2.5 Nearest neighbor methods

- Analogous to the linear perceptron, we consider a problem setting that is characterized by a data set M of known cases
  - Each case $x \in M$ is defined as a vector $x \in \mathbb{R}^n$ of attribute values describing the respective setting
  - Moreover, each case is classified, i.e., the class of each case in the data set is known
  - Based on these cases, further cases have to be classified
- For this purpose, the learning algorithm of the linear perceptron iteratively transforms the knowledge exhausted from the data set into a single weight vector
- This is a **significant compression of the available data** into one separating vector, i.e., from a considerable set of vectors into one vector that separates the entire data set into two parts
- An **alternative approach is to keep all available vectors** (learning them by heart) for the purpose of a **direct detailed comparison with new cases** in order to derive a more reliable classification

Nearest neighbor methods – motivation

- For this purpose, a considered vector to be classified is categorized according to the **known classification of its direct neighborhood** in the given data set
- The **neighborhood of a vector results from an applied distance measure**
- As knowledge is not processed or transformed before it is applied to classify new cases, this technique is categorized as a **special form of lazy learning with a significant memory consumption**

Problem

- Given: Data set M with N vectors of the $\mathbb{R}^n$, i.e., $x_1, x_2, \dots, x_N \in \mathbb{R}^n$ with a known classification $c: \{1, \dots, N\} \mapsto \{1, \dots, C\}$ into $C \in \mathbb{N}$ predetermined classes and a new vector $s \in \mathbb{R}^n$
- Sought: Classification of the vector $s \in \mathbb{R}^n$ by comparing it with the known cases of the data set
- Possible applications
  - Diagnosis systems in medical applications
  - Pattern recognition (see the last example of the perceptron algorithm)
  - Classification of customers in social networks

Distance measures

- The distance measure determines for each known element of the given data set the similarity of this already classified case to cases currently not classified
- For this purpose, various distance measures can be applied
- For instance, the **Euclidean distance measure** is frequently applied, i.e.,

$$\forall x, y \in \mathbb{R}^n: d(x,y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- By using existing weights for the different attributes, we obtain

$$\forall x, y, w \in \mathbb{R}^n: d_w(x,y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} w_i \cdot (x_i - y_i)^2}$$

Nearest neighbor

- The nearest neighbor classifier determines the classification of a given case $x \in \mathbb{R}^n$ **solely by evaluating the classification of its nearest neighbor** $y \in M$
- Thus, we can formalize this method as follows:
- Given: Data set $M \subseteq \mathbb{R}^n$ with classification mapping $c: \{1, \dots, N\} \mapsto \{1, \dots, C\}$ into $C \in \mathbb{N}$ predetermined classes, a distance measure $d(x, y) \in \mathbb{R}$ for two vectors $x, y \in \mathbb{R}^n$, and a new vector $s \in \mathbb{R}^n$ to be classified

Nearest Neighbor(M, s)

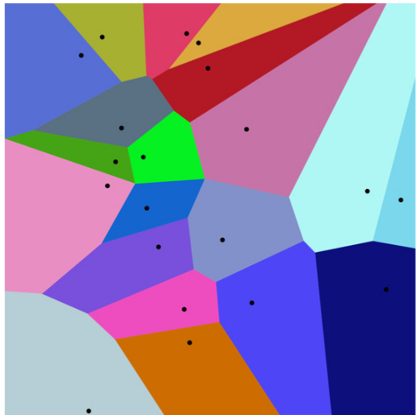
$y \coloneqq argmin\{d(s, x) \mid x \in M\};$

return$\big(c(y)\big)$

Voronoi diagram

- A Voronoi diagram of N points (denoted as seeds, sites, or generators) in the $\mathbb{R}^n$ is a partitioning of the $\mathbb{R}^n$ such that
  - Each seed constitutes a different subsets of the $\mathbb{R}^n$ and
  - each point $x \in \mathbb{R}^n$ belongs to the subset constituted by the seed that is closest located to x (of all seeds)
- Therefore, based on the given data set M, the nearest neighbor method provides a partitioning (and subsequent clustering) of all unclassified vectors according to the Voronoi diagram
- As each subset in a Voronoi diagram is obtained from the intersection of half spaces, such subsets are convex polygons. Moreover, line segments of the Voronoi diagram are all the points that are equidistant to the two nearest seeds. The Voronoi vertices are the points equidistant to three (or more) seeds
- Hence, separations done by the nearest neighbor method are much more flexible than the linear separations of the linear perceptron

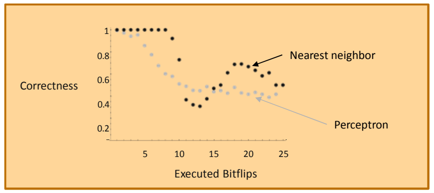
Voronoi diagram – illustration



File:Euclidean Voronoi diagram.svg
Created: 22 February 2015

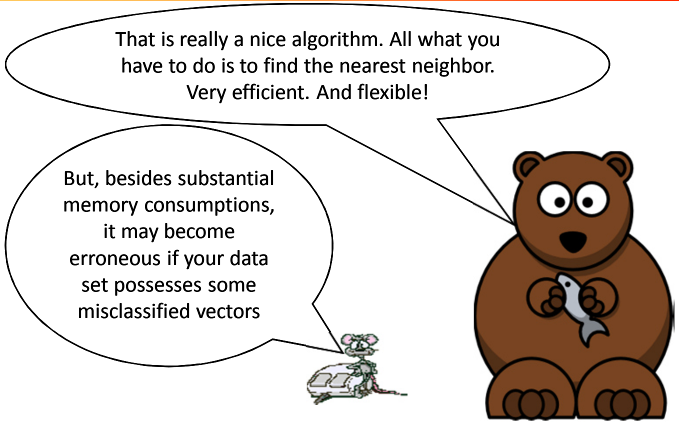
Comparison with the linear perceptron

- By comparing the application of the nearest neighbor method to the pattern recognition example (complicated by inverted bits) with the linear perceptron, Ertel (2016) reports the correctness values depicted below in dependence of the number of inverted bits
- It is worth mentioning that the Hamming distance between the second case of set M_+ and the cases 4 and 5 (belonging to set M_-) is 9
- Therefore, the 100 percent correctness significantly falls with increasing the number of inverted bits to this threshold (> 8)



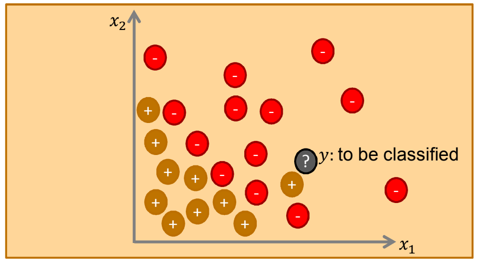
(See Ertel (2016) p.210)

Nearest neighbor – Easy to use



One neighbor decisions may be erroneous

- Even if there is only a small number of erroneously classified vectors in the data set, these few cases may spread out
- The following figure illustrates how falsely classified cases may spread out if the vector to be newly classified (by the nearest neighbor method) is closest located to such an erroneous case.
- Such an erroneous adaption is a form of overfitting



k-nearest neighbor

- In order to **reduce the number of misclassified cases**, the nearest neighbor method is often extended to the **k-nearest neighbor method**
- Depending on the given parameter k, this method classifies a new vector **according to the known classification of the k nearest neighbors**
- Here, the classification is assigned that is **most frequently present** among these k nearest neighbors
- This leads to the following modified procedure

k-**Nearest Neighbor**(M, k, s)

Compute $V \subseteq M$ as the set of k nearest neighbors of s in M;
Set $\forall l \in \{1, \dots, C\}$: $V_l := \{x \mid x \in V \wedge c(x) = l\}$
Set $m := argmax\{|V_l| \mid l \in \{1, \dots, C\}\}$
return(m)

Finding appropriate values for parameter k

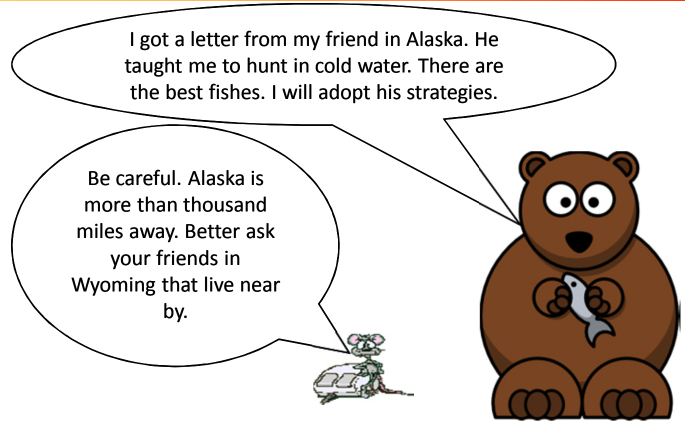
- The choice of the parameter k may have considerable consequences for the efficiency of the approach
- Small values of k may not sufficiently eliminate the negative influence of erroneously classified cases in the data set
- Large values of k may increase the impact of cases that are not representative for the case to be classified. This results from the fact that (more) remote cases are additionally integrated. As these farer away located cases do not provide adequate decision support for the classification of the currently considered case, the classification may be distorted
- Note that the latter problem can be mitigated by additionally applying distance-dependent weights (see below)

Approximation

- Another application problem of the k-nearest neighbor method emerges when the number of classes that the cases have to be assigned to increase significantly
- Then, the classification is frequently complicated by the fact that, due to the significant number of classes, the number of relevant known cases is strongly limited
- Moreover, due to the numerous classes, the respective applications may benefit from a continuous classification provided by a continuous function
- For this purpose, the average value of the classification of all relevant k cases in set $V = \{x_1, x_2, \ldots, x_k\} \subseteq M$ is computed, i.e., we classify the considered vector x by

$$c(x) = \frac{1}{k} \cdot \sum_{i=1}^{k} c(x_i)$$

k-nearest neighbor – Every case is significant?



Considering the distance

- First of all, it has to be stated that the k considered neighbors that are used by the k-nearest neighbor approach for classifying a considered case are equally weighted, i.e., irrespective of their significance or representativity all these cases are equally handled
- Therefore, by increasing the parameter k, the number of cases (integrated in the classification) that possess a considerable distance to the case to be classified may substantially increase
- Hence, the significance or representativity of these cases may become quite small

Integrating weights

- Therefore, for the determination of the sought class $c(x)$ each known case $x_i \in V$ is weighted according to its distance to x, i.e., by the weight

$$w_i = \frac{1}{1 + \alpha \cdot d(x, x_i)}$$

- The factor α determines how fast the influence of $x_i \in V$ is reduced with an increased distance to x
- Hence, in case of the approximation, we obtain

$$c(x) = \frac{\sum_{i=1}^{k} w_i \cdot c(x_i)}{\sum_{i=1}^{k} w_i}$$

- In case of a discrete classification, it is possible to assign case x to the class with a maximum total weight (see next slide)

Discrete classification with weights

k-**Nearest Neighbor**(M, k, s)

Compute $V \subseteq M$ as the set of k nearest neighbors of s in M;

Set $\forall l \in \{1, \ldots, C\}$: $V_l := \{x \mid x \in V \wedge c(x) = l\}$

Set $m := argmax\left\{\sum_{x_j \in V_l} w_j \cdot c(x_j) \mid l \in \{1, \ldots, C\}\right\}$

return(m)

- A further extension is to integrate an approach of exponential smoothing into the weighting of neighboring cases
- Specifically, depending on a discretization of the continuous distance $\chi(d(x_i, x))$, the weight of each case in the data set is discounted exponentially with a discount rate α

$$w_i = \frac{1}{1 + \alpha^{\chi(d(x_i,x))}}$$

Computational effort

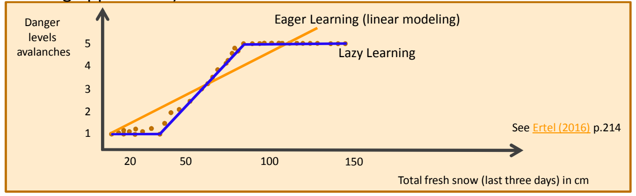
- The k-nearest neighbor method requires a considerable memory consumption since it is necessary to store all cases given by the data set M
- Moreover, the classification of a currently considered case x may become quite time consuming as the determination of the k closest located cases of set M (i.e., the determination of set V) requires to consider each case, i.e., we have $O(|M|)$. Depending on the used data structure, it can become to $O\big(|M| \cdot log_2(|M|)\big)$. In addition to this, the classification takes at least time proportional to $O(C)$. Therefore, all in all, a minimum time complexity of $O(|M| + k)$ occurs
- For application with large data sets, this may be too time consuming, in particular, if a considerable number of classifications have to be done in real-time

Eager Learning – Lazy Learning

- As k-nearest neighbor does not further process or modify the given data set in order to exploit knowledge, **all effort is relinked to the final evaluation or classification step**
- Therefore, the k-nearest neighbor method is denoted as a lazy learning approach
- In contrast to this, eager learning approaches spend much more effort in the learning phase that exploits knowledge from the given data set in order to enable fast classifications
- Eager Learning approaches are for instance:
  - Perceptron
  - Decision tree
  - Bayes networks
  - Neural networks

Comparing eager and lazy learning

- Ertel (2016) gives the following comparison of eager and lazy learning
  - Eager learning usually transforms the raw data in the data set into a modeling
  - I.e., eager learning compresses the data to a mathematical structure, as, for instance, a linear function
  - In contrast to this, nearest neighbor (as a lazy learning method) does a local assignment that is often more precise (outperforms the ones done by eager learning approaches)



See Ertel (2016) p.214

https://www.natural-hazards.ch/home/dealing-with-natural-hazards/avalanches/danger-levels-avalanches.html

When to use nearest neighbor methods?

- Therefore, the nearest neighbor method can be reasonably applied if the entire data set can be efficiently stored and evaluated in the available time
- Particularly, if the classification has to guarantee a high local precision the nearest neighbor method outperforms many eager learning approaches
- However, if one of the first two requirements is not fulfilled or if knowledge stored in the raw data set has to be transformed into an understandable modeling (for analysis purposes), nearest neighbor methods are not the best choice

2.6 Ensemble Learning and Random Forests

- In what follows, we consider a somewhat surprising kind of approaches
- These approaches are not original in terms of generating and applying a new sophisticated technique that provides more reliable classifications or predictions, but are innovative in the sense that they propose to orchestrate a variety of known approaches providing numerous results in parallel in order to derive (out of these set of results) a more reliable decision
- First, we would like to motivate the basic idea behind this concept

2.6.1 Motivation – Playing a game

Motivation – Three choices

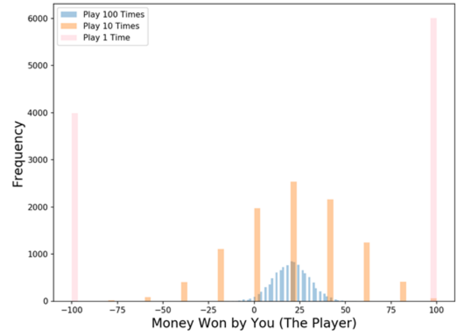
We analyze the three games

- Clearly, the expected values are identical in all games
  - Expected Value Game 1: $0.60 \cdot 100 + 0.40 \cdot (-100) = 20$
  - Expected Value Game 2: $(0.60 \cdot 10 + 0.40 \cdot (-10)) * 10 = 20$
  - Expected Value Game 3: $(0.60 \cdot 1 + 0.40 \cdot (-1)) * 100 = 20$
- But, the distributions are different in the three games
- This can be visualized by doing 10,000 Monte Carlo simulations of the three games
- This is illustrated by the figure depicted on the following slide

(See Understanding Random Forest)

The three distributions

By comparing the distributions, it becomes clear that **the bear makes money** in **60 %** of the simulations playing **game 1**, in **63 %** of the simulations playing **game 2**, but **even in 97 % of the simulations playing game 3**



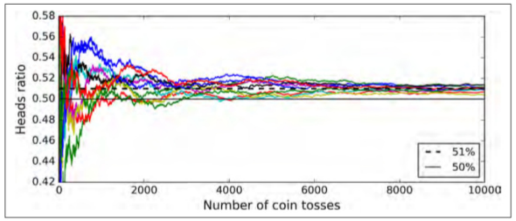
(See Understanding Random Forest)

Ensemble Learning – Motivation

- Although the expected values are identical, the positive effect becomes significantly reliable by splitting the game into more and more rounds
- This is the basic idea of ensemble learning
- By putting together various independent classifiers, we get a much more reliable classification
- Note that this does not even require classifiers of high quality
- In order to understand this, let us consider various binary classifiers that are independent and classify a given case into one of the two possible classes
- We assume that each classifier correctly classifies with a probability of only 51 percent, i.e., slightly better than guessing

The law of large numbers is helpful

- This can be illustrated by Monte Carlo simulations of tossing a slightly biased coin (51:49 for head)
- With a large number of coin tosses (>6,000) we observe that all conducted simulations attain a heads ratio of over 50 percent
- With other words, although that each classification is only slightly better than a 50:50 guess, a large number of independent repetitions results in a reliable classification whenever we decide for the majority of votes



See Géron (2017) p.183

Ensemble learning

- This is just the basic idea of ensemble learning
- Instead of applying one approach or method to decide about a considered classification, apply numerous
- However, one main prerequisite (of applying the aforementioned law of large numbers) is that these classifiers are independent
- Hence, Géron (2017, 2019) states
  - Ensemble methods work best when the predictors are as independent from one another as possible
  - For this purpose, it is reasonable to train the classifiers by using very different algorithms
  - As this increases the chance that the classifiers will make very different types of errors, the ensemble's accuracy is improved
- Roughly speaking, in order to establish a variety of different predictors (classifiers), ensemble learning proposes two concepts: **Bagging** and **Boosting**

2.6.2 Bagging (according to Breiman (1996))

- We consider a learning set $\mathcal{L} = \{ (y_i, x_i) \mid i = 1, \dots, N \}$ with vectors of attribute values $\forall i \in \{1, \dots, N\}: x_i \in \mathbb{R}^n$ and a corresponding classification y_i that is either a class label (i.e., $y_i \in \{1, \dots, C\}$, with $C \in \mathbb{N}$) or a numerical response (i.e., $y_i \in \mathbb{R}$).
- We assume there is a predictor $\varphi(x, \mathcal{L})$ that predicts the y-value according to the input $x \in \mathbb{R}^n$ and based on the learning set $\mathcal{L}$
- Now, we assume that there is a sequence of learning sets $\{\mathcal{L}_1, \dots, \mathcal{L}_k, \dots\}$ each consisting of N independent observations from the same underlying distribution as $\mathcal{L}$
- The mission is to use the learning sets $\{\mathcal{L}_1, \dots, \mathcal{L}_k, \dots\}$ in order to obtain an improved predictor than the single learning set predictor $\varphi(x, \mathcal{L})$ introduced above

Bagging (according to Breiman (1996))

- If y is numerical, we replace $\varphi(x, \mathcal{L})$ by the average of $\varphi(x, \mathcal{L}_k)$ over all generated learning sets $\mathcal{L}_k$. By theoretically considering all possible learning sets we approach the averaging value $\varphi_A(x) \coloneqq E_{\mathcal{L}}\big(\varphi(x, \mathcal{L})\big)$, with the expectation $E_{\mathcal{L}}$ over all learning sets $\mathcal{L}$ for $\varphi(x, \mathcal{L})$
- If y is a class label, we conduct a voting of all predictors and take the one with the most votes, i.e., with $N_j = |\{\mathcal{L}_k \mid \varphi(x, \mathcal{L}_k) = j\}|, \forall j \in \{1, \dots, C\}$, we set $\varphi_A(x) \coloneqq argmax\{N_j \mid j \in \{1, \dots, C\}\}$
- However, in real-world applications, we have only one learning set $\mathcal{L}$ without the luxury of replicates

Bagging (according to Breiman (1996))

- We do the following to imitate the aforementioned process
  - Take repeated bootstrap samples $\{\mathcal{L}^{(B)}\}$ from $\mathcal{L}$ and form $\{\varphi(x, \mathcal{L}^{(B)})\}$
  - If y is numerical, we set $\varphi_B(x) = av_B\left(\varphi(x, \mathcal{L}^{(B)})\right)$ (i.e., we take the average value over all bootstrap samples $\{\mathcal{L}^{(B)}\}$)
  - If y is a class label, we let the predictors of set $\{\varphi(x, \mathcal{L}^{(B)})\}$ vote to determine $\varphi_B(x)$
- This procedure is denoted as "**b**ootstrap **agg**regat**ing**" while the acronym **bagging** is used
- The bootstrap samples $\{\mathcal{L}^{(B)}\}$ each consisting of $N' \leq N$ cases are **drawn at random** from $\mathcal{L}$, **BUT with replacement** (otherwise, for the common setting $N' = N$ there would be all identical to $\mathcal{L}$ as this set also comprises N cases)
- Thus, each item $(y_n, x_n) \in \mathcal{L}$ may appear repeated times or not at all in some $\mathcal{L}^{(B)}$

Bagging (according to Breiman (1996))

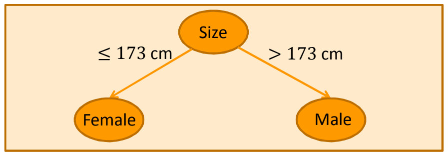
- Intention: The bootstrap samples $\{\mathcal{L}^{(B)}\}$ are replicate data sets drawn from the bootstrap distribution approximating the distribution underlying $\mathcal{L}$
- Frochte (2018) p.155 reports that, for the common setting $N' = N$, the proportion of items of the original training set $\mathcal{L}$ that are inserted in $\mathcal{L}^{(B)}$ approximates $1 - \frac{1}{e}$ for large values of N. These are about 0.63 percent. The remaining 37 percent are repeated items
- If the setting $N' < N$ is applied, the procedure is also denoted as **Subagging**

2.6.3 Boosting

- Boosting and Bagging are strongly related
- In both cases, the different classifications of various given predictors are combined into one (hopefully better) prediction
- The basic idea of boosting is to generate a strong or stronger predictor by using various weaker ones
- In contrast to bagging where all predictors are independently generated in parallel, **boosting derives the predictors iteratively while using the temporary results provided by the preceding steps** in order to derive more reliable classifications
- By doing so, Boosting is a general method for improving the performance of any learning algorithm (see Freund and Schapire (1996))

Decision stumps

- Are one-level decision trees that are used as predictors
- Thus, a stump comprises only one “inner node”, namely the root node itself
- This single node is directly connected with the terminal nodes
- Consequently, there is only one rule (one input feature) that is applied to decide about a classification
- Depending on the classification, various settings are thinkable. For instance, if there is a nominal feature there may be a stump with a leaf for each possible value whereas, for continuous features, threshold values are applied in order to separate the cases into items with attribute values below or above the threshold



Boosting approach AdaBoost

- In what follows, we consider in detail the AdaBoost algorithms originally proposed by Freund and Schapire (see the papers: Freund and Schapire (1996), Freund and Schapire (1997), and Freund and Schapire (1999)) as well as extended and adpated by various authors
- For instance Friedman, J.; Hastie, T.; Tibshirani, R. (1998) state that “Breiman (1996) (referring to a NIPS workshop) called AdaBoost with trees the “best off-the-shelf classifier in the world”
- Therefore, the following part tries to provide an overview of and an introduction to this specific approach
  - First, AdaBoost is introduced and defined as a binary classifier (this part is mainly adopted from the talks of Matas and Šochman and Šochman and Matas
  - Second, we consider/mention some extensions
  - Third, the paper Freund and Schapire (1996) is considered as it comprises an empirical comparison of boosting and bagging. Both related methods are tested with different predictors

Some facts

"Historic" development

- 1990 – Boost-by-majority algorithm (Freund)
- 1995 – AdaBoost (Freund & Schapire)
- 1997 – Generalized version of AdaBoost (Schapire & Singer)
- 2001 – AdaBoost in Face Detection (Viola & Jones)

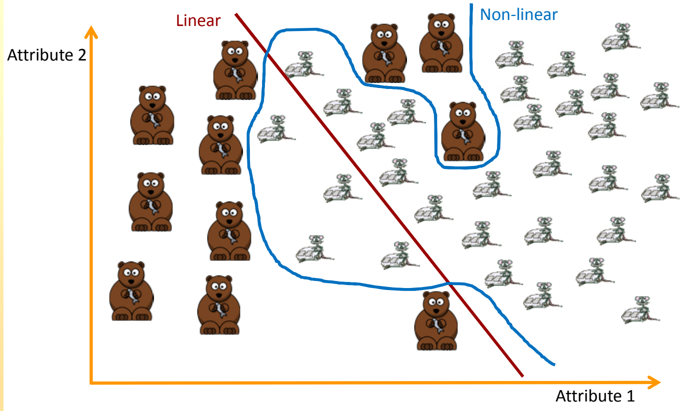
Properties

- AdaBoost combines several (weak) classifiers
- AdaBoost is frequently able to reduce bias or variance
- AdaBoost is close to sequential decision making by producing a sequence of gradually more complex classifiers

Introducing AdaBoost

- First, we introduce AdaBoost as a binary classifier
- I.e., it predicts the classification of cases according to two classes
- For technical reasons, in what follows, the two classes are denoted by the values "$+1$" and "-1"

AdaBoost covers non-linear classifications



Given and sought

- Given: $(x_1, y_1), \dots, (x_m, y_m); \forall i \in \{1, \dots, m\}$: $x_i \in X, y_i \in \{-1, +1\}$
- Sought: A predictor (final classifier) $H(x) = sign(\sum_{t=1}^{T} \alpha_t \cdot h_t(x))$, with
  $$\forall x \in \mathbb{R}: sign(x) = \begin{cases} 1 & if\, x > 0 \\ 0 & if\ x = 0 \\ -1 & if\ x < 0 \end{cases}$$
- Note that the "zero case" is infeasible since we require a binary classification into $\{-1, +1\}$
- Therefore, this case is handled by randomly drawing -1 or 1 (each case has probability 0.5)

The basic procedure AdaBoost – Part 1

1. Initialize the weight in the first round of each case by setting $D_1(i) = \frac{1}{m}$

Repeat the following steps for round $t = 1, \dots, T$:

2. Find a weak predictor $h_t: X \mapsto \{-1, +1\}$ that minimizes the resulting error, i.e., if $\mathcal{X}$ denotes the set of all feasible predictor functions, h_t is defined as follows:

$$h_t = argmin\{\epsilon_j \mid \epsilon_j = \sum_{i=1}^{m} D_t(i)[\![y_i \neq h_j(x_i)]\!] \wedge h_j \in \mathcal{X}\},$$

with for each predicate a it holds that: $[\![a]\!] = \begin{cases} 1 & if\ a\ is\ true \\ 0 & if\ a\ is\ not\ true \end{cases}$

Thus, the chosen predictor h_t causes a prediction error ϵ_t

3. If $\epsilon_t \geq \frac{1}{2}$ (not better than guessing) then stop
4. Set $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

The basic procedure AdaBoost – Part 2

5. Update the given distribution (i.e., the weights of the m training cases)

$$D_{t+1}(i) = D_t(i) \cdot \frac{e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{Z_t} = \frac{D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{Z_t}$$

In this calculation Z_t is a normalization factor that ensures that D_{t+1} provides a distribution of all cases, i.e., $Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}$

6. End of round t
7. Output the final classifier: $H(x) = sign(\sum_{t=1}^{T} \alpha_t \cdot h_t(x))$

Furthermore, we define $f(x) = \sum_{t=1}^{T} \alpha_t \cdot h_t(x)$

Weak Learner

- The boosting algorithm has access to another unspecified learning algorithm, called the weak learning algorithm (WeakLearn)
- The booster algorithm provides WeakLearn in each round t with a derived distribution D_t defined for the training set S
- In response, the classifier computes a classifier $h_t: X \mapsto \{-1, +1\}$ which should correctly classify a fraction of the training set that has large probability with respect to the distribution D_t
- For this purpose, the goal of the weak learner is to find a classification that minimizes the training error $\epsilon_t = Prob_{i \sim D_i}[h_t(x_i) \neq y_i]$ (this error is generated according to the provided distribution D_t
- The distribution is updated in each round in order to focus the computation of the weak learner to the cases that are wrongly classified
- This process continues for T rounds, and, at last, the booster combines the weak predictions $h_1, \dots, h_T$ into a single final combined one

Updating the distribution D_t

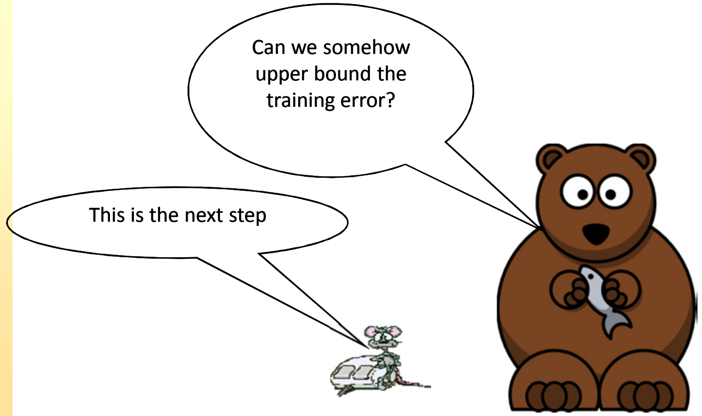
- With $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$, with $\frac{1-\epsilon_t}{\epsilon_t} > 1$ as $\epsilon_t < \frac{1}{2}$
- AdaBoost applies the update formula $D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{Z_t}$
- If $h_t(x_i) \neq y_i$ (false classification) it either holds that $h_t(x_i) = 1$ and $y_i = -1$ or vice versa $h_t(x_i) = -1$ and $y_i = 1$. Hence, if $h_t(x_i) \neq y_i$, we have $y_i \cdot h_t(x_i) = -1$ and $D_{t+1}(i) = \frac{D_t(i) \cdot e^{\alpha_t}}{Z_t} = D_t(i) \cdot \left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{\frac{1}{2}} > D_t(i)$
- Conversely, If $h_t(x_i) = y_i$ (correct classification) it either holds that $h_t(x_i) = 1$ and $y_i = 1$ or vice versa $h_t(x_i) = -1$ and $y_i = -1$. Hence, if $h_t(x_i) = y_i$, we have $y_i \cdot h_t(x_i) = 1$ and $D_{t+1}(i) = \frac{D_t(i) \cdot (e^{-\alpha_t})}{Z_t} = D_t(i) \cdot \left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{-\frac{1}{2}} < D_t(i)$

Updating the distribution D_t

Idea behind this updating

- If the prediction $h_t(x_i)$ of the ith case is not correct, the respective weight in the distribution is increased by the factor $\left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{\frac{1}{2}} > 1$. Due to a incorrect classification, it is interpreted as complex. Hence, it has to spent more attention in the next round
- Conversely, the weight of this case in the distribution is reduced as this case is assumed to be less complex
- Moreover, all information about previously selected features is captured in D_t

Quality of the approach



Upper bound of the training error

2.6.3.1 **Theorem**

By assuming the notation above, the following bound holds on the training error of H

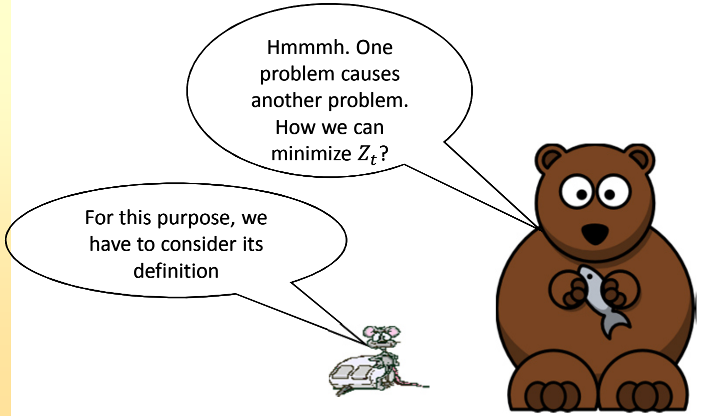
$$\frac{1}{m} \cdot |\{\, i \mid i \in \{1, \dots, m\} \wedge H(x_i) \neq y_i \,\}| \leq \prod_{t=1}^{T} Z_t$$

Proof of Theorem 2.6.3.1

- We have $D_{T+1}(i) = D_1(i) \cdot \frac{e^{-\sum_{t=1}^{T} \alpha_t \cdot y_i \cdot h_t(x_i)}}{\prod_{t=1}^{T} Z_t} = \frac{e^{-\sum_{t=1}^{T} \alpha_t \cdot y_i \cdot h_t(x_i)}}{m \cdot \prod_{t=1}^{T} Z_t}$
- We conclude that $D_{T+1}(i) = \frac{e^{-y_i \cdot f(x_i)}}{m \cdot \prod_{t=1}^{T} Z_t} \Rightarrow D_{T+1}(i) \cdot m \cdot \prod_{t=1}^{T} Z_t = e^{-y_i \cdot f(x_i)}$ (*)
- If $H(x_i) \neq y_i$ (false classification) it either holds that $f(x_i) > 0$ and $y_i = -1$ or vice versa $f(x_i) < 0$ and $y_i = 1$. Hence, if $H(x_i) \neq y_i$, we have $y_i \cdot f(x_i) \leq 0$. This implies $e^{-y_i \cdot f(x_i)} \geq 1$
- Thus, we obtain $[\![H(x_i) \neq y_i]\!] \leq e^{-y_i \cdot f(x_i)}$ (**)
- We use (*) and (**) in order to conclude

$$\frac{1}{m} \cdot |\{\, i \mid i \in \{1, \dots, m\} \wedge H(x_i) \neq y_i \,\}| \leq \frac{1}{m} \cdot \sum_{i=1}^{m} e^{-y_i \cdot f(x_i)}$$
$$= \frac{1}{m} \cdot \sum_{i=1}^{m} \left(m \prod_{t=1}^{T} Z_t \right) \cdot D_{T+1}(i)$$
$$= \prod_{t=1}^{T} Z_t \cdot \sum_{i=1}^{m} D_{T+1}(i) = \prod_{t=1}^{T} Z_t$$

Quality of the approach



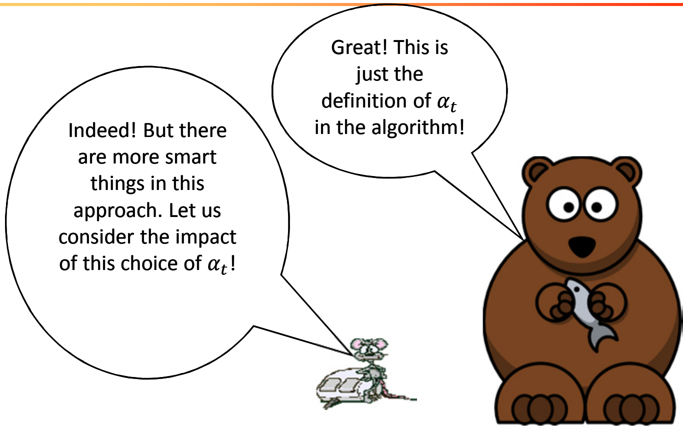
Consequences

- The upper bound of the training error can be minimized
- This can be done by minimizing Z_t in each training round t
  - For this purpose, we chose an optimal h_t
  - and an optimal α_t

2.6.3.2 Optimizing α_t

- Our aim is to minimize $Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}$
- Hence, we consider the first derivative of this function
$$\frac{dZ_t}{\alpha_t} = (-1) \cdot \sum_{i=1}^{m} D_t(i) \cdot y_i \cdot h_t(x_i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}$$
- Due to fact that $y_i \cdot h_t(x_i) = -1$ if $y_i \neq h_t\ (x_i)$ and $y_i \cdot h_t(x_i) = 1$ if $y_i = h_t\ (x_i)$, we conclude that
$$= \sum_{i \mid y_i \neq h_t(x_i)} D_t(i) \cdot e^{\alpha_t} - \sum_{i \mid y_i = h_t(x_i)} D_t(i) \cdot e^{-\alpha_t}$$
- With $\epsilon_j = \sum_{i=1}^{m} D_t(i) [\![y_i \neq h_j(x_i)]\!]$, we obtain
$$= (1-\epsilon_t) \cdot e^{\alpha_t} - \epsilon_t \cdot e^{-\alpha_t}$$
- Now, we set this derivative to zero
$$(1-\epsilon_t) \cdot e^{\alpha_t} - \epsilon_t \cdot e^{-\alpha_t} = 0 \Leftrightarrow (1-\epsilon_t) \cdot e^{\alpha_t} = \epsilon_t \cdot e^{-\alpha_t}$$
$$\Leftrightarrow ln(1-\epsilon_t) + \alpha_t = ln(\epsilon_t) - \alpha_t \Leftrightarrow ln(1-\epsilon_t) + 2\alpha_t = ln(\epsilon_t)$$
$$\Leftrightarrow 2\alpha_t = ln(\epsilon_t) - ln(1-\epsilon_t) \Leftrightarrow \alpha_t = \frac{1}{2} \cdot ln\left(\frac{\epsilon_t}{1-\epsilon_t}\right)$$

Quality of the approach



2.6.3.3 Substituting $\alpha_t = \frac{1}{2} \cdot ln\left(\frac{\epsilon_t}{1-\epsilon_t}\right)$

- By using $\alpha_t = \frac{1}{2} \cdot ln\left(\frac{\epsilon_t}{1-\epsilon_t}\right)$, we compute Z_t
- Thus, we obtain

$$Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}$$

$$= \sum_{i \mid y_i \neq h_t(x_i)} D_t(i) \cdot e^{\alpha_t} + \sum_{i \mid y_i = h_t(x_i)} D_t(i) \cdot e^{-\alpha_t}$$

$$= \epsilon_t \cdot e^{\alpha_t} + \frac{(1-\epsilon_t)}{e^{\alpha_t}} = \frac{\epsilon_t \cdot e^{2\alpha_t} + (1-\epsilon_t)}{e^{\alpha_t}}$$

$$= \frac{\epsilon_t \cdot \frac{\epsilon_t}{1-\epsilon_t} + (1-\epsilon_t)}{\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}}$$

And minimizing $Z_t(\alpha_t)$

$$= \epsilon_t \cdot \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + (1-\epsilon_t) \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \frac{\epsilon_t^2 + (1-\epsilon_t)^2}{\sqrt{\epsilon_t} \cdot \sqrt{1-\epsilon_t}}$$

We consider the first derivative (the second is positive)

$$\frac{\partial Z_t\left(\alpha_t = \frac{1}{2} \cdot ln\left(\frac{\epsilon_t}{1-\epsilon_t}\right)\right)}{\partial \epsilon_t} = -\frac{4\epsilon_t^3 - 6\epsilon_t^2 + 1}{2(1-\epsilon_t)^{\frac{3}{2}} \cdot x^{\frac{3}{2}}}$$

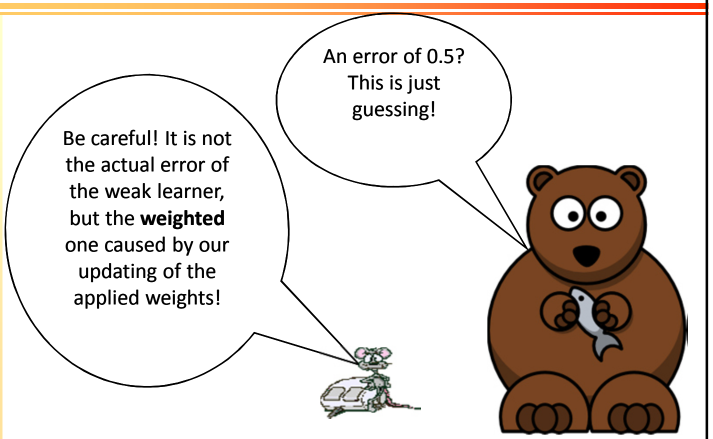
set it to zero, and obtain

$$\epsilon_t = \frac{1}{2} \vee \epsilon_t = \frac{\sqrt{3}+1}{2} > 1 \vee \epsilon_t = -\frac{\sqrt{3}-1}{2} < 0$$

Hence, the only feasible optimal solution for $0 \leq \epsilon_t \leq 1$ is

$$\epsilon_t = \frac{1}{2}$$

Quality of the approach



Consequence

- By analyzing the derivation of $\alpha_t = \frac{1}{2} \cdot ln\left(\frac{\epsilon_t}{1-\epsilon_t}\right)$, we set

$$\frac{dZ_t}{\alpha_t} = (-1) \cdot \sum_{i=1}^{m} D_t(i) \cdot y_i \cdot h_t(x_i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)} = 0$$

- This leads to

$$\sum_{i \mid y_i \neq h_t(x_i)} D_t(i) \cdot e^{\alpha_t} - \sum_{i \mid y_i = h_t(x_i)} D_t(i) \cdot e^{-\alpha_t} = 0$$

$$\Leftrightarrow \sum_{i \mid y_i \neq h_t(x_i)} D_t(i) \cdot e^{\alpha_t} = \sum_{i \mid y_i = h_t(x_i)} D_t(i) \cdot e^{-\alpha_t}$$

- With $D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{Z_t}$, we obtain

$$\Leftrightarrow \sum_{i \mid y_i \neq h_t(x_i)} D_{t+1}(i) \cdot Z_t = \sum_{i \mid y_i = h_t(x_i)} D_{t+1}(i) \cdot Z_t$$

$\Leftrightarrow \sum_{i \mid y_i \neq h_t(x_i)} D_{t+1}(i) = \sum_{i \mid y_i = h_t(x_i)} D_{t+1}(i)$, with $Z_t \neq 0$

- With other words, the sum of updated weights of the correctly classified cases coincides with the sum of updated weights of the incorrectly classified cases

Quality of the approach

AdaBoost – Example

Index	X-Coordinate	Y-Coordinate	Classification	Initial weight
0	1.7	3.5	1	0.05
1	4.4	2.2	1	0.05
2	9.5	3.7	1	0.05
3	13.0	4.5	1	0.05
4	16.2	5.9	1	0.05
5	6.5	6.1	1	0.05
6	11.3	6.8	1	0.05
7	4.3	8.3	1	0.05
8	3.0	10.5	1	0.05
9	3.9	15.5	1	0.05
10	8.5	3.5	-1	0.05
11	11.3	3.5	-1	0.05
12	14.0	6.5	-1	0.05
13	7.1	8.5	-1	0.05
14	14.0	8.8	-1	0.05
15	10.0	9.8	-1	0.05
16	14.2	11.8	-1	0.05
17	10.0	13.4	-1	0.05
18	16.3	14.4	-1	0.05
19	13.8	16.2	-1	0.05

Weak learner

- We apply as a weak learner a simple stump
  - It considers both attributes and identifies the best threshold to separate all cases
  - I.e., 40 possible thresholds are compared, while the separation is implemented that attains a smallest weighted error
- In what follows, we consider the output of a Python program

Iteration 1 – weak classifier

- X-coordinate
  - Threshold x-coordinate=6.5
  - Best Threshold x-coordinate quality=0.2 Threshold x-coordinate flag=-1
- Y-coordinate
  - Threshold y-coordinate=8.3
  - Threshold y-coordinate quality=0.25 Threshold y-coordinate flag=-1
- We take the x-coordinate. Flag=-1
- Results of weak classifier
  - Case 2 NOT correctly classified. Current error=0.05
  - Case 3 NOT correctly classified. Current error=0.1
  - Case 4 NOT correctly classified. Current error=0.15
  - Case 6 NOT correctly classified. Current error=0.2
- Total error=0.2
- Current list of classifiers: [['x-coordinate', 0.2, 6.5, -1, 1]]
- Current alphalist: [0.6931471805599453]

Iteration 1 – Quality of the combined classifier

- Classified classification of case 0 1 0.6931471805599453 Correct
- Classified classification of case 1 1 0.6931471805599453 Correct
- **Classified classification of case 2 -1 -0.6931471805599453 NOT correct!**
- **Classified classification of case 3 -1 -0.6931471805599453 NOT correct!**
- **Classified classification of case 4 -1 -0.6931471805599453 NOT correct!**
- Classified classification of case 5 1 0.6931471805599453 Correct
- **Classified classification of case 6 -1 -0.6931471805599453 NOT correct!**
- Classified classification of case 7 1 0.6931471805599453 Correct
- Classified classification of case 8 1 0.6931471805599453 Correct
- Classified classification of case 9 1 0.6931471805599453 Correct
- Classified classification of case 10 -1 -0.6931471805599453 Correct
- Classified classification of case 11 -1 -0.6931471805599453 Correct
- Classified classification of case 12 -1 -0.6931471805599453 Correct
- Classified classification of case 13 -1 -0.6931471805599453 Correct
- Classified classification of case 14 -1 -0.6931471805599453 Correct
- Classified classification of case 15 -1 -0.6931471805599453 Correct
- Classified classification of case 16 -1 -0.6931471805599453 Correct
- Classified classification of case 17 -1 -0.6931471805599453 Correct
- Classified classification of case 18 -1 -0.6931471805599453 Correct
- Classified classification of case 19 -1 -0.6931471805599453 Correct

Total error=0.2 (Clearly, is identical with the first weak classifier)

Iteration 1 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.03125
1	4.4	2.2	1	0.03125
2	**9.5**	**3.7**	**1**	**0.12500**
3	**13.0**	**4.5**	**1**	**0.12500**
4	**16.2**	**5.9**	**1**	**0.12500**
5	6.5	6.1	1	0.03125
6	**11.3**	**6.8**	**1**	**0.12500**
7	4.3	8.3	1	0.03125
8	3.0	10.5	1	0.03125
9	3.9	15.5	1	0.03125
10	8.5	3.5	-1	0.03125
11	11.3	3.5	-1	0.03125
12	14.0	6.5	-1	0.03125
13	7.1	8.5	-1	0.03125
14	14.0	8.8	-1	0.03125
15	10.0	9.8	-1	0.03125
16	14.2	11.8	-1	0.03125
17	10.0	13.4	-1	0.03125
18	16.3	14.4	-1	0.03125
19	13.8	16.2	-1	0.03125

Iteration 2 – weak classifier

- X-coordinate
  - Threshold x-coordinate=13.0
  - Best Threshold x-coordinate quality=0.28124999999999994 Threshold x-coordinate flag=-1
- Y-coordinate
  - Threshold y-coordinate=8.3
  - Threshold y-coordinate quality=0.15624999999999997 Threshold y-coordinate flag=-1
- We take the y-coordinate. Flag=-1
- Results of weak classifier
  - Case 8 NOT correctly classified. Current error=0.031249999999999993
  - Case 9 NOT correctly classified. Current error=0.062499999999999986
  - Case 10 NOT correctly classified. Current error=0.09374999999999997
  - Case 11 NOT correctly classified. Current error=0.12499999999999997
  - Case 12 NOT correctly classified. Current error=0.15624999999999997
- Total error=0.15624999999999997
- ['y-coordinate', 0.15624999999999997, 8.3, -1, 2]
- Current list of classifiers: [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2]]
- Current alphalist: [0.6931471805599453, 0.8431994767851144]

Iteration 2 – Quality of the combined classifier

- Classified classification of case 0 1 1.5363466573450597 Correct
- Classified classification of case 1 1 1.5363466573450597 Correct
- Classified classification of case 2 1 0.15005229622516914 Correct
- Classified classification of case 3 1 0.15005229622516914 Correct
- Classified classification of case 4 1 0.15005229622516914 Correct
- Classified classification of case 5 1 1.5363466573450597 Correct
- Classified classification of case 6 1 0.15005229622516914 Correct
- Classified classification of case 7 1 1.5363466573450597 Correct
- Classified classification of case 8 -1 -0.15005229622516914 NOT correct!
- Classified classification of case 9 -1 -0.15005229622516914 NOT correct!
- Classified classification of case 10 1 0.15005229622516914 NOT correct!
- Classified classification of case 11 1 0.15005229622516914 NOT correct!
- Classified classification of case 12 1 0.15005229622516914 NOT correct!
- Classified classification of case 13 -1 -1.5363466573450597 Correct
- Classified classification of case 14 -1 -1.5363466573450597 Correct
- Classified classification of case 15 -1 -1.5363466573450597 Correct
- Classified classification of case 16 -1 -1.5363466573450597 Correct
- Classified classification of case 17 -1 -1.5363466573450597 Correct
- Classified classification of case 18 -1 -1.5363466573450597 Correct
- Classified classification of case 19 -1 -1.5363466573450597 Correct

Total error=0.25

Iteration 2 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.018519
1	4.4	2.2	1	0.018519
2	9.5	3.7	1	0.074074
3	13.0	4.5	1	0.074074
4	16.2	5.9	1	0.074074
5	6.5	6.1	1	0.018519
6	11.3	6.8	1	0.074074
7	4.3	8.3	1	0.018519
8	**3.0**	**10.5**	**1**	**0.100000**
9	**3.9**	**15.5**	**1**	**0.100000**
10	**8.5**	**3.5**	**-1**	**0.100000**
11	**11.3**	**3.5**	**-1**	**0.100000**
12	**14.0**	**6.5**	**-1**	**0.100000**
13	7.1	8.5	-1	0.018519
14	14.0	8.8	-1	0.018519
15	10.0	9.8	-1	0.018519
16	14.2	11.8	-1	0.018519
17	10.0	13.4	-1	0.018519
18	16.3	14.4	-1	0.018519
19	13.8	16.2	-1	0.018519

Iteration 3 – weak classifier

- X-coordinate
  - Threshold x-coordinate=6.5
  - Best Threshold x-coordinate quality=0.2962962962962963 Threshold x-coordinate flag=-1
- Y-coordinate
  - Threshold y-coordinate=3.5
  - Threshold y-coordinate quality=0.26666666666666666 Threshold y-coordinate flag=1
- We take the y-coordinate. Flag=1
- Results of weak classifier
  - Case 2 NOT correctly classified. Current error=0.07407407407407407
  - Case 3 NOT correctly classified. Current error=0.14814814814814814
  - Case 4 NOT correctly classified. Current error=0.2222222222222222
  - Case 5 NOT correctly classified. Current error=0.24074074074074073
  - Case 6 NOT correctly classified. Current error=0.31481481481481477
  - Case 7 NOT correctly classified. Current error=0.33333333333333326
  - Case 8 NOT correctly classified. Current error=0.43333333333333324
  - Case 9 NOT correctly classified. Current error=0.5333333333333332
  - Case 10 NOT correctly classified. Current error=0.6333333333333332
  - Case 11 NOT correctly classified. Current error=0.7333333333333332
  - Total error=0.7333333333333332
  - ['y-coordinate', 0.26666666666666666, 3.5, 1, 3]
- Current list of classifiers:
- [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2], ['y-coordinate', 0.26666666666666666, 3.5, 1, 3]]
- Current alphalist: [0.6931471805599453, 0.8431994767851144, 0.50580045583924]

Iteration 3 – Quality of the combined classifier

- Classified classification of case 0 1 1.0305462015058198 Correct
- Classified classification of case 1 1 1.0305462015058198 Correct
- Classified classification of case 2 1 0.6558527520644092 Correct
- Classified classification of case 3 1 0.6558527520644092 Correct
- Classified classification of case 4 1 0.6558527520644092 Correct
- Classified classification of case 5 1 2.0421471131842996 Correct
- Classified classification of case 6 1 0.6558527520644092 Correct
- Classified classification of case 7 1 2.0421471131842996 Correct
- Classified classification of case 8 1 0.3557481596140709 Correct
- Classified classification of case 9 1 0.3557481596140709 Correct
- Classified classification of case 10 -1 -0.3557481596140709 Correct
- Classified classification of case 11 -1 -0.3557481596140709 Correct
- Classified classification of case 12 1 0.6558527520644092 NOT correct!
- Classified classification of case 13 -1 -1.0305462015058198 Correct
- Classified classification of case 14 -1 -1.0305462015058198 Correct
- Classified classification of case 15 -1 -1.0305462015058198 Correct
- Classified classification of case 16 -1 -1.0305462015058198 Correct
- Classified classification of case 17 -1 -1.0305462015058198 Correct
- Classified classification of case 18 -1 -1.0305462015058198 Correct
- Classified classification of case 19 -1 -1.0305462015058198 Correct

Total error=0.05

Iteration 3 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.034722
1	4.4	2.2	1	0.034722
2	**9.5**	**3.7**	**1**	**0.050505**
3	**13.0**	**4.5**	**1**	**0.050505**
4	**16.2**	**5.9**	**1**	**0.050505**
5	**6.5**	**6.1**	**1**	**0.012626**
6	**11.3**	**6.8**	**1**	**0.050505**
7	**4.3**	**8.3**	**1**	**0.012626**
8	**3.0**	**10.5**	**1**	**0.068182**
9	**3.9**	**15.5**	**1**	**0.068182**
10	**8.5**	**3.5**	**-1**	**0.068182**
11	**11.3**	**3.5**	**-1**	**0.068182**
12	14.0	6.5	-1	0.187500
13	7.1	8.5	-1	0.034722
14	14.0	8.8	-1	0.034722
15	10.0	9.8	-1	0.034722
16	14.2	11.8	-1	0.034722
17	10.0	13.4	-1	0.034722
18	16.3	14.4	-1	0.034722
19	13.8	16.2	-1	0.034722

Iteration 4 – weak classifier

- X-coordinate
  - Threshold x-coordinate=6.5
  - Threshold x-coordinate quality=0.202020202020202 Threshold x-coordinate flag=-1
- Y-coordinate
  - Threshold y-coordinate=6.1
  - Threshold y-coordinate quality=0.3358585858585858 Threshold y-coordinate flag=-1
- We take the x-coordinate. Flag=-1
- Results of weak classifier
  - Case 2 NOT correctly classified. Current error=0.0505050505050505
  - Case 3 NOT correctly classified. Current error=0.101010101010101
  - Case 4 NOT correctly classified. Current error=0.1515151515151515
  - Case 6 NOT correctly classified. Current error=0.202020202020202
  - Total error=0.202020202020202
  - ['x-coordinate', 0.202020202020202, 6.5, -1, 4]
- Current list of classifiers: [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2], ['y-coordinate', 0.26666666666666666, 3.5, 1, 3], ['x-coordinate', 0.202020202020202, 6.5, -1, 4]]
- Current alphalist: [0.6931471805599453, 0.8431994767851144, 0.50580045583924, 0.6868577894565153]

Iteration 4 – Quality of the combined classifier

- Classified classification of case 0 1 1.717403990962335 Correct
- Classified classification of case 1 1 1.717403990962335 Correct
- Classified classification of case 2 -1 -0.03100503739210614 NOT correct!
- Classified classification of case 3 -1 -0.03100503739210614 NOT correct!
- Classified classification of case 4 -1 -0.03100503739210614 NOT correct!
- Classified classification of case 5 1 2.729004902640815 Correct
- Classified classification of case 6 -1 -0.03100503739210614 NOT correct!
- Classified classification of case 7 1 2.729004902640815 Correct
- Classified classification of case 8 1 1.0426059490705861 Correct
- Classified classification of case 9 1 1.0426059490705861 Correct
- Classified classification of case 10 -1 -1.0426059490705861 Correct
- Classified classification of case 11 -1 -1.0426059490705861 Correct
- Classified classification of case 12 -1 -0.0310050373921061 Correct
- Classified classification of case 13 -1 -1.717403990962335 Correct
- Classified classification of case 14 -1 -1.717403990962335 Correct
- Classified classification of case 15 -1 -1.717403990962335 Correct
- Classified classification of case 16 -1 -1.717403990962335 Correct
- Classified classification of case 17 -1 -1.717403990962335 Correct
- Classified classification of case 18 -1 -1.717403990962335 Correct
- Classified classification of case 19 -1 -1.717403990962335 Correct

Total error=0.2

Iteration 4 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.021756
1	4.4	2.2	1	0.021756
2	**9.5**	**3.7**	**1**	**0.125000**
3	**13.0**	**4.5**	**1**	**0.125000**
4	**16.2**	**5.9**	**1**	**0.125000**
5	6.5	6.1	1	0.007911
6	**11.3**	**6.8**	**1**	**0.125000**
7	4.3	8.3	1	0.007911
8	3.0	10.5	1	0.042722
9	3.9	15.5	1	0.042722
10	8.5	3.5	-1	0.042722
11	11.3	3.5	-1	0.042722
12	14.0	6.5	-1	0.117484
13	7.1	8.5	-1	0.021756
14	14.0	8.8	-1	0.021756
15	10.0	9.8	-1	0.021756
16	14.2	11.8	-1	0.021756
17	10.0	13.4	-1	0.021756
18	16.3	14.4	-1	0.021756
19	13.8	16.2	-1	0.021756

Iteration 5 – weak classifier

- X-coordinate
  - Threshold x-coordinate=13.0
  - Threshold x-coordinate quality=0.2757120253164556 Threshold x-coordinate flag=-1
- Y-coordinate
  - Threshold y-coordinate=8.3
  - Threshold y-coordinate quality=0.2883702531645569 Threshold y-coordinate flag=-1
- We take the x-coordinate. Flag=-1
- Results of weak classifier
  - Case 4 NOT correctly classified. Current error=0.12499999999999994
  - Case 10 NOT correctly classified. Current error=0.1677215189873417
  - Case 11 NOT correctly classified. Current error=0.21044303797468344
  - Case 13 NOT correctly classified. Current error=0.2321993670886075
  - Case 15 NOT correctly classified. Current error=0.25395569620253156
  - Case 17 NOT correctly classified. Current error=0.2757120253164556
  - Total error=0.2757120253164556
  - [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2], ['y-coordinate', 0.26666666666666666, 3.5, 1, 3], ['x-coordinate', 0.202020202020202, 6.5, -1, 4], ['x-coordinate', 0.2757120253164556, 13.0, -1, 5]]
  - Current alphalist: [0.6931471805599453, 0.8431994767851144, 0.50580045583924, 0.6868577894565153, 0.4829160669569937]

Iteration 5 – Quality of the combined classifier

- Classified classification of case 0 1 2.200320057919329 Correct
- Classified classification of case 1 1 2.200320057919329 Correct
- Classified classification of case 2 1 0.45191102956488755 Correct
- Classified classification of case 3 1 0.45191102956488755 Correct
- Classified classification of case 4 -1 -0.5139211043490999 NOT correct!
- Classified classification of case 5 1 3.211920969597809 Correct
- Classified classification of case 6 1 0.45191102956488755 Correct
- Classified classification of case 7 1 3.211920969597809 Correct
- Classified classification of case 8 1 1.5255220160275798 Correct
- Classified classification of case 9 1 1.5255220160275798 Correct
- Classified classification of case 10 -1 -0.5596898821135925 Correct
- Classified classification of case 11 -1 -0.5596898821135925 Correct
- Classified classification of case 12 -1 -0.5139211043490999 Correct
- Classified classification of case 13 -1 -1.2344879240053415 Correct
- Classified classification of case 14 -1 -2.200320057919329 Correct
- Classified classification of case 15 -1 -1.2344879240053415 Correct
- Classified classification of case 16 -1 -2.200320057919329 Correct
- Classified classification of case 17 -1 -1.2344879240053415 Correct
- Classified classification of case 18 -1 -2.200320057919329 Correct
- Classified classification of case 19 -1 -2.200320057919329 Correct

Total error=0.05

Iteration 5 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.015019
1	4.4	2.2	1	0.015019
2	9.5	3.7	1	0.086292
3	13.0	4.5	1	0.086292
4	**16.2**	**5.9**	**1**	**0.226686**
5	6.5	6.1	1	0.005461
6	11.3	6.8	1	0.086292
7	4.3	8.3	1	0.005461
8	3.0	10.5	1	0.029492
9	3.9	15.5	1	0.029492
10	**8.5**	**3.5**	**-1**	**0.077475**
11	**11.3**	**3.5**	**-1**	**0.077475**
12	14.0	6.5	-1	0.081103
13	**7.1**	**8.5**	**-1**	**0.039455**
14	14.0	8.8	-1	0.015019
15	**10.0**	**9.8**	**-1**	**0.039455**
16	14.2	11.8	-1	0.015019
17	**10.0**	**13.4**	**-1**	**0.039455**
18	16.3	14.4	-1	0.015019
19	13.8	16.2	-1	0.015019

Iteration 6 – weak classifier

- X-coordinate
  - Threshold x-coordinate=14.2
  - Threshold x-coordinate quality=0.37383943200436914 Threshold x-coordinate flag=1
- Y-coordinate
  - Threshold y-coordinate=3.5
  - Threshold y-coordinate quality=0.2895823326466632 Threshold y-coordinate flag=1
- We take the y-coordinate. Flag=1
- Results of weak classifier
  - Case 2 NOT correctly classified. Current error=0.08629164391043145
  - Case 3 NOT correctly classified. Current error=0.1725832878208629
  - Case 4 NOT correctly classified. Current error=0.39926908409059036
  - Case 5 NOT correctly classified. Current error=0.40473058054061767
  - Case 6 NOT correctly classified. Current error=0.49102222445104915
  - Case 7 NOT correctly classified. Current error=0.49648372090107645
  - Case 8 NOT correctly classified. Current error=0.5259758017312239
  - Case 9 NOT correctly classified. Current error=0.5554678825613714
  - Case 10 NOT correctly classified. Current error=0.6329427749573542
  - Case 11 NOT correctly classified. Current error=0.710417667353337
  - Total error=0.710417667353337
  - ['y-coordinate', 0.2895823326466632, 3.5, 1, 6]
- Current list of classifiers: [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2], ['y-coordinate', 0.26666666666666666, 3.5, 1, 3], ['x-coordinate', 0.202020202020202, 6.5, -1, 4], ['x-coordinate', 0.2757120253164556, 13.0, -1, 5], ['y-coordinate', 0.2895823326466632, 3.5, 1, 6]]
- Current alphalist: [0.6931471805599453, 0.8431994767851144, 0.50580045583924, 0.6868577894565153, 0.4829160669569937, 0.4487067041788279]

Iteration 6 – Quality of the combined classifier

- Classified classification of case 0 1 1.751613353740501 Correct
- Classified classification of case 1 1 1.751613353740501 Correct
- Classified classification of case 2 1 0.9006177337437155 Correct
- Classified classification of case 3 1 0.9006177337437155 Correct
- Classified classification of case 4 -1 -0.06521440017027197 NOT correct!
- Classified classification of case 5 1 3.6606276737766366 Correct
- Classified classification of case 6 1 0.9006177337437155 Correct
- Classified classification of case 7 1 3.6606276737766366 Correct
- Classified classification of case 8 1 1.9742287202064077 Correct
- Classified classification of case 9 1 1.9742287202064077 Correct
- Classified classification of case 10 -1 -1.0083965862924205 Correct
- Classified classification of case 11 -1 -1.0083965862924205 Correct
- Classified classification of case 12 -1 -0.06521440017027197 Correct
- Classified classification of case 13 -1 -0.7857812198265135 Correct
- Classified classification of case 14 -1 -1.751613353740501 Correct
- Classified classification of case 15 -1 -0.7857812198265135 Correct
- Classified classification of case 16 -1 -1.751613353740501 Correct
- Classified classification of case 17 -1 -0.7857812198265135 Correct
- Classified classification of case 18 -1 -1.751613353740501 Correct
- Classified classification of case 19 -1 -1.751613353740501 Correct

Total error=0.05

Iteration 6 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.025932
1	4.4	2.2	1	0.025932
2	**9.5**	**3.7**	**1**	**0.060733**
3	**13.0**	**4.5**	**1**	**0.060733**
4	**16.2**	**5.9**	**1**	**0.159544**
5	**6.5**	**6.1**	**1**	**0.003844**
6	**11.3**	**6.8**	**1**	**0.060733**
7	**4.3**	**8.3**	**1**	**0.003844**
8	**3.0**	**10.5**	**1**	**0.020757**
9	**3.9**	**15.5**	**1**	**0.020757**
10	**8.5**	**3.5**	**-1**	**0.054528**
11	**11.3**	**3.5**	**-1**	**0.054528**
12	14.0	6.5	-1	0.140035
13	7.1	8.5	-1	0.068124
14	14.0	8.8	-1	0.025932
15	10.0	9.8	-1	0.068124
16	14.2	11.8	-1	0.025932
17	10.0	13.4	-1	0.068124
18	16.3	14.4	-1	0.025932
19	13.8	16.2	-1	0.025932

Iteration 7 – weak classifier

- X-coordinate
  - Threshold x-coordinate=14.2
  - Threshold x-coordinate quality=0.3091976806419835 Threshold x-coordinate flag=1
- Y-coordinate
  - Threshold y-coordinate=6.1
  - Threshold y-coordinate quality=0.2151460337068737 Threshold y-coordinate flag=-1
- We take the y-coordinate. Flag=-1
- Results of weak classifier
  - Case 6 NOT correctly classified. Current error=0.06073303626577251
  - Case 7 NOT correctly classified. Current error=0.06457689932056825
  - Case 8 NOT correctly classified. Current error=0.08533375981646518
  - Case 9 NOT correctly classified. Current error=0.10609062031236212
  - Case 10 NOT correctly classified. Current error=0.16061832700961792
  - Case 11 NOT correctly classified. Current error=0.2151460337068737
  - Total error=0.2151460337068737
  - ['y-coordinate', 0.2151460337068737, 6.1, -1, 7]
  - Current list of classifiers: [['x-coordinate', 0.2, 6.5, -1, 1], ['y-coordinate', 0.15624999999999997, 8.3, -1, 2], ['y-coordinate', 0.26666666666666666, 3.5, 1, 3], ['x-coordinate', 0.20202020202020202, 6.5, -1, 4], ['x-coordinate', 0.2757120253164556, 13.0, -1, 5], ['y-coordinate', 0.2895823326466632, 3.5, 1, 6], ['y-coordinate', 0.2151460337068737, 6.1, -1, 7]]
- Current alphalist: [0.6931471805599453, 0.8431994767851144, 0.50580045583924, 0.6868577894565153]

Iteration 7 – Quality of the combined classifier

- Classified classification of case 0 1 2.398703676828247 Correct
- Classified classification of case 1 1 2.398703676828247 Correct
- Classified classification of case 2 1 1.5477080568314616 Correct
- Classified classification of case 3 1 1.5477080568314616 Correct
- Classified classification of case 4 1 0.5818759229174741 Correct
- Classified classification of case 5 1 4.307717996864382 Correct
- Classified classification of case 6 1 0.2535274106559693 Correct
- Classified classification of case 7 1 3.0135373506888903 Correct
- Classified classification of case 8 1 1.3271383971186617 Correct
- Classified classification of case 9 1 1.3271383971186617 Correct
- Classified classification of case 10 -1 -0.3613062632046743 Correct
- Classified classification of case 11 -1 -0.3613062632046743 Correct
- Classified classification of case 12 -1 -0.7123047232580182 Correct
- Classified classification of case 13 -1 -1.4328715429142598 Correct
- Classified classification of case 14 -1 -2.398703676828247 Correct
- Classified classification of case 15 -1 -1.4328715429142598 Correct
- Classified classification of case 16 -1 -2.398703676828247 Correct
- Classified classification of case 17 -1 -1.4328715429142598 Correct
- Classified classification of case 18 -1 -2.398703676828247 Correct
- Classified classification of case 19 -1 -2.398703676828247 Correct

Total error=0.0

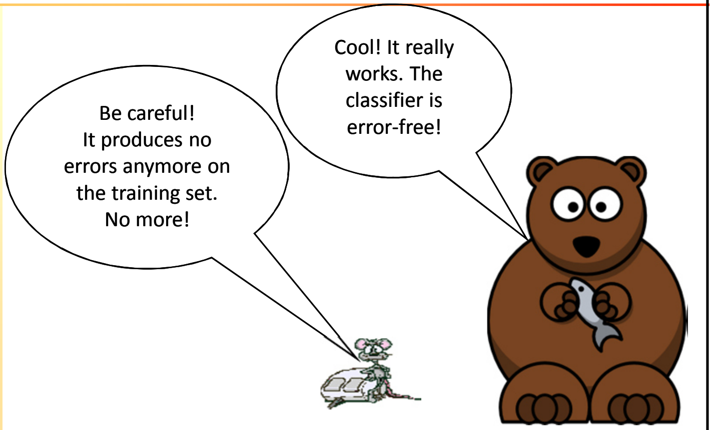
Iteration 7 – Updated weights

Index	X-Coordinate	Y-Coordinate	Classification	Weight
0	1.7	3.5	1	0.016521
1	4.4	2.2	1	0.016521
2	9.5	3.7	1	0.038691
3	13.0	4.5	1	0.038691
4	16.2	5.9	1	0.101639
5	6.5	6.1	1	0.002449
6	11.3	6.8	1	0.141144
7	4.3	8.3	1	0.008933
8	3.0	10.5	1	0.048239
9	3.9	15.5	1	0.048239
10	8.5	3.5	-1	0.126723
11	11.3	3.5	-1	0.126723
12	14.0	6.5	-1	0.089211
13	7.1	8.5	-1	0.043399
14	14.0	8.8	-1	0.016521
15	10.0	9.8	-1	0.043399
16	14.2	11.8	-1	0.016521
17	10.0	13.4	-1	0.043399
18	16.3	14.4	-1	0.016521
19	13.8	16.2	-1	0.016521

Termination with error 0

As we have no remaining error within the training set, the algorithm stops

Quality of the approach



Extensions

- Due to its impressive performance, the AdaBoost algorithm was extended by many scientific contributions
- Enabling general classifications
  - Instead of binary classifications, various authors propose AdaBoost extensions that are able to deal with more than two classes (see Freund and Schapire (1996), Zhu, Zou, Rosset and Hastie (2009))
  - This will be considered more in detail in the next part of this section
- Online versions of ensemble learning
  - In order to derive reliable predictors also under restrictive time restrictions, various authors generated AdaBoost variants/extensions
  - These versions derive the combined predictors by exploring the available data sets only once or in a considerably reduced number of iterations (see Oza (2001))

Multi-class AdaBoost by Freund and Schapire (1996)

- In what follows, we consider a simple extension of AdaBoost to general classifications
- For this purpose, the authors generate and introduce two different approaches, namely
  - AdaBoost.M1 and
  - AdaBoost.M2

AdaBoost.M1

Input:
- Sequence of m cases $S = \langle(x_1, y_1), \dots, (x_m, y_m)\rangle$ with labels $y_i \in Y = \{1, \dots C\}$ determining the respective classification of the case $x_i \in X = \{x_1, \dots, x_m\}$
- Weak learning algorithm (predictor) denotes as WeakLearn
- Integer T determining the number of iterations to be performed

Initialize $D_1(i) := \frac{1}{m}$ (weights of the cases to be considered), $\forall i \in \{1, \dots, m\}$

DO FOR ALL $t = 1,2, \dots, T$:

1. Call WeakLearn($D_t(1), \dots, D_t(n)$) /* based on the weights $D_t(1), \dots, D_t(n)$ */
2. Get back the prediction $h_t: X \mapsto Y$
3. Calculate the error ϵ_t of the predictor h_t by the formula $\epsilon_t = \sum_{i|h_t(x_i)\neq y_i} D_t(i)$
4. IF $\epsilon_t > \frac{1}{2}$ THEN set $T := t-1$; Abort loop;
5. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

AdaBoost.M1 – Continuation

6. Update the current distribution (of weights):

$$D_{t+1}(i) := \frac{D_t(i)}{Z_t} \cdot \begin{cases} \beta_t & if\ h_t(x_i) = y_i \\ 1 & otherwise \end{cases}$$

In this definition Z_t is a normalization constant in order to guarantee that D_{t+1} is a distribution

END DO FOR ALL

Output the final predictions of all generated predictors

$$h_{fin}(x) = argmax\left\{\sum_{t|h_t(x)=y} \log\left(\frac{1}{\beta_t}\right) \mid y \in Y\right\}$$

Shortcomings of AdaBoost.M1

- AdaBoost.M1 forces the weak learner to give an unambiguous decision concerning the classification of each training case. However, frequently it is more realistic that the weak learner has reliable knowledge about some not applying classifications, while vague knowledge is given concerning some other cases that are much more likely to apply. Such a situation can be mapped adequately by using a set of "plausible" labels
- For this purpose, AdaBoost.M2 will indicate a "degree of plausibility"
- One further main disadvantage of AdaBoost.M1 is that this procedure is unable to handle weak predictions with an error exceeding $\frac{1}{2}$
- Note that this is **acceptable for binary classifications only** (pure guessing would attain 50 percent), but if the number of classes increases, this limitation is quite restrictive. Here, the expected error of simple guessing one of C classes would be $1 - \frac{1}{C}$
- All these shortcomings leads to the generation of AdaBoost.M2

AdaBoost.M2

Input:
- Sequence of m cases $\langle(x_1, y_1), \dots, (x_m, y_m)\rangle$ with labels $y_i \in Y = \{1, \dots C\}$ determining the respective classification of the case $x_i \in X = \{x_1, \dots, x_m\}$
- Weak learning algorithm (predictor) denotes as WeakLearn
- Integer T determining the number of iterations to be performed

Let $B := \{(i, y) \mid i \in \{1, \dots, m\} \wedge y \in Y \wedge y \neq y_i\}$ /* all possible mislabels */

Initialize $D_1(i, y) := \frac{1}{|B|}$ (weights of the mislabels to be considered), $\forall (i, y) \in B$

DO FOR ALL $t = 1,2, \dots, T$:

1. Call WeakLearn($D_t(i, y)$) /* based on the mislabel weights $D_t(i, y), \forall (i, y) \in B$ */
2. Get back the prediction $h_t: X \times Y \mapsto [0,1]$
3. Calculate the pseudo-loss ϵ_t of the predictor h_t by the formula
$$\epsilon_t = \frac{1}{2} \cdot \sum_{(i,y)\in B} D_t(i, y) \cdot \left(1 - h_t(x_i, y_i) + h_t(x_i, y)\right)$$
4. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

AdaBoost.M2 – Continuation

5. Update the current distribution (of weights of pseudo-losses):

$$D_{t+1}(i,y) := \frac{D_t(i,y)}{Z_t} \cdot \beta_t^{\frac{1}{2}\cdot(1-h_t(x_i,y_i)+h_t(x_i,y))}$$

In this definition Z_t is a normalization constant in order to guarantee that D_{t+1} is a distribution

END DO FOR ALL

Output the final predictions of all generated predictors

$$h_{fin}(x) = argmax\left\{\sum_{t=1}^{T}\left(\log\frac{1}{\beta_t}\right)\cdot h_t(x,y) \mid y \in Y\right\}$$

Derivation of the pseudo-loss computation

- In each iteration, the weak learner generates $h_t: X \times Y \mapsto \{0,1\}$
- I.e., $h_t(x,y)$ measures the degree to which it is believed that y is the correct label associated with instance x. Note that h_t is not a distribution
- Thus, if for a given $x \in X$ we have $h_t(x,y)$ is identical for all $y \in Y$, we say that the hypothesis is uninformative on instance x
- On the other side, any deviation from strict equality is potentially informative, because it predicts some labels to be more plausible than others
- In order to motivate the pseudo-loss computation of AdaBoost.M2, we pose for each incorrect label $y \neq y_i$ the question: "*Which is the label of* x_i *:* y_i *or* y?"
- To answer the question, we have to transform the degrees of h_t into expected classification values, i.e., we have to find a modeling of using these degrees
  - For this purpose, we do the following game
  - We draw a bit $b_t(x,y) \in \{0,1\}$ randomly such that $b_t(x,y)$ is one with probability $h_t(x,y)$ and 0 otherwise, i.e., with probability $1-h_t(x,y)$.
  - We do the same for $h_t(x,y_i)$

Derivation of the pseudo-loss computation

- Clearly, if both randomly drawn bits are unequal, we got a decision
  - If $b_t(x,y) = 1 = 1 - b_t(x,y_i)$ applies, the incorrect classification y is assumed
  - If $b_t(x,y) = 0 = 1 - b_t(x,y_i)$ applies, the correct classification y_i is assumed
- However, if both bits are of equal values, i.e., $b_t(x,y) = b_t(x,y_i)$, the classification is done randomly on the basis of a uniform distribution, i.e., with a probability of $\frac{1}{2}$ in both cases, y or y_i is chosen
- Therefore, due to these (game) assumptions, the probability of choosing the incorrect answer y to the *question above* is the probability of the case $b_t(x,y) = 1 \wedge b_t(x,y_i) = 0$ plus half of the probability of the case $b_t(x,y) = b_t(x,y_i)$
- Hence, we can compute

$$\begin{aligned}&h_t(x_i,y)\cdot\left(1-h_t(x_i,y_i)\right)+\\&\frac{1}{2}\cdot\left(h_t(x_i,y)\cdot h_t(x_i,y_i)+\left(1-h_t(x_i,y)\right)\cdot\left(1-h_t(x_i,y_i)\right)\right)\\&=h_t(x_i,y)-h_t(x_i,y)\cdot h_t(x_i,y_i)+\frac{1}{2}h_t(x_i,y)\cdot h_t(x_i,y_i)\\&+\frac{1}{2}-\frac{1}{2}h_t(x_i,y)-\frac{1}{2}h_t(x_i,y_i)+\frac{1}{2}h_t(x_i,y)\cdot h_t(x_i,y_i)\end{aligned}$$

Derivation of the pseudo-loss computation

$$\begin{aligned}&=h_t(x_i,y)-h_t(x_i,y)\cdot h_t(x_i,y_i)+\frac{1}{2}h_t(x_i,y)\cdot h_t(x_i,y_i)\\&+\frac{1}{2}-\frac{1}{2}h_t(x_i,y)-\frac{1}{2}h_t(x_i,y_i)+\frac{1}{2}h_t(x_i,y)\cdot h_t(x_i,y_i)\\&=\frac{1}{2}h_t(x_i,y)-h_t(x_i,y)\cdot h_t(x_i,y_i)+h_t(x_i,y)\cdot h_t(x_i,y_i)+\frac{1}{2}-\frac{1}{2}h_t(x_i,y_i)\\&=\frac{1}{2}h_t(x_i,y)+\frac{1}{2}-\frac{1}{2}h_t(x_i,y_i)=\frac{1}{2}\cdot\left(1-h_t(x_i,y_i)+h_t(x_i,y)\right)\end{aligned}$$

Pseudo loss – Observations

- We consider the pseudo-loss computation for $x_i \in X$ with C classifications and state the following
  - If $\forall y \in Y$ it holds that $h_t(x_i, y) = \frac{1}{C}$ we obtain
  $$\frac{1}{2} \cdot (1 - h_t(x_i, y_i) + h_t(x_i, y)) = \frac{1}{2} \cdot \left(1 - \frac{1}{C} + \frac{1}{C}\right) = \frac{1}{2}$$
  - Hence, if this holds for all $x_i \in X$ (uninformative case), we obtain
  $$\epsilon_t = \frac{1}{2} \cdot \sum_{(i,y)\in B} D_t(i, y) \cdot (1 - h_t(x_i, y_i) + h_t(x_i, y))$$
  $$= \frac{1}{2} \cdot \sum_{(i,y)\in B} D_t(i, y) \cdot \left(1 - \frac{1}{C} + \frac{1}{C}\right) = \frac{1}{2} \cdot \sum_{(i,y)\in B} D_t(i, y) = \frac{1}{2}$$
  - Moreover, if we have $\epsilon_t > \frac{1}{2}$ we can modify h_t by setting $\forall y \in Y: h_t(x_i, y) := 1 - h_t(x_i, y)$ and obtain the pseudo-loss $1 - \epsilon_t < \frac{1}{2}$
  - Hence, we can assume that $\epsilon_t \leq \frac{1}{2}$ holds

Pseudo loss – Observations

- We assume that $\frac{1}{2} \cdot (1 - h_t(x_i, y_i) + h_t(x_i, y)) > \frac{1}{2}$ holds
- Then, we conclude that $1 - h_t(x_i, y_i) + h_t(x_i, y) > 1$
- Thus, we obtain $-h_t(x_i, y_i) + h_t(x_i, y) > 0$ and $h_t(x_i, y_i) - h_t(x_i, y) < 0$
- Hence, we obtain
$$\frac{1}{2} \cdot \left(1 - (1 - h_t(x_i, y_i)) + (1 - h_t(x_i, y))\right)$$
$$= \frac{1}{2} \cdot (1 + h_t(x_i, y_i) - h_t(x_i, y))$$
$$\leq \frac{1}{2} + \frac{1}{2} \cdot (h_t(x_i, y_i) - h_t(x_i, y)) < \frac{1}{2}$$
- Therefore, as stated above, by setting $\forall y \in Y: h_t(x_i, y) := 1 - h_t(x_i, y)$, we obtain the pseudo-loss $1 - \epsilon_t < \frac{1}{2}$

AdaBoost.M2 vs AdaBoost.M1

- The main difference between both approaches is that the second version gives the weak learner more expressive power concerning the classification of the training cases
  - The error measurement
- This requires a more sophisticated assessment of the performance
  - Specifically, instead of measuring the error, i.e., the total weighted incorrect classification
  $$\epsilon_t = \sum_{i|h_t(x_i)\neq y_i} D_t(i)),$$
  - AdaBoost.M2 sums up the total weighted pseudo-loss
  $$\epsilon_t = \frac{1}{2} \cdot \sum_{(i,y)\in B} D_t(i, y) \cdot (1 - h_t(x_i, y_i) + h_t(x_i, y))$$

AdaBoost.M2 vs AdaBoost.M1

- Moreover, the update of the distribution is adapted accordingly
  - By using the computed error or pseudo-loss ϵ_t, both approaches generate the factor $\beta_t = \frac{\epsilon_t}{1-\epsilon_t} \leq 1$.
  - By using this factor, the distribution of the preceding round is updated
  - Update in AdaBoost.M1
  $$D_{t+1}(i) := \frac{D_t(i)}{Z_t} \cdot \begin{cases} \beta_t & if\ h_t(x_i) = y_i \\ 1 & otherwise \end{cases}$$
  - Update in AdaBoost.M2
  $$D_{t+1}(i, y) := \frac{D_t(i, y)}{Z_t} \cdot \beta_t^{\frac{1}{2} \cdot (1 - h_t(x_i, y_i) + h_t(x_i, y))}$$

Computational validations

- Freund and Schapire (1996) report the measured computational results attained by the proposed approaches for various experiments taken from the UCI benchmark
- As mentioned above, these tests provide the following:
  - A comparison between Ada.Boost.M1 and Ada.Boost.M2, i.e., particularly, the impact of replacing error by pseudo-loss
  - A comparison of boosting and bagging on the basis of different weak learners
  - A consideration of the performance of the decision-tree approach C4.5 with and without boosting
  - A study of the performance of a learning algorithm which combines AdaBoost and a variant of the nearest neighbor classifier
- Firstly, we briefly introduce (sketch) the various weak learners applied by Freund and Schapire (1996) in the computational tests

Weak learners tested by Freund and Schapire (1996)

FindAttrTest

- Searches for the single attribute test that causes minimal error (or pseudo-loss when AdaBoost.M2 is applied)
- E.g., for a binary classifier, an attribute a with a value v is determined such that each new case x is classified as follows:
  - If case x does not possess a value of attribute a the classification is randomly chosen
  - If attribute a is discrete and case x possesses the value v for attribute a the classification is p_0
  - If attribute a is continuous and case x possesses a value smaller or equal to v for attribute a the classification is p_0
  - In all other cases, the classification is p_1
- FindAttrTest searches exhaustively for the classifier of the form given above with minimum error or pseudo-loss with respect to the distribution provided by the booster

Weak learners tested by Freund and Schapire (1996)

FindAttrTest

- Hence, this method has to check exhaustively all attributes and cases in the training set
- Therefore, with m training cases and n attributes this search can be executed with an asymptotic running time $O(n \cdot m)$. For extensions dealing with k classes, we have to add a factor of $O(k)$

Weak learners tested by Freund and Schapire (1996)

FindDecRule

- This algorithm requires an unweighted training set, so we use the resampling version of boosting
- First, the given training set is randomly divided into a growing set using 70% of the data, and a pruning set with the remaining 30% of given cases
- First phase
  - The growing set of cases of the data set is used to grow a list of attribute-value tests. The latter is initially empty, i.e., does not contain any test criterion
  - Analogous to FindAttrTest, each test compares a chosen attribute a to a value v
  - The procedure adds only one test at a time. An entropy-based potential function is used to decide about the growth of the list of tests. Specifically, the test is added that causes the greatest drop in potential
  - After the test is chosen, only one branch is expanded, namely, the branch with the highest remaining potential. The list continues to be grown in this fashion until no test remains which will further reduce the potential

Weak learners tested by Freund and Schapire (1996)

FindDecRule

- Second phase
  - The list is pruned by selecting the prefix of the list with minimum error (or pseudo-loss) on the pruning set
  - I.e., a sequence of test criteria is determined that causes minimum error

Weak learners tested by Freund and Schapire (1996)

C4.5

- This is the sophisticated decision tree algorithm proposed by Quinland (1993) and introduced in this course
- During the tests, all the default options including pruning are turned on
- As C4.5 expects an unweighted training sample, resampling is applied
- Moreover, AdaBoost.M2 is not applied as C4.5 is designed to minimize error, not pseudo-loss. Note that Freund and Schapire (1996) argue that pseudo-loss is not really helpful when using a weak learning algorithm as strong as C4.5, since such an algorithm will usually be able to find a hypothesis with error less than $\frac{1}{2}$

Adapting the weak learners – integrating plausibility

- The algorithm Adaboost.M2 introduced above requires that a weak learner generates a more detailed output function $h_t: X \times Y \mapsto [0,1]$
- However, so far, the learning algorithms presented in this lecture predict only an assigned single class label, but do not generate a detailed function defining the plausibility of the identified class label and all other (dismissed) class labels
- Fortunately, these learning algorithms can be modified or utilized accordingly in order to provide the required extended plausibility function
- For this purpose, we give an example how the FindAttrTest approach can be utilized

Boosting with trees in sklearn

- The FindAttrTest learner tested by Freund and Schapire (1996) extends the idea of decision stumps in order to provide a suitable plausibility function
- Unfortunately, the provided description of the implementation details is rather vague (see the preceding slides summarizing the description provided by the paper)
- Therefore, we will present an approach actually implemented in the python library *scikit learn,* which utilizes decision trees to generate the needed plausibility function
- This approach should be very similar to the one described by Freund and Schapire (1996)

Boosting in scikit learn

- The python library *scikit learn* implements the so called AdaBoost-SAMME and AdaBoost-SAMME.R approaches presented in Zhu et. al (2006) and Zhu et. al (2009)
- The second approach, SAMME.R uses similar to AdaBoost.M2 real-valued confidence-rated predictions such as weighted probability estimates, to update the weights
- We do not go into detail of these approaches and merely use the weighted class probability estimates as the plausibility estimates $h_t(x_i, y)$ in the AdaBoost.M2 approach

A rough implementation sketch

- The actual implementation of the scikit learn library is not easily readable on a slide
- Thus, we provide only a code snippet

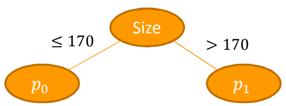
```
# Given:
#   X: list of lists with feature values of the items
#   Y: list of known classes of the items in X (coded as integers 0,...,k)
tree = DecisionTreeClassifier(...)
tree.fit(X,Y) # builds a decision stump
prediction = tree.predict(X) # returns predicted classes (dtype=int)
nodes = tree.apply(X) # returns a list containing a node index for each
                      # element x of X, indicating the node x is
                      # classified by
proba = []
for i,x in enumerate(X):
  proba_x = [0.] * no_classes
  leaf = nodes[i]
  for j,x in enumerate(nodes):
    if x == leaf:
      proba_x[Y[j]] += 1.
  s = sum(proba_x)
  proba_x = [ x / s for x in proba_x ]
  proba.append(proba_x)
```


An example

- In the following example, we determine probability estimates for a decision stump using the attribute “size” with value 170 as a threshold
- Adult individuals are classified by their gender „male“ or „female“, while children are classified simply as „child“

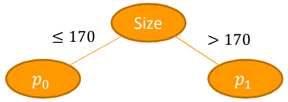
Size	125	143	150	163	167	173	180	182	187
Group	child	male	child	female	female	male	male	female	male

- Since for the attribute “size” there are no unknown values we commence with a binary decision tree with nodes p_0 and p_1



An example (continued)

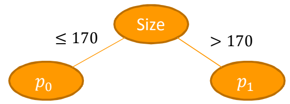
Given the decision tree below we can classify our items as either p_0 or p_1



Size	125	143	150	163	167	173	180	182	187
Group	child	male	child	female	female	male	male	female	male
Node	p_0	p_0	p_0	p_0	p_0	p_1	p_1	p_1	p_1

- Probability estimates for an item x_i with class y_i can now be derived from this result by computing the probability as the fraction of items of class y classified by the leaf node which also classifies x_i

An example (continued)



Size	125	143	150	163	167	173	180	182	187
Group	child	male	child	female	female	male	male	female	male
Node	p_0	p_0	p_0	p_0	p_0	p_1	p_1	p_1	p_1

Summary for node p_0				
	male	female	child	total
#	1	2	2	5
norm.	0.2	0.4	0.4	1.0

Summary for node p_1				
	male	female	child	total
#	3	1	0	4
norm.	0.75	0.25	0	1.0

An example (continued)

The needed plausibility values $h(x_i, y)$ are determined for each x_i according to the probability values of the leaf nodes

Summary for node p_0				
	male	female	child	total
#	1	2	2	5
norm.	0.2	0.4	0.4	1.0

Summary for node p_1				
	male	female	child	total
#	3	1	0	4
norm.	0.75	0.25	0	1.0

Size	125	143	150	163	167	173	180	182	187
Group	child	male	child	female	female	male	male	female	male
Node	p_0	p_0	p_0	p_0	p_0	p_1	p_1	p_1	p_1
$h(x_i, \boldsymbol{m})$	0.2	0.2	0.2	0.2	0.2	0.75	0.75	0.75	0.75
$h(x_i, \boldsymbol{f})$	0.4	0.4	0.4	0.4	0.4	0.25	0.25	0.25	0.25
$h(x_i, \boldsymbol{c})$	0.4	0.4	0.4	0.4	0.4	0.0	0.0	0.0	0.0

Further implementation issues

- Many learning algorithms can be modified to handle examples that are weighted by a distribution such as the one created by the boosting algorithm

  If this applies, the booster's distribution D_t is supplied directly to the weak learner (denoted as *boosting by reweighting*)
- However, some learning algorithms require an unweighted set of examples
  - In this case, a set of examples is chosen from the given data set independently at random according to the given distribution D_t with replacement
  - Note that the number of examples to be chosen on each round is a matter of discretion
  - Freund and Schapire (1996) chose m examples on each round, where m is the size of the original training set
  - This method is denoted as boosting by resampling

Further implementation issues

- Note that boosting by resampling is also possible when using the pseudo-loss (Freund and Schapire (1996) p.4)
  - In this case, a set of mislabels are chosen from the set B of all mislabels with replacement according to the given distribution D_t
  - Freund and Schapire (1996) used a sample of size $|B| = m \cdot (C - 1)$

The tested bagging approach

- The bagging algorithm is the one proposed by Breiman (1996)
- The method works by training each copy of the algorithm on a bootstrap sample, i.e., a sample of size m chosen uniformly at random with replacement from the original training set
- The multiple hypotheses that are computed are then combined using simple voting, i.e.,

  the final composite hypothesis classifies an example x to the class most often assigned by the underlying "weak" hypotheses
- In order to compare AdaBoost.M2, which uses pseudo-loss, to bagging, we also extended bagging in a natural way for use with a weak learning algorithm that minimizes pseudo-loss rather than ordinary error
- Such a weak learning algorithm expects to be provided with a distribution over the set B of all mislabels

The tested bagging approach

- On each round of bagging, we construct this distribution using the bootstrap method; that is, we select $|B|$ mislabels from B (chosen uniformly at random with replacement) and assign each mislabel weight $\frac{1}{|B|}$ times the number of times it was chosen
- The weak learner then provides the classification by combining the voting in a natural manner; namely, given x, the combined hypothesis outputs the label y which maximizes $\sum_t h_t(x, y)$

Differences between bagging and boosting

For either error or pseudo-loss, it can be summarized the following

1. bagging always uses resampling rather than reweighting
2. bagging does not modify the distribution over examples or mislabels, but instead always uses the uniform distribution
3. in forming the final hypothesis, bagging gives equal weight to each of the weak hypotheses

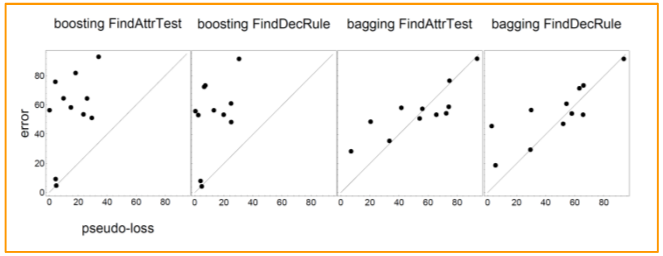
The real-world problems of the benchmark

name	# examples train	# examples test	# classes	# attributes disc.	# attributes cont.	missing values
soybean-small	47	-	4	35	-	-
labor	57	-	2	8	8	×
promoters	106	-	2	57	-	-
iris	150	-	3	-	4	-
hepatitis	155	-	2	13	6	×
sonar	208	-	2	-	60	-
glass	214	-	7	-	9	-
audiology.stand	226	-	24	69	-	×
cleve	303	-	2	7	6	×
soybean-large	307	376	19	35	-	×
ionosphere	351	-	2	-	34	-
house-votes-84	435	-	2	16	-	×
votes1	435	-	2	15	-	×
crx	690	-	2	9	6	×
breast-cancer-w	699	-	2	-	9	×
pima-indians-di	768	-	2	-	8	-
vehicle	846	-	4	-	18	-
vowel	528	462	11	-	10	-
german	1000	-	2	13	7	-
segmentation	2310	-	7	-	19	-
hypothyroid	3163	-	2	18	7	×
sick-euthyroid	3163	-	2	18	7	×
splice	3190	-	3	60	-	-
kr-vs-kp	3196	-	2	36	-	-
satimage	4435	2000	6	-	36	-
agaricus-lepiot	8124	-	2	22	-	-
letter-recognit	16000	4000	26	-	16	-

See Freund and Schapire (1996) p.5
and https://archive.ics.uci.edu/ml/index.php

Error vs. pseudo-loss with Boosting and Bagging

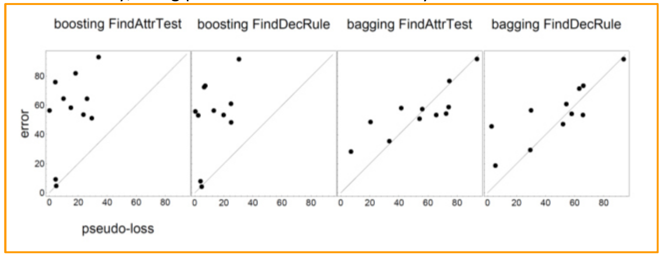
- Firstly, the two weak learners with boosting and bagging are directly compared for the two criteria error and pseudo-loss (average values of multiple repetitions) by the resulting errors (each point is one benchmark)
- It becomes obvious that boosting using pseudo-loss clearly outperforms boosting using error



See Freund and Schapire (1996) p.5

Error vs. pseudo-loss with Boosting and Bagging

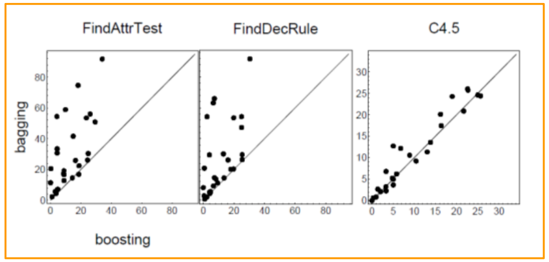
- For boosting, it becomes obvious that using pseudo-loss did dramatically better than error on every non-binary problem (except it did slightly worse on “iris” with three classes)
- As the figure shows, using pseudo-loss with bagging gave mixed results in comparison to ordinary error. Overall, pseudo-loss gave better results, but occasionally, using pseudo-loss hurt considerably



See Freund and Schapire (1996) p.5

Boosting versus Bagging

- For boosting, the error rate achieved using pseudo-loss is given. For bagging, the error rate achieved using either error or pseudo-loss, whichever gave the better result on that particular benchmark, is reported
- For the binary problems, and experiments with C4.5, only error was used



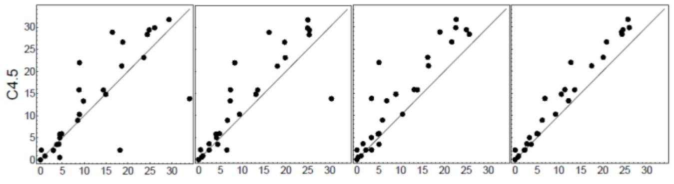
See Freund and Schapire (1996) p.6

Boosting versus Bagging

- For the simpler weak learning algorithms (FindAttr-Test and FindDecRule), boosting did significantly and uniformly better than bagging
- The boosting error rate was worse than the bagging error rate (using either pseudo-loss or error) on a very small number of benchmark problems, and on these, the difference in performance was quite small
- On average, for FindAttrTest, boosting improved the error rate over using FindAttrTest alone by 55.2%, compared to bagging which gave an improvement of only 11.0% using pseudo-loss or 8.4% using error. For FindDecRule, boosting improved the error rate by 53.0%, bagging by only 18.8% using pseudo-loss, 13.1% using error
- When using C4.5 as the weak learning algorithm, boosting and bagging seem more evenly matched, although boosting still seems to have a slight advantage. On average, boosting improved the error rate by 24.8%, bagging by 20.0%. Boosting beat bagging by more than 2% on 6 of the benchmarks, while bagging did not beat boosting by this amount on any benchmark. For the remaining 20 benchmarks, the difference in performance was less than 2%

C4.5 versus Boosting and Bagging

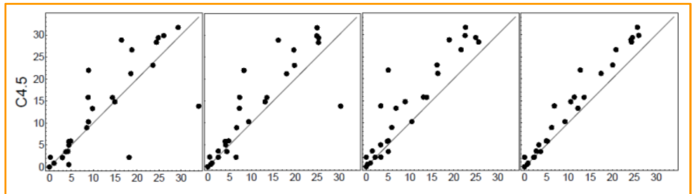
- Using boosting with FindAttrTest does quite well as a learning algorithm in its own right, in comparison to C4.5.
- This algorithm beat C4.5 on 10 of the benchmarks (by at least 2%), tied on 14, and lost on 3



See Freund and Schapire (1996) p.6

C4.5 versus Boosting and Bagging

- C4.5's improvement in performance over FindAttrTest was 49.3%
- Using boosting with FindDecRule did somewhat better
- The win-tie-lose numbers for this algorithm (compared to C4.5) were
  - 13-12-2,
  - and its average improvement over FindAttrTest was 58.1%



See Freund and Schapire (1996) p.6

Test error rates of various algorithms

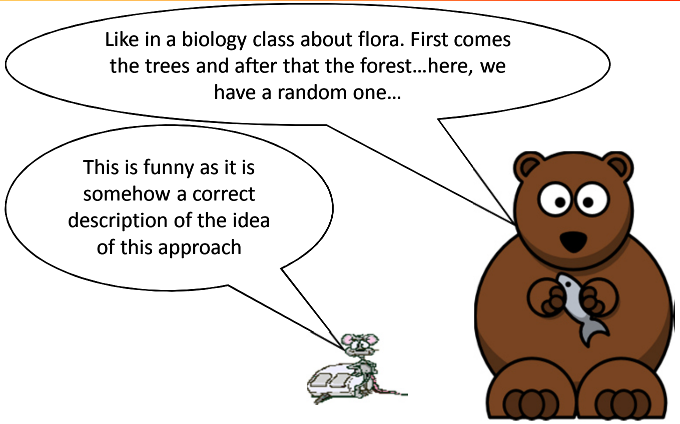
name	FindAttrTest error –	FindAttrTest error boost	FindAttrTest error bag	FindAttrTest pseudo-loss boost	FindAttrTest pseudo-loss bag	FindDecRule error –	FindDecRule error boost	FindDecRule error bag	FindDecRule pseudo-loss boost	FindDecRule pseudo-loss bag	C4.5 error –	C4.5 error boost	C4.5 error bag
soybean-small	57.6	56.4	48.7	0.2	20.5	51.8	56.0	45.7	0.4	2.9	2.2	3.4	2.2
labor	25.1	8.8	19.1			24.0	7.3	14.6			15.8	13.1	11.3
promoters	29.7	8.9	16.6			25.9	8.3	13.7			22.0	5.0	12.7
iris	35.2	4.7	28.4	4.8	7.1	38.3	4.3	18.8	4.8	5.5	5.9	5.0	5.0
hepatitis	19.7	18.6	16.8			21.6	18.0	20.1			21.2	16.3	17.5
sonar	25.9	16.5	25.9			31.4	16.2	26.1			28.9	19.0	24.3
glass	51.5	51.1	50.9	29.4	54.2	49.7	48.5	47.2	25.0	52.0	31.7	22.7	25.7
audiology.stand	53.5	53.5	53.5	23.6	65.7	53.5	53.5	53.5	19.9	65.7	23.1	16.2	20.1
cleve	27.8	18.8	22.4			27.4	19.7	20.3			26.6	21.7	20.9
soybean-large	64.8	64.5	59.0	9.8	74.2	73.6	73.6	73.6	7.2	66.0	13.3	6.8	12.2
ionosphere	17.8	8.5	17.3			10.3	6.6	9.3			8.9	5.8	6.2
house-votes-84	4.4	3.7	4.4			5.0	4.4	4.4			3.5	5.1	3.6
votes1	12.7	8.9	12.7			13.2	9.4	11.2			10.3	10.4	9.2
crx	14.5	14.4	14.5			14.5	13.5	14.5			15.8	13.8	13.6
breast-cancer-w	8.4	4.4	6.7			8.1	4.1	5.3			5.0	3.3	3.2
pima-indians-di	26.1	24.4	26.1			27.8	25.3	26.4			28.4	25.7	24.4
vehicle	64.3	64.4	57.6	26.1	56.1	61.3	61.2	61.0	25.0	54.3	29.9	22.6	26.1
vowel	81.8	81.8	76.8	18.2	74.7	82.0	72.7	71.6	6.5	63.2	2.2	0.0	0.0
german	30.0	24.9	30.4			30.0	25.4	29.6			29.4	25.0	24.6
segmentation	75.8	75.8	54.5	4.2	72.5	73.7	53.3	54.3	2.4	58.0	3.6	1.4	2.7
hypothyroid	2.2	1.0	2.2			0.8	1.0	0.7			0.8	1.0	0.8
sick-euthyroid	5.6	3.0	5.6			2.4	2.4	2.2			2.2	2.1	2.1
splice	37.0	9.2	35.6	4.4	33.4	29.5	8.0	29.5	4.0	29.5	5.8	4.9	5.2
kr-vs-kp	32.8	4.4	30.7			24.6	0.7	20.8			0.5	0.3	0.6
satimage	58.3	58.3	58.3	14.9	41.6	57.6	56.5	56.7	13.1	30.0	14.8	8.9	10.6
agaricus-lepiot	11.3	0.0	11.3			8.2	0.0	8.2			0.0	0.0	0.0
letter-recognit	92.9	92.9	91.9	34.1	93.7	92.3	91.8	91.8	30.4	93.7	13.8	3.3	6.8

See Freund and Schapire (1996) p.7

2.6.4 Random Forest

- Instead of applying various methods, Random Forest approaches solely uses decision trees as predictors
- In order to increase the variability of these trees and their predictions, besides randomly generating the training sets, the attributes assigned to inner nodes are also randomly chosen and are therefore tree-dependent

From tree to forest



A possible random forest procedure

1. Let n be the number of available attributes in the training set $\mathcal{L}$ (formerly denoted as the data set M). The training set comprises $|\mathcal{L}| = N$ cases
2. Determine the number of decision trees (classifiers) $|B|$ to be generated
3. For $i = 1$ to $|B|$ do
   - Generate randomly by **b**ootstrap **agg**regat**ing** the training set $\mathcal{L}^{(i)}$
   - Select randomly $\tilde{n} \leq n$ attributes from training set $\mathcal{L}$ and insert it into set $\mathcal{M}^{(i)}$
   - Train CART on training set $\mathcal{L}^{(i)}$ by solely using the attributes of set $\mathcal{M}^{(i)}$
   - No pruning is applied
4. End For

Finding appropriate values for $\tilde{n}$

- Frochte (2018) p.155 reports that appropriate values for a sufficiently reliable approach are $\tilde{n} = \lfloor \log_2 n \rfloor$ or $\tilde{n} = \lfloor \sqrt{n} \rfloor$
- By considering current libraries, for instance scikit-learn (Python), this choice depends on the sought classification
  - For class labels, $\tilde{n} = \lfloor \sqrt{n} \rfloor$ is proposed as a default value
  - For numerical classifications, the default value is $\tilde{n} = n$

Optimizing the number of generated trees $|B|$

- Besides choosing the number of drawn attributes, the **size of the forest, i.e., the number of generated trees** is another important parameter of the random forest algorithm
- Due to the random generation of the different training sets $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(B)}$, we face the situation that each tree is trained on the basis of a specific set $\mathcal{L}^{(i)}$, but this set does not comprise all cases. Hence, remaining cases of other sets can be used for parameter tuning or testing
- For Optimizing the number of generated trees $|B|$, Frochte (2018) p.156 illustrates three possible approaches
  - First, we generate the so-called **out-of-bag error** of cases x_i that belong to some set $\mathcal{L}^{(i)}$ by testing trees $j \neq i$ that do not have trained on x_i, i.e., $x_i \notin \mathcal{L}^{(j)}$. By doing so for all cases in all generated sets, we obtain an average error. As long as this error is decreased by adding a tree, we do so (i.e., by setting $N \coloneqq N + 1$)

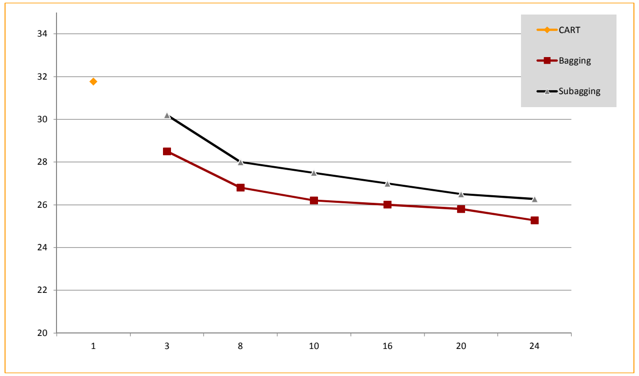
Optimizing the number of generated trees $|B|$

- Second, as trees can be generated in parallel by different resources (Computers, CPUs, or cores), an obvious limitation of the number of generated trees may be also the number of available computational resources that are available for the time in question
- Third, if there are experiences with applications of the random forest approach, it is reasonable to consider these empirical values

Evaluation of Random Forest

- Frochte (2018) reports on page 159 some measured computational results comparing the performance of a single CART decision tree, a random forest with up to 24 trees generated by bagging, and a random forest with up to 24 trees generated by subagging
- Subagging is conducted with 50% (perc=0.5), i.e., only 50 percent of the cases is randomly drawn for each generated tree

Evaluation of Random Forest



See Frochte (2018) p.159

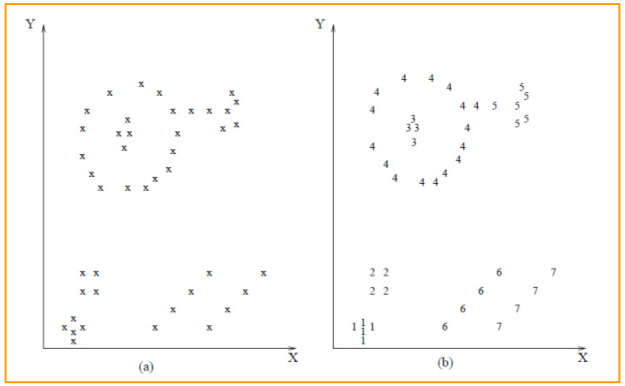
Evaluation of Random Forest – Results

- The generation of a considerable number of decision trees by applying bagging and subagging proves to be a promising approach for improving the validity of decision tree classifiers
- Particularly, the full bagging approach reduces the average error from 31,76 produced by CART with a single tree to 25,27 attained by the bagging approach using 24 trees
- By using only 50 percent of the stored cases in the data set, subagging averagely attains 26,27 errors with 24 generated trees

2.7 Clustering approaches

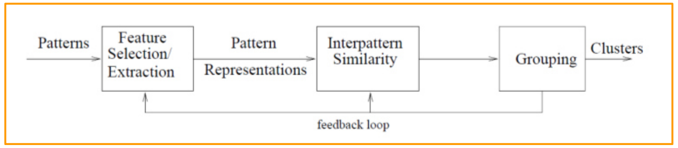
- Data clustering is a process that tries to identify groups or clusters within multidimensional data (the following general depictions are based on Jain, Murty, and Flynn (1999) and Omram, Engelbrecht, and Salman, A. (2007))
- For this purpose, similarity between items (cases) is determined by suitable application-dependent distance measures
- Clustering approaches have various applications in Artificial Intelligence approaches. Among others, these are, for instance,
  - image segmentation
  - vector and color image quantization (data compression techniques)
  - data mining, or machine learning
- Each cluster itself is characterized by the assigned items (cases) and the resulting cluster center (also denoted as the barycenter or centroid of the cluster)
- Clustering comprises considerable complexity as it is an unsupervised approach that has to identify and exploit possible patterns in the data

Clustering – Illustration



See Jain, A.K.; Murty, M.N.; Flynn, P.J. (1999) p.266

Clustering steps



See Jain, A.K.; Murty, M.N.; Flynn, P.J. (1999) p.267

Basic steps (see Jain, A.K.; Murty, M.N.; Flynn, P.J. (1999) p.267)

1. pattern representation (optionally including feature extraction and/or selection)
2. definition of a pattern proximity measure appropriate to the data domain,
3. clustering or grouping,
4. data abstraction (if needed), and
5. assessment of output (if needed)

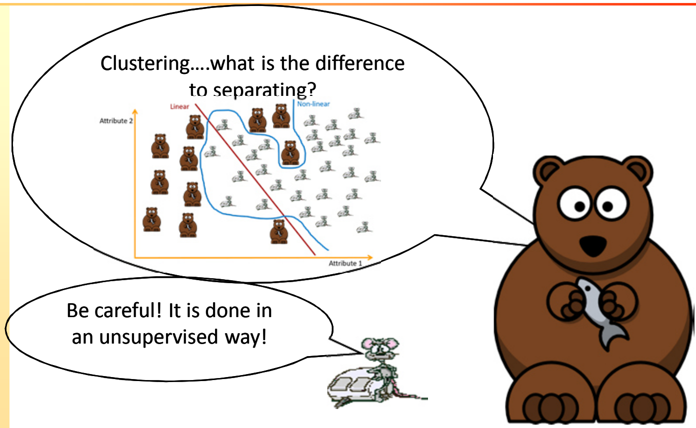
Clustering steps – some details

- **Pattern representation** comprises the determination of the
  - Number of classes
  - Number of available patterns
  - Number, type, and scale of the features
  - Note that some of these information may not be controllable, but externally given by the data set
- **Feature selection** has to identify the most effective subset of the original features to use in clustering
- **Feature extraction** is the use of one or more transformations of the input features to produce new salient features
- **Pattern proximity** is usually measured by a distance function defined on pairs of patterns. The literature provides a variety of distance measures
- **Grouping step** provides a clustering of cases into groups. This clustering can be hard or fuzzy (with variable degree of membership). There are various grouping algorithms in the literature

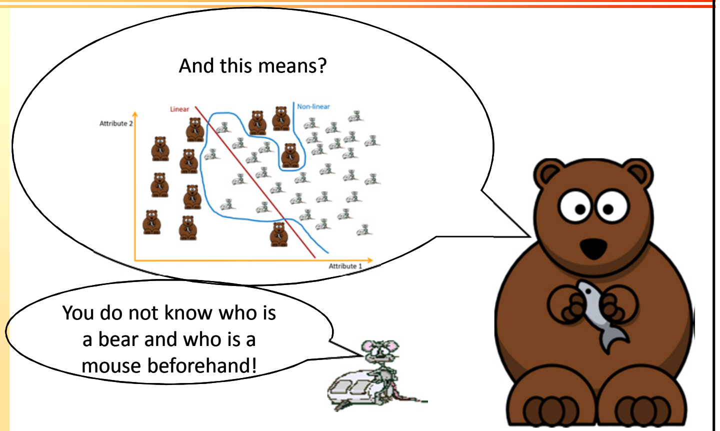
Clustering steps – some details

- **Data abstraction** is the process of extracting a simple and compact representation of a data set. This is done either from
  - the perspective of automatic analysis (so that a machine can perform further processing efficiently)
  - or it is human-oriented (so that the representation obtained is easy to comprehend and intuitively appealing)
- In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes or representative patterns such as the centroid
- **Cluster validity** analysis assesses the output quality of a clustering approach. Note that a clustering output is valid if it cannot reasonably have occurred by chance. There are three basic types of validation studies
  - *external assessment* of validity compares the recovered structure to an a priori structure
  - *internal examination* tries to determine whether the structure is intrinsically appropriate for the data
  - *relative test* compares two structures and measures their relative merit

What is different about clustering?



From tree to forest



2.7.1 Basics

- We define the following terms and notations
  - A pattern (feature vector) x is a single item used by the applied clustering algorithm. It is usually defined by a vector of d measurements, i.e., $\mathbb{x} = (x_1, x_2, \dots, x_d)$
  - Each scalar component x_i of a pattern x is denoted as a feature (or an attribute)
  - The parameter d gives the dimensionality of the pattern space
  - A pattern set is denoted by $\mathcal{H} = \{\mathbb{x}_1, \mathbb{x}_2, \dots, \mathbb{x}_n\}$ and comprises n patterns. Each pattern $\mathbb{x}_i$ is denoted by $(x_{i,1}, x_{i,2}, \dots, x_{i,d})$, i.e., a pattern set to be clustered can be viewed as an $n \times d$ pattern matrix
  - A class refers to a state of nature that governs the pattern generation process in some cases. More concretely, a class can be viewed as a source of patterns whose distribution in feature space is governed by a probability density specific to the class

Basic parameters

- A hard clustering assigns a class label l_i to each pattern $\mathbb{x}_i$ that identifies its class unambiguously. The set of all labels for a pattern set $\mathcal{H} = \{\mathbb{x}_1, \mathbb{x}_2, \dots, \mathbb{x}_n\}$ is denoted as $\mathcal{L}(\mathcal{H}) = \{l_1, l_2, \dots, l_n\}$ with $l_i \in \{1,2,\dots,k\}$ giving the index of the cluster $\mathbb{x}_i$ is assigned to while k determines the total number of clusters
- Fuzzy clustering procedures assign to each pattern $\mathbb{x}_i$ a fractional degree of membership $f_{i,j}$ in each output cluster $j \in \{1,2,\dots,k\}$
- A distance measure (a specialization of a proximity measure) is a metric (or quasi-metric) on the feature space used to quantify the similarity of patterns

Distance measures (or metric)

- As mentioned above, clustering has to group items into clusters in order to attain the following objectives
  - Inner cluster homogeneity: Items that are assigned to the same cluster possess similar attribute values
  - Inter cluster heterogeneity: The attribute values of items that are assigned to different clusters differ considerably. As a consequence, cluster centers or centroids are distinguishable and enable to derive significant cognitions
- For this purpose, we have to mathematically define what is similarity, or, with other words, how can we differences between our items (cases) in the data set
- This requires the introduction and application of a metric

Metric

2.7.1.1 **Definition**

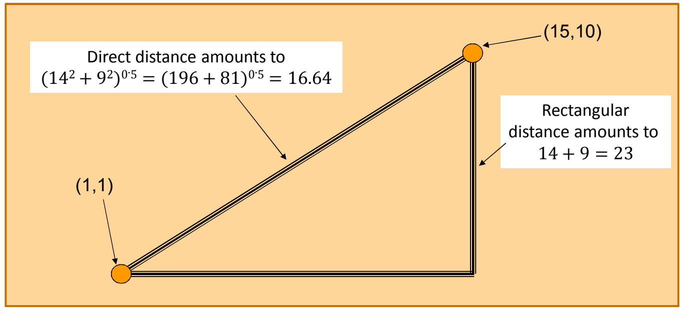
A metric on a set X is a function (also denoted as a distance function or just a distance) $d: X \times X \mapsto [0,\infty) \subseteq \mathbb{R}$ with $[0,\infty)$ being all positive real numbers such that the following restrictions are fulfilled

1. $\forall x,y \in X: d(x,y) \geq 0$
2. $\forall x,y \in X: d(x,y) = 0 \Rightarrow x = y$
3. $\forall x,y \in X: d(x,y) = d(y,x)$
4. $\forall x,y,z \in X: d(x,y) \leq d(x,z) + d(z,y)$

2.7.1.2 The Minkowski distance

- One general distance measure is the well-known Minkowski distance
$$d^{\alpha}(\mathbb{x}_1, \mathbb{x}_2) = \left(\sum_{j=1}^{d} |x_{1,j} - x_{2,j}|^{\alpha}\right)^{1/\alpha}$$
- Due to the parameter α, it allows for various applications
- For instance, by setting α to 2, we obtain the Euclidean distance measure
$$d^{2}(\mathbb{x}_1, \mathbb{x}_2) = \sqrt{\sum_{j=1}^{d} (x_{1,j} - x_{2,j})^2} = \|\mathbb{x}_1 - \mathbb{x}_2\|$$
- Alternatively, if we set α to 1, we obtain the Rectangular distance measure (also denoted as the Manhattan distance measure)
$$d^{1}(\mathbb{x}_1, \mathbb{x}_2) = \sum_{j=1}^{d} |x_{1,j} - x_{2,j}|$$

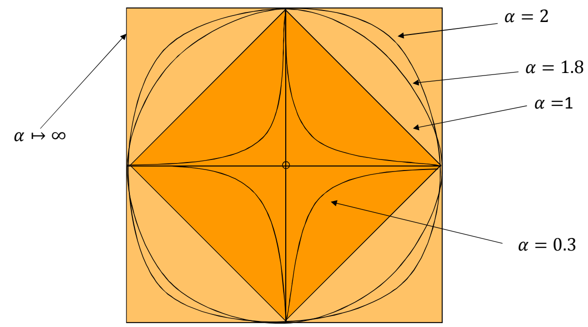
Illustration



- The Rectangular distance measure just adds the distances over all attribute value pairs
- In contrast to this, the Euclidean distance measures the direct distance between the vectors

Illustration

We consider the $\mathbb{R}^2$ and illustrate all points $(x_1, x_2) \in \mathbb{R}^2$ with distance $d^{\alpha}(0, \mathbb{x}) = 1$ to the origin $(0,0)$



Consequences

2.7.1.3 **Lemma**
The Minkowski distance is a metric for $\alpha \geq 1$, but not for $\alpha < 1$.

Proof:
We start with $\alpha \geq 1$. Please note that the following proof (copied from a former script) denotes the Minkowski distance d^{α} by l_p:

Now, we have to show:

$$\forall x, y, z \in IR^n : l_p(x,y) + l_p(y,z) \geq l_p(x,z)$$

Note, that the function

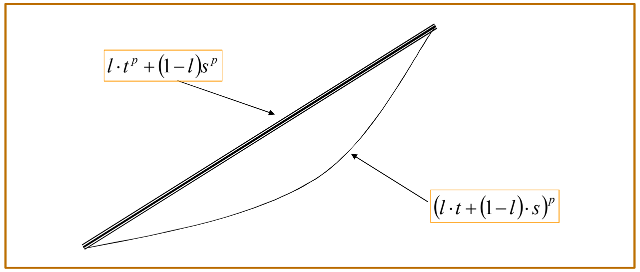
$$f(t) = t^p \text{ is convex for all } t \geq 0 \text{ and } p \geq 1$$

$$\text{since for } t > 0 : \frac{\partial f(t)}{\partial t} = p \cdot t^{p-1} \wedge \frac{\partial \frac{\partial f(t)}{\partial t}}{\partial t} = p \cdot (p-1) \cdot t^{p-2} \geq 0$$

Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

Due to the convexity, we obtain that $\forall l \in \mathbb{R}$ with $0 \leq l \leq 1$, it holds that $(l \cdot t + (1 - l) \cdot s)^p \leq l \cdot t^p + (1 - l) \cdot s^p$

Illustration:



Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

We substitute :

$$t=\frac{|x_i-y_i|}{l_p(x,y)},\ s=\frac{|y_i-z_i|}{l_p(y,z)} \text{ and } l=\frac{l_p(x,y)}{l_p(x,y)+l_p(y,z)}$$

and therefore, receive the following modified formula :

$$\forall l\in IR:(0\leq l\leq 1):\left(l\cdot \underbrace{t}_{\frac{|x_i-y_i|}{l_p(x,y)}}+(1-l)\cdot \underbrace{s}_{\frac{|y_i-z_i|}{l_p(y,z)}}\right)^p\leq l\cdot \underbrace{t}_{\frac{|x_i-y_i|}{l_p(x,y)}}{}^p+(1-l)\underbrace{s}_{\frac{|y_i-z_i|}{l_p(y,z)}}{}^p$$

$$\Leftrightarrow\left(l\cdot\frac{|x_i-y_i|}{l_p(x,y)}+(1-l)\cdot\frac{|y_i-z_i|}{l_p(y,z)}\right)^p\leq l\cdot\left(\frac{|x_i-y_i|}{l_p(x,y)}\right)^p+(1-l)\cdot\left(\frac{|y_i-z_i|}{l_p(y,z)}\right)^p$$

Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

$$\Leftrightarrow\left(l\cdot\frac{|x_i-y_i|}{l_p(x,y)}+(1-l)\cdot\frac{|y_i-z_i|}{l_p(y,z)}\right)^p\leq l\cdot\left(\frac{|x_i-y_i|}{l_p(x,y)}\right)^p+(1-l)\cdot\left(\frac{|y_i-z_i|}{l_p(y,z)}\right)^p$$

Substitute $l=\frac{l_p(x,y)}{l_p(x,y)+l_p(y,z)}$:

$$\Leftrightarrow\left(\frac{l_p(x,y)}{l_p(x,y)+l_p(y,z)}\cdot\frac{|x_i-y_i|}{l_p(x,y)}+\frac{l_p(y,z)}{l_p(x,y)+l_p(y,z)}\cdot\frac{|y_i-z_i|}{l_p(y,z)}\right)^p$$

$$\leq\frac{l_p(x,y)}{l_p(x,y)+l_p(y,z)}\cdot\left(\frac{|x_i-y_i|}{l_p(x,y)}\right)^p+\frac{l_p(y,z)}{l_p(x,y)+l_p(y,z)}\cdot\left(\frac{|y_i-z_i|}{l_p(y,z)}\right)^p$$

$$\Leftrightarrow\left(\frac{|x_i-y_i|}{l_p(x,y)+l_p(y,z)}+\frac{|y_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p$$

$$\leq\frac{l_p(x,y)}{l_p(x,y)+l_p(y,z)}\cdot\left(\frac{|x_i-y_i|}{l_p(x,y)}\right)^p+\frac{l_p(y,z)}{l_p(x,y)+l_p(y,z)}\cdot\left(\frac{|y_i-z_i|}{l_p(y,z)}\right)^p$$

Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

$$\Leftrightarrow\left(\frac{|x_i-y_i|+|y_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p\leq\frac{l_p(x,y)\cdot\left(\frac{|x_i-y_i|}{l_p(x,y)}\right)^p+l_p(y,z)\cdot\left(\frac{|y_i-z_i|}{l_p(y,z)}\right)^p}{l_p(x,y)+l_p(y,z)}$$

$$\Leftrightarrow\left(\frac{|x_i-y_i|+|y_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p\leq\frac{l_p(x,y)\cdot\frac{|x_i-y_i|^p}{l_p(x,y)^p}+l_p(y,z)\cdot\frac{|y_i-z_i|^p}{l_p(y,z)^p}}{l_p(x,y)+l_p(y,z)}$$

$$\Leftrightarrow\left(\frac{|x_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p\leq\left(\frac{|x_i-y_i|+|y_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p\leq\frac{l_p(x,y)\cdot\frac{|x_i-y_i|^p}{l_p(x,y)^p}+l_p(y,z)\cdot\frac{|y_i-z_i|^p}{l_p(y,z)^p}}{l_p(x,y)+l_p(y,z)}$$

$$\Leftrightarrow\left(\frac{|x_i-z_i|}{l_p(x,y)+l_p(y,z)}\right)^p\leq\frac{l_p(x,y)\cdot\frac{|x_i-y_i|^p}{l_p(x,y)^p}+l_p(y,z)\cdot\frac{|y_i-z_i|^p}{l_p(y,z)^p}}{l_p(x,y)+l_p(y,z)}$$

Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

$$\Leftrightarrow\frac{|x_i-z_i|^p}{(l_p(x,y)+l_p(y,z))^p}\leq\frac{l_p(x,y)\cdot\frac{|x_i-y_i|^p}{l_p(x,y)^p}+l_p(y,z)\cdot\frac{|y_i-z_i|^p}{l_p(y,z)^p}}{l_p(x,y)+l_p(y,z)}$$

Summing over $i=1,2,\ldots,n$ *we get :*

$$\Leftrightarrow\frac{l_p(x,z)^p}{(l_p(x,y)+l_p(y,z))^p}\leq\frac{l_p(x,y)\cdot\frac{l_p(x,y)^p}{l_p(x,y)^p}+l_p(y,z)\cdot\frac{l_p(y,z)^p}{l_p(y,z)^p}}{l_p(x,y)+l_p(y,z)}$$

$$\Leftrightarrow\frac{l_p(x,z)^p}{(l_p(x,y)+l_p(y,z))^p}\leq\frac{l_p(x,y)+l_p(y,z)}{l_p(x,y)+l_p(y,z)}=1$$

$$\Leftrightarrow l_p(x,z)^p\leq(l_p(x,y)+l_p(y,z))^p$$

$$\Leftrightarrow l_p(x,z)\leq l_p(x,y)+l_p(y,z)\qquad q.e.d.$$

Proof of Lemma 2.7.1.3 – $\alpha \geq 1$

As d^α also fulfills the first three remaining restrictions of Definition 2.7.1.1, d^α is a metric for $\alpha \geq 1$

Proof of Lemma 2.7.1.3 – $\alpha < 1$

- We consider the case $\alpha < 1$ and the distance between $(0,0)$ and $(1,1)$
- Obviously, it holds that $d^\alpha\big((0,0),(1,1)\big) = \sqrt[\alpha]{1^\alpha + 1^\alpha} = \sqrt[\alpha]{2} = 2^{1/\alpha} > 2$, as, due to $\alpha < 1$, it holds that $1/\alpha > 1$
- However, the additional vector $(0,1)$ possesses the distance 1 to both, namely to $(0,0)$ and $(1,1)$
  - $d^\alpha\big((0,0),(0,1)\big) = \sqrt[\alpha]{0^\alpha + 1^\alpha} = \sqrt[\alpha]{1} = 1$
  - $d^\alpha\big((0,1),(1,1)\big) = \sqrt[\alpha]{1^\alpha + 0^\alpha} = \sqrt[\alpha]{1} = 1$
- Hence, it holds that
$$2 = d^\alpha\big((0,0),(0,1)\big) + d^\alpha\big((0,1),(1,1)\big) < d^\alpha\big((0,0),(1,1)\big)$$
- Therefore, the fourth restriction of Definition 2.7.1.1 is not fulfilled and d^α is not a metric for $\alpha < 1$

Considering $\lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2)$

2.7.1.3 **Lemma**

$$\forall \mathbb{x}_1, \mathbb{x}_2 \in \mathbb{R}^d \colon \lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) = \max\big\{ \left|x_{1,j} - x_{2,j}\right| \mid 1 \leq j \leq d \big\}$$

Proof:

We denote as $\hat{d} = \left|x_{1,j} - x_{2,j}\right| = \max\big\{ \left|x_{1,k} - x_{2,k}\right| \mid k \in \{1, \dots, d\} \big\}, j \in \{1, \dots, d\}$

$$\lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) = \lim_{\alpha \mapsto \infty} \left(\sum_{j=1}^{d} \left|x_{1,j} - x_{2,j}\right|^\alpha \right)^{1/\alpha} = \lim_{\alpha \mapsto \infty} \left(\hat{d}^\alpha \cdot \frac{\sum_{j=1}^{d} \left|x_{1,j} - x_{2,j}\right|^\alpha}{\hat{d}^\alpha} \right)^{1/\alpha} =$$

$$\lim_{\alpha \mapsto \infty} \hat{d} \cdot \left(\frac{\sum_{j=1}^{d} \left|x_{1,j} - x_{2,j}\right|^\alpha}{\hat{d}^\alpha} \right)^{\frac{1}{\alpha}} \leq \lim_{\alpha \mapsto \infty} \hat{d} \cdot \left(\frac{\sum_{j=1}^{d} \hat{d}^\alpha}{\hat{d}^\alpha} \right)^{\frac{1}{\alpha}} \leq \lim_{\alpha \mapsto \infty} \hat{d} \cdot \left(\sum_{j=1}^{d} 1 \right)^{\frac{1}{\alpha}} \leq \lim_{\alpha \mapsto \infty} \hat{d} \cdot \sqrt[\alpha]{d}$$

Hence, we conclude that

$$\lim_{\alpha \mapsto \infty} \hat{d} \cdot 1^{\frac{1}{\alpha}} \leq \lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) \leq \lim_{\alpha \mapsto \infty} \hat{d} \cdot \sqrt[\alpha]{d}$$

$$\Leftrightarrow \hat{d} \cdot \lim_{\alpha \mapsto \infty} 1^{\frac{1}{\alpha}} \leq \lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) \leq \hat{d} \cdot \lim_{\alpha \mapsto \infty} \sqrt[\alpha]{d}$$

$$\Leftrightarrow \hat{d} \leq \lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) \leq \hat{d}$$

And thus, we obtain $\lim_{\alpha \mapsto \infty} d^\alpha(\mathbb{x}_1, \mathbb{x}_2) = \hat{d}$

2.7.2 k-means clustering

- The well-known k-means clustering approach is widely used in practice and science
- Therefore, there are **various variants and extensions**
- In the **classic version**, one is given an integer $k \in \mathbb{N}$ and a set $\mathcal{X} \subset \mathbb{R}^d$ of n data points defined by vectors of attribute values in the $\mathbb{R}^d$
- The goal is to determine k center points (building the set $\mathcal{C}$) such that the squared Euclidean distance of each data point to the closest located chosen center point is minimized. Hence, it holds that
$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2$$
- Clearly, the determination defines a clustering of the data points as we assign each data point to the closest located chosen center
- Unfortunately, solving this problem exactly is NP-hard (see Drineas, Frieze, Kannan, Vempala and Vinay (2004)), even with just two clusters

The basic k-means algorithm

We give the description of this basic algorithm that can be found in the paper of Arthur and Vassilvitskii (2007). This description bases on the method originally proposed by Lloyd (1982)

Note that C_i defines a set of points, while c_i gives its current center of mass

1. Arbitrarily choose k initial centers $\mathcal{C} = \{c_1, c_2, \dots, c_k\}$
2. For each $i \in \{1, \dots, k\}$, update the cluster C_i to be the set of points in $\mathcal{X}$ that are closer to c_i than they are to c_j for all $j \neq i$
3. For each $i \in \{1, \dots, k\}$, set c_i to the center of mass of all points currently assigned to set C_i, i.e., compute $c_i = \frac{1}{|C_i|} \cdot \sum_{x \in C_i} x$
4. Repeat the steps 2 and 3 as long as C changes during the last iteration

Originally, it was standard practice to choose the initial centers uniformly at random from $\mathcal{X}$.

Observations

- For Step 2, ties may be broken arbitrarily, as long as the method is consistent
- The execution of the steps 2 and 3 guarantee to decrease ϕ
- This results from the following cognitions

2.7.2.1 **Lemma**

Let S be a set of points in the $\mathbb{R}^d$ with a mass denoted as $c(S)$. Moreover, let z be an arbitrary point in the $\mathbb{R}^d$. Then, it holds that

$$\sum_{x \in S} \|x - z\|^2 - \sum_{x \in S} \|x - c(S)\|^2 = |S| \cdot \|c(S) - z\|^2$$

Reduction of the objective value by step 3

- We make use of Lemma 2.7.2.1 in order to show that step 3 reduces the objective solution value
- For this purpose, we consider step 3 and assume that z is the initial center of a cluster S. Then, $\sum_{x \in S} \|x - z\|^2$ gives the contribution of all current members of S to ϕ. By adding a new element y (z' is the center of its previous cluster), the center moves to $c(S \cup \{y\})$ and we obtain the reduction by step 3

$$\begin{aligned} &\|y - z'\|^2 + \sum_{x \in S} \|x - z\|^2 - \|y - c(S \cup \{y\})\|^2 - \sum_{x \in S} \|x - c(S \cup \{y\})\|^2 \\ &> \sum_{x \in S \cup \{y\}} \|x - z\|^2 - \sum_{x \in S \cup \{y\}} \|x - c(S \cup \{y\})\|^2 \\ &= (|S| + 1) \cdot \|c(S \cup \{y\}) - z\|^2 \geq 0 \text{ (Lemma 2.7.2.1)} \end{aligned}$$

as $\|y - z'\|^2 > \|y - z\|^2 \geq \|y - c(S \cup \{y\})\|^2$ holds for the former center z' of the cluster that y was assigned to before

Observations

- Hence, the algorithm iteratively makes local improvements to an arbitrary clustering until it is no longer possible to do so
- Arthur and Vassilvitskii (2007) state that “the k-means algorithm is attractive in practice because it is simple and it is generally fast. Unfortunately, it is guaranteed only to find a local optimum, which can often be quite poor.”

2.7.3 k-means++ algorithm

- Arthur and Vassilvitskii (2007) propose the following extension of the k-means algorithm
- The main intention was to improve the performance of k-means by augmenting it with a very simple, randomized seeding technique
- Specifically, the initialization of the k-means procedure is done by choosing random starting centers with very specific probabilities

k-means++ clustering – Replacing step 1

Step 1:

a. Choose an initial center c_1 uniformly at random from set $\mathcal{X}$

b. Choose the next center c_i, selecting $c_i = x' \in \mathcal{X}$ with probability $\frac{D(x')^2}{\sum_{x \in \mathcal{X}} D(x)^2}$

c. Repeat step 1.b until altogether k centers are chosen

The steps 2-4 are identical with the ones of the original k-means clustering approach

The abbreviation $D(x)$ (used in step 1.b) denotes the shortest distance of data point $x \in \mathcal{X}$ to the closest center already generated in the previous iterations of the steps 1.a and 1.b.

Arthur and Vassilvitskii (2007) denote the weighting in step 1.b as the D^2-weighting

Main theoretical result of k++

Arthur and Vassilvitskii (2007) prove that the following Theorem holds even after conducting the step 1 (the modified step)

2.7.3.1 **Theorem**

If the clustering C is constructed with k-means++, then the corresponding objective function ϕ satisfies $E[\phi] \leq 8(ln(k) + 2) \cdot \phi_{OPT}$, with ϕ_{OPT} being the objective value of the optimal clustering

Note that Arthur and Vassilvitskii (2007) also show that – within a constant factor – this bound is tight

Computational results – k-means versus k++

- Arthur and Vassilvitskii (2007) evaluated the performance of k-means and k-means++ on four data sets, namely
  - *Norm25* is a synthetic data set, i.e., 25 “true” centers are uniformly drawn at random from a 15-dimensional hypercube of side length 500. Then, points are added from Gaussian distributions of variance 1 around each true center. This procedure resulted in a number of well separated Gaussians with the true centers providing a good approximation to the optimal clustering
  - In contrast to this, the remaining datasets from real-world examples off the UC-Irvine Machine Learning Repository
    - *Cloud data set*: 1,024 points in 10 dimensions, and it is Philippe Collard’s first cloud cover data base

Computational results – k-means versus k-means++

- *Intrusion data set* consists of 494,019 points in 35 dimensions, and it represents features available to an intrusion detection system
- *Spam data set* consists of 4,601 points in 58 dimensions, and it represents features available to an e-mail spam detection system
- For each data set, the authors tested the settings $k = 10, 25,$ and 50.
- Since randomized seeding processes are tested, 20 trials for each case were conducted, while the minimum and the average potential (actually divided by the number of points), as well as the mean running time are reported
- Percentage improvements are $100 \cdot \left(1 - \frac{k\text{-means++ value}}{k\text{-means value}}\right)$ (%)

Results – k-means versus k-means++

- *Norm25* data set (see above)

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$1.365 \cdot 10^5$	8.47%	$1.174 \cdot 10^5$	0.93%	0.12	46.72%
25	$4.233 \cdot 10^4$	99.96%	$1.914 \cdot 10^4$	99.92%	0.90	87.79%
50	$7.750 \cdot 10^3$	99.81%	$1.474 \cdot 10^1$	0.53%	2.04	−1.62%

- *Cloud* data set ($n = 1{,}024, d = 10$)

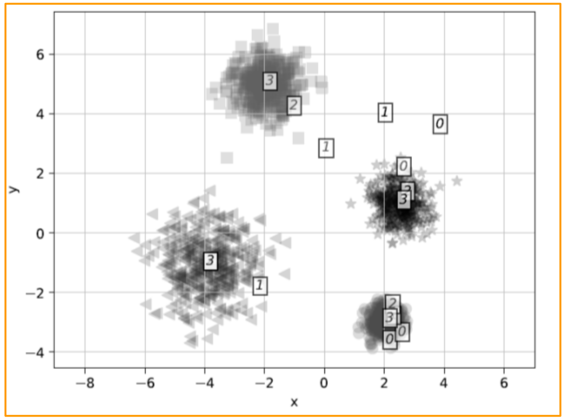
k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$7.921 \cdot 10^3$	22.33%	$6.284 \cdot 10^3$	10.37%	0.08	51.09%
25	$3.637 \cdot 10^3$	42.76%	$2.550 \cdot 10^3$	22.60%	0.11	43.21%
50	$1.867 \cdot 10^3$	39.01%	$1.407 \cdot 10^3$	23.07%	0.16	41.99%

Results – *Cloud* data set ($n = 4{,}601, d = 58$)

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$3.698 \cdot 10^4$	49.43%	$3.684 \cdot 10^4$	54.59%	2.36	69.00%
25	$3.288 \cdot 10^4$	88.76%	$3.280 \cdot 10^4$	89.58%	7.36	79.84%
50	$3.183 \cdot 10^4$	95.35%	$2.384 \cdot 10^4$	94.30%	12.20	75.76%

- In almost all settings, k-means++ clearly outperforms k-means for all measured criteria, i.e., k-means++ consistently outperformed k-means, both by achieving a lower potential value, in some cases by several orders of magnitude, and also by having a faster running time
- The D^2 seeding is slightly slower than uniform seeding, but it still leads to a faster algorithm since it helps the local search converge after fewer iterations
- Arthur and Vassilvitskii (2007) report that the synthetic example is a case where the standardized k-means algorithm performs very badly. Although there is an "obvious" clustering, the uniform seeding will inevitably merge some of these clusters, and the local search will never be able to split them apart

Applying k-means – example 1

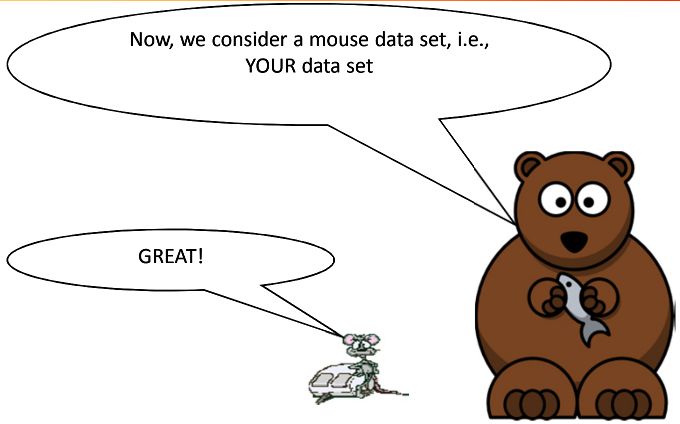


See Frochte (2018) p.311

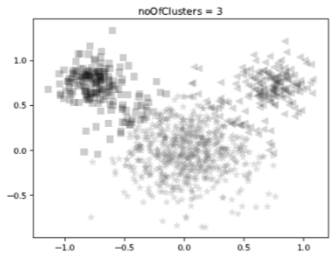
Applying k-means – example 1

- This first example shows that although the starting values are not useful, the algorithm was able to identify suitable clusters
- This was possible by conducting only four iterations

Validation of the approach



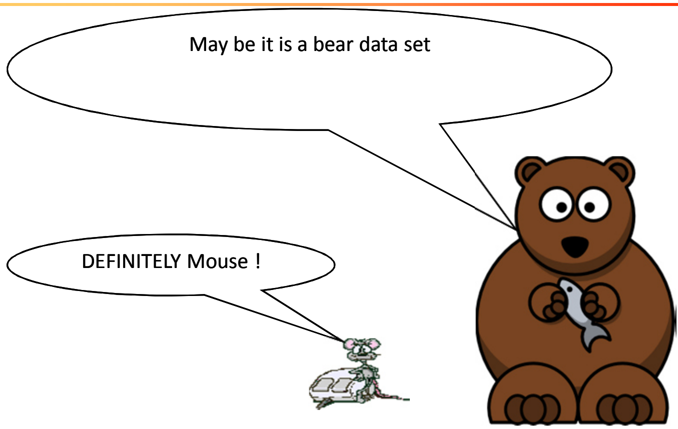
Applying k-means – example 2



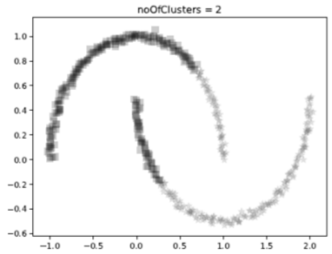
See Frochte (2018) p.312

- Mouse data set (can you identify the mouse?)
- Separation of ears and face by predetermining the finding of three clusters
- Result shows that k-means has a tendency towards equally sized clusters
- As a consequence, the ears include parts of the face

Validation of the approach



Applying k-means – example 3



See Frochte (2018) p.312

- Although the graphical result confuses the human visual impression, it results from the definition of the applied distance function
- More intuitive would be a clustering that addresses density issues
- In this case, the both paths would be interpreted as the respective clusters
- Further examples can be found in Frochte (2018) on page 312

2.7.4 Fuzzy C-means

- The k-means and k-means++ algorithms assign each case (point) unambiguously to one cluster
- I.e., the clustering is deterministic
- However, in many applications, such a deterministic assignment is not useful
- For instance, if we group existing cities into, let say, four categories that assess their meaning and size
  - Metropolis (world city)
  - Meaningful center
  - Medium-sized
  - Small-sized
- In this case, an assignment is not always clear
- The assignment of New York City or Peking seems to be quite clear. But, what about Berlin, Munich, or Madrid?

Applying a fuzzy assignment

- For this purpose, a Fuzzy variant of the k-means algorithm is originally proposed by Dunn (1973)
- Note that various extensions to this basic approach exist
- To introduce this algorithm, we define
  - Given
    - There are n cases (i.e., data points x_j with $j \in \{1, \dots, n\}$) to be clustered into groups. Each data point is a vector of m attribute values. However, the clustering is done in a fuzzy way
    - C clusters to be build by assigning subsets of data points
  - Sought
    - Matrix $W = (w_{i,j})_{1 \le i \le C, 1 \le j \le n}$ indicating the degree of membership by which the jth case (point) belongs to cluster i
    - It holds that $\forall i \in \{1, \dots, C\}: \forall j \in \{1, \dots, n\}: 0 \le w_{i,j} \le 1$ and $w_{i,j} \in \mathbb{R}$

Objective function and restrictions

- The procedure coincides with the basic ideas applied by the k-means clustering procedure while the pursued objective function is modified to

$$J = \sum_{i=1}^{C} \sum_{j=1}^{n} (w_{i,j})^m \cdot d(x_j, \mu_i)$$

- with $m > 1$ defining a predetermined parameter (frequently denoted as the fuzzifier) for weighting the degree of membership
- As the membership of each relation is defined by $w_{i,j}$ these entries of the matrix W have to be updated during each iteration, while the following restrictions have to be obeyed
  1. No cluster is empty, i.e., it holds that $\forall i \in \{1, \dots, C\}: \sum_{j=1}^{n} w_{i,j} > 0$
  2. The membership of each case over all clusters is a distribution, i.e., it holds that $\forall j \in \{1, \dots, n\}: \sum_{i=1}^{C} w_{i,j} = 1$
- Clearly, while restriction 1 deals with the rows of matrix W, the second restriction considers its columns

The procedure

1. Initialize C clusters by defining μ_i for $i \in \{1, \dots, C\}$
2. Compute the current membership degree of each case j according to every cluster i by calculating

$$w_{i,j} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_j - \mu_i\|}{\|x_j - \mu_k\|} \right)^{\frac{2}{m-1}}} = \frac{1}{\|x_j - \mu_i\|^{\frac{2}{m-1}} \cdot \sum_{k=1}^{C} \|x_j - \mu_k\|^{\frac{-2}{m-1}}}$$

3. Update the new barycenters μ_i for $i \in \{1, \dots, C\}$ of the C clusters by computing

$$\mu_i = \sum_{j=1}^{n} \frac{(w_{i,j})^m}{\sum_{j=1}^{n} (w_{i,j})^m} \cdot x_j = \frac{1}{\sum_{j=1}^{n} (w_{i,j})^m} \cdot \sum_{j=1}^{n} (w_{i,j})^m \cdot x_j$$

Checking the first restriction

We compute $\forall i \in \{1, \dots, C\}$:

$$\sum_{j=1}^{n} w_{i,j} = \sum_{j=1}^{n} \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_j - \mu_i\|}{\|x_j - \mu_k\|} \right)^{\frac{2}{m-1}}} = \sum_{j=1}^{n} \frac{1}{\|x_j - \mu_i\|^{\frac{2}{m-1}}} \cdot \sum_{k=1}^{C} \|x_j - \mu_k\|^{\frac{2}{m-1}}$$

$$= \frac{\sum_{k=1}^{C} \|x_j - \mu_k\|^{\frac{2}{m-1}} > 0}{\sum_{j=1}^{n} \|x_j - \mu_i\ \|^{\frac{2}{m-1}} > 0} > 0$$

Thus, the update of the membership values fulfills the first restriction

Checking the second restriction

We compute $\forall j \in \{1, \dots, n\}$:

$$\sum_{i=1}^{C} w_{i,j} = \sum_{i=1}^{C} \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_j - \mu_i\|}{\|x_j - \mu_k\|} \right)^{\frac{2}{m-1}}} = \sum_{i=1}^{C} \frac{1}{\|x_j - \mu_i\|^{\frac{2}{m-1}}} \cdot \sum_{k=1}^{C} \|x_j - \mu_k\|^{\frac{2}{m-1}}$$

$$= \frac{\sum_{k=1}^{C} \|x_j - \mu_k\|^{\frac{2}{m-1}}}{\sum_{i=1}^{C} \|x_j - \mu_i\ \|^{\frac{2}{m-1}}} = 1$$

Thus, the update of the membership values fulfills the second restriction

Updating the membership values - simple example

- Let us assume that for $m = 2$ and three clusters (i.e., $C = 3$), we consider case 1 that possesses a small distance (let say ϵ) to cluster 1 and a much larger distance 1 to the other two clusters 2 and 3
- Then, the update results to $w_{1,1} = \frac{1}{\left(\frac{\epsilon}{\epsilon}\right)^2 + \left(\frac{\epsilon}{1}\right)^2 + \left(\frac{\epsilon}{1}\right)^2}$
- Due to the fact that ϵ is very small in comparison to 1, we obtain a value for $w_{1,1}$ close to one, i.e., indicating that the assignment of case 1 to cluster 1 is quite sure
- For instance, if we set $\epsilon = 0.01$ we obtain $w_{1,1} = \frac{1}{(1)^2 + 0.0002} = \frac{1}{1.0002} = \frac{10{,}000}{10{,}002}$
- The other degrees of membership of the first case (data point) are as follows

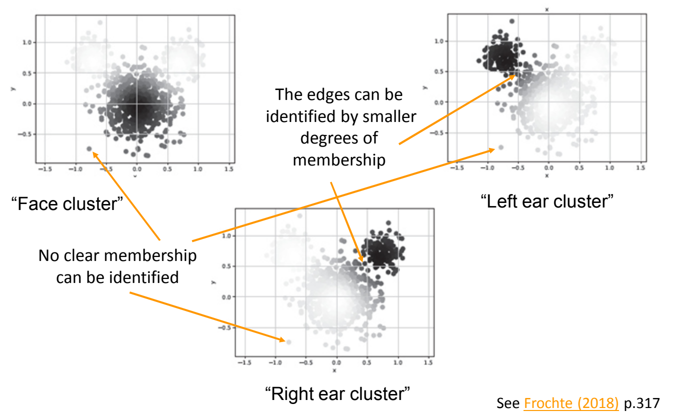
  $w_{2,1} = w_{3,1} = \frac{1}{\left(\frac{1}{\epsilon}\right)^2 + \left(\frac{1}{1}\right)^2 + \left(\frac{1}{1}\right)^2} = \frac{1}{\left(\frac{1}{0.01}\right)^2 + 1 + 1} = \frac{1}{10{,}002}$
- It holds that $\frac{10{,}000}{10{,}002} + 2 \cdot \frac{1}{10{,}002} = \frac{10{,}000+2}{10{,}002} = \frac{10{,}002}{10{,}002} = 1$

Updating the membership values - simple example

- However, if we have an equal distance of, let say ½, to all three clusters, we obtain the following degrees of membership that are not surprising

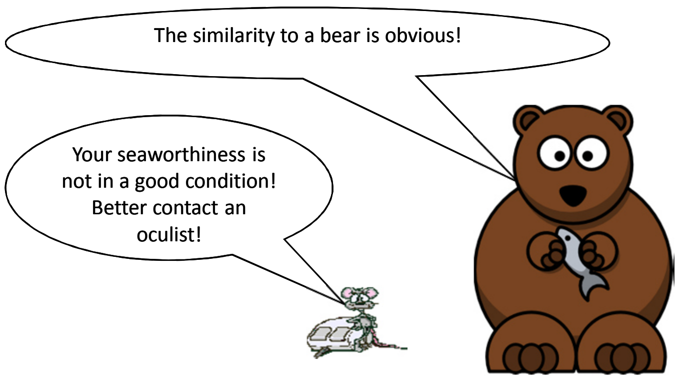
$$w_{1,1} = w_{2,1} = w_{3,1} = \frac{1}{\left(\frac{1/2}{1/2}\right)^2 + \left(\frac{1/2}{1/2}\right)^2 + \left(\frac{1/2}{1/2}\right)^2} = \frac{1}{3}$$

The probabilities obtained for the mouse data set



See Frochte (2018) p.317

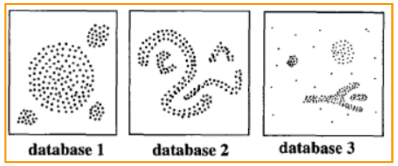
Validation of the approach



2.7.5 Density-Based Spatial Clustering (with noise)

- In what follows, we consider the Density-Based Algorithm for Discovering Clusters (DBSCAN)
- It was originally proposed by Ester, Kriegel, Sander, and Xu (1996)
- This basic approach was extended by other studies
- In what follows, the basic approach proposed by Ester, Kriegel, Sander, and Xu (1996) is introduced
- The procedure addresses the knowledge discovery in spatial databases (KDD)

Motivation of the approach



- By briefly analyzing the sample sets of points depicted above, we can easily detect clusters of points and noise points not belonging to any of those clusters
- The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster
- Furthermore, the density within the areas of noise is lower than the density in any of the clusters

Requirements (see Ester, Kriegel, Sander, and Xu (1996) p.226)

Clustering algorithms are attractive for the task of class identification. However, the application to large spatial databases rises the following requirements for clustering algorithms (see Ester, Kriegel, Sander, and Xu (1996) p.226)

1. Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases
2. Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
3. Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects

ϵ-neighborhood of a point

2.7.5.1 **Definition**: (ϵ-neighborhood of a point)

The ϵ-*neighborhood* of a point p, denoted by $N_\epsilon(p)$ is a subset of the data set D that is defined by $N_\epsilon(p) = \{q \in D \mid dist(p,q) < \epsilon\}$

One may think that it is enough to require for each point in a cluster that there are at least P^{min} points in an ϵ-neighborhood of that point

This is too naive since there are two kinds of points in a cluster, namely, points inside of the cluster (core points) and points on the border of the cluster (border points).

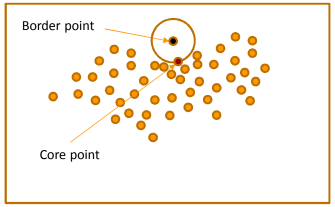
Direct density reachability

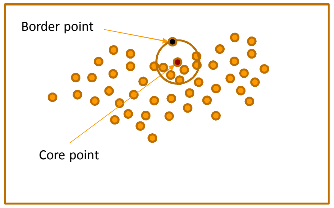
2.7.5.2 **Definition**: (directly density-reachable)

A point p is **directly density-reachable** from a point q with respect to ϵ and P^{min} if the following two criteria are fulfilled

1. $p \in N_\epsilon(q)$ and
2. $|N_\epsilon(q)| \geq P^{min}$

Border points and core points





- Obviously, the criterion “directly density-reachable” is not symmetric if both core and border points are considered (this obviously results from the second criterion not fulfilled by border points)
- But, it is symmetric for two core points
- See Ester, Kriegel, Sander, and Xu (1996) p.228

Density reachable

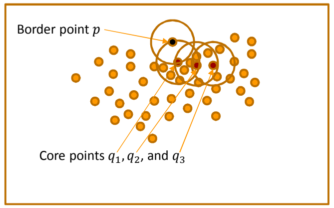
2.7.5.3 **Definition**: (density-reachable)

A point p is ***density reachable*** from a point q with respect to ϵ and P^{min} if there is a chain of points $q = p_1, p_2, \ldots, p_{n-1}, p_n = p$ such that p_{i+1} is directly density-reachable from p_i for all $i \in \{1, \ldots, n-1\}$.

Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric.

See Ester, Kriegel, Sander, and Xu (1996) p.228

Density reachable – Non-symmetric relation



- p is density-reachable from q_3 as q_2 is directly density-reachable from q_3, q_1 is directly density-reachable from q_2, and p is directly density-reachable from q_1
- But, q_3 is NOT density-reachable from p as a border point does not fulfill the second criterion of Definition 2.7.5.2

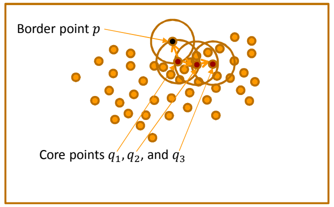
Density connected

2.7.5.4 **Definition**: (density-connected)

A point p is density connected to a point q with respect to ϵ and P^{min} if there is a point o such that both, p and q are density-reachable from o with respect to ϵ and P^{min}.

Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive, i.e., it holds that each point p is density connected to p as p is density-reachable from p

Density connectivity – Symmetric relation



- p is density connected to q_3 as p is directly density-reachable from q_1 and q_3 is directly density-reachable from q_1
- q_3 is density connected to p as p is directly density-reachable from q_1 and q_3 is directly density-reachable from q_1

Definition of a density cluster

2.7.5.5 **Definition**: (cluster)

Let D be a database of points. A *cluster* C with respect to ϵ and P^{min} is a non-empty subset of D, i.e., $C \subseteq D$ such that the following two restrictions are fulfilled:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p with respect to ϵ and P^{min} then $q \in C$ (Maximality restriction)
2. $\forall p, q \in C$: p is density-connected to q with respect to ϵ and P^{min} (Connectivity restriction)

See Ester, Kriegel, Sander, and Xu (1996) p.228

Definition of noise

2.7.5.6 **Definition**: (noise)

Let $C_1, \ldots, C_k$ be the clusters of the database D with respect to ϵ and P^{min}, $i = 1, \ldots, k$.

Then, we define the *noise* as the set of points in the database D not belonging to any cluster $C_1, \ldots, C_k$

Thus, it holds that

$noise = \{p \in D \mid p \notin C_1 \cup C_2 \cup \cdots \cup C_k\}$

Observation

- Each cluster C with respect to ϵ and P^{min} contains at least P^{min} points due to the following facts
  - $C \subseteq D$ is a non-empty subset of the database
  - Thus, there is a point $p \in C$
  - Thus, p is at least density connected to itself by a point $o \in C$ (note that this covers the case $o = p$)
  - But, then node o fulfills the second criterion of Definition 2.7.5.2 and we obtain $|N_\epsilon(o)| \geq P^{min}$ with $o \in C$
  - Then, there are $|N_\epsilon(o)|$ points that are directly density reachable from $o \in C$
  - Thus, all these nodes also belong to C and we conclude that $|C| \geq P^{min}$

Conclusions

- The following Lemmata are important for validating the correctness of the clustering algorithm
- Intuitively, they state the following
  - Given the parameters ϵ and P^{min}, we can discover a cluster in a two-step approach
  - First, choose an arbitrary point from the database satisfying the core point condition as a seed
  - Second, retrieve all points that are density-reachable from the seed obtaining the cluster containing the seed

Conclusions

2.7.5.7 **Lemma**

Let p be a point in D and $|N_\epsilon(p)| \geq P^{min}$. Then, the set

$$O = \left\{ \begin{matrix} o \mid o \in D \wedge o \text{ is density} - \text{reachable from } p \\ \text{with respect to } \epsilon \text{ and } P^{min} \end{matrix} \right\}$$

is a cluster with respect to ϵ and P^{min}. It is not obvious that a cluster C with respect to ϵ and P^{min} is uniquely determined by *any* of its core points. However, each point in C is density-reachable from any of the core points of C and, therefore, a cluster C contains exactly the points which are density-reachable from an arbitrary core point of C.

Conclusions

2.7.5.8 **Lemma**

Let C be a cluster with respect to ϵ and P^{min} and let p be any point in C fulfilling $|N_\epsilon(p)| \geq P^{min}$.

Then, C equals to the set

$$O = \left\{ \begin{matrix} o \mid o \text{ is density} - \text{reachable from } p \\ \text{with respect to } \epsilon \text{ and } P^{min} \end{matrix} \right\}$$

Parameters

- D Cases (i.e., data points) in
- P^{min} Minimum size of a cluster (to be predetermined by the user)
- ϵ Minimum distance between two clusters (to be predetermined by the user)

Algorithm – DBSCAN(D, ϵ, P^{min})

1. $C^{\text{id}} = 0$
2. FOR all unvisited $x \in D$ DO
    1. Label x as a visited node
    2. Set $N = \{x' \in D \mid \|x' - x\| < \epsilon\}$
    3. IF $|N| < P^{min}$

       THEN label x as a noise node

       ELSE $C^{\text{id}} = C^{\text{id}} + 1;$

       Add node x to cluster with id C^{id}

       ExpandCluster$(N, C^{\text{id}}, \epsilon, P^{min})$

       END IF
3. END FOR

END OF FUNCTION

Procedure ExpandCluster$(N, C^{id}, \epsilon, P^{min})$

FOR all $y \in N$ DO

IF y is not visited (labeled)

THEN label y as visited

$N^y := \{z \in D \mid \|z - y\| < \epsilon\}$

IF $|N^y| \geq P^{min}$

THEN $N := N \cup N^y$

END IF

IF y is not assigned to a cluster

THEN Assign y to cluster C^{id}

Overwrite a potential noise label of y

END IF

END IF

END FOR

Derivation of the parameters – 1

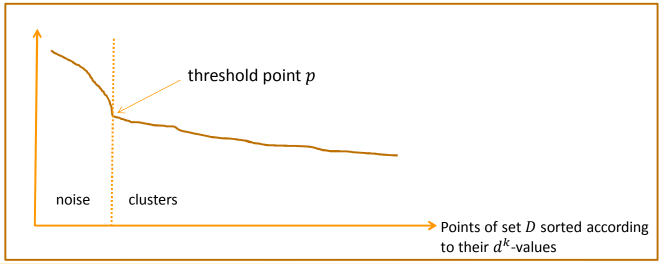
- Ester, Kriegel, Sander, and Xu (1996) propose the following procedure to derive the needed values for the applied parameters ϵ and P^{min}
- This procedure considers the "thinnest" cluster in the considered database
- For this, purpose,
    - let d be the distance of a point p to its kth nearest neighbor
    - Thus, the d-neighborhood of point p comprises at least $k+1$ nodes altogether
    - But, note that it is quite unlikely that the d-neighborhood of point p comprises more than $k+1$ nodes as this is only possible if there are several nodes with equal distance to p
    - Furthermore, we can state that changing the parameter k for a node in a cluster frequently does not result in large changes of the resulting d-value

Derivation of the parameters – 2

- The latter results from the fact that clusters possesses a significant density and such a significant change of d would imply that all points are located more or less on a straight line which is in general not true for a cluster
- For a given k, a function d^k is introduced with $d^k: D \to \mathbb{R}$
- This function defines for each node $p \in D$ the distance of the kth nearest neighbor
- By sorting all nodes p in set D in sequence of non-increasing d^k-values, we can define a graph of these values starting with the one that possesses the largest d^k-value
- Note that this graph (sorted d^k graph) may give us some hint concerning the density distribution in the considered data set
- Specifically, if we choose a node p, set the parameter ϵ to $d^k(p)$ and P^{min} to k, all nodes q with $d^k(q) \leq d^k(p)$ are core points
- Ester, Kriegel, Sander, and Xu (1996) propose to find a threshold node p possessing the maximal d^k value in the thinnest cluster of set D

Derivation of the parameters – 3

- This threshold point determines the first “valley” of the sorted d^k graph
- This is illustrated by the figure given below
- All nodes on the left of the threshold point p (i.e., nodes with larger d^k value) are considered as noise
- All other nodes are considered as cluster nodes



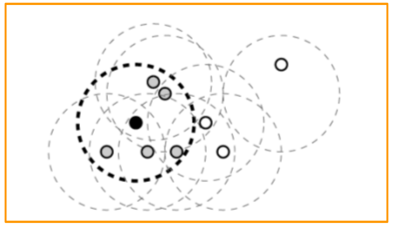
Derivation of the parameters – 4

- Ester, Kriegel, Sander, and Xu (1996) state that in general it is very difficult to detect the first "valley" automatically, but it is relatively simple for a user to see this valley in a graphical representation
- Therefore, Ester, Kriegel, Sander, and Xu (1996) propose to follow an interactive approach for determining the threshold point
- DBSCAN needs two parameters, ϵ and P^{min}.
- However, Ester, Kriegel, Sander, and Xu (1996) state that their conducted experiments indicate that the sorted d^k graphs for $k > 4$ do not significantly differ from the sorted d^4 graphs and, furthermore, they need considerably more computation.
- Therefore, Ester, Kriegel, Sander, and Xu (1996) propose to set $P^{min} = 4$ thus, eliminating this parameter for all databases (for the two-dimensional data).

Derivation of the parameters – 5

- Ester, Kriegel, Sander, and Xu (1996) propose the following interactive approach for determining the remaining parameter ϵ of DBSCAN
  - The system computes and displays the sorted d^4 graphs for the database
  - If the user can estimate the percentage of noise, this percentage is entered and the system derives a proposal for the threshold point from it
  - The user either accepts the proposed threshold or selects another point as the threshold point. The sorted d^4 graphs value of the threshold point is used as the ϵ-value for DBSCAN

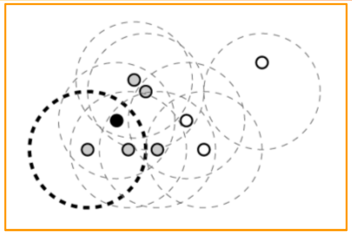
Illustration of the DBSCAN computation - 1



See Frochte (2018) p.318

- Start with the black point as its neighborhood is large enough
- Now, we have to check for each point of this neighborhood whether these points are also corner points
- This is done by calling the procedure ExpandCluster

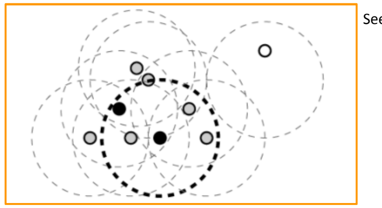
Illustration of the DBSCAN computation - 2



See Frochte (2018) p.318

- As the neighborhood is too tittle, the two nodes within the dotted circle are not labeled as new core points
- However, this does not apply to the next node (see the next slide)

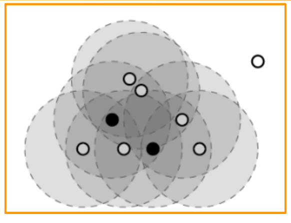
Illustration of the DBSCAN computation - 3



See Frochte (2018) p.318

- A second core node is identified
- Therefore, its neighborhood is scanned by a second call of the procedure ExpandCluster
- All new nodes of this neighborhood are inserted into the cluster
- Hence, one node remains and is labeled as noise

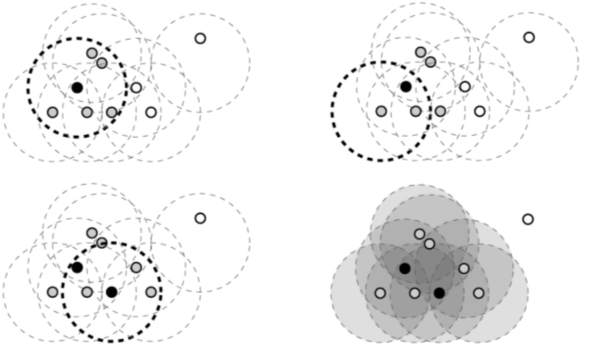
Illustration of the DBSCAN computation - 4



See Frochte (2018) p.318

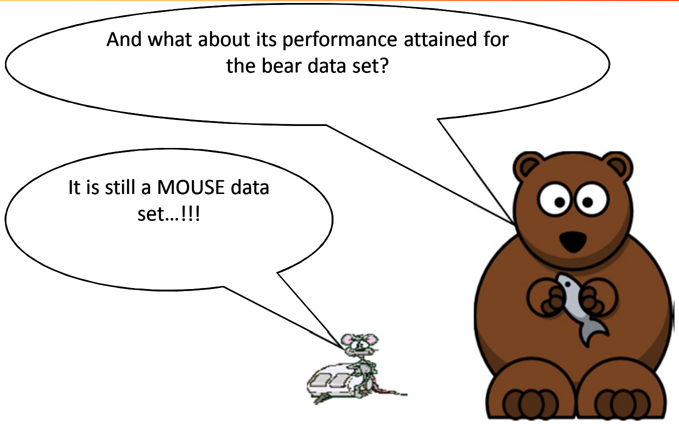
- The cluster is completely generated in this small example
- One node is classified as noise
- All other nodes belong to the cluster

Overview

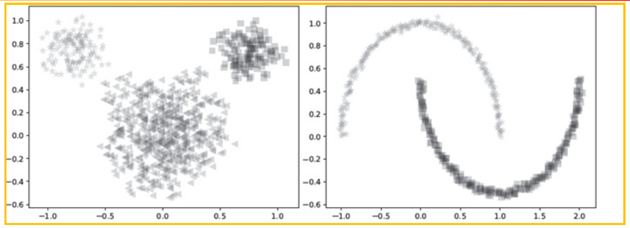


See Frochte (2018) p.318

Validation of the approach



Applied to the illustrative data sets

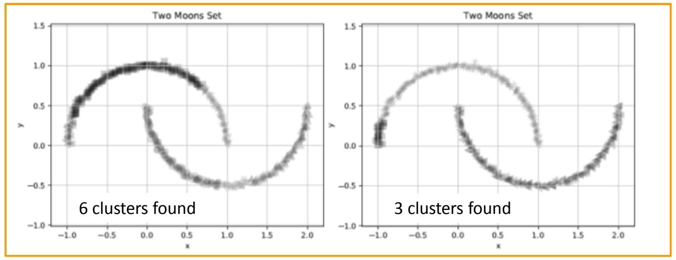


See Frochte (2018) p.323

- Mouse data set:
  - Applied setting is $\epsilon = 0.4$ and $P^{min} = 10$. Then, noise was about 3.8 percent
  - However, if ϵ is increased while keeping $P^{min} = 10$, the left ear is integrated into the face cluster
- Moon data set:

  Applied setting is $\epsilon = 0.08$ and $P^{min} = 5$. Then, no noise was observed

Two moons data set – Further results



Results for $\epsilon = 0.06$ (left) and $\epsilon = 0.07$ (right), see Frochte (2018) p.323

- It can be observed that the obtained results are quite sensitive concerning the predetermined parameter values
- In case of $\epsilon = 0.07$ the upper moon is divided into two clusters
- This separation is continued if ϵ is further decreased

Two moon data sets – detailed results

ϵ	Number of clusters	Noise
0.02	5	95 %
0.03	24	66 %
0.04	38	16 %
0.05	11	1.4 %
0.06	6	0 %
0.07	3	0 %
0.08	2	0 %

- This table further underscores the sensitivity of the attained results form choosing suitable parameter values
- The expected (human eye corresponding) result with two distinctive clusters is attained by setting $\epsilon = 0.08$

2.7.6 Hierarchical Clustering

- In what follows, we consider a different technique applied for clustering
- It generates clusters hierarchically, i.e., it decides about a combination of groups of cases (points) by measuring the similarity of their members
- Hierarchical clustering can be found, for instance, in biology where animal species are hierarchically grouped according to characterizing attributes
  - Group of animals
    - Invertebrates
    - Fish
    - Amphibians
    - Reptiles
    - Birds
    - Mammals
  - Clearly, all these subgroups are further separated into smaller groups
- Hence, in order to apply such a clustering, distance measures have to be extended to groups of cases (instead of comparing just single cases)

Extending distance measures to groups

In literature, there a various proposals for such a necessary extension. Among them, the following approaches are frequently applied in hierarchical clustering approaches:

- Single Linkage
- Complete Linkage
- Average Linkage
- Centroid method

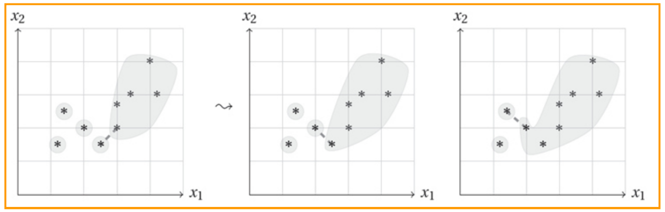
In what follows, we introduce and consider the four alternative measures more in detail

Single Linkage

- This approach measures the distance between two groups (clusters) by the smallest distance between two members
- Therefore, the highest similarity of two members is identified and applied
- Specifically, single linkage uses the minimal distance between two nodes of different clusters in question, i.e., by considering the clusters C_1 and C_2, we obtain

$$\boxed{D^{sl}(C_1, C_2) = \min\{d(a,b) \mid a \in C_1 \land b \in C_2\}}$$

Problem of single linkage – chain building



See Frochte (2018) p.325

- Due to the orientation towards the mutually closest located nodes, it can be frequently observed that the clusters are not really of compact shape but may degenerate to chains
- This is depicted by the above figures that are taken from the textbook of Frochte (2018)
- Possible extensions would be to consider for a suitably chosen parameter $k \in \mathbb{N}$, the kth nearest tuple or the average distance of the k nearest tuples

Complete Linkage

- In contrast to the single linkage approaches, complete linkage methods measure the distance between two groups of nodes by identifying the tuple of nodes with maximum distance
- Thus, similarity is determined by the distance between the two most dissimilar elements of the respective groups
- Specifically, complete linkage defines the distance between the two clusters in question, i.e., between C_1 and C_2, by calculating the maximum distance of two nodes in these clusters, i.e.,

$$D^{cl}(C_1, C_2) = max\{d(a,b) \mid a \in C_1 \wedge b \in C_2\}$$

Average Linkage

- A straightforward compromise between single and complete linkage is the so-called average linkage that determines the distance between two groups of nodes by the average distance between all occurring pairs of nodes
- Specifically, the distance measure of average linkage between two clusters, i.e., between the clusters C_1 and C_2, is defined through

$$D^{al}(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \cdot \sum_{a \in C_1} \sum_{b \in C_2} d(a,b)$$

Centroid method

- This method starts the distance measuring process by identifying the centroid of the two clusters, let say C_1 and C_2, by generating the mean values of each stored attribute
- Subsequently, the distance between these two centroids is taken as the distance value of the two clusters
- By assuming that each node a in the two clusters is defined by a vector of altogether m attributes, it can be unambiguously defined through

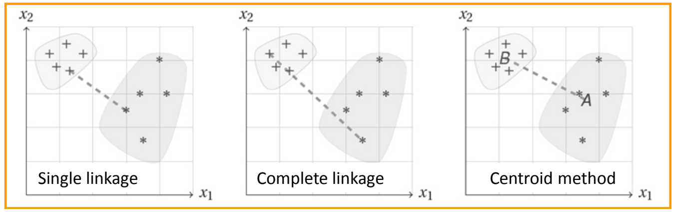
$$a = (a_1, a_2, \dots, a_m)$$

- Hence we have

$$D^{cm}(C_1, C_2) = d\big(c(C_1), c(C_2)\big) \text{ with}$$

$$c(C) = (c_1, c_2, \dots, c_m) \text{ where } \forall i \in \{1, \dots, m\}: c_i = \frac{1}{|C|} \cdot \sum_{a \in C} a_i$$

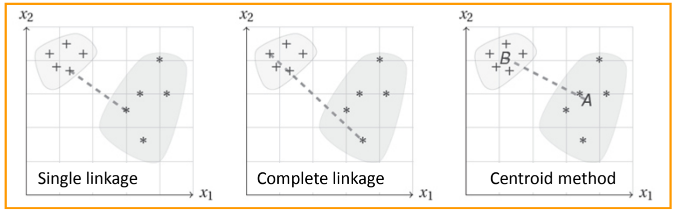
Illustration



See Frochte (2018) p.325

- The figures given above try to illustrate the three distance measure computations single linkage, complete linkage, and the centroid method
- Due to the large number of possible tuples, an illustration of the average linkage approach is not provided

Computational aspects



See Frochte (2018) p.325

- As all pairs of nodes have to be considered, a trivial approach would lead to an asymptotical effort of $\mathcal{O}(n^2 \cdot n) = \mathcal{O}(n^3)$
- However, by using specific settings and applying some specifically designed data structures, special cases can by handled in quadratic time for single linkage (see Sibson (1973)) and complete linkage (see Defays (1977))

Computational aspects – Updating

- Due to the significant computational effort, it is useful to reuse existing (i.e., already computed) distance values in later iterations
- For this purpose, the well-known update formula of Lance and Williams can be applied
- Situation
  - Two clusters, namely C_1 and C_2 were united during the last iteration
  - Then, depending on the applied approach (single linkage or complete linkage), the distances between the new cluster and all other clusters can be updated by using the current distance values of the preceding iteration
  - This is done by using the following updating formula:

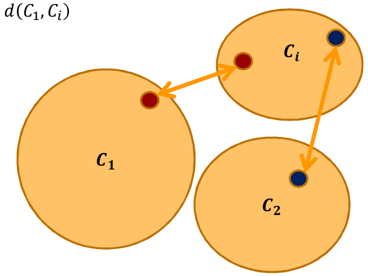
$$D(C_1 \cup C_2, C_i) =$$
$$\alpha_1 \cdot d(C_1, C_i) + \alpha_2 \cdot d(C_2, C_i) + \beta \cdot d(C_1, C_2) + \gamma \cdot |d(C_1, C_i) - d(C_2, C_i)|$$

Values for the formula of Lance and Williams

Method	α_1	α_2	β	γ
Single linkage	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete linkage	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Average	$\frac{\lvert C_1\rvert}{\lvert C_1\rvert+\lvert C_2\rvert}$	$\frac{\lvert C_2\rvert}{\lvert C_1\rvert+\lvert C_2\rvert}$	0	0
Centroid (Euclidean distance)	$\frac{\lvert C_1\rvert}{\lvert C_1\rvert+\lvert C_2\rvert}$	$\frac{\lvert C_2\rvert}{\lvert C_1\rvert+\lvert C_2\rvert}$	$\frac{\lvert C_1\rvert \cdot \lvert C_2\rvert}{(\lvert C_1\rvert+\lvert C_2\rvert)^2}$	0

Single linkage

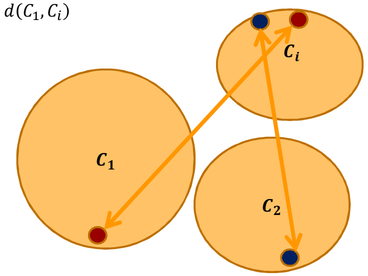
- $d(C_1 \cup C_2, C_i) = 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) - 0.5 \cdot |d(C_1, C_i) - d(C_2, C_i)|$
- Clearly, it either holds that $d(C_1 \cup C_2, C_i) = d(C_1, C_i)$ or $d(C_1 \cup C_2, C_i) = d(C_2, C_i)$
- We assume (without limiting the generality of the foregoing) $d(C_1 \cup C_2, C_i) = d(C_1, C_i)$
- Hence, we obtain $0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) - 0.5 \cdot |d(C_1, C_i) - d(C_2, C_i)|$
  $= 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) - 0.5 \cdot \left(d(C_2, C_i) - d(C_1, C_i)\right)$
  $= 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_1, C_i)$
  $= d(C_1, C_i)$



Complete linkage

- $d(C_1 \cup C_2, C_i) = 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) + 0.5 \cdot |d(C_1, C_i) - d(C_2, C_i)|$
- Clearly, it either holds that $d(C_1 \cup C_2, C_i) = d(C_1, C_i)$ or $d(C_1 \cup C_2, C_i) = d(C_2, C_i)$
- We assume (without limiting the generality of the foregoing) $d(C_1 \cup C_2, C_i) = d(C_1, C_i)$
- Hence, we obtain $0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) + 0.5 \cdot |d(C_1, C_i) - d(C_2, C_i)|$

$$= 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_2, C_i) + 0.5 \cdot \big(d(C_1, C_i) - d(C_2, C_i)\big)$$
$$= 0.5 \cdot d(C_1, C_i) + 0.5 \cdot d(C_1, C_i)$$
$$= d(C_1, C_i)$$



Average linkage

- $d(C_1 \cup C_2, C_i) = \frac{|C_1|}{|C_1|+|C_2|} \cdot d(C_1, C_i) + \frac{|C_2|}{|C_1|+|C_2|} \cdot d(C_2, C_i)$
- Clearly, it holds that

$$d(C_1 \cup C_2, C_i)$$
$$= d(C_1, C_i) \cdot \frac{|C_1| \cdot |C_i|}{(|C_1|+|C_2|) \cdot |C_i|} + d(C_2, C_i) \cdot \frac{|C_2| \cdot |C_i|}{(|C_1|+|C_2|) \cdot |C_i|}$$
$$= d(C_1, C_i) \cdot \frac{|C_1|}{(|C_1|+|C_2|)} + d(C_2, C_i) \cdot \frac{|C_2|}{(|C_1|+|C_2|)}$$
$$= \frac{|C_1|}{|C_1|+|C_2|} \cdot d(C_1, C_i) + \frac{|C_2|}{|C_1|+|C_2|} \cdot d(C_2, C_i)$$

- Hence, this is just the formula defined above

Centroid method with Euclidean distance

- It holds that $d(C_1 \cup C_2, C_i)$

$$= \frac{|C_1|}{|C_1|+|C_2|} \cdot d(C_1, C_i) + \frac{|C_2|}{|C_1|+|C_2|} \cdot d(C_2, C_i) - \frac{|C_1| \cdot |C_2|}{(|C_1|+|C_2|)^2} \cdot d(C_1, C_2)$$

- The new centroid of $C_1 \cup C_2$ will be at

$$c(C_1 \cup C_2) = \frac{|C_1| \cdot c(C_1)}{|C_1|+|C_2|} + \frac{|C_2| \cdot c(C_2)}{|C_1|+|C_2|} = \frac{|C_1| \cdot c(C_1) + |C_2| \cdot c(C_2)}{|C_1|+|C_2|}$$

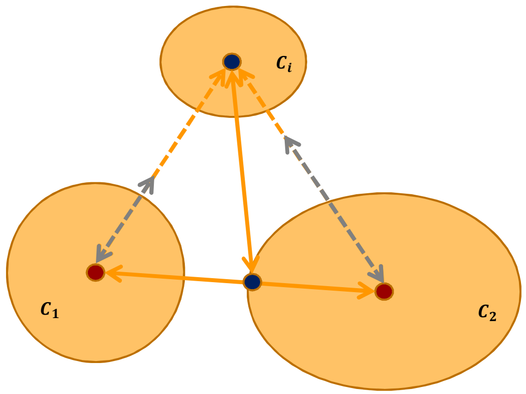
- Thus, we obtain the Euclidean distance between $C_1 \cup C_2$ and C_i as

$$d(C_1 \cup C_2, C_i) = \left(c(C_i) - \frac{|C_1| \cdot c(C_1) + |C_2| \cdot c(C_2)}{|C_1|+|C_2|}\right)^2$$

And by multiplying up and rearranging, we finally obtain

$$= \frac{|C_1|}{|C_1|+|C_2|} \cdot \big(c(C_i) - c(C_1)\big)^2 + \frac{|C_2|}{|C_1|+|C_2|} \cdot \big(c(C_i) - c(C_2)\big)^2$$
$$- \frac{|C_1|}{|C_1|+|C_2|} \cdot \frac{|C_2|}{|C_1|+|C_2|} \cdot \big(c(C_1) - c(C_2)\big)^2$$
$$= \frac{|C_1|}{|C_1|+|C_2|} \cdot d(C_1, C_i) + \frac{|C_2|}{|C_1|+|C_2|} \cdot d(C_2, C_i) - \frac{|C_1| \cdot |C_2|}{(|C_1|+|C_2|)^2} \cdot d(C_1, C_2)$$

Centroid method – illustration of the updating



Agglomerative or divisive clustering approaches

- Agglomerative clustering approaches
  - It is a bottom up approach that starts with a setting where each case (node) constitutes an individual cluster
  - The two closest located clusters are united to one cluster as long as a predetermined criterion is not met
  - Such a predefined criterion may be
    - a minimum number of required clusters
    - a maximum distance value (exceeded also by the closest located clusters)
    - a maximum number of allowed conducted cluster unifications
    - …

Agglomerative or divisive clustering approaches

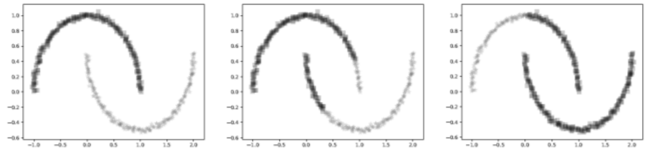
- Divisive clustering approaches
  - These approaches are just the opposite of agglomerative procedures
  - Specifically, divisive approaches start with one cluster comprising all cases (nodes) of the considered data set
  - At each stage of the algorithm, one cluster is divided into two new clusters
  - For this purpose, divisive clustering approaches require a sophisticated method to efficiently identify a suitable cluster and the respective subsets of cases in this cluster
  - Note that an exhaustive enumeration of all possible separations for n clusters possessing approximately m elements, we have a prohibitive effort of

    $O\big(n \cdot (2^m - 1)\big)$ possibilities to be considered
  - Clearly, for real-world applications, this requires the application of sophisticated approaches reducing the computational effort

Divisive clustering approaches

- Despite its computational burdens, bisecting divisive clustering approaches are quite attractive in many applications as (see Savaresi et al. (2002))
  - by recursively using a bisecting divisive clustering procedure, the data-set can be partitioned into any given number of clusters.
  - Interestingly enough, the so-obtained clusters are structured as a hierarchical binary tree (or a binary taxonomy)
- As a consequence, specific approaches are proposed that pursue the finding of an efficient separation of given clusters

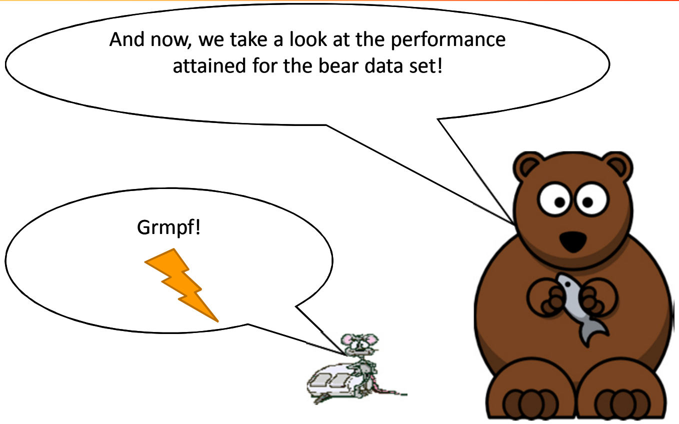
Validation – Two moons set



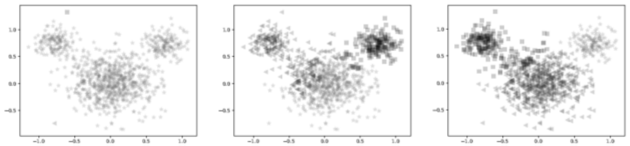
See Frochte (2018) p.330

- From left to right, the results attained by applying single linkage, complete linkage, and average linkage for 2 clusters
- Clearly, single linkage directly follows the shape of the two moons and provides the expected two moons
- In contrast to this, the approaches complete and average linkage separate at least one of the two moons as the distance measure considers the situation in a more global perspective

Validation of the approach



Validation – Mouse data set



- Again, from left to right, the results attained by applying single linkage, complete linkage, and average linkage for 3 clusters
- The single linkage result produces almost one chain and therefore mainly one single cluster. In contrast to the two moon example, in case of the mouse data set, this is not reasonable
- Best results are attained by applying the average linkage approach
- But, this result does not attain the quality reached by applying the DBSCAN approach