Introduction to Artificial Intelligence

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 400

Subsection 1

Learning, Learning Agents and Data Mining

Introduction

- From the dictum

  > Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

  and, given that computers' learning ability is inferior to that of humans, it follows that the study of learning mechanisms and of machine learning algorithms is a very important branch of AI.
- There is also demand for machine learning in software development, e.g., programming the behavior of an autonomous robot.
- Machine learning algorithms are used to program robots, often by combining programmed and learned behavior.
- We study the most important machine learning algorithms, but, for now, restrict to supervised and unsupervised learning algorithms.
- As an important class of learning algorithms, neural networks will be studied later. Another type, reinforcement learning, especially important for autonomous robots, will also be studied later.

What Is Learning? Generalization

- Memorization is uninteresting for AI since it amounts to little more than saving text in a file.
- In contrast, the acquisition of mathematical skills is usually not done by memorization.
- How do we learn mathematics?
  Answer: The teacher explains the process and the students practice it on examples until they no longer make mistakes on new examples.
- After 50 examples the student (hopefully) understands addition well enough that (s)he can apply what was learned to infinitely many new, previously unseen, examples.
- This process is known as **generalization**.

Example: Classifying Apples

- A fruit farmer wants to automatically divide harvested apples into the merchandize classes A and B. The sorting device is equipped with sensors to measure two features, size and color, and then decide which of the two classes the apple belongs to.
- Systems which are capable of dividing feature vectors into a finite number of classes are called **classifiers**.
- To configure the machine, apples are hand-picked by a specialist, that is, they are classified:

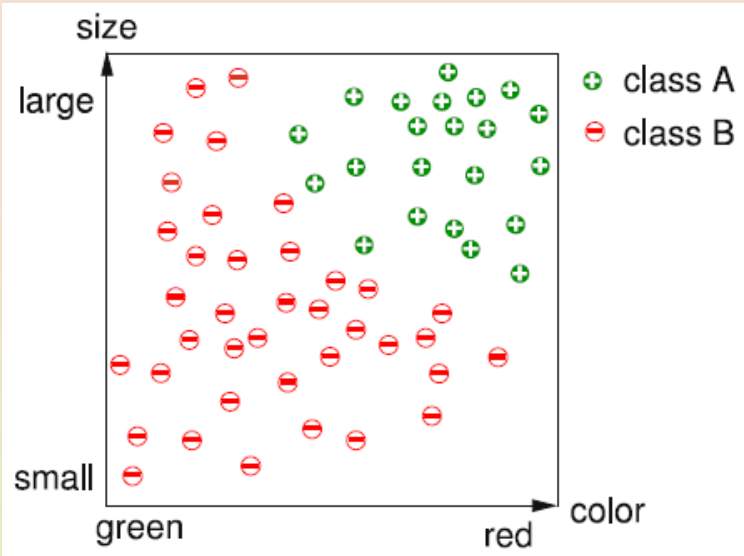
Size (cm)	8	8	6	3	$\cdots$
Color	0.1	0.3	0.9	0.8	$\cdots$
Merchandize Class	B	A	A	B	$\cdots$

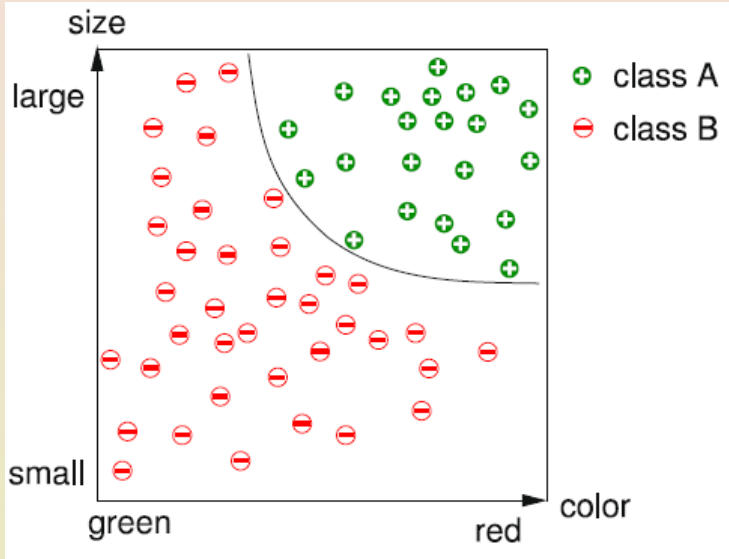
The size is given in the form of diameter in centimeters and the color by a numeric value between 0 (for green) and 1 (for red).

- The task in machine learning consists of generating a function from the collected, classified data which calculates the class value (A or B) for a new apple from the two features size and color.

Example: Classifying Apples (Cont'd)

- Given the plotted data on the left





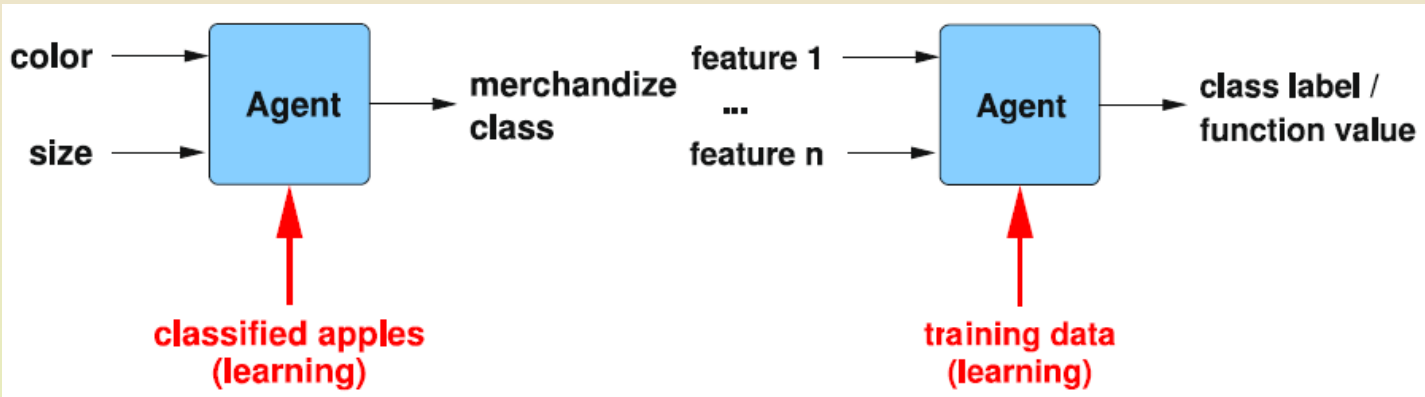
  a function is shown by the dividing line drawn through the diagram on the right.
- All apples with a feature vector to the bottom left of the line are put into class B, and all others into class A.
- Classification is a more difficult and much less visualizable task when the objects are described by not just two, but by many features.
- In practice 30 or more features are usually used.

Classifiers Using Many Features

- For n features, the task consists of finding an $n-1$ dimensional hyperplane within the n-dimensional feature space which divides the classes as well as possible.
- A "good" division means that the percentage of falsely classified objects is as small as possible.
- A **classifier** maps a feature vector to a class value.
- Here it has a fixed, usually small, number of alternatives.
- The desired mapping is also called **target function**.
- If the target function does not map onto a finite domain, then it is not a classification, but rather an approximation problem.
  - Determining the market value of a stock from given features is such an approximation problem.

The Learning Agent

- A **learning agent** is a function which maps a feature vector to a discrete class value or, in general, to a real number.
- This function is not preprogrammed; it is created and/or changes itself during the learning phase, influenced by the training data.



- During learning, the agent is fed with the already classified data.
- The agent constructs as good a mapping as possible from the feature vector to the function value (e.g. merchandise class).

Learning Agent

- Tom Mitchell gives the definition of "machine learning":

> **Machine Learning** is the study of computer algorithms that improve automatically through experience.

- Modifying, we define:

Definition of a Learning Agent

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

- It is important to test the generalization capability of the learning algorithm on unknown data, the **test data**.
- Otherwise, every system that just saved the training data would appear to perform optimally just by calling up the saved data.

Features of a Learning Agent

- A learning agent is characterized by the following terms:
  - **Task**: the task of the learning algorithm is to learn a mapping.
  - **Variable agent** (more precisely a class of agents): the choice of learning algorithm(s) determines the class of all learnable functions.
  - **Training data** (experience): the training data (sample) contain the knowledge which the learning algorithm is supposed to extract and learn. With the choice of training data one must ensure that it is a representative sample for the task to be learned.
  - **Test data**: important for evaluating whether the trained agent can generalize well from the training data to new data.
  - **Performance measure**: for the apple sorting device, the number of correctly classified apples.
- Knowing the performance measure is usually much easier than knowing the agent's function.

Data Mining

- **Data mining** is the task of a learning machine to extract knowledge from training data.
- It is desirable for the learning machine to also make the extracted knowledge readable for humans.
- Induction of decision trees constitute an example of this type of method.
- In electronic business, in the areas of advertisement and marketing, whenever large amounts of data are available, one can attempt to use these data for the analysis of customer preferences in order to show customer tailored advertisements.
- Preference learning is dedicated to this purpose.

Data Mining Tasks

- The process of acquiring and representing knowledge from data, as well as their application, is called **data mining**.
  - The methods used are taken from statistics or machine learning and should be applicable to very large amounts of data at reasonable cost.
  - In the context of acquiring information, for example on the Internet or in an intranet, text mining plays an increasingly important role. Typical tasks include finding similar text in a search engine or the classification of texts, which for example is applied in spam filters for email. The naive Bayes algorithm for the classification of text is popular.
  - A relatively new challenge for data mining is the extraction of structural, static, and dynamic information from graph structures such as social networks, traffic networks, or Internet traffic.
- Because the two described tasks of machine learning and data mining are formally very similar, the *basic methods used in both areas are for the most part identical*.

Subsection 2

Data Analysis

Statistics

- Statistics helps to describe data with simple parameters.
- From these we choose a few which are especially important for the analysis of training data and test these on a subset of the data.
- For each variable x_i, its **average** $\bar{x}_i$ is defined as $\bar{x}_i := \frac{1}{N}\sum_{p=1}^{N} x_i^p$.
- The **standard deviation** s_i is a measure of its average deviation from the average value: $s_i := \sqrt{\frac{1}{N-1}\sum_{p=1}^{N}(x_i^p - \bar{x}_i)^2}$.
- Whether two variables x_i and x_j are statistically dependent (correlated) is important for the analysis of multidimensional data.
- The **covariance** $\sigma_{ij} = \frac{1}{N-1}\sum_{p=1}^{N}(x_i^p - \bar{x}_i)(x_j^p - \bar{x}_j)$ gives information about this.
- In this sum, a summand for the p-th data vector is positive exactly when both deviations of the i-th and j-th components from the average have the same sign. It is negative if they have different signs.

The Covariance and the Correlation Coefficient

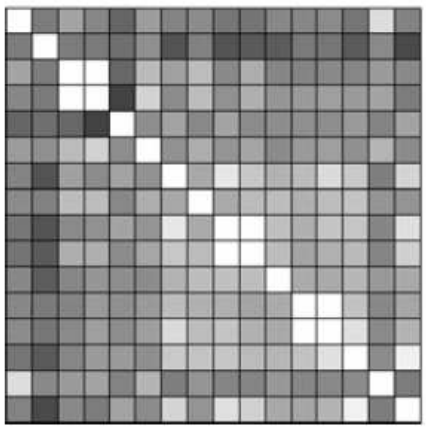
- One aspect influencing covariance is the sign of the product of deviations of the i-th and j-th components from the average.
- The covariance also depends on the absolute value of the variables, which makes comparison of the values difficult.
- To compare the degree of dependence in the case of multiple variables, we define the **correlation coefficient** $K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}$ for two values x_i and x_j, which is nothing but a normalized covariance.
- The **matrix K of all correlation coefficients** (i) contains values between -1 and 1, (ii) is symmetric, and (iii) all of its diagonal elements are 1.
- The matrix K becomes somewhat more readable when we represent it as a **density plot**.
  - Instead of the numerical values, the matrix is filled with gray values.
  - It conveys quickly which variables display a weak or strong dependence.

Correlation Coefficients: Illustration

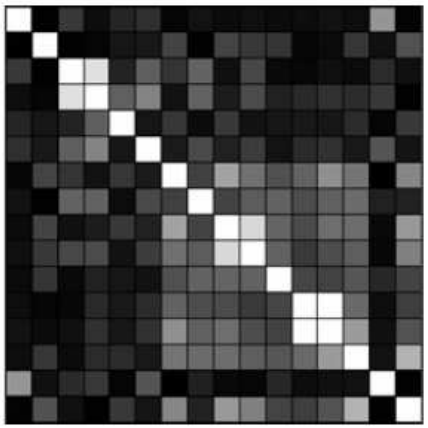
- Consider a medical application involving 16 variables:
- The correlation table looks like:

Var. num.	Description	Values
1	Age	Continuous
2	Sex (1=male, 2=female)	1, 2
3	Pain quadrant 1	0, 1
4	Pain quadrant 2	0, 1
5	Pain quadrant 3	0, 1
6	Pain quadrant 4	0, 1
7	Local muscular guarding	0, 1
8	Generalized muscular guarding	0, 1
9	Rebound tenderness	0, 1
10	Pain on tapping	0, 1
11	Pain during rectal examination	0, 1
12	Axial temperature	Continuous
13	Rectal temperature	Continuous
14	Leukocytes	Continuous
15	Diabetes mellitus	0, 1
16	Appendicitis	0, 1

1.	−0.009	0.14	0.037	−0.096	0.12	0.018	0.051	−0.034	−0.041	0.034	0.037	0.05	−0.037	0.37	0.012
−0.009	1.	−0.0074	−0.019	−0.06	0.063	−0.17	0.0084	−0.17	−0.14	−0.13	−0.017	−0.034	−0.14	0.045	−0.2
0.14	−0.0074	1.	0.55	−0.091	0.24	0.13	0.24	0.045	0.18	0.028	0.02	0.045	0.03	0.11	0.045
0.037	−0.019	0.55	1.	−0.24	0.33	0.051	0.25	0.074	0.19	0.087	0.11	0.12	0.11	0.14	−0.0091
−0.096	−0.06	−0.091	−0.24	1.	0.059	0.14	0.034	0.14	0.049	0.057	0.064	0.058	0.11	0.017	0.14
0.12	0.063	0.24	0.33	0.059	1.	0.071	0.19	0.086	0.15	0.048	0.11	0.12	0.063	0.21	0.053
0.018	−0.17	0.13	0.051	0.14	0.071	1.	0.16	0.4	0.28	0.2	0.24	0.36	0.29	−0.0001	0.33
0.051	0.0084	0.24	0.25	0.034	0.19	0.16	1.	0.17	0.23	0.24	0.19	0.24	0.27	0.083	0.084
−0.034	−0.17	0.045	0.074	0.14	0.086	0.4	0.17	1.	0.53	0.25	0.19	0.27	0.27	0.026	0.38
−0.041	−0.14	0.18	0.19	0.049	0.15	0.28	0.23	0.53	1.	0.24	0.15	0.19	0.23	0.02	0.32
0.034	−0.13	0.028	0.087	0.057	0.048	0.2	0.24	0.25	0.24	1.	0.17	0.17	0.22	0.098	0.17
0.037	−0.017	0.02	0.11	0.064	0.11	0.24	0.19	0.19	0.15	0.17	1.	0.72	0.26	0.035	0.15
0.05	−0.034	0.045	0.12	0.058	0.12	0.36	0.24	0.27	0.19	0.17	0.72	1.	0.38	0.044	0.21
−0.037	−0.14	0.03	0.11	0.11	0.063	0.29	0.27	0.27	0.23	0.22	0.26	0.38	1.	0.051	0.44
0.37	0.045	0.11	0.14	0.017	0.21	−0.0001	0.083	0.026	0.02	0.098	0.035	0.044	0.051	1.	−0.0055
0.012	−0.2	0.045	−0.0091	0.14	0.053	0.33	0.084	0.38	0.32	0.17	0.15	0.21	0.44	−0.0055	1.



$K_{ij} = -1$: black, $K_{ij} = 1$: white



$|K_{ij}| = 0$: black, $|K_{ij}| = 1$: white

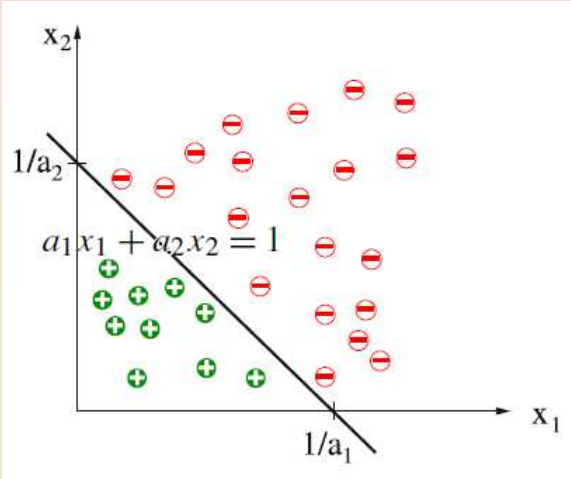
- The density plots look like:
- Variables 7, 9, 10 and 14 most strongly correlate with "appendicitis"
- Since 9 and 10 are strongly correlated, one may be sufficient.

Subsection 3

The Perceptron: A Linear Classifier

Linear Separability

- A set of training data is linearly separable if it can be separated by a straight line.
- In n dimensions a **hyperplane** is needed for the separation, i.e., a linear subspace of dimension $n-1$.



- In $\mathbb{R}^n$ it can be described by an equation $\sum_{i=1}^{n} a_i x_i = \theta$.

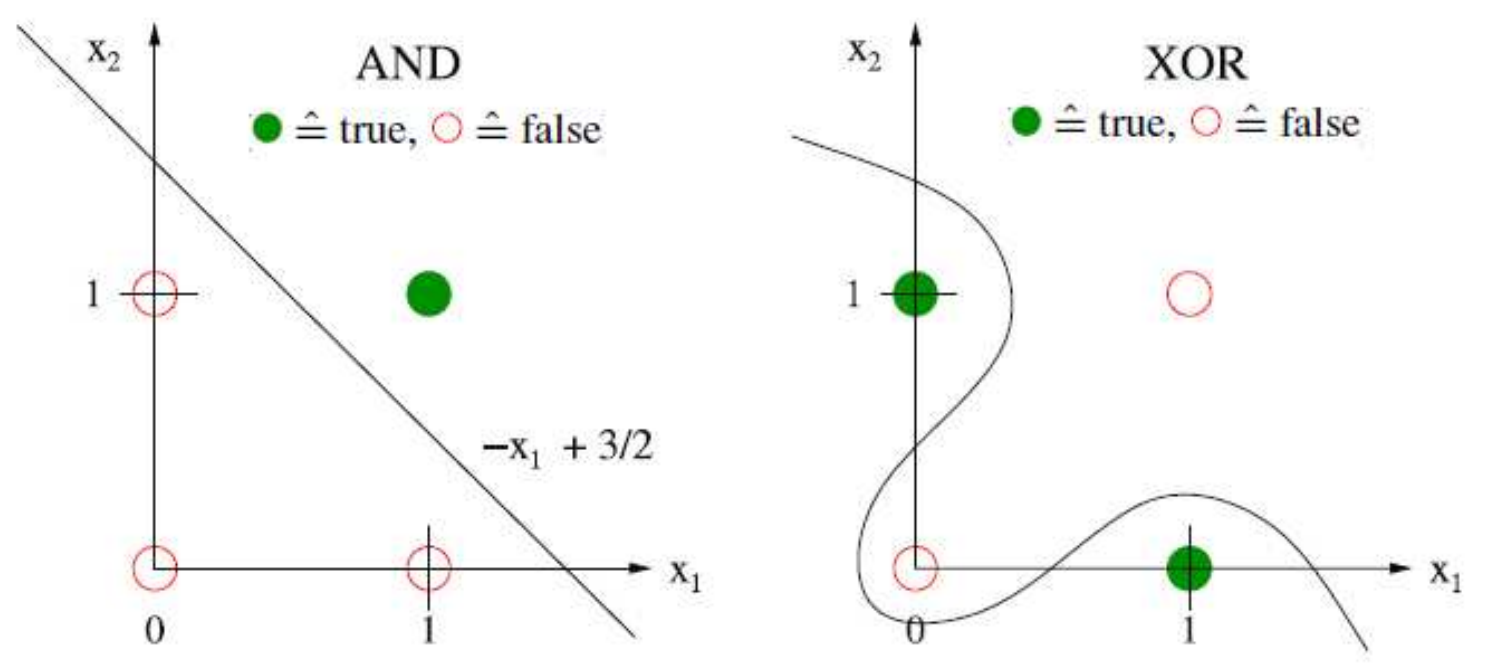
Linear Separability in $\mathbb{R}^n$

Two sets $M_1 \subset \mathbb{R}^n$, $M_2 \subset \mathbb{R}^n$ are called **linearly separable** if there exist real numbers $a_1, \ldots, a_n, \theta$, such that

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

θ is called the **threshold**.

Linear Separability: Illustraion



- The AND function is linearly separable, but the XOR function is not.
- For AND, the line $x_2 = -x_1 + 3/2$ separates true and false interpretations of the formula $x_1 \wedge x_2$.
- Clearly the XOR function has a more complex structure than the AND function in terms of linear separability.

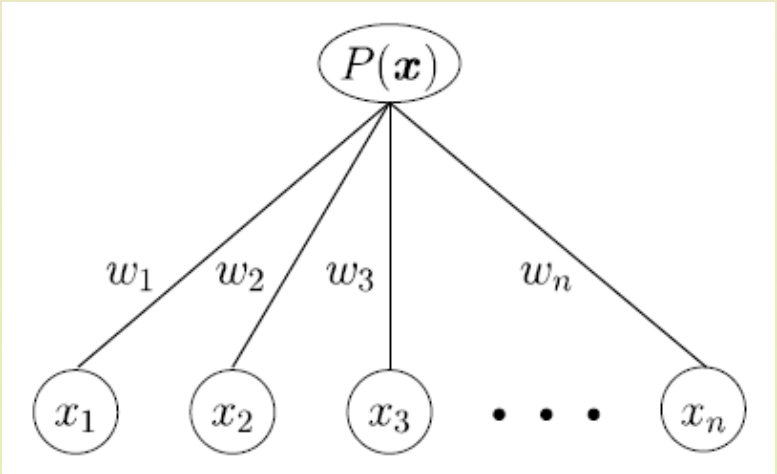
The Perceptron: Definition

Definition of Perceptron

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

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

- The perceptron is a very simple classification algorithm.
- It is equivalent to a two-layer neural network with activation by a threshold function.
- Each node in the network represents a neuron, and every edge a synapse.



The Perceptron Learning Rule

- We view the perceptron as a learning agent, i.e., as a mathematical function which maps a feature vector to a function value.
- All points x above the hyperplane $\sum_{i=1}^{n} w_i x_i = 0$ are classified as positive ($P(x) = 1$), and all others as negative ($P(x) = 0$).
- We will show that the absence of an arbitrary threshold represents no restriction of power.
- Let M_+ and M_- be the sets of positive and negative training points:

PerceptronLearning$[M_+, M_-]$

$\mathbf{w}$ = arbitrary vector of real numbers
Repeat
  For all $\mathbf{x} \in M_+$
    If $\mathbf{w} \cdot \mathbf{x} \leq 0$ **then** $\mathbf{w} = \mathbf{w} + \mathbf{x}$
  For all $\mathbf{x} \in M_-$
    If $\mathbf{w} \cdot \mathbf{x} > 0$ **then** $\mathbf{w} = \mathbf{w} - \mathbf{x}$
Until all $\mathbf{x} \in M_+ \cup M_-$ are correctly classified

How the Perceptron Rule Works

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

- The perceptron should output the value 1 for all $x \in M_+$.
- This is true when $\mathbf{w} \cdot \mathbf{x} > 0$.

- If this is not the case, then $\mathbf{x}$ is added to the weight vector $\mathbf{w}$, so the weight vector is changed in exactly the right direction.
- If we apply the perceptron to new weight vector $\mathbf{w} + \mathbf{x}$, we get

$$(\mathbf{w} + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x} + |\mathbf{x}|^2,$$

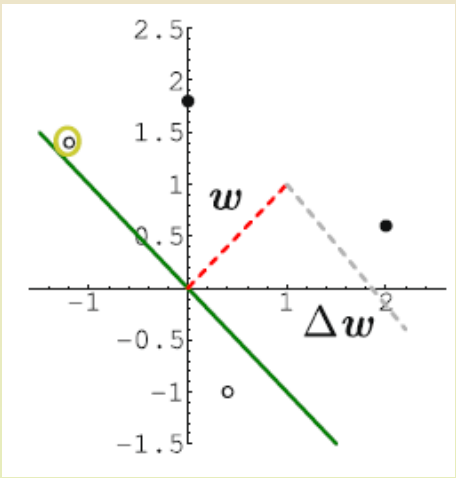
  i.e., the change in $\mathbf{w} \cdot \mathbf{x}$ tends towards a positive direction.
- For negative training data, the perceptron calculates an ever smaller value for the dot product.

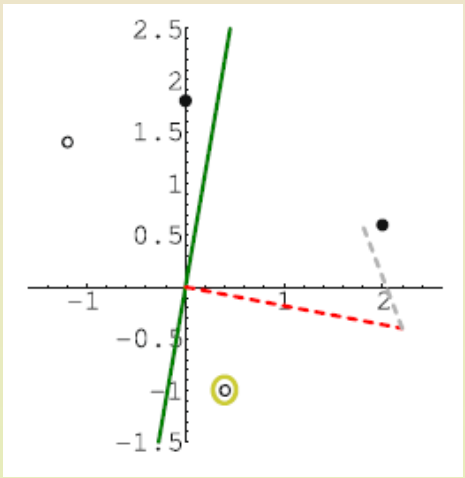
An Example

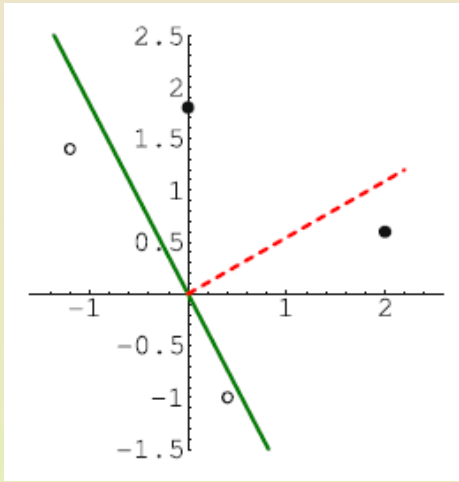
- A perceptron is to be trained on the sets

$$M_+ = \{(0, 1.8), (2, 0.6)\}$$
$$M_- = \{(-1.2, 1.4), (0.4, -1)\}.$$

$\mathbf{w} = (1, 1)$ is used as an initial weight vector. The training data and the line $\mathbf{w} \cdot \mathbf{x} = x_1 + x_2 = 0$:







In the first iteration the only falsely classified training example is $(-1.2, 1.4)$ because $(-1.2, 1.4) \cdot (1, 1) = 0.2 > 0$. This results in $\mathbf{w} = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$.

After five changes, the line separates the two classes.

The Perceptron Learning Algorithm Theorem

- The perceptron always converges for linearly separable data:

Theorem (Guarantee of Success of PerceptronLearning)

Let M_+ and M_- be linearly separable classes by a hyperplane $\mathbf{w} \cdot \mathbf{x} = 0$. Then PerceptronLearning converges for every initialization of the vector $\mathbf{w}$. The perceptron P with the weight vector so calculated divides the classes M_+ and M_-, i.e.,

$$P(x) = 1 \Leftrightarrow x \in M_+ \quad \text{and} \quad P(x) = 0 \Leftrightarrow x \in M_-.$$

- Perceptrons cannot divide arbitrary linearly separable sets, rather only those which are divisible by a line (or in $\mathbb{R}^n$ a hyperplane) through the origin, because the constant θ is missing from $\sum_{i=1}^{n} w_i x_i = 0$.
- With a trick, however, we can generate the constant term.

Representability by Perceptrons

- We hold the last component x_n of the input vector $\mathbf{x}$ constant to 1.
- Then $w_n =: -\theta$ works like a threshold: $\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n-1} w_i x_i - \theta > 0 \iff \sum_{i=1}^{n-1} w_i x_i > \theta$.
- In this case $x_n = 1$ is called a **bias unit**.
- In the application of the perceptron learning algorithm, a bit with the constant value 1 is appended to the training data vector.
- We observe that the weight w_n, or the threshold θ, is learned during the learning process.
- Thus, a perceptron $P_\theta : \mathbb{R}^{n-1} \to \{0, 1\}$

$$P_\theta(x_1, \ldots, x_{n-1}) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n-1} w_i x_i > \theta \\ 0, & \text{otherwise} \end{cases}$$

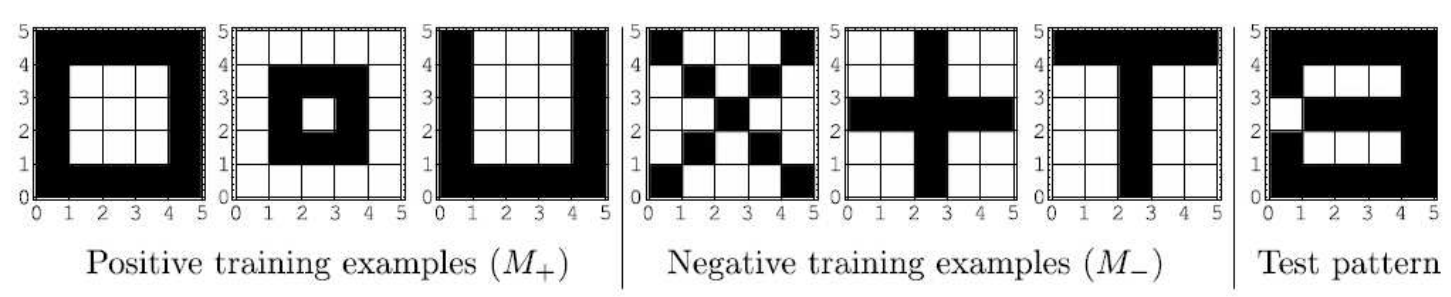
can be simulated by a perceptron $P : \mathbb{R}^n \to \{0, 1\}$ with the threshold 0.

Theorem (Perceptron Representability)

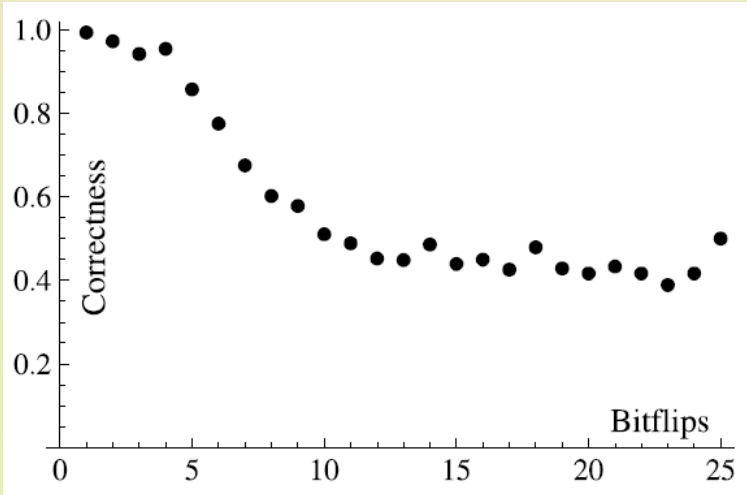
A function $f : \mathbb{R}^n \to \{0, 1\}$ can by represented by a perceptron if and only if the two sets of positive and negative input vectors are linearly separable.

Example: Pattern Recognition

- We train a perceptron with a threshold on six simple, graphical binary patterns with 5×5 pixels each:



- The training data can be learned by PerceptronLearning in four iterations over all patterns.
- Patterns, with a variable number of inverted bits (in sequence one after the other) introduced as noise, are used as test patterns.
- The percentage of correctly classified patterns is plotted vs. the number of false bits:



Power, Convergence and Generalizations

- The two-layer perceptron is one of the simplest neural network based learning algorithms and can only divide linearly separable classes.
- Multilayered networks are significantly more powerful.
- Despite the simple structure, the perceptron converges very slowly. It can be accelerated by normalization of the weight-altering vector.
- The formulas $\mathbf{w} = \mathbf{w} \pm \mathbf{x}$ are replaced by $\mathbf{w} = \mathbf{w} \pm \frac{\mathbf{x}}{|\mathbf{x}|}$, so that every data point has the same weight during learning.
- Speed of convergence depends on the initialization of $\mathbf{w}$.
  - In the best case scenario, $\mathbf{w}$ would not need to be changed at all.
  - We can get closer to this goal by using the heuristic initialization $\mathbf{w}_0 = \sum_{\mathbf{x} \in M_+} \mathbf{x} - \sum_{\mathbf{x} \in M_-} \mathbf{x}$.
- The perceptron is equivalent to naive Bayes, the simplest type of Bayesian network.
- A generalization in the form of the back-propagation algorithm can divide non linearly separable sets through the use of multiple layers.

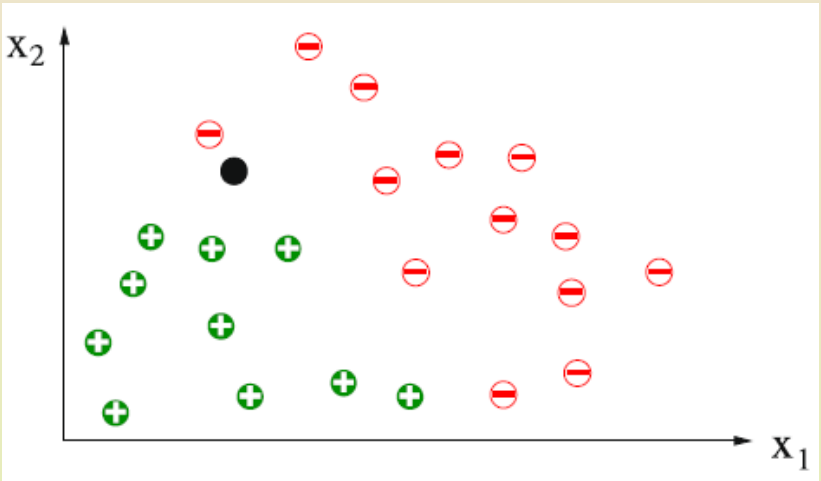
Subsection 4

The Nearest Neighbor Method

Similarity and Distance

- Sometimes, similarity provides a clue for good classification.
- But, how can similarity be formalized?
- If the training samples are presented in a multidimensional feature space, we may declare that "the smaller their distance in the feature space, the more similar" they are.

Applying this to a simple two-dimensional example, the next neighbor to the black point is a negative example. So, it is assigned to the negative class.



- The distance $d(\mathbf{x}, \mathbf{y})$ between two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ may be measured by $d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$.
- It is often sensible to scale the features differently by weights w_i. The formula then reads $d_w(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}$.

Nearest Neighbor Classification Algorithm

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

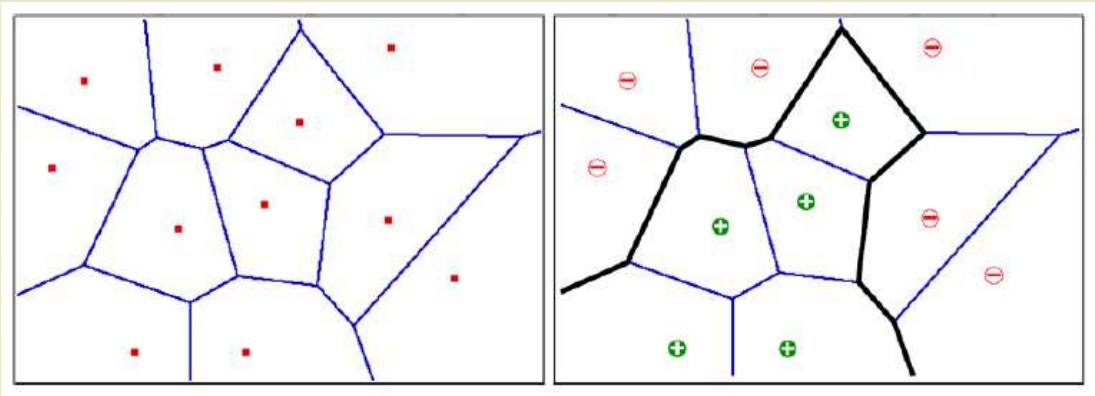
NearestNeighbor$[M_+, M_-, \mathbf{s}]$

$\mathbf{t} = \text{argmin}_{\mathbf{x} \in M_+ \cup M_-} \{d(\mathbf{s}, \mathbf{x})\}$
If $\mathbf{t} \in M_+$ **then Return** ("+")
Else Return("-")

- Unlike perceptron, the nearest neighbor does not generate a line dividing the training data points, but an imaginary line certainly exists.
- We can generate this by first generating the Voronoi diagram.
  - In the Voronoi diagram, each data point is surrounded by a convex polygon, which thus defines a neighborhood around it.

Voronoi Diagrams

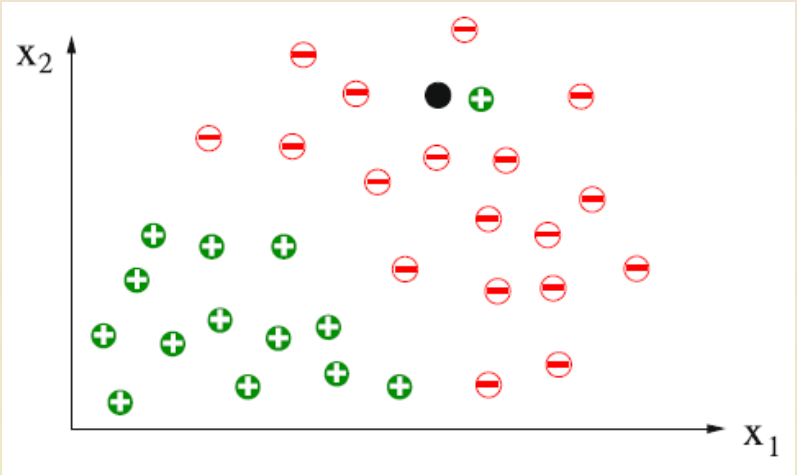
- In the Voronoi diagram the nearest neighbor of a new point among the data points is the one that lies in the same neighborhood.
- If the Voronoi diagram for a set of training data is determined, then it is simple to find the nearest neighbor for a new point to be classified.
- The class membership will then be taken from the nearest neighbor.



- The nearest neighbor method is significantly more powerful than the perceptron, since it is capable of correctly realizing arbitrarily complex dividing lines (or hyperplanes).
- However, the drawback is that, in certain circumstances, a single erroneous point can lead to very bad classification results.

Case for k-Nearest Neighbor Algorithm

- With nearest neighbor a single erroneous point can lead to very bad classification results:
  The new point is immediately next to a positive point that is an outlier of the positive class, so it will be classified positive rather than negative as would be intended here.



- An erroneous fitting to random errors (noise) is called **overfitting**.
- To prevent false classifications due to single outliers, it is recommended to smooth out the division surface somewhat.
- This can be accomplished with the K-NearestNeighbor algorithm, which makes a majority decision among the k nearest neighbors instead of relying on just one.

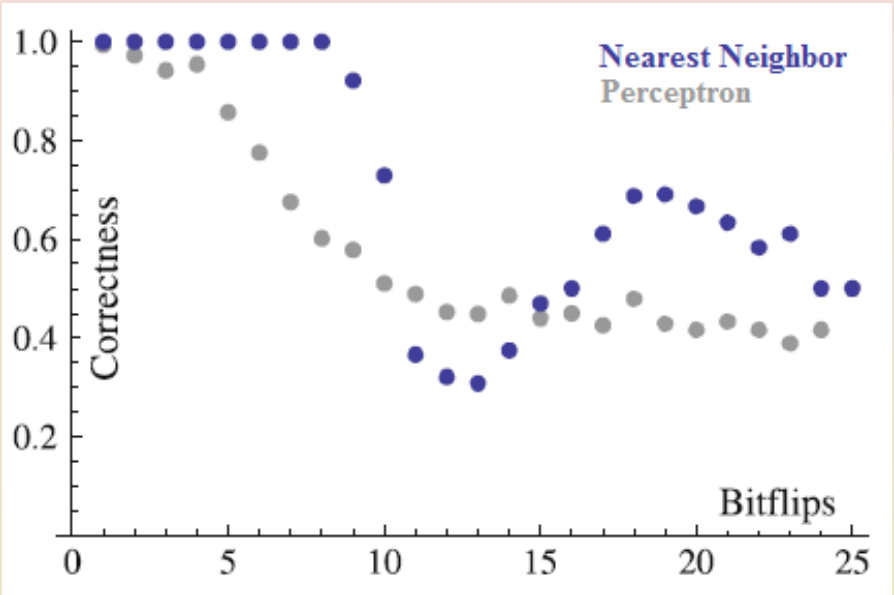
The k-Nearest Neighbor Algorithm

K-NearestNeighbor($M_+, M_-, \mathbf{s}$)

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

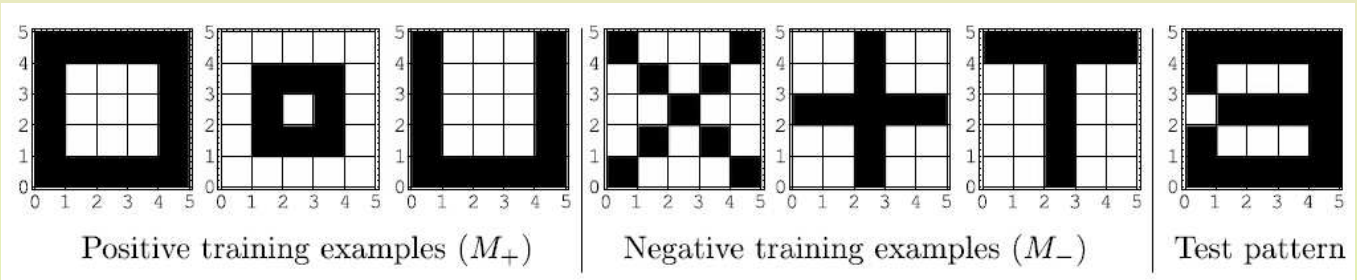
- Example: How does NearestNeighbor fair in our pattern recognition example?
  - We use the Hamming distance as the distance metric.
  - We use training examples with n consecutive inverted bits as test data.
  - We plot the percentage of correctly classified test examples vs. the number of inverted bits b.

Pattern Recognition: Nearest Neighbor vs. Perceptron



- For up to eight inverted bits, all patterns are correctly identified. Past that point, the number of errors quickly increases.

- This is unsurprising because training pattern 2 has a hamming distance of 9 from patterns 4,5.



- Quite clearly we see that nearest neighbor classification is superior to the perceptron in this application for up to eight false bits.

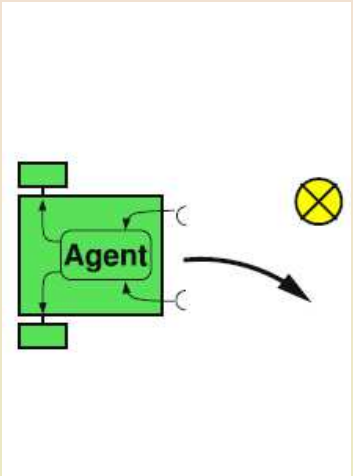
Nearest Neighbor for Many Classes

- The nearest neighbor classification can be applied to more than two classes.
- The class of the feature vector to be classified is simply set as the class of the nearest neighbor.
- For the k nearest neighbor method, the class is to be determined as the class with the most members among the k nearest neighbors.
- If the number of classes is large, then it usually no longer makes sense to use classification algorithms:
  - The size of the necessary training data grows quickly with the number of classes.
  - In certain circumstances important information is lost during classification of many classes.

Training A Robot: The Setup

- An autonomous Braitenberg style robot with simple sensors is supposed to learn to move away from light.

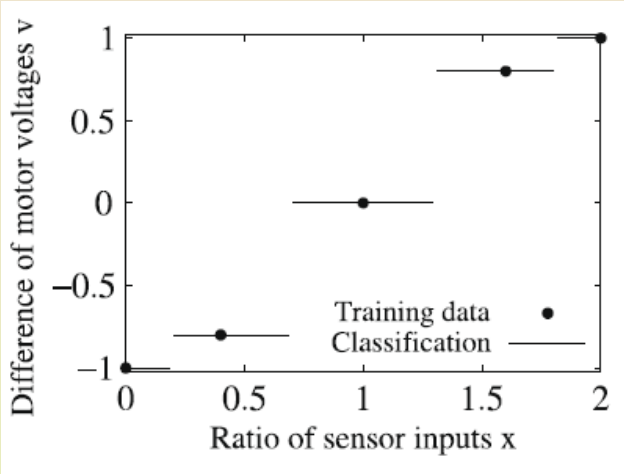
  This means it should learn as optimally as possible to map its sensor values onto a steering signal which controls the driving direction. The robot is equipped with two simple light sensors on its front side.

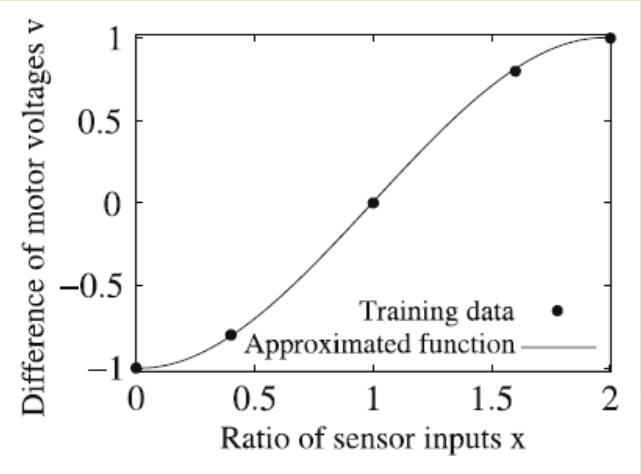


  From the two sensor signals (s_ℓ for the left and s_r for the right sensor), the relationship $x = \frac{s_r}{s_\ell}$ is calculated. To control the electric motors of the two wheels from this value x, the difference $v = U_r - U_\ell$ of the two voltages U_r and U_ℓ of the left and right motors, respectively, is calculated. The learning agent's task is now to avoid a light signal. It must therefore learn a mapping f which calculates the "correct" value $v = f(x)$.

Training A Robot: Training

- For a few values x, we find as optimal a value v as we can.
- During nearest neighbor classification each x-point is classified exactly like its nearest neighbor among the training data. This results into a step function with large jumps. For finer steps, need more training data.





- On the other hand, we can obtain a continuous function if we approximate a smooth function to fit the five points.
- Requiring the function f to be continuous leads to very good results, even with no additional data points.

Multi-Dimensional k-Nearest Neighbor

- Approximation is problematic in higher dimensions.
- The special difficulty in AI is that model-free approximation methods are needed, i.e., a good approximation must be produced without knowledge about special properties of the data or the application.
- Very good results have been achieved here with neural networks and other nonlinear function approximators, presented later.
- The k-nearest neighbor method can be applied in a simple way to the approximation problem.
- In the algorithm K-NearestNeighbor, after the set $V = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ is determined, the k-nearest neighbors average function value

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

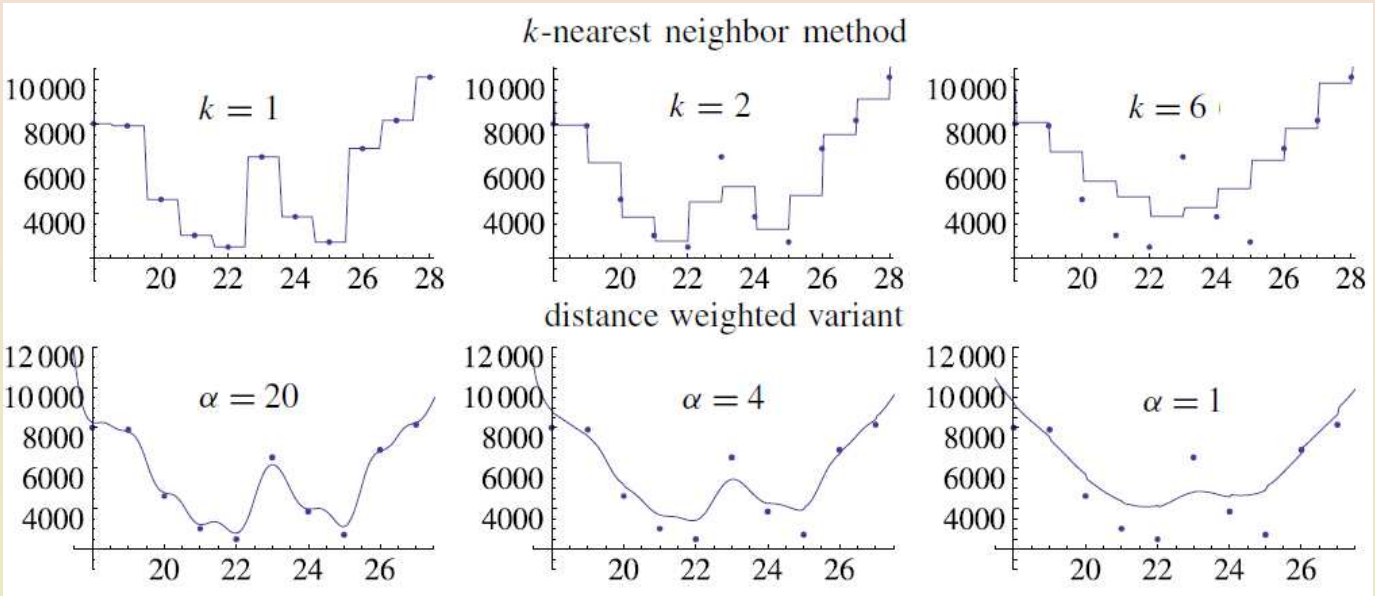
is calculated and taken as an approximation $\hat{f}$ for the query vector $\mathbf{x}$. The larger k becomes, the smoother the function $\hat{f}$ is.

Weighting Distances

- As k becomes large, there typically exist more neighbors with a large distance than those with a small distance.
- So, the calculation of $\hat{f}$ is dominated by neighbors that are far away.
- To prevent this, the k neighbors are weighted such that the more distant neighbors have lesser influence on the result.
- During the majority decision in K-NearestNeighbor, the "votes" are weighted with the weight $w_i = \frac{1}{1 + \alpha d(\mathbf{x}, \mathbf{x}_i)^2}$, which decreases with the square of the distance. The constant α determines the speed of decrease of the weights.
- Thus, $\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{k} w_i f(\mathbf{x}_i)}{\sum_{i=1}^{k} w_i}$.
- The influence of points asymptotically approaches zero as distance increases, so it becomes possible to use many or even all training data to classify or approximate a given input vector.

Weighting Distances: Illustration

- The distance weighted method gives a much smoother approximation than k-nearest neighbor.



- With respect to approximation quality, this simple method competes well with sophisticated approximation algorithms such as nonlinear neural networks, support vector machines, and Gaussian processes.
- The width parameter α, which has to be set manually, greatly influences the results. Optimization methods have been developed to automatically set α.

Time Requirements

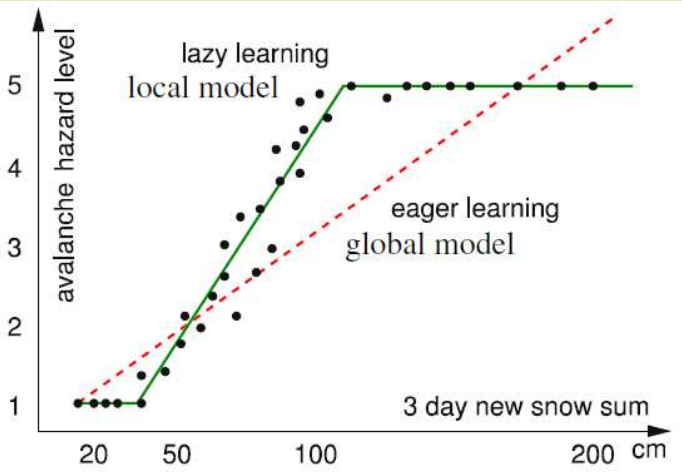
- Training is accomplished in all variants of the nearest neighbor method by simply saving all training vectors together with their labels (class values), or the function value $f(\mathbf{x})$.
- Thus, there is no other learning algorithm that learns as quickly.
- Answering a query for classification or approximation of a vector $\mathbf{x}$ can become very expensive.
  - Just finding the k nearest neighbors for n training data requires a cost which grows linearly with n.
  - For classification or approximation, there is additionally a cost which is linear in k.
  - The total computation time thus grows as $\Theta(n+k)$.
- For large amounts of training data, this can create problems.

Lazy versus Eager Learning

- Because nothing happens in the learning phase of the presented nearest neighbor methods, such algorithms are also denoted lazy learning, in contrast to eager learning, in which the learning phase can be expensive, but application to new examples is very efficient.
- The perceptron and all other neural networks, decision tree learning, and the learning of Bayesian networks are eager learning methods.
- Since the lazy learning methods need access to the memory with all training data for approximating a new input, they are also called memory-based learning.

Lazy versus Eager Learning: Swiss Avalanche Hazard

- In determining the current avalanche hazard from the amount of newly fallen snow in a certain area of Switzerland, values determined by experts are entered, which we want to use as training data.
  - During the application of an eager learning algorithm which undertakes a linear approximation of the data, we get the dashed line:



  - During lazy learning, nothing is calculated before a query for the current hazard level arrives. Then the answer is calculated from several nearest neighbors, that is, locally.

- The advantage of the lazy method is its locality.
- Nearest neighbor methods provide a good local approximation, when speed is not an issue.

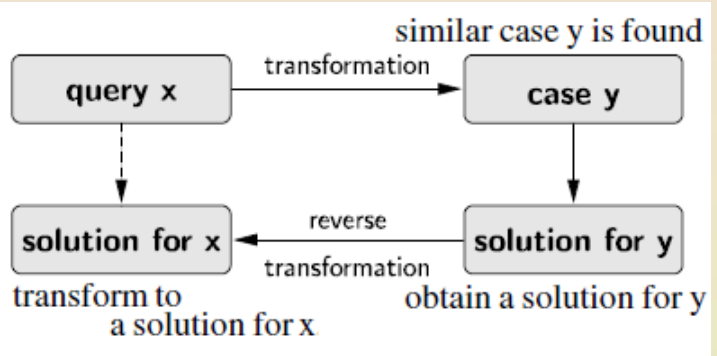
Case-Based Reasoning

- To be able to use the described methods, the training data must be available in the form of vectors of integers or real numbers.
- They are unsuitable for applications in which the data are represented symbolically, for example as first order logic formulas.
- In case-based reasoning (CBR), the nearest neighbor method is extended to symbolic problem descriptions and their solutions.
- Construction of CBR diagnostic systems is a difficult task:
  - Modeling: The domain must be modeled formally. Logical monotony presents difficulties. The developer must predict and map all possible special cases and problem variants.
  - Similarity: Finding a suitable similarity metric for symbolic features.
  - Transformation: Even if a similar case is found, it is not yet clear how the transformation mapping and its inverse should look.
- There are practical CBR systems, but still behind human experts.
- Alternatives to CBR, such as Bayesian networks, decision trees or neural networks, work well, if the symbolic representation can be mapped to discrete or continuous numerical features.

Example: Diagnosing Bike Problems

- Consider the diagnosis of a bicycle light going out:

Feature	Query	Case from case base
Defective part:	Rear light	Front light
Bicycle model:	Marin Pine Mountain	VSF T400
Year:	1993	2001
Power source:	Battery	Dynamo
Bulb condition:	ok	ok
Light cable condition:	?	ok
	Solution	
Diagnosis:	?	Front electrical contact missing
Repair:	?	Establish front electrical contact



- Consider the query of a defective rear light.
- In the right column, a case similar to the query is given. It is based to training data in the nearest neighbor method.
- Nearest neighbor would mislead into trying to repair the front light.
- A reverse transformation of the solution back to the query is needed.
- The transformation is: rear light mapped to front light.

Subsection 5

Decision Tree Learning

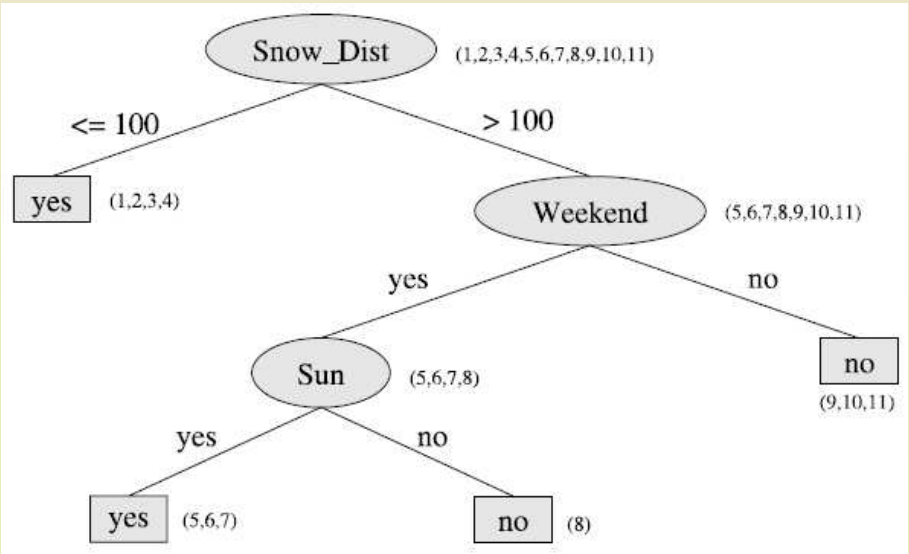
Introduction to Decision Tree Learning

- Decision tree learning is an important algorithm for extracting knowledge from data because it is powerful, simple and efficient.
- Compared to perceptron and nearest neighbor, it has the advantage that the extracted knowledge can be easily understood, interpreted, and controlled by humans in the form of a readable decision tree.
- This makes it also an important tool for data mining.
- We focus on the C4.5 decision tree learning algorithm (Quinlan 1993)
  - It is an improvement over ID3 (Iterative Dichotomiser 3, 1986).
  - A more efficient successor that can take into account the costs of decisions, is C5.0.
- The CART (Classification and Regression Trees, 1984) system of Breiman works similarly.
- In 1964, the CHAID (Chi-square Automatic Interaction Detectors) system was introduced by Sonquist and Morgan.
- Also interesting is the data mining tool KNIME (Konstanz Information Miner), which uses the WEKA Java library to induct decision trees.

Example of a Decision Tree

- A devoted skier wants a decision tree to help him decide whether to go skiing. The two-class problem ski yes/no is based on the variables:

Variable	Value	Description
Ski (goal variable)	Yes, no	Should I drive to the nearest ski resort with enough snow?
Sun (feature)	Yes, no	Is there sunshine today?
Snow_Dist (feature)	$\leq$100, >100	Distance to the nearest ski resort with good snow conditions (over/under 100 km)
Weekend (feature)	Yes, no	Is it the weekend today?



- A **decision tree** is a tree whose inner nodes represent features (attributes).
- Each edge stands for an attribute value.
- At each leaf node a class value is given.

Data for a Decision Tree

- The data used for the construction of the decision tree are:
- Each row contains the data for one day, so it represents a sample.
- Since row 6 and row 7 contradict each other, no deterministic classification algorithm can correctly classify all of the data.

Day	*Snow_Dist*	*Weekend*	*Sun*	*Skiing*
1	≤100	Yes	Yes	Yes
2	≤100	Yes	Yes	Yes
3	≤100	Yes	No	Yes
4	≤100	No	Yes	Yes
5	>100	Yes	Yes	Yes
6	>100	Yes	Yes	Yes
7	>100	Yes	Yes	No
8	>100	Yes	No	No
9	>100	No	Yes	No
10	>100	No	Yes	No
11	>100	No	No	No

- The number of falsely classified data must therefore be 1, i.e., the tree classifies the data optimally.
- How is such a tree created from the data?

Naive Construction of a Decision Tree

- We first restrict to discrete attributes with finitely many values.
- The number of attributes is also finite and each attribute can occur at most once per path, so there are finitely many different decision trees.
- A simple, obvious algorithm for the construction of a tree would
  - generate all trees;
  - for each tree calculate the number of erroneous classifications of the data;
  - at the end choose the tree with the minimum number of errors.
- Thus we would even have an optimal algorithm (in the sense of errors for the training data) for decision tree learning.
- The obvious disadvantage of this algorithm is its unacceptably high computation time, as soon as the number of attributes becomes somewhat larger.

Heuristic Construction Based on Information Gain

- We will develop a heuristic algorithm which, starting from the root, recursively builds a decision tree:
  - First the attribute with the highest information gain (Snow_Dist) is chosen for the root node from the set of all attributes.
  - For each attribute value ($\leq 100, > 100$) there is a branch in the tree.
  - Now for every branch this process is repeated recursively:
    - During generation of the nodes, the attribute with the highest information gain among the attributes which have not yet been used is always chosen, in the spirit of a greedy strategy.
- At each step the algorithm selects the attribute with the highest information gain.
- Entropy is the metric used for the information content of a set of training data D.

Certain and Uniform Distributions

- If we only look at the binary variable skiing in the above example, then D can be described as

$$D = (\text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{no}, \text{no}, \text{no}, \text{no}, \text{no}),$$

  with estimated probabilities $p_1 = P(\text{yes}) = \frac{6}{11}$ and $p_2 = P(\text{no}) = \frac{5}{11}$.
- In general, for an n class problem $p = (p_1, \ldots, p_n)$ with $\sum_{i=1}^{n} p_i = 1$.
  - If $p = (1, 0, 0, \ldots, 0)$, i.e. the first of the n events will certainly occur and all others will not, the uncertainty about the outcome is minimal.
  - In contrast, if $p = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$, the uncertainty is maximal because no event can be distinguished from the others.
- Shannon asked how many bits are needed to encode such an event.
  - In the certain case, zero bits are needed because we know that the case $i = 1$ always occurs.
  - In the uniformly distributed case, there are n equally probable possibilities. For binary encodings, $\log_2 n$ bits are needed. Because all individual probabilities are $p_i = \frac{1}{n}$, $\log_2 \frac{1}{p_i}$ bits are needed in this case.

Entropy: The General Case

- In the general case $p = (p_1, \ldots, p_n)$, the expectation value H for the number of bits is calculated.
- We weight all values $\log_2 \frac{1}{p_i} = -\log_2 p_i$ with their probabilities and obtain

$$H = \sum_{i=1}^{n} p_i(-\log_2 p_i) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

- The more bits we need to encode an event, clearly the higher the uncertainty about the outcome.

Definition (Entropy)

The **entropy** H as a metric for the uncertainty of a probability distribution is defined by

$$H(p) = H(p_1, \ldots, p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i.$$

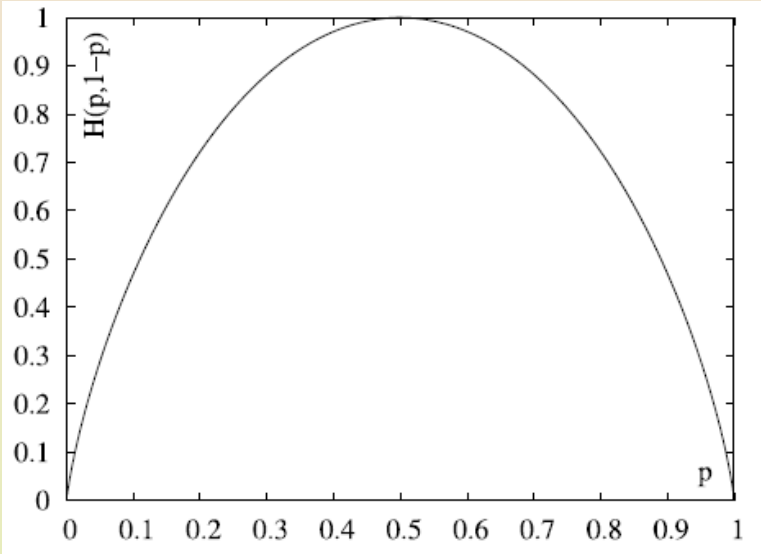
- To cope with zeros, we define $0 \cdot \log_2 0 := 0$.

Entropy for a Two-Outcome Distribution

- Clearly, $H(1, 0, \ldots, 0) = 0$.
- The entropy in the hypercube $[0, 1]^n$ under the constraint $\sum_{i=1}^{n} p_i = 1$ takes on its maximum value with the uniform distribution $(\frac{1}{n}, \ldots, \frac{1}{n})$.

- In the case of two possible outcomes, the result is

$$\begin{aligned} H(p) &= H(p_1, p_2) \\ &= H(p_1, 1 - p_1) \\ &= -(p_1 \log_2 p_1 + \\ &\quad (1 - p_1) \log_2 (1 - p_1)). \end{aligned}$$



- The graph, with its maximum at $p_1 = \frac{1}{2}$, is shown as a function of p_1:
- Because each classified dataset D is assigned a probability distribution p by estimating the class probabilities, we can extend the concept of entropy to data by the definition $H(D) = H(p)$.

Information Content and Information Gain

- The information content $I(D)$ of the data set D is meant to be the opposite of uncertainty:

Definition (Information Content)

The **information content** of a dataset is defined as $I(D) := 1 - H(D)$.

- The more an attribute raises the information content of the distribution by dividing the data, the better that attribute is.

Definition (Information Gain)

The **information gain** $G(D, A)$ through the use of the attribute A is determined by the difference of the average information content of the data set $D = D_1 \cup D_2 \cup \cdots \cup D_n$, divided by the n-value attribute A, and the information content $I(D)$ of the undivided data set:

$$G(D, A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D).$$

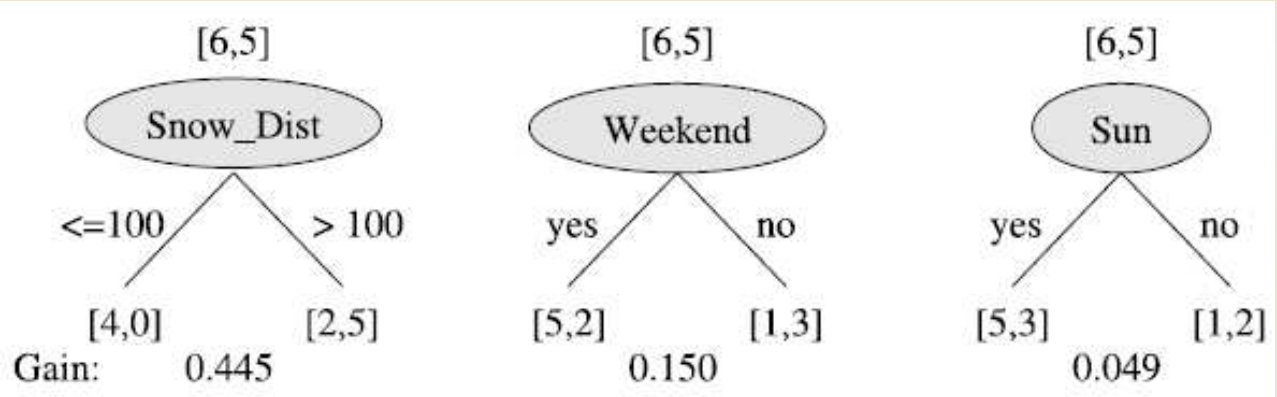
Gain in terms of Entropy

- We express $G(D, A)$ directly in terms of entropy:

$$\begin{aligned} G(D,A) &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D) \\ &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D)) \\ &= 1 - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i). \end{aligned}$$

The Skiing Example I

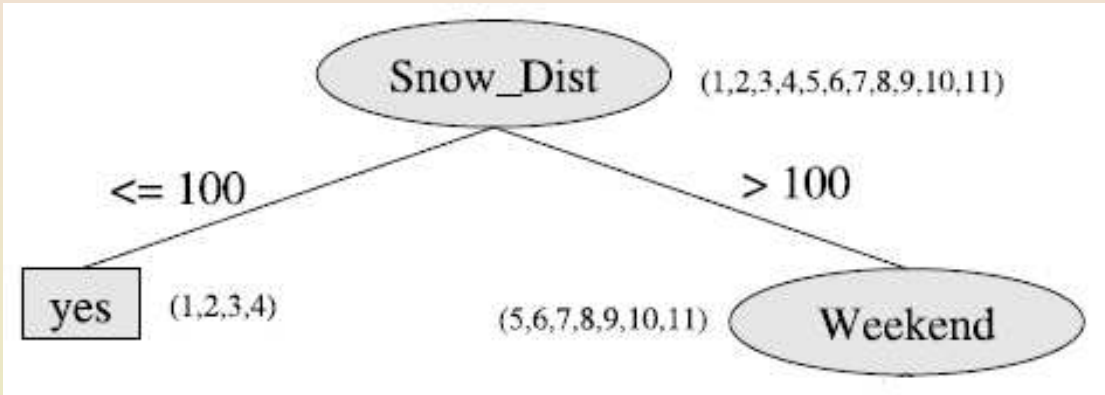
- Note that $H(\frac{6}{11}, \frac{5}{11}) = -\frac{6}{11}\log_2\frac{6}{11} - \frac{5}{11}\log_2\frac{5}{11} \approx 0.994$.
- Now $G(D, \text{Snow\_Dist}) = H(D) - (\frac{4}{11}H(D_{\leq 100}) + \frac{7}{11}H(D_{>100})) \approx 0.994 - \frac{4}{11}\cdot 0 + \frac{7}{11}\cdot 0.863 = 0.445$.
- Similarly, $G(D, \text{Weekend}) \approx 0.150$ and $G(D, \text{Sun}) \approx 0.049$.



- Based on this information gain analysis, the attribute Snow_Dist becomes the root node of the decision tree.
- The two attribute values ≤ 100 and > 100 generate two edges in the tree, which correspond to the subsets $D_{\leq 100}$ and $D > 100$.
  - For $D_{\leq 100}$ the classification is yes and the tree terminates.
  - For $D_{>100}$ there is no clear result and the algorithm proceeds.

The Skiing Example II

Day	*Snow_Dist*	*Weekend*	*Sun*	*Skiing*
1	≤100	Yes	Yes	Yes
2	≤100	Yes	Yes	Yes
3	≤100	Yes	No	Yes
4	≤100	No	Yes	Yes
5	>100	Yes	Yes	Yes
6	>100	Yes	Yes	Yes
7	>100	Yes	Yes	No
8	>100	Yes	No	No
9	>100	No	Yes	No
10	>100	No	Yes	No
11	>100	No	No	No



- From the two attributes still available, Sun and Weekend, the better one must be chosen:
  $G(D_{>100}, \text{Weekend}) = 0.292$ and $G(D_{>100}, \text{Sun}) = 0.170$.
- The node is assigned the attribute Weekend.
  - For Weekend = no the tree terminates with the decision Ski = no.
  - For Weekend = yes, Sun results in a gain of 0.171.
- After this, construction of the tree terminates because no further attributes are available.

The C4.5 Decision Tree Generator: Input

- This decision tree can be generated by C4.5.
- The training data are saved in a data file `ski.data`:

```
<=100, yes,  yes,  yes
<=100, yes,  yes,  yes
<=100, yes,  no,   yes
<=100, no,   yes,  yes
>100,  yes,  yes,  yes
>100,  yes,  yes,  yes
>100,  yes,  yes,  no
>100,  yes,  no,   no
>100,  no,   yes,  no
>100,  no,   yes,  no
>100,  no,   no,   no
```

```
|Classes: no: do not ski,
|         yes: go skiing
no,yes.
|
|Attributes
|
Snow_Dist:          <=100,>100.
Weekend:            no,yes.
Sun:                no,yes.
```

- The information about attributes and classes is stored in the file `ski.names` (lines beginning with "|" are comments):
- C4.5 is run

```
unixprompt> c4.5 -f ski -m 1
```

  - The option -f is for the name of the input file;
  - the option -m specifies the minimum number of training data points required for generating a new branch in the tree. Because the number of training data points is extremely small, -m 1 is sensible.

The C4.5 Decision Tree Generator: Output

- The following decision tree (formatted w/ indentations) is generated.

```
Decision Tree:

Snow_Dist = <=100: ja (4.0)
Snow_Dist = >100:
|   Weekend = no: no (3.0)
|   Weekend = yes:
|   | Sun = no: no (1.0)
|   | Sun = yes: yes (3.0/1.0)
```

```
Simplified Decision Tree:

Snow_Dist = <=100: yes (4.0/1.2)
Snow_Dist = >100: no (7.0/3.4)
```

- Additionally, a simplified tree with only one attribute is given. This pruned tree is important for an increasing amount of training data.
- The error rate for both trees on the training data is given.

```
Evaluation on training data (11 items):

       Before Pruning              After Pruning
       ---------------    ---------------------------
       Size     Errors      Size     Errors     Estimate

        7     1(9.1%)        3      2(18.2%)    (41.7%)
```

- The numbers in parentheses after the decisions give the size of the underlying data set and the number of errors.
  - The line Sun = yes : yes (3.0/1.0) in the first tree indicates that out of three training examples at that leaf, one is falsely classified.

Decision Tree Construction Algorithm

GenerateDecisionTree(Data,Node)

$A_{\max}$ = Attribute with maximum information gain
If $G(A_{\max}) = 0$
  Then Node becomes leaf node with most frequent class in Data
  Else assign the attribute $A_{\max}$ to Node
    For each value $a_1, \ldots, a_n$ of $A_{\max}$, generate
      a successor node: $K_1, \ldots, K_n$
    Divide Data into $D_1, \ldots, D_n$ with $D_i = \{x \in \text{Data} : A_{\max}(x) = a_i\}$
    For all $i \in \{1, \ldots, n\}$
      If all $x \in D_i$ belong to the same class C_i
        Then generate leaf node K_i of class C_i
        Else GenerateDecisionTree(D_i, K_i)

Discretizing Continuous Attributes

- The C4.5 algorithm converts continuous attributes to binary ones by choosing thresholds.
- The threshold $\Theta_{D,A}$ for an attribute A is determined as follows:
  - For all values v which occur in the training data D, the binary attribute $A > v$ is generated and its information gain is calculated.
  - The threshold $\Theta_{D,A}$ is then set to the value v with the maximum information gain: $\Theta_{D,A} = \mathrm{argmax}_v\{G(D, A > v)\}$.
- Sometimes, a decision based on a binary discretization is too imprecise.
- But there is no need for finer discretization because each continuous attribute is tested on each newly generated node and can thus occur repeatedly in one tree with a different threshold $\Theta_{D,A}$.
- Ultimately, a very good discretization whose fineness fits the problem is obtained.

Occam's Razor

- Occam's Razor: Of two scientific theories which explain the same situation equally well, the simpler one is preferred.
- Occam's razor, is important for machine learning and data mining.
- Both a decision tree and the data themselves may explain the same situation equally well.
- If the tree classifies all data without any errors, but is much more compact and thus more easily understood by humans, then it is preferable according to Occam's razor.
- The same is true for two decision trees of different sizes.
- The goal of every algorithm for generating a decision tree must be to generate the smallest possible decision tree for a given error rate. Among all trees with a fixed error rate, the smallest tree should always be selected.

Overfitting

- It is important that the learned tree not just memorize the training data, but that it also generalize well.
- To test the ability of a tree to generalize, we divide the available data into a set of training data and a set of test data.
- A rule of thumb is, given a big data set, to use two-thirds of the data for learning and the remaining third for testing.
- Aside from better comprehensibility, Occam's razor has another important justification, generalization ability:
  - The more complex the model (here a decision tree), the more details are represented, but to the same extent the less is the model transferable to new data.
  - There is an optimum size after which the error rate starts to increase again. This effect we called **overfitting**.

Overfitting and Optimization

Definition (Overfitting)

A specific learning algorithm or learning agent A is **overfit to a set of training data** if there is another agent A' whose error on the training data is larger than that of A, but whose error on the whole distribution of data is smaller than the error of A.

- To find the point of minimum error on the test data **cross validation** may be used:
  - During construction, the error on the test data is measured in parallel.
  - When the error rises significantly, the tree with the min error is saved.
- This algorithm is used by the CART system mentioned earlier.
- C4.5 works somewhat differently:
  - Using GenerateDecisionTree, it builds a possibly overfit tree.
  - Using pruning, it cuts away nodes of the tree until the error on the test data, estimated by the error on the training data, begins to rise.
  - Like the construction of the tree, this is also a greedy algorithm: if a node is pruned, it cannot be re-inserted, even if this would be better.

Missing Values

- Individual attribute values may be missing from the training data.
- Such data can still be used during construction of the decision tree.
- We can assign the attribute the most frequent value from the whole data set or the most frequent of all data points from the same class.
- It is even better to substitute the probability distribution of all attribute values for the missing attribute value and to split the training example into branches according to this distribution.
- This is a reason for the occurrence of non-integer values in the expressions in parentheses next to the leaf nodes of the C4.5 tree.
- Missing values can occur not only during learning, but also during classification. These are handled in the same way as during learning.

Summary

- Learning of decision trees, due to its simplicity and speed, is a favorite approach to classification tasks.
- For the user it is also important that the decision tree as a learned model can be understood and potentially changed.
- A decision tree can be automatically transformed into a series of if-then-else statements and efficiently coded into an existing program.
- Because a greedy algorithm is used for construction of the tree as well as during pruning, the trees are in general suboptimal.
- The discovered decision tree usually has a relatively small error rate.
- However, there is potentially a better tree, because the heuristic greedy search of C4.5 prefers small trees and attributes with high information gain at the top of the tree.
- For attributes with many values, the presented formula for the information gain shows weaknesses and potential alternatives have been suggested.

Subsection 6

Learning of Bayesian Networks

Automatic Generation of Bayesian Networks

- After having seeing how to build a Bayesian network manually, we introduce algorithms for the induction of Bayesian networks.
- Similar to the learning process described previously, a Bayesian network is automatically generated from a file containing training data.
- This process is typically decomposed into two parts.
  1. Learning the network structure: For given variables, the network topology is generated from the training data. This first step is by far the more difficult one and will be given closer attention later.
  2. Learning the conditional probabilities: For known network topologies, the CPTs must be filled with values. If enough training data are available, all necessary conditional probabilities can be estimated by counting the frequencies in the data. This step can be automated relatively easily.

The Topology of the Network

- To build a Bayesian network the causal dependencies of the variables are taken into account to ensure simplicity and good quality.
- A human relies on background knowledge, which is unavailable to the machine, so this procedure cannot be easily automated.
- Finding an optimal structure for a Bayesian network can be formulated as a classic search problem:
  - Let a set of variables $V_1, \ldots, V_n$ and a file with training data be given.
  - We are looking for a set of directed edges without cycles between the nodes $V_1, \ldots, V_n$, i.e., a directed acyclic graph (DAG) which reproduces the underlying data as well as possible.
  - The number of different DAGs grows more than exponentially with the number of nodes, so an uninformed combinatorial search in the space of all graphs with a given set of variables is hopeless.
  - Therefore heuristic algorithms must be used, which poses the question of an evaluation function for Bayesian networks.
  - We could measure the classification error for a set of test data, (as in C4.5), but the calculated probabilities must be mapped to a decision.

Evaluating the Quality of a Network

- A direct measurement of the quality of a network can be taken over the probability distribution.
- We assume that, before the construction of the network from the data, we could determine (estimate) the distribution.
- Then we
  - begin the search in the space of all DAGs,
  - estimate the value of the CPTs for each DAG (that is, for each Bayesian network) using the data,
  - calculate the distribution using the CPTs and compare it to the distribution known from the data.
- For the comparison of distributions we will obviously need a distance metric.

Distance Metrics for Comparing Distributions

- Because, for constant predetermined variables, the distribution is clearly represented by a vector of constant length, we can calculate the **Euclidean norm** of the difference of the two vectors as a distance between distributions:

$$d_q(x, y) = \sum_i (x_i - y_i)^2.$$

- Often, instead of the square distance, the so-called **Kullback-Leibler distance**

$$d_k(x, y) = \sum_i y_i (\log_2 y_i - \log_2 x_i),$$

  an information theory metric, is used.
- It is to be expected that networks with many edges approximate the distribution better than those with few edges.

Heuristic Evaluation to Avoid Overfitting

- If all edges in the network are constructed, then it becomes very confusing and creates the risk of overfitting, as is the case in many other learning algorithms.
- To avoid overfitting, we give small networks a larger weight using a heuristic evaluation function

$$f(N) = \text{Size}(N) + w \cdot d_k(\mathbf{P}_N, \mathbf{P}),$$

where $\text{Size}(N)$ is the number of entries in the CPTs, $\mathbf{P}_N$ is the distribution of network N and w is a weight factor, which must be manually fit.

- The learning algorithm for Bayesian networks thus calculates the heuristic evaluation $f(N)$ for many different networks and then chooses the network with the smallest value.

Summary

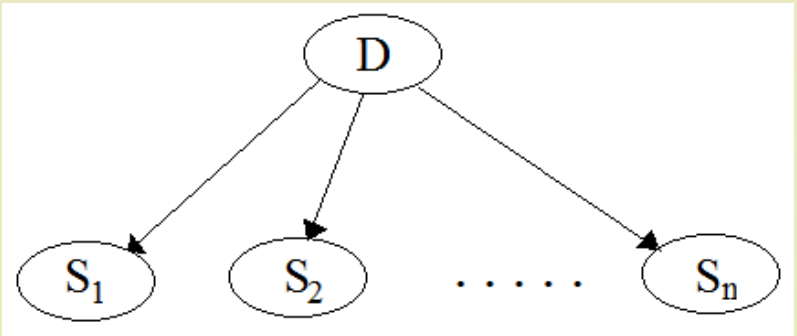
- The main difficulty is in reducing the search space for the network topology we are searching for.
- As a simple algorithm:
  - Start from a (e.g., causal) ordering of the variables $V_1, \ldots, V_n$ and include in the graph only those edges for which $i < j$.
  - Based, for example, on a greedy search, one edge after another is removed until the value f no longer decreases.
- This algorithm is not practical for larger networks in this form.
  - The large search space, the manual tuning of the weight w, and the necessary comparison with a goal distribution P may be too demanding or the available data set could be too small.
- Research is ongoing, with a large number of algorithms suggested:
  - Examples are the EM algorithm, Markov chain Monte Carlo methods, and Gibbs sampling.
- Besides batch learning (network is generated once from the whole data set), incremental algorithms exist, in which each new case is used to improve the network.

Subsection 7

The Naive Bayes Classifier

The Naive Bayes Classifier

- Consider variables $S_1, \ldots, S_n$ and the k-valued variable D with values $d_1, \ldots, d_k$.
- We calculate
$$P(D|S_1, \ldots, S_n) = \frac{P(S_1, \ldots, S_n|D) \cdot P(D)}{P(S_1, \ldots, S_n)}.$$
- If there are no independent variables, all combinations would need to be determined for all probabilities of the distribution $P(S_1, \ldots, S_n, D)$.
- If we assume all S_i's are conditionally independent given D,
$$P(S_1, \ldots, S_n|D) = P(S_1|D) \cdot \ldots \cdot P(S_n|D).$$
So $P(D|S_1, \ldots, S_n) = \frac{P(D) \prod_{i=1}^{n} P(S_i|D)}{P(S_1, \ldots, S_n)}.$



- The **naive Bayes classifier**:
$d_{\text{Naive\_Bayes}} = \text{argmax}_{i \in \{1,\ldots,k\}} P(D = d_i) \prod_{j=1}^{n} P(S_j|D)$.

Estimating the Probabilities

- If one of the factors $P(S_i|D)$ becomes zero, we get a zero;
- If the $P(S_i|D)$ are small, this can cause problems because of $P(S_i = x|D = y) = \frac{|S_i=x \wedge D=y|}{|D=y|}$.
- Example: Assume for S_i, $P(S_i = x|D = y) = 0.01$ and that there are 40 training cases with $D = y$ and similarly for $D = z$. With high probability there is no training case with $S_i = x$ and $D = y$, and we estimate $P(S_i = x|D = y) = 0$. For $D = z$, suppose the estimate is greater than zero for all $P(S_i = x|D = z)$. Thus the value $D = z$ is always preferred, not reflecting the actual probability distribution.
- When estimating probabilities, the formula $P(A|B) \approx \frac{|A \wedge B|}{|B|} = \frac{n_{AB}}{n_B}$ is replaced by $P(A|B) \approx \frac{n_{AB}+mp}{n_B+m}$, where $p = P(A)$ is the a priori probability for A, and m is a constant which can be freely chosen and is known as the "equivalent data size". The larger m becomes, the larger the weight of the a priori probability compared to the value determined from the measured frequency.

Text Classification Using Naive Bayes

- Naive Bayes is very successful in text classification, e.g., automatic filtering of email into desired and undesired (spam).
  - The user must at first manually classify a large number of emails.
  - Then the filter is trained and, to stay up to date, regularly retrained.
  - For this the user should correctly classify all emails which were falsely classified by the filter, that is, put them in the appropriate folders.
- Beside spam filtering, other applications include
  - filtering of undesired entries in discussion forums,
  - tracking websites with criminal content such as militant or terrorist activities, child pornography or racism.
- It can also be used to customize search engines, adapting to the habits and wishes of each individual user.
- Example: We'll do text analysis on a short text by Alan Turing.

A Text for Classification

> We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.

- Suppose that texts should be divided into two classes: "I" for interesting and "$\neg I$" for uninteresting.
- Suppose that a database exists of texts which are already classified.
- Which attributes should be used? In a classical approach to the construction of a Bayesian network, we define a set of attributes (e.g., length of the text, the average sentence length, the relative frequency of specific punctuation marks, frequency of key words, etc.)

Setting Up Naive Bayes

- For each of the n word positions in the text, an attribute s_i is defined. All words that occur in the text are possible values for all s_i.
- For the classes I and $\neg I$ the values

$$P(I|s_1, \ldots, s_n) = c \cdot P(I) \prod_{i=1}^{n} P(s_i|I)$$

  and $P(\neg I|s_1, \ldots, s_n)$ must be calculated and then the class with the maximum value selected.
- In the example with 113 words, this yields

$$\begin{aligned} P(I|s_1, \ldots, s_n) &= c \cdot P(I) \cdot P(s_1 = \text{“We”}\,|I) \cdot P(s_2 = \text{“may”}\,|I) \cdot \\ &\quad \ldots \cdot P(s_{113} = \text{“should”}\,|I) \\ P(\neg I|s_1, \ldots, s_n) &= c \cdot P(\neg I) \cdot P(s_1 = \text{“We”}\,|\neg I) \cdot P(s_2 = \text{“may”}\,|\neg I) \cdot \\ &\quad \ldots \cdot P(s_{113} = \text{“should”}\,|\neg I) \end{aligned}$$

- The conditional probabilities $P(s_i|I), P(s_i|\neg I)$ and the a priori probabilities $P(I), P(\neg I)$ must be calculated.

Equivalent Form Based on Independence

- We assume that the $P(s_i|I)$ are not dependent on position in the text, so $P(s_{61} = \text{“}and\text{”}|I) = P(s_{69} = \text{“and”}|I) = P(s_{86} = \text{“and”}|I)$. So, the expression $P(\text{and}|I)$ is a binary variable giving the probability of the occurrence of “and” at an arbitrary position.
- The implementation can be slightly accelerated if we find the frequency n_i of every word w_i which occurs in the text and use

$$P(I|s_1, \ldots, s_n) = c \cdot P(I) \prod_{i=1}^{\ell} P(w_i|I)^{n_i}.$$

- The index i in the product only goes to the number ℓ of different words which occur in the text.
- Despite its simplicity, naive Bayes delivers excellent results for text classification. Spam filters which work with naive Bayes achieve error rates of well under one percent.

Subsection 8

Clustering

Idea Behind Clustering

- If we search in a search engine for "mars", we will get results both about he planet and the chocolate which are semantically different.
- In the set of discovered documents there are two different clusters.
- The user is usually interested in only one of the clusters.
- Accordingly, it would be preferable if the search engine separated the clusters and presented them separately to the user.
- The distinction of clustering in contrast to supervised learning is that the training data are unlabeled.
  - Thus the pre-structuring of the data by the supervisor is missing.
  - Rather, finding structures is the whole point of clustering.
- In a cluster, the distance of neighboring points is typically smaller than the distance between points of different clusters.
- The choice of a suitable distance metric for points, that is, for objects to be grouped and for clusters, is of fundamental importance.
- We assume, as before, that every data object is described by a vector of numerical attributes.

Distances Used in Clustering

- For each application, the various distance metrics are defined for the distance d between two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$.
- The most common is the Euclidean distance

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

- The sum of squared distances is $d_q(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$. For algorithms in which only distances are compared, it is equivalent to the Euclidean distance.
- Also used is the Manhattan distance

$$d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|.$$

- Another is the distance of the maximum component

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1,\ldots,n} |x_i - y_i|.$$

Clustering in Text Classification

- During text classification, the normalized projection of the two vectors on each other, that is, the normalized scalar product $\frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}||\mathbf{y}|}$ is frequently calculated, where $|\mathbf{x}|$ is the Euclidian norm of $\mathbf{x}$.
- Because this formula is a metric for the similarity of the two vectors, its inverse $d_s(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x}||\mathbf{y}|}{\mathbf{x} \cdot \mathbf{y}}$ is taken as a distance metric.
- In the search for a text, the attributes $x_1, \ldots, x_n$ are calculated similarly to naive Bayes as components of the vector $\mathbf{x}$:
  - For a dictionary with 50,000 words, the value x_i equals the frequency of the i-th dictionary word in the text.
  - Since normally almost all components are zero, during the calculation of the scalar product, nearly all terms of the summation are zero.
  - By exploiting this kind of information, the implementation can be made significantly faster.

The k-Means Algorithm

- Whenever the number of clusters is already known in advance, the k-means algorithm can be used.
- k clusters are defined by their average value.
- First the k cluster midpoints $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ are randomly or manually initialized.
- Then the following two steps are repeatedly carried out:
  - Classification of all data to their nearest cluster midpoint;
  - Recomputation of the cluster midpoint $\boldsymbol{\mu} = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathbf{x}_i$.

K-MEANS$(\mathbf{x}_1, \ldots, \mathbf{x}_n, k)$

Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ (e.g. randomly)
Repeat
  classify $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to each's nearest $\boldsymbol{\mu}_i$
  recalculate $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
Until no change in $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
Return$(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$

Features and Alternatives

- The K-Means algorithm does not guarantee convergence, but it usually converges very quickly.
- This means that the number of iteration steps is typically much smaller than the number of data points.
- Its complexity is $O(ndkt)$, where n is the total number of points, d the dimensionality of the feature space, and t the number of iteration steps.
- In many cases, the necessity of giving the number of classes in advance poses an inconvenient limitation. The HierarchicalClustering algorithm is more flexible, alleviating this problem.
- Before that, we will mention the EM algorithm, which is a continuous variant of K-Means: it does not make a firm assignment of the data to classes, but, instead, it returns, for each point, the probability of it belonging to the various classes.

The EM Algorithm

- We assume that the type of probability distribution is known.
- Often the normal distribution is used.
- The task of the EM algorithm is to determine the parameters (mean $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$ of the k multi-dimensional normal distributions) for each cluster.
- Similarly to k-means, the two following steps are repeatedly executed:
  - Expectation: For each data point the probability $P(C_j|x_i)$ that it belongs to each cluster is calculated.
  - Maximization: Using the newly calculated probabilities, the parameters of the distribution are recalculated.
- Thereby a softer clustering is achieved, which in many cases leads to better results.
- EM: Alternation between Expectation and Maximization.
- The EM algorithm is used, e.g., in learning Bayesian networks.

Hierarchical Clustering

- In hierarchical clustering:
  - We begin with n clusters consisting of one point each.
  - Then the nearest neighbor clusters are combined until all points have been combined into a single cluster, or until a termination criterion has been reached.

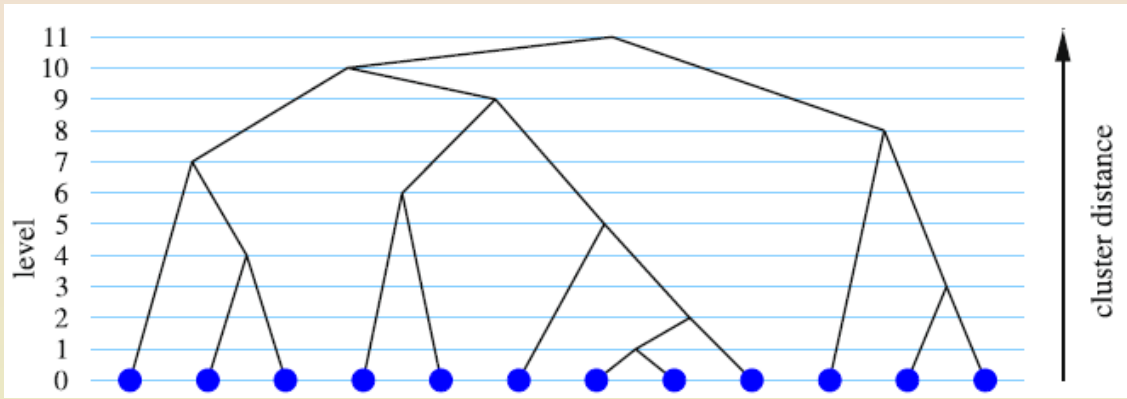
HierarchicalClustering$(\mathbf{x}_1, \ldots, \mathbf{x}_n, k)$

Initialize $C_1 = \{\mathbf{x}_1\}, \ldots, C_n = \{\mathbf{x}_n\}$
Repeat
  Find two clusters C_i and C_j with the smallest distance
  Combine C_i and C_j
Until Termination condition reached
Return(tree with clusters)

- The termination condition could be chosen as a desired number of clusters or a maximum distance between clusters.

Illustration of Hierarchical Clustering

- A schematic illustration of hierarchical clustering uses a binary tree, in which from bottom to top in each step two subtrees are connected.



- How are the distances between the clusters calculated?
- The distance metrics for points cannot be used on clusters.
- A convenient and often used metric is the distance between the two closest points in the two clusters C_i and C_j:

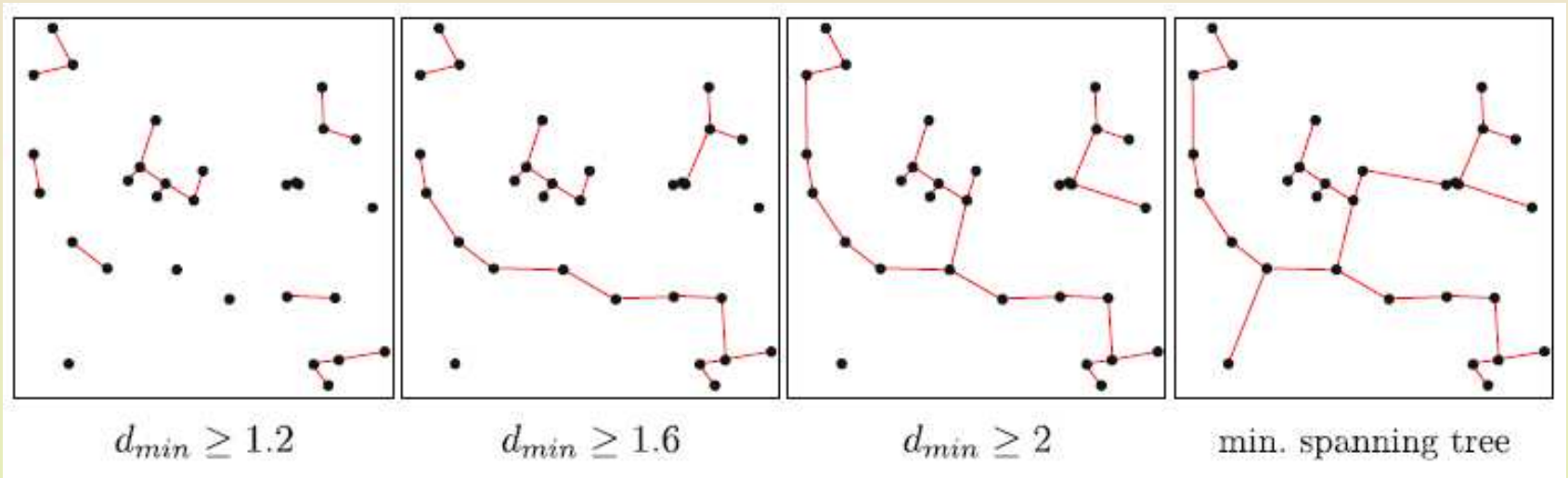
$$d_{\min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}).$$

Hierarchical Clustering With $d_{\min}$

- The distance between the two closest points in the two clusters C_i and C_j:

$$d_{\min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}).$$

- We obtain the nearest neighbor algorithm:



- This algorithm generates a minimum spanning tree.
- For graphs with clusters which are not clearly separated, the result depends heavily on the algorithm or the chosen distance metric.

Efficiently Implementing Hierarchical Clustering

- For an efficient implementation, we first create an adjacency matrix in which the distances between all points is saved.
- This requires $O\left(n^2\right)$ time and memory.
- If the number of clusters does not have an upper limit, the loop will iterate $n-1$ times and the computation time needed is $O\left(n^3\right)$.
- To calculate the distance between two clusters, we can also use the distance between the two farthest points

$$d_{\max}(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}).$$

- Alternatively, the distance of the cluster's midpoint may be used

$$d_{\mu}(C_i, C_j) = d(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j).$$

- Many other variants of the clustering algorithm are possible.

Subsection 9

Data Mining Tools and Summary

Data Mining in Practice

- All the learning algorithms seen so far can be used in data mining.
- For the user it is rather inconvenient to
  - get used to new software tools for each application;
  - have to put the data into the appropriate format each time.
- A number of data mining systems address these problems.
- Most of these systems offer a convenient graphical user interface with diverse tools for visualization of the data, for preprocessing, such as manipulation of missing values, and for analysis.
  - WEKA offers a large number of algorithms and simplifies the development of new algorithms.
  - KNIME offers a convenient user interface and all the types of tools mentioned above. KNIME also uses WEKA modules. Furthermore it offers a simple way of controlling the data flow of the chosen visualization, preprocessing, and analysis tools with a graphical editor.
  - Other systems include RapidMiner, Clementine and the KXEN analytic framework.

Summary: Supervised and Unsupervised Learning

- We have thoroughly covered several algorithms from the established field of supervised learning, including decision tree learning, Bayesian networks, and the nearest neighbor method.
- These algorithms are stable and efficiently usable in applications and, thus, belong to the standard repertoire in AI and data mining.
- The same is true for the clustering algorithms, which work without a "supervisor" and can be found, for example, in search engines.
- Reinforcement learning as another field of machine learning uses no supervisor either. In contrast to supervised learning, only now and then is positive or negative feedback received from the environment.
- In semi-supervised learning only very few out of a large number of training data are labeled.
- For supervised learning of data with continuous labels any function approximation algorithm, such as neural networks, least squares, support vector machines, can be employed.

Summary: Taxonomy of Learning Algorithms

- Supervised learning
  - Lazy learning
    - k nearest neighbor method (classification + approximation);
    - Locally weighted regression (approximation)
    - Case-based learning (classification + approximation)
  - Eager learning
    - Decision trees induction (classification)
    - Learning of Bayesian networks (classification + approximation)
    - Neural networks (classification + approximation)
    - Gaussian processes (classification + approximation)
    - Wavelets, splines, radial basis functions, . . .
- Unsupervised learning (clustering)
  - Nearest neighbor algorithm
  - Farthest neighbor algorithm
  - k-means
  - Neural networks
- Reinforcement learning
  - Value iteration
  - Q learning
  - TD learning
  - Policy gradient methods
  - Neural networks

Summary: Unknown Set of Attributes

- What has been said about supervised learning is only true, however, when working with a fixed set of known attributes.
- An interesting, still open field, is automatic feature selection.
- In learning with decision trees, we presented an algorithm for the calculation of the information gain of attributes that sorts the attributes according to their relevance. and uses only those which improve the quality of the classification.
- With this type of method it is possible to automatically select the relevant attributes from a potentially large base set. This base set, however, must be manually selected.
- Still open and also not clearly defined is the question of how the machine can find new attributes.

Summary: Feature Selection Through Clustering

- Clustering provides one approach to feature selection.
- Before training a machine to pick apples among a collection of fruits, we let clustering run on the data.
- For (supervised) learning of the classes apple and non apple, the input is no longer all of the pixels, rather only the classes found during clustering, potentially together with other attributes.
- Clustering can be used for automatic, creative "discovery" of features. On the other hand, it is uncertain whether the discovered features are relevant.
- If the video camera used for apple recognition only transmits black and white images, the ability to classify is reduced.
- In that case, it would be nice if the machine would be creative on its own account, for example by suggesting that the camera should be replaced with a color camera.
- This is beyond today's AI capabilities.