set) to each parameter-configuration
2. Optimal parameters minimize/maximize the objective function

How does an objective function look like?

- Different choices possible
- Most intuitive: Each misclassified object contributes a constant (loss) value

$\leadsto$ 0-1 loss
Optimal Parameter Estimation

0-1 Loss Objective for Linear Classifier

\[ L(w, w_0) = \sum_{i=1}^{N} I(y_i \neq K_{H(w, w_0)}(x_i)) \]

\[ \min_{w, w_0} L(w, w_0) \]

where \( I(condition) = 1 \) if condition holds, 0 otherwise

- Minimize the overall number of training errors, but . . .
  - NP-hard to optimize in general (non-smooth, non-convex)
  - Small changes of \( w, w_0 \) can lead to large changes of the loss
### Alternative Convex Loss Functions

<table>
<thead>
<tr>
<th>Loss Function</th>
<th>Formula</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum-of-squares loss</td>
<td>$(w^T x_i + w_0 - y_i)^2$</td>
<td>(SVM)</td>
</tr>
<tr>
<td>Hinge loss</td>
<td>$\max \left{ 0, (1 - y_i(w^T x_i + w_0)) \right}$</td>
<td>(AdaBoost)</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>$\exp(-y_i(w^T x_i + w_0))$</td>
<td></td>
</tr>
<tr>
<td>Cross-entropy error</td>
<td>$-y_i \log g(x_i) + (1 - y_i) \log(1 - g(x_i))$</td>
<td>(Logistic Regression)</td>
</tr>
<tr>
<td></td>
<td>where $g(x_i) = \frac{1}{1 + \exp(-(w^T x_i + w_0))}$</td>
<td></td>
</tr>
</tbody>
</table>

... and many more

- Optimizing different loss function leads to several classification algorithms
- Next, we derive the optimal parameters for the sum-of-squares loss
Optimal Parameters for SSE loss

Objective Function

\[
SSE(w, w_0) = \frac{1}{2} \sum_{i=1}^{N} (w^T x_i + w_0 - y_i)^2
\]

- Minimize the error function for getting optimal parameters
- Use standard optimization technique:
  1. Calculate first derivative
  2. Set derivative to zero and compute the global minimum (SSE is a convex function)
Optimal Parameters for SSE Loss

- Transform the problem for simpler computations:
  - \( w^T x + w_0 = \sum_{i=1}^{d} w_i x_i + w_0 = \sum_{i=0}^{d} w_i x_i \) with \( x_0 = 1 \)
  - For \( w \), let \( \tilde{w} = (w_0, \ldots, w_d)^T \)

- Combine the values to matrices

\[
\tilde{X} = \begin{pmatrix}
1 & x_{1,1} & \cdots & x_{1,d} \\
\vdots & \vdots & \ddots & \vdots \\
1 & x_{N,1} & \cdots & x_{N,d}
\end{pmatrix}, \quad Y = \begin{pmatrix}
y_1 \\
\vdots \\
y_N
\end{pmatrix}
\]

- Then the sum-of-squares error is equal to

\[
SSE(\tilde{w}) = \frac{1}{2}(\tilde{X}\tilde{w} - Y)^T(\tilde{X}\tilde{w} - Y)
\]
Optimal Parameters for SSE Loss

- Take the derivative:
  \[
  \frac{\partial}{\partial \tilde{w}} \text{SSW}(\tilde{w}) = \tilde{X}^T (\tilde{X} \tilde{w} - Y)
  \]

- Solve \(\frac{\partial}{\partial \tilde{w}} \text{SSE}(\tilde{w}) = 0\):
  \[
  \tilde{X}^T (\tilde{X} \tilde{w} - Y) = 0
  \Leftrightarrow \tilde{X}^T \tilde{X} \tilde{w} = \tilde{X}^T Y
  \Leftrightarrow \tilde{w} = \left(\tilde{X}^T \tilde{X}\right)^{-1} \tilde{X}^T Y
  \]
  Left-inverse of \(\tilde{X}\) ("Moore-Penrose Inverse")
Optimal Parameters for SSE Loss

- Set $\hat{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y$
- Classify new point $x$ with $x_0 = 1$ using

Classification Rule

$$K_{H(\hat{w})}(x) = \text{sign}(\hat{w}x)$$
Example SSE

Data (consider only age and max. speed):

\[
\tilde{X} = \begin{pmatrix}
1 & 23 & 180 \\
1 & 17 & 240 \\
1 & 43 & 246 \\
1 & 68 & 183 \\
1 & 32 & 110
\end{pmatrix}, \quad
\begin{pmatrix}
1 \\
1 \\
-1 \\
-1
\end{pmatrix}
\]

Encode classes as \{high = 1, low = -1\}

\[
(\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T = \begin{pmatrix}
0.7491 & -0.0836 & -0.8603 & -0.4736 & 1.6684 \\
-0.0087 & -0.0114 & 0.0049 & 0.0194 & -0.0043 \\
-0.0012 & 0.0036 & 0.0046 & -0.0002 & -0.0068
\end{pmatrix}
\]

\[
\Rightarrow \hat{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y = \begin{pmatrix}
w_0 \\
w_{age} \\
w_{maxspeed}
\end{pmatrix} = \begin{pmatrix}
-1.3896 \\
-0.0302 \\
0.0141
\end{pmatrix}
\]
Example SSE

- **Model parameters:**

  \[
  \hat{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y = \begin{pmatrix} w_0 \\ w_{age} \\ w_{maxspeed} \end{pmatrix} = \begin{pmatrix} -1.3896 \\ -0.0302 \\ 0.0141 \end{pmatrix}
  \]

  \[\implies K_H(\hat{w})(x) = \text{sign} \left( \begin{pmatrix} -0.0302 \\ 0.0141 \end{pmatrix}^T x - 1.3896 \right)\]

- **Query:** \(q = (age = 60; max\_speed = 190)\)

  \[\text{sign}(\hat{w}^T q) = \text{sign}(-0.5323) = -1\]

  \[\implies \text{class} = \text{low}\]
Extension to Multiple Classes

Assume we have more than two \((k > 2)\) classes. What to do?

- **One-vs-the-rest (“one-vs-all”)**
  \(\rightarrow k\) classifiers

- **One-vs-one (Majority vote of classifiers)**
  \(\Rightarrow \frac{k(k-1)}{2}\) classifiers

- **Multiclass linear classifier**
  \(\rightarrow k\) classifiers
Extension to Multiple Classes

**Idea of Multiclass Linear Classifier**

- Take $k$ linear functions of the form $H_{w_j, w_{j,0}}(x) = w_j^T x + w_{j,0}$
- Decide for class $y_j$:

\[
y_j = \arg\max_{j=1,\ldots,k} H_{w_j, w_{j,0}}(x)
\]

- Advantage: No ambiguous regions except for points on decision hyperplanes
- The optimal parameter estimation is also extendable to $k$ classes $y_1, \ldots, y_k$
## Discussion (SSE)

### Advantages

- Simple approach
- Closed form solution for parameters
- Easily extendable to non-linear spaces (later on)

### Disadvantages

- Sensitive to outliers ⇨ not a stable classifier
  - How to define and efficiently determine the maximum stable hyperplane?
- Only good results for linearly separable data
- Expensive computation of selected hyperplanes

~> Approach to solve problems: Support Vector Machines (SVMs) [Vapnik 1979, 1995]
Agenda

1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification
   4.2 Regression

5. Advanced Topics
Maximum Margin Hyperplane

Question

How to define the notion of the "best" hyperplane differently?

Criteria

- Stability at insertion
- Distance to the objects of both classes
Support Vector Machines: Principle

Basic Idea

Linear separation with the *Maximum Margin Hyperplane (MMH)*:

- Distance to points from any of the two sets is maximal, i.e., at least $\xi$
- Minimal probability that the separating hyperplane has to be moved due to an insertion

$\leadsto$ Best generalization behavior; MMH is “maximally stable”
Support Vector Machines: Principle

Support Vectors

MMH only depends on points \( p_i \) whose distance to the hyperplane is exactly \( \xi \). These \( p_i \) are called *support vectors* (SV).
Let \( x_i \in \mathbb{R}^d \) denote the data points, and \( y_i = +1 \), if first class, else \( y_i = -1 \).

A hyperplane in Hesse normal form is represented by a normal vector \( w \in \mathbb{R}^d \) of unit length (i.e., \( \|w\|_2 = 1 \)), and a (signed) distance from the origin \( b \in \mathbb{R} \).

In the following slides, we will define the requirements which the MMH shall fulfil.
Requirements of the MMH

The parameters \((w, b)\) of the MMH shall fulfil the following two requirements:

**No Error**

The classification is accurate for all points, i.e.

\[
y_i \cdot (\langle w, x_i \rangle + b) > 0 \iff \begin{cases} y_i = -1 & \langle w, x_i \rangle + b < 0 \\ y_i = +1 & \langle w, x_i \rangle + b > 0 \end{cases}
\]

**Requirement: Maximal Margin**

Let \(\xi = \min_{x_i \in TR} |\langle w, x_i \rangle + b|\) denote the minimum distance of any training object \(x_i\) to the hyperplane \(H(w, b)\). The margin \(\xi\) should be as large as possible.
Computation of the MMH

- **Task**: Maximise $\xi$ subject to $y_i \cdot (\langle w, x_i \rangle + b) > \xi$ for all $i \in \{1, \ldots, n\}$.
- Scaling the constraints by $\xi^{-1}$ yields $y_i \cdot (\langle \xi^{-1}w, x_i \rangle + \xi^{-1}b) > 1$ for all $i \in \{1, \ldots, n\}$.
- Define $w' = \xi^{-1}w$, and $b' = \xi^{-1}b$.
- Maximizing $\xi$ corresponds to minimizing $\langle w', w' \rangle = \frac{\langle w, w \rangle}{\xi^2}$.

**Primary Optimization Problem**

$$\begin{align*}
\text{min} & \quad \|w'\|_2^2 \\
\text{s.t.} & \quad y_i \cdot (\langle w', x_i \rangle + b') > 1 \quad i \in \{1, \ldots, n\}
\end{align*}$$
Primary Optimization Problem

\[
\begin{align*}
\text{min} & \quad \|w'\|^2_2 \\
\text{s.t.} & \quad y_i \cdot (\langle w', x_i \rangle + b') > 1 \quad i \in \{1, \ldots, n\}
\end{align*}
\]

- Convex optimization problem: Quadratic programming problem with linear constraints
  \[\implies\text{Solution can be obtained by Lagrangian Theory.}\]
Soft Margin Optimization

- Problem of MMH optimization: How to treat non-(linearly separable) data?

- Two typical problems:
  - Data points not linearly separable
  - Complete separation not optimal (overfitting)

- Trade-off between training error and size of margin
Soft Margin Optimization

- Additionally regard the number of training errors when optimizing:
  - $\xi_i$ is the distance from $x_i$ to the margin (often called slack variable):
    - $\xi_i = 0 \implies x_i$ on correct side
    - $\xi_i > 0 \implies x_i$ on wrong side

- Introduce parameter $C$ to weight the misclassification against the size of the margin.
Primary Optimization Problem With Soft Margin

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} \|w'\|_2^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.} & \quad y_i \cdot (\langle w', x_i \rangle + b') > 1 - \xi_i \\
& \quad \xi_i \geq 0
\end{align*}
\]
Soft Margin Optimization

Wolfe-Dual with Lagrange Multipliers

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\
\text{s.t.} & \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \\
& \quad 0 \leq \alpha_i \leq C \\
& \quad i \in \{1, \ldots, n\}
\end{align*}
\]

- \( \alpha_i = 0 \): \( x_i \) is not a support vector
- \( \alpha_i = C \): \( x_i \) is support vector with \( \xi_i > 0 \)
- \( 0 < \alpha_i < C \): \( x_i \) is support vector with \( \xi_i = 0 \)
Soft Margin SVM

**Decision Rule**

\[ H(x) = \text{sign} \left( \sum_{x_i \in SV} \alpha_i y_i \langle x_i, x \rangle + b \right) \]

where \( SV \) denotes the set of support vectors.
SVM: Discussion

Pro

▶ generate classifiers with a high classification accuracy
▶ relatively weak tendency to overfitting (generalization theory)
▶ efficient classification of new objects due to often small number of support vectors
▶ compact models

Contra

▶ training times may be long (appropriate feature space may be very high-dimensional)
▶ expensive implementation
Agenda

1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification
   4.2 Regression

5. Advanced Topics
Non-Linearly Separable Data Sets

Problem
For real data sets, a linear separation with a high classification accuracy often is not possible.

Idea
Transform the data non-linearly into a new space, and try to separate the data in the new space linearly (extension of the hypotheses space).

Example for quadratically separable data set
**Extension of the Hypotheses Space**

### Principle

<table>
<thead>
<tr>
<th>input space $\phi$</th>
<th>extented feature space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$</td>
<td></td>
</tr>
</tbody>
</table>

$\leadsto$ Try to linearly separate in the extended feature space.

### Example

$$\phi(x, y) = (1, x, y, x^2, xy, y^2)$$

Here: A hyperplane in the extended feature space is a polynomial of degree 2 in the input space.
Extension of the Hypotheses Space: Example (1)

Input Space (2 attributes):
\[ x = (x_1, x_2) \]
\[ x_2 = x_1^2 + 0.5 \]

Extended Space (6 attributes):
\[ \phi(x) = (1, x, y, x^2, xy, y^2) \]
\[ x_2 = (x_1^2) + 0.5 \]
Extension of the Hypotheses Space: Example (2)

Input Space (2 attributes):
\[ x = (x_1, x_2) \]

Extended Space (3 attributes):
\[ \phi(x) = (x_1^2, x_2^2, x_1x_2) \]

\[ x_1^2 + x_2^2 = 0.25 \]

\[ (x_2^2) = -(x_1^2) + 0.25 \]
Extension of Linear Discriminant Function Classifier

- Linear classifier can be easily extended to non-linear spaces
- Recap: linear classifier $K_{H(w, w_0)}(x) = sign(w^T x + w_0)$
- Extend to non-linear case:
  - Transform all data points $x$ to new feature space $\phi(x)$
  - Data Matrix $X$ becomes a matrix $\Phi$
  - The optimal hyperplane vector becomes ...

\[ \tilde{w}_{opt, \phi} = (\Phi^T \Phi)^{-1} \Phi^T Y \]

- ... and that’s all!
- New classification rule: $K_{H(w_\phi, w_0, \phi)}(x) = sign(w_\phi^T \phi(x) + w_{0, \phi})$
- SVM can be extended in a similar way
Non-linear Classification: Discussion

**Pro**
- By explicit feature transformation a much richer hypotheses space
- Simple extension of existing techniques
- Efficient evaluation, if transformed feature space not too high-dimensional

**Contra**
- Explicit mapping to other feature spaces can become problematic
- Meaningful transformation is usually not known a-priori
- Complex data distributions may require very high-dimensional features spaces → High memory consumption, High computational costs
Explicit Mapping

Explicit mapping of the data into the new feature space:

- After transformation, any vector-based distance is applied
- Resulting feature space may be very high dimensional ⇒ Potential problems: Inefficient calculation, storage overhead

Often, we do not need the transformed data points themselves, but just the distances between them!
"Kernel Trick"

Just implicitly map the data to a feature space: Determine a function $K_\phi$, which computes the distance in the kernel space without explicitly computing $\phi(\cdot)$

$$K_\phi(x, y) = \langle \phi(x), \phi(y) \rangle$$
Kernel: Example

- Assume the original domain is $\mathcal{X} = \mathbb{R}^2$
- We transform a point $x = (x_1, x_2)$ to $\phi(x) = (x_1^2, x_2^2, x_1x_2)$, i.e. the novel feature space is $\mathcal{H} = \mathbb{R}^3$, and $\kappa : \mathcal{X} \to \mathcal{H}$.

**Input Space (2 attributes):**

$$x = (x_1, x_2)$$

**Extended Space (3 attributes):**

$$\phi(x) = (x_1^2, x_2^2, x_1x_2)$$

\[x_1^2 + x_2^2 = 0.25\]
Kernel: Example

- Original point $x = (x_1, x_2)$; transformed point $\phi(x) = (x_1^2, x_2^2, \sqrt{2} \cdot x_1 x_2)$
- We want to calculate the dot product in the novel feature space $\mathcal{H}$:

$$\langle \phi(x), \phi(y) \rangle = \left\langle \left( x_1^2, x_2^2, \sqrt{2} \cdot x_1 x_2 \right), \left( y_1^2, y_2^2, \sqrt{2} \cdot y_1 y_2 \right) \right\rangle$$

$$= x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2$$

$$= (x_1 y_1 + x_2 y_2)^2$$

$$= \langle x, y \rangle^2$$

- We do not have to explicitly map the points to the feature space $\mathcal{H}$!
- Simply calculate squared dot product in the original domain $\mathcal{X}$!
- $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, (x, y) \mapsto \langle x, y \rangle^2$ is called a (valid) kernel
Why is the dot product useful?

- Kernels correspond to dot products in some feature space
- With the dot product we are able to compute:
  - The norm/length of a vector $\|x\| = \sqrt{\langle x, x \rangle}$
  - The distance between two vectors:
    $$\|x - y\|^2 = \langle x - y, x - y \rangle = \langle x, x \rangle + \langle y, y \rangle - 2 \langle x, y \rangle$$
  - The angle between two vectors:
    $$\angle(x, y) = \arccos \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|}$$
Formal Definitions

Definition: Kernel Function

A kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function, i.e., $\kappa(x, y) = \kappa(y, x)$, mapping pairs of objects $x, y \in \mathcal{X}$ to real numbers.

Definition: Mercer Kernel

For all finite $\{x_1, \ldots, x_n\} = X \subseteq \mathcal{X}$, let $\kappa(X) := (\kappa(x_i, x_j))_{i,j} \in \mathbb{R}^{n \times n}$. A kernel function $\kappa$ is called Mercer kernel, valid kernel, admissible kernel, or positive semi-definite, if for all such finite $X$, the matrix $\kappa(X)$ is positive semi-definite, i.e. for all $c \in \mathbb{R}^n$, it holds

$$c^T \kappa(X) c \geq 0$$
Definition: Dot Product

A dot product in a vector space $\mathcal{H}$ is a function $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ satisfying:

- $\langle x, x \rangle = 0$ for $x = 0$
- $\langle x, x \rangle > 0$ for $x \neq 0$
- $\langle x, y \rangle = \langle y, x \rangle$ (Symmetry)
- $\langle \alpha x + \beta y, z \rangle = \alpha \langle x, z \rangle + \beta \langle y, z \rangle$ (Bi-linearity)

Definition: Hilbert Space

A vector space $\mathcal{H}$ endowed with a dot product $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ for which the induced norm gives a complete metric space, is termed *Hilbert Space*.
Interpretation of Kernel Functions

Theorem

Let $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a valid kernel on $\mathcal{X}$. There exists a possibly infinite-dimensional Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$ such that $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$ for all $x, y \in \mathcal{X}$ where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the dot product in a Hilbert space $\mathcal{H}$.

$\Rightarrow$ every kernel $\kappa$ can be seen as a dot product in some feature space $\mathcal{H}$.

Advantages

- Feature space $\mathcal{H}$ can be infinite-dimensional
- Not really necessary to know which feature space $\mathcal{H}$ we have
- Computation of kernel is done in original domain $\mathcal{X}$
Wolfe-Dual Optimization Problem with Lagrange Multipliers

\[
\begin{align*}
\max & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \\
\text{s.t.} & \sum_{i=1}^{n} \alpha_i y_i = 0 \\
& 0 \leq \alpha_i \leq C \quad i \in \{1, \ldots, n\}
\end{align*}
\]

Decision Rule

\[
H(x) = \sign \left( \sum_{x_i \in SV} \alpha_i y_i \kappa(x_i, x) + b \right)
\]
Example for Mercer Kernels

Radial Basis Kernel
\[ \kappa(x, y) = \exp(-\gamma \|x - y\|^2) \]

Polynomial Kernel (degree 2)
\[ \kappa(x, y) = (\langle x, y \rangle + 1)^d \]
Discussion

**Pro**

- Kernel methods provide a simple method for dealing with non-linearity
- Implicit mapping allows for mapping to arbitrary-dimensional spaces: Computational effort depends on the number of training examples, but not on the feature space dimensionality

**Contra**

- Resulting models rarely provide an intuition
- Choice of kernel can be difficult
1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification

4.2 Regression

5. Advanced Topics
Decision Tree Classifiers

- Approximating discrete-valued target function
- Learned function is represented as a tree:
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution

- Tree can be transformed into decision rules:
  - if age > 60 then risk = low
  - if age ≤ 60 and car type = truck then risk = low
  - if age ≤ 60 and car type ≠ truck then risk = high

### Advantages

- Decision trees represent explicit knowledge
- Decision trees are intuitive to most users

<table>
<thead>
<tr>
<th>age</th>
<th>car_type</th>
<th>max_speed</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>family</td>
<td>180</td>
<td>high</td>
</tr>
<tr>
<td>17</td>
<td>sportive</td>
<td>240</td>
<td>high</td>
</tr>
<tr>
<td>43</td>
<td>sportive</td>
<td>246</td>
<td>high</td>
</tr>
<tr>
<td>68</td>
<td>family</td>
<td>183</td>
<td>low</td>
</tr>
<tr>
<td>32</td>
<td>truck</td>
<td>110</td>
<td>low</td>
</tr>
</tbody>
</table>
Decision Tree Classifier: Splits

Goal

- Each tree node defines an axis-parallel \((d - 1)\)-dimensional hyperplane, that splits the data space.
- *Find such splits which lead to as homogenous groups as possible.*
Decision Tree Classifiers: Basics

- Decision tree generation (training phase) consists of two phases
  1. Tree construction
     - At start, all the training examples are at the root
     - Partition examples recursively based on selected attributes
  2. Tree pruning
     - Identify and remove branches that reflect noise or outliers

- Use of decision tree: Classifying an unknown sample
  - Traverse the tree and test the attribute values of the sample against the decision tree
  - Assign the class label of the respective leaf to the query object
Algorithm for Decision Tree Construction

▶ Basic algorithm (a greedy algorithm)
  ▶ Tree is created in a *top-down recursive divide-and-conquer* manner
  ▶ Attributes may be categorical or continuous-valued
  ▶ At the start, all the training examples are assigned to the root node
  ▶ Recursively partition examples at each node and push them down to the new nodes
  ▶ Select test attributes and determine split points or split sets for the respective values based on a heuristic or statistical measure (*split strategy*, e.g., information gain)

▶ Conditions for stopping partitioning
  ▶ All samples for a given node belong to the same class
  ▶ There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  ▶ There are no samples left
Algorithm for Decision Tree Construction

- Most algorithms are versions of this basic algorithm (greedy, top-down)
  - E.g.: ID3, or its successor C4.5

### ID3 Algorithm

```java
procedure ID3(Examples, TargetAttr, Attributes) ⊳ specialized to learn boolean-valued functions

Create Root node for the tree

if all Examples are positive then return Root with label = +
else if all Examples are negative then return Root with label = −
else if Attributes = ∅ then return Root with label = most common value of TargetAttr in Examples
else

A = "best" decision attribute for next node ⊳ how to determine the "best" attribute?
Assign A as decision attribute for Root
for each possible value vi of A do ⊳ how to split the possible values?

Generate branch corresponding to test A = vi
Examplesvi = examples that have value vi for A
if Examplesvi = ∅ then
   Add leaf node with label = most common value of TargetAttr in Examples
else
   Add subtree ID3(Examplesvi, TargetAttr, Attributes \ {A})
```

Supervised Methods Classification
Example: Decision for "playing_tennis"

- Query: How about playing tennis today?
- Training data:

<table>
<thead>
<tr>
<th>day</th>
<th>forecast</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>tennis decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- Build decision tree ...
Split Strategies: Quality of Splits

**Given**

- A set $T$ of training objects
- A (disjoint, complete) partitioning $T_1, \ldots, T_m$ of $T$
- The relative frequencies $p_i$ of class $c_i$ in $T$ and in the partitions $T_1, \ldots, T_m$

**Wanted**

- A measure for the heterogeneity of a set $S$ of training objects with respect to the class membership
- A split of $T$ into partitions $\{T_1, \ldots, T_m\}$ such that the heterogeneity is minimized

~ Proposals: *Information gain, Gini index, Misclassification error*
Attribute Selection Measures: Information Gain

- Used in ID3/C4.5

_entropy_ $\text{Entropy}$

- Minimum number of bits to encode a message that contains the class label of a random training object
- The entropy of a set $T$ of training objects is defined as

$$
\text{entropy}(T) = -\sum_{i=1}^{k} p_i \log_2 p_i
$$

for $k$ classes with frequencies $p_i$
- $\text{entropy}(T) = 0$ if $p_i = 1$ for any class $c_i$
- $\text{entropy}(T) = 1$ if $p_i = \frac{1}{k}$ for all classes $c_i$
Let $A$ be the attribute that induced the partitioning $\{T_1, \ldots, T_m\}$ of $T$. The information gain of attribute $A$ w.r.t. $T$ is defined as

$$\text{information gain}(T, A) = \text{entropy}(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \text{entropy}(T_i)$$
Attribute Selection: Example (Information Gain)

\[
\text{information gain}(T, \text{forecast}) = 0.94 - \frac{5}{14} \times 0.971 - \frac{4}{14} \times 0 - \frac{5}{14} \times 0.971 = 0.246
\]

\[
\text{information gain}(T, \text{temperature}) = 0.94 - \frac{4}{14} \times 0.811 - \frac{6}{14} \times 0.981 - \frac{4}{14} \times 1 = 0.029
\]

\[
\text{information gain}(T, \text{humidity}) = 0.94 - \frac{7}{14} \times 0.985 - \frac{7}{14} \times 0.592 = 0.151
\]

\[
\text{information gain}(T, \text{wind}) = 0.94 - \frac{8}{14} \times 0.811 - \frac{6}{14} \times 1 = 0.048
\]

Result: "forecast" yields the highest information gain
Example: Decision Tree for "playing_tennis"

<table>
<thead>
<tr>
<th>day</th>
<th>forecast</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>tennis decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

Final decision tree:
Attribute Selection Measures: Gini Index

- Used in IBM’s IntelligentMiner

### Gini Index

The Gini index for a set \( T \) of training objects is defined as

\[
gini(T) = 1 - \sum_{i=1}^{k} p_i^2
\]

- Small value of Gini index \( \equiv \) low heterogeneity
- Large value of Gini index \( \equiv \) high heterogeneity

### Gini Index (of an attribute \( A \))

Let \( A \) be the attribute that induced the partitioning \( \{T_1, \ldots, T_m\} \) of \( T \). The Gini index of attribute \( A \) w.r.t. \( T \) is defined as

\[
gini_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} gini(T_i)
\]
Attribute Selection Measures: Misclassification Error

### Misclassification Error

The Misclassification Error for a set \( T \) of training objects is defined as

\[
Error(T) = 1 - \max_{c_i} p_i
\]

- Small value of Error \(\equiv\) low heterogeneity
- Large value of Error \(\equiv\) high heterogeneity

### Misclassification Error (of an attribute \( A \))

Let \( A \) be the attribute that induced the partitioning \( \{ T_1, \ldots, T_m \} \) of \( T \). The Misclassification Error of attribute \( A \) w.r.t. \( T \) is defined as

\[
Error_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} Error(T_i)
\]
Attribute Selection Measures: Comparison

For two-class problems:

![Graph showing Entropy, Gini, and Misclassification error for two-class problems.](diagram.png)
Split Strategies: Types of Splits

- Categorical attributes
  - Split criteria based on equality "attribute = a"
  - Based on subset relationships "attribute ∈ set"
    - Many possible choices (subsets)
      - Choose the best split according to, e.g., gini index

- Numerical attributes
  - Split criteria of the form "attribute < a"
    - Many possible choices for the split point
      - One approach: Order test samples w.r.t. their attribute value; consider every mean value between two adjacent samples as possible split point; choose best one according to, e.g., gini index
  - Partition the attribute value into a discrete set of intervals (Binning)
Avoid Overfitting in Classification

▶ The generated tree may overfit the training data
  ▶ Too many branches, some may reflect anomalies due to noise or outliers
  ▶ Result has poor accuracy for unseen samples

▶ Two approaches to avoid overfitting for decision trees:
  1. Post-pruning = pruning of overspecialized branches
  2. Pre-pruning = halt tree construction early
Avoid Overfitting in Classification

Post-pruning

Pruning of overspecialized branches:

- Remove branches from a "fully grown" tree and get a sequence of progressively pruned trees
- Use a set of data different from the training data to decide which is the “best pruned tree”
Avoid Overfitting in Classification

Pre-pruning

Halt tree construction early, do not split a node if this would result in the goodness measure falling below a threshold.

▶ Choice of an appropriate value for \textit{minimum support}
  ▶ Minimum support: minimum number of data objects a leaf node contains
  ▶ In general, \textit{minimum support} $\gg 1$

▶ Choice of an appropriate value for \textit{minimum confidence}
  ▶ Minimum confidence: minimum fraction of the majority class in a leaf node
  ▶ Typically, \textit{minimum confidence} $\ll 100\%$
  ▶ Leaf nodes can absorb errors or noise in data records

▶ Discussion
  ▶ With Pre-pruning it is difficult to choose appropriate thresholds
  ▶ Pre-pruning has less information for the pruning decision than post-pruning $\rightsquigarrow$ can be expected to produce decision trees with lower classification quality
  ▶ Tradeoff: tree construction time vs. classification quality
Reduced-Error Pruning

1. Decompose classified data into training set and test set
2. Create a decision tree $E$ for the training set
3. Prune $E$ using the test set $T$
   - Determine the interior node $v$ of $E$ whose pruning reduces the number of misclassified data points on $T$ the most (i.e., replace the subtree $S$ with root $v$ by a leaf. Determine the value of the leaf by majority voting)
   - Prune
   - Finish if no such interior node exists
4. Only applicable if a sufficient number of classified data is available

---

Pruning of Decision Trees: Approach Post-pruning

Minimal Cost Complexity Pruning \(^1\)

- Does not require a separate test set
  - Applicable to small training sets as well
- Pruning of the decision tree by using the training set
  - Classification error is no appropriate quality measure
- New quality measure for decision trees:
  - Trade-off of classification error and tree size
  - Weighted sum of classification error and tree size
- General observation
  - The smaller decision trees yield the better generalization

Minimal Cost Complexity Pruning: Notions

- Size $|E|$ of a decision tree $E$: number of leaf nodes
- Cost-complexity quality measure of $E$ with respect to training set $T$, classification error $F_T$ and complexity parameter $\alpha \geq 0$:

$$CC_T(E, \alpha) = F_T(E) + \alpha|E|$$

- For the smallest minimal subtree $E(\alpha)$ of $E$ w.r.t. $\alpha$, it is true that:
  1. There is no subtree of $E$ with a smaller cost complexity
  2. If $E(\alpha)$ and $B$ both fulfill (1), then is $E(\alpha)$ a subtree of $B$

- $\alpha = 0$: $E(\alpha) = E$
  - Only error matters

- $\alpha \rightarrow \infty$: $E(\alpha) = \text{root node of } E$
  - Only tree size matters

- $0 < \alpha < \infty$: $E(\alpha)$ is a proper substructure of $E$
  - The root node or more than the root node
Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand

Example

```plaintext
if forecast = "overcast" then playing_tennis = "yes"
if forecast = "sunny" and humidity = "high" then playing_tennis = "no"
if forecast = "sunny" and humidity = "normal" then playing_tennis = "yes"
if forecast = "rainy" and wind = "strong" then playing_tennis = "no"
if forecast = "rainy" and wind = "weak" then playing_tennis = "yes"
```
Enhancement: Handle Missing Attribute Values

- If node \( n \) tests attribute \( A \):
  - Assign most common value of \( A \) among other examples sorted to node \( n \)
  - Assign the most common value of the attribute among other examples with the same target value sorted to node \( n \)
  - Assign probability \( p_i \) to each of the possible values \( v_i \) of attribute \( A \) among other examples sorted to node \( n \)
    - Assign fraction \( p_i \) of example to each descendant in tree
    - Classify new examples in the same fashion: Classification decision is the one with the highest probability (sum over all instance fragments of each class decision)

Supervised Methods
Classification
January 25, 2019
Decision Tree Classifiers: Summary

**Pro**

- Relatively fast learning speed (in comparison to other classification methods)
- Fast classification speed
- Convertible to simple and easy to understand classification rules
- Often comparable classification accuracy with other classification methods

**Contra**

- Not very stable, small changes of the data can lead to large changes of the tree
Agenda

1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification

4.2 Regression

5. Advanced Topics
Nearest Neighbor Classifiers

Motivation

▶ Assume data in a non-vector representation: graphs, forms, XML-files, etc.
▶ No simple way to use linear classifiers or decision trees

Solutions

▶ Use appropriate kernel function for kernel machines (e.g. kernel SVM)
  ↞ Not always clear how to define a kernel
▶ Transformation of objects to some vector space (e.g. representation learning)
  ↞ Difficult to determine appropriate transformation & vector space
▶ Here: Nearest neighbor classifier
  ↞ Direct usage of the similarity of objects for classification
Nearest Neighbor Classifiers

**Procedure**

Assign query object $q$ to the class $c_j$ of the closest training object $x \in D$:

$$class(q) = class(NN(q)) \quad NN(q) = \{x \in D \mid \forall x' \in D : d(q, x) \leq d(q, x')\}$$

$\Rightarrow$ Instance-Based Learning

**Example**

```
cat  cat  q  dog  dog  wolf
  cat   cat
```

Classifier decides that query object $q$ is a dog.
Instance-Based Methods

Eager Evaluation

- Create models from data (training phase) and then use these models for classification (test phase)
- Examples: Decision tree, Bayes classifier

Instance-Based Learning

- Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical Approaches: $k$-nearest neighbor approach:
  - Instances represented as points in a metric space (e.g. Euclidean)
  - Classification by label of NN requires fast NN queries
  - Training phase = Index construction (e.g. R-tree)
Nearest Neighbor Classifiers: Notions

Notions

- Distance Function: Defines the (dis-)similarity for pairs of objects
- Decision Set: The set of $k$ nearest neighboring objects to be used in the decision rule

Decision Rule

Given the class labels of the objects from the decision set, how to determine the class label to be assigned to the query object?
Decision Rules

**Majority Vote (Default)**

Choose majority class in the decision set, i.e. the class with the most representatives in the decision set.

**Weighted Decision Rules**

Choose *weighted* majority of decision set class labels. Weight variants:

- *Reciprocal squared distance*: $d(q, x)^{-2}$
- *Inverse A-Priori Probability*: Use inverse frequency of classes in the training set
Example: $k = 5$ – Influence of Weighting

Majority Vote

Reciprocal Squared Distance
Example: Majority Vote – Influence of $k$

$k = 1$

$k = 5$
Choosing an appropriate $k$: Tradeoff between *overfitting* and *generalization*:

**Influence of $k$**
- $k$ too small: High sensitivity against outliers
- $k$ too large: Decision set contains many objects from other classes

**Rules of Thumb**
- Based on theoretical considerations: Choose $k$, such that it grows slowly with $n$, e.g. $k \approx \sqrt{n}$ or $k \approx \log n$
- Empirically, $1 \ll k < 10$ yields a high classification accuracy in many cases
NN Classifier: Variants

- **$k$-NN Classifier**: Consider the $k$ nearest neighbors for the class assignment decision.
- **Weighted $k$-NN Classifier**: Use weights for the classes of the $k$ nearest neighbors.
- **Mean-based NN Classifier**: Determine mean vector $m_i$ for each class $c_j$ (in training phase); Assign query object to the class $c_j$ of the nearest mean vector $m_i$.
- **Generalization**: Representative-based NN Classifier; Use more than one representative per class.
NN Classifier: Discussion

**Pro**

- **Applicability**: Training data and distance function required only
- High classification *accuracy* in many applications
- Easy *incremental* adaptation to new training objects useful also for prediction
- *Robust* to noisy data by averaging $k$-nearest neighbors

**Contra**

- Naïve implementation is inefficient: Requires $k$-nearest neighbor query processing $\leadsto$ support by database techniques may help to reduce from $O(n)$ to $O(\log n)$
- Does not produce explicit knowledge about classes, *but* provides some explanation information
- Curse of dimensionality: Distance between neighbors could be dominated by irrelevant attributes $\leadsto$ Apply dimensionality reduction first
Agenda

1. Introduction

2. Basics

3. Unsupervised Methods

4. Supervised Methods
   4.1 Classification
      4.1.1 Bayesian Classifiers
      4.1.2 Linear Discriminant Functions
      4.1.3 Support Vector Machines
      4.1.4 Kernel Methods
      4.1.5 Decision Tree Classifiers
      4.1.6 Nearest Neighbor Classifiers
      4.1.7 Ensemble Classification
   4.2 Regression

5. Advanced Topics
Problem

▶ No single classifier performs good on every problem
▶ For some techniques, small changes in the training set lead to very different classifiers

Idea

Improve performance by combining different classifiers \( \leadsto \) ensemble classification. Different possibilities exist. Discussed here:

▶ Bagging (Bootstrap aggregation)
▶ Boosting
Bagging

How to obtain different classifiers?
Easiest way: Train the same classifier $K$ on different datasets

Bagging (or Bootstrap Aggregation)

- Randomly select $m$ different subsets from the training set
- On each subset, independently train a classifier $K_i$ ($i = 1, \ldots, m$)
- Overall decision:

$$K(x) = \text{sign} \left( \frac{1}{m} \sum_{i=1}^{m} K_i(x) \right)$$
Linear combination of several weak learners (different classifiers)

Given $m$ weak learners $K_i$ and weights $\alpha_i$ for $i = 1, \ldots, m$

Overall decision

$$K(x) = \text{sign} \left( \sum_{i=1}^{m} \alpha_i K_i(x) \right)$$

Important difference: classifiers are trained in sequence!

Repeatedly misclassified points are weighted stronger
AdaBoost

Widely used boosting method: AdaBoost \(^{21}\): Meta-algorithm that iteratively generates a chain of weak learners

**General Idea**

- Assume \((t-1)\) weak learners are already given. The \(t\)th learner should focus on instances that were previously misclassified.
- Assign a weight \(w_i\) to each instance \(x_i\) to represent its importance
- Start with equal weight for each instance, adapt weights according to the performance of previously trained classifiers

AdaBoost – Algorithm

Given: \( n \) data points \( x_1, \ldots, x_n \), labels \( y_1, \ldots, y_n \)

Initialize \( w_1 = \ldots = w_n = \frac{1}{n} \)

for \( i = 1, \ldots, m \) do

Fit a classifier \( K_i(x) \) to the training data by minimizing weighted error function

\[
J_i = \sum_{j=1}^{n} w_j \mathbb{I}(K_i(x_j) \neq y_j)
\]

where \( \mathbb{I} \) is the indicator function

Compute weighting coefficient

\[
\alpha_i = \ln \left( \frac{1 - \epsilon_i}{\epsilon_i} \right) \quad \text{where} \quad \epsilon_i = \frac{J_i}{\sum_{j=1}^{n} w_j}
\]

Update all data weights:

\[
w_j := w_j \exp \left( \alpha_i \mathbb{I}(K_i(x_j) \neq y_j) \right)
\]

Final Model

\[
K(x) = \text{sign} \left( \sum_{i=1}^{m} \alpha_i K_i(x) \right)
\]
## Classification: Summary

<table>
<thead>
<tr>
<th>Model</th>
<th>Compactness</th>
<th>Model Interpretability</th>
<th>Decision Interpretability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Model</td>
<td>hyperplane compact (# dims) medium/low</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>hyperplane/non-linear(kernel) compact (# SV) medium/low</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>Decision Tree</td>
<td>set of (axis-parallel) hyperplanes compact (pruned) good</td>
<td>good (rules)</td>
<td></td>
</tr>
<tr>
<td>(k)NN</td>
<td>no model no model no model</td>
<td>medium/good (example)</td>
<td></td>
</tr>
<tr>
<td>Bayes</td>
<td>statistical distribution density model dependent model dependent</td>
<td>medium/good (probabilities)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Robustness</th>
<th>Training Time</th>
<th>Test Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Model</td>
<td>arbitrary (kernel) low high</td>
<td>low/high (for high-dim)</td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>arbitrary (kernel) high medium</td>
<td>low/medium</td>
<td></td>
</tr>
<tr>
<td>Decision Tree</td>
<td>categorical &amp; vector low low/medium</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>(k)NN</td>
<td>arbitrary (distance) high no training</td>
<td>low (index)/ very high</td>
<td></td>
</tr>
<tr>
<td>Bayes</td>
<td>arbitrary (probability distribution) high model-dependent</td>
<td>model-dependent; often low</td>
<td></td>
</tr>
</tbody>
</table>
Classification: Conclusion

- Classification is an extensively studied problem (mainly in statistics and machine learning)
- Classification is probably one of the most widely used data mining techniques with a lot of extensions
- Scalability is an important issue for database applications: thus combining classification with database techniques should be a promising topic
- Research directions: classification of complex data, e.g., text, spatial, multimedia, etc.;
  
  *Example:* $k$NN-classifiers rely on distances but do not require vector representations of data
- Results can be improved by ensemble classification