Business Analytics & Machine Learning

Ensemble Methods and Clustering

Prof. Dr. Martin Bichler

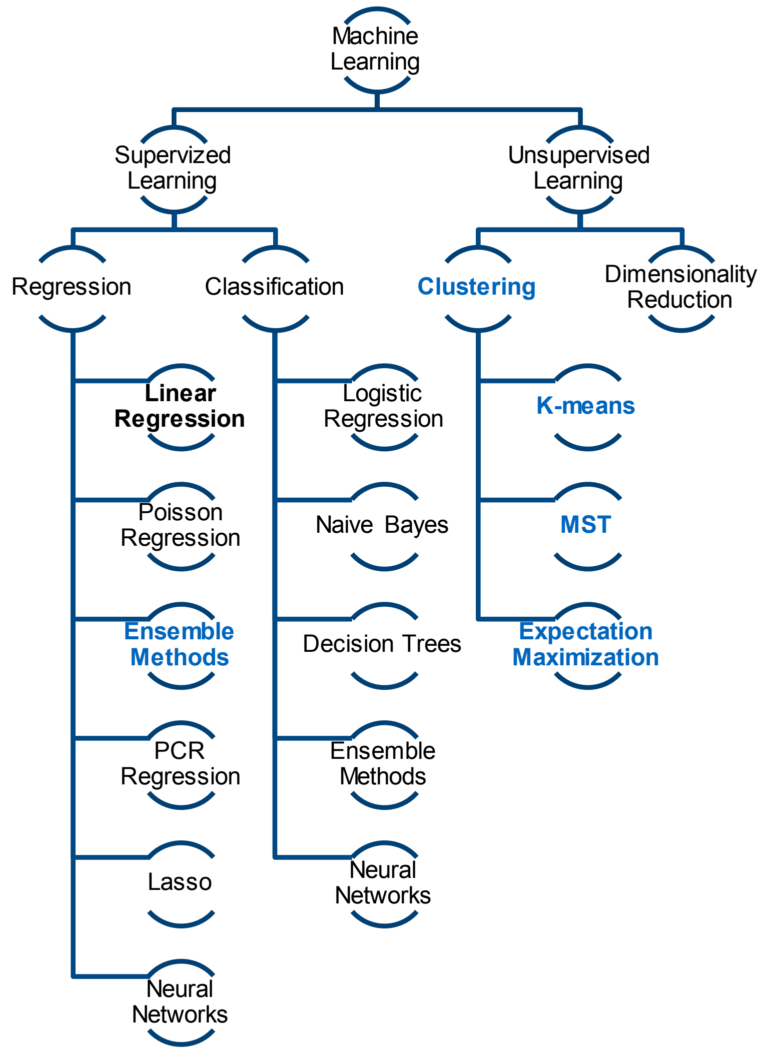
Department of Computer Science

School of Computation, Information, and Technology

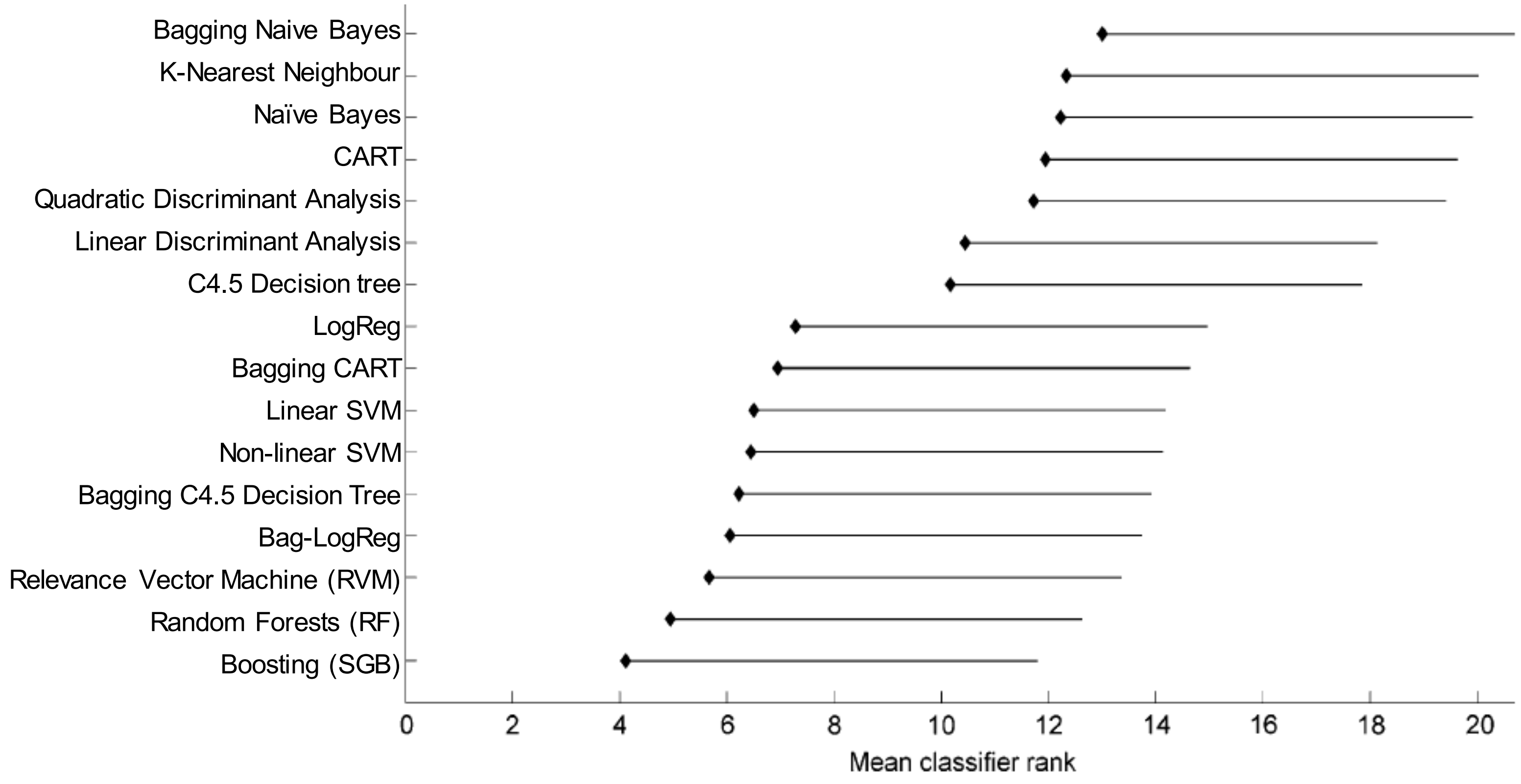
Technical University of Munich

Course Content

- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- **Ensemble Methods and Clustering**
- Dimensionality Reduction
- Association Rules and Recommenders
- Convex Optimization
- Neural Networks
- Reinforcement Learning



An Empirical Study of Classifier Performance



Stefan Lessmann and Stefan Voß. "Customer-centric decision support“. Business & Information Systems Engineering 2.2 (2010): 79-93.

Ensembles: Combining Multiple Models

Basic idea: build different “experts”, let them vote

Advantage:

- often improves predictive performance

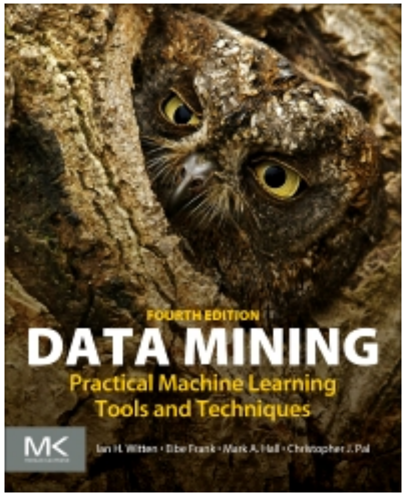
Disadvantage:

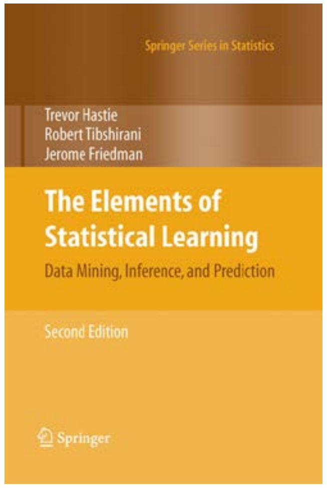
- usually produces output that is very hard to analyze
- but: there are approaches that aim to produce a single comprehensible structure

Recommended Literature

- **Data Mining: Practical Machine Learning Tools and Techniques**
  - Ian H. Witten, Eibe Frank, Mark A. Hall
  - http://www.cs.waikato.ac.nz/ml/weka/book.html
  - Section 12

- **The Elements of Statistical Learning**
  - Trevor Hastie, Robert Tibshirani, Jerome Friedman
  - https://web.stanford.edu/~hastie/ElemStatLearn/
  - Section: 8.7, 8.8, 10, 15, 16


FOURTH EDITION
DATA MINING
Practical Machine Learning Tools and Techniques
MK
Ian H. Witten • Eibe Frank • Mark A. Hall • Christopher J. Pal


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

Autline for Today

- **Ensemble Methods**
  - Bagging
  - Random Forrests
  - Boosting
  - Stacking
- Clustering
  - partitional clustering via k-means
  - hierarchical clustering via MST
  - probabilistic clustering via Expectation Maximization (EM)

Bagging

Combining predictions by voting/averaging

- each model receives equal weight

"Idealized" version:

- sample several training sets of **size n** from the population (instead of just having one training set of size n)
- build a classifier for each training set
- combine the classifiers' predictions

If the learning scheme is **unstable** bagging almost always improves the performance

- unstable learner: small change in training data can make big change in model (e.g., decision trees)

More on Bagging

Problem: we only have one training dataset!

Solution: generate new datasets of size n by sampling from the original dataset **with replacement (as in the bootstrap)**.

← pick a random sample from the original dataset at each time.

Even though the datasets are all dependent, bagging often **reduces variance**.

Advantages:

- can be applied to numeric prediction and classification
- can help a lot if the data is noisy
- usually, the more classifiers the better, with diminishing returns
- Bagging can easily be parallelized because ensemble members are created independently

Bagging – **B**oostrap **Agg**regat**ing**

Model generation:

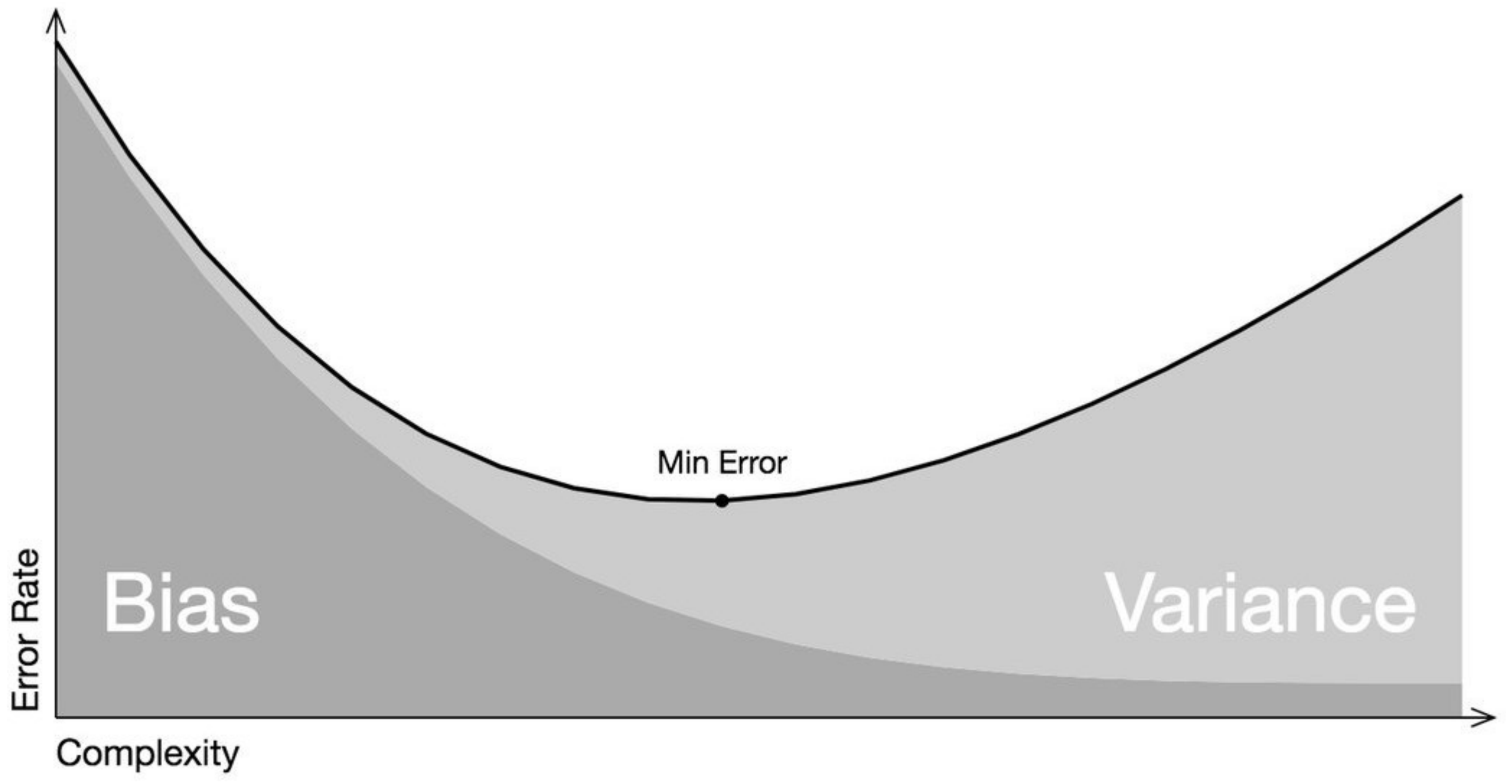
```
Let $n$ be the number of instances in the training data
For each of $t$ iterations:
    Sample $n$ instances from training set
        (with replacement)
    Apply learning algorithm to the sample
    Store resulting model
```

Classification:

```
For each of the $t$ models:
  Predict class of instance using model
Return class that is predicted most often
```

↙ majority vote
(we could change this)

Remember: Bias-Variance Tradeoff



Bias-Variance Decomposition

We can decompose the expected error of any individual ensemble member as follows:

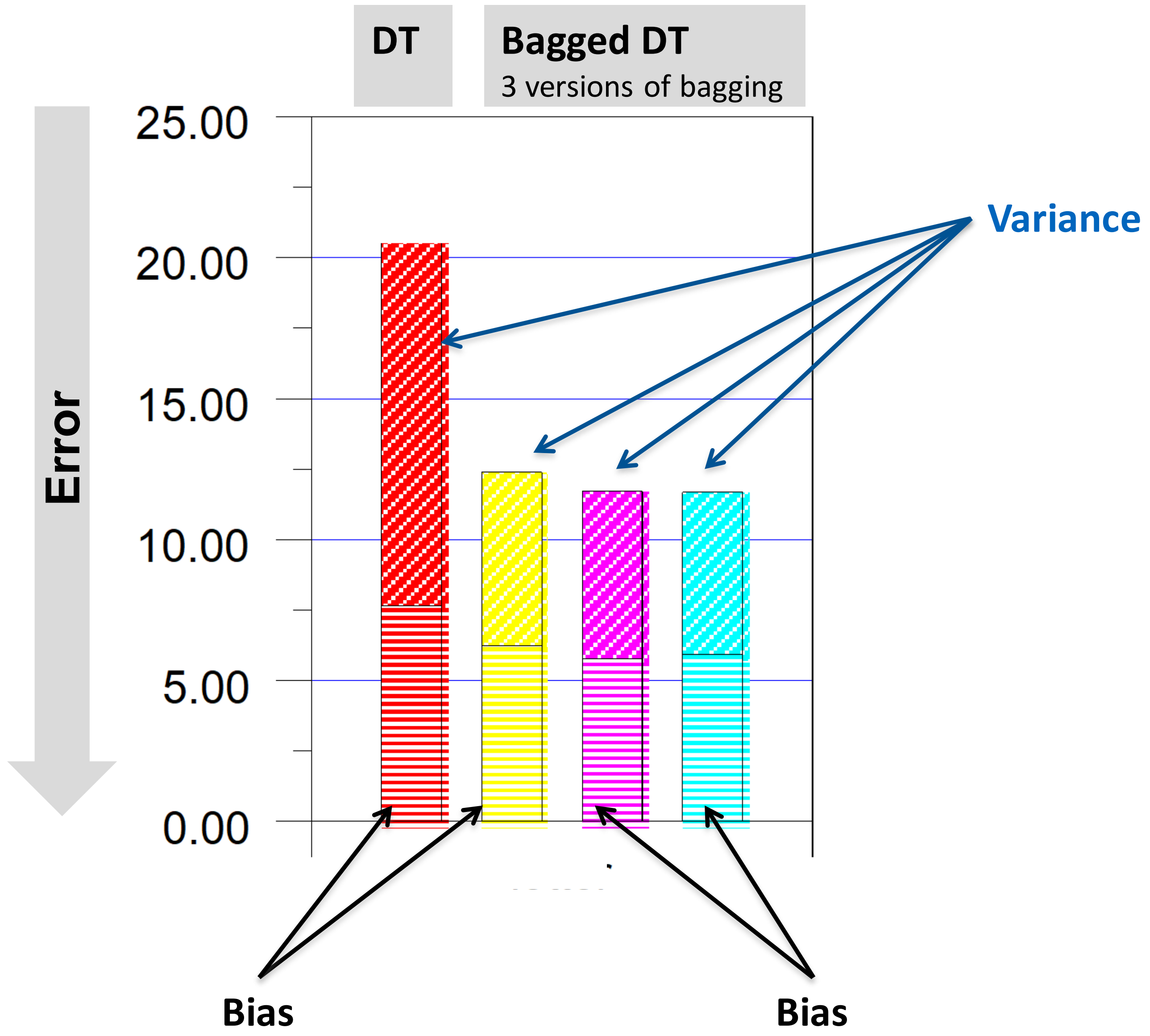
- **Bias** = expected error of the ensemble classifier on new data (high bias = underfitting)
- **Variance** = component of the expected error due to the particular training set being used to build our classifier (high variance = overfitting)
- Total expected error = bias + variance

Combining multiple classifiers generally decreases the expected error by **reducing variance**.

- We assume noise inherent in the data is part of the bias component as it cannot normally be measured.

The **bias-variance decomposition** to understand both components:

- Training error reflects bias but not variance; test error reflects both.
- For example, bootstrap training data sets B from a data set D and evaluate against samples in D-B to estimate bias and variance empirically.



Eric Bauer and Ron Kohavi. “An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants”. Machine Learning 36, 105–139 (1999).

Random Forests

We can also **randomize a learning algorithm** instead of input

- pick m options at random from the full set of options, then choose the best of those m choices
- e.g., randomize the attribute selection in decision trees

Can be combined with bagging

- when using decision trees, this yields the **random forest** method for building ensemble classifiers

Random forests randomize data and features!

- Random decision forests correct for decision trees' habit of overfitting
- Initial proposal by Tin Kam Ho (1995)
- "Random Forests – Random Features" (Leo Breiman, 1997) http://oz.berkeley.edu/~breiman/random-forests.pdf

Random Forest (Breiman, 2001)

Model generation:

```
Select ntree, the number of trees to grow, and mtry, a
   number no larger than number of variables
For i = 1 to ntree:
   Draw a bootstrap sample from the data. Call those
       not in the bootstrap sample the "out-of-bag"
       data
   Grow a "random" tree, where at each node, the best
       split is chosen among mtry randomly selected
       variables. The tree is grown to maximum size
       and not pruned back
   Store the resulting decision tree
```

Classification:

```
For each of the t decision trees:
    Predict class of instance
Return class that was predicted most often
```

Random Forest in Python

```
from sklearn.ensemble import RandomForestClassifier

# Create the forest object
model_random_forest = RandomForestClassifier(max_depth=3, …)

# Train the forest
model_random_forest.fit(X_train, y_train)

# Importance of each predictor
print(model_random_forest.feature_importances_)

# Use the forest for predictions on new data
y_pred = model_random_forest.predict(X_test)
```

Boosting

Boosting combines several weak learners into a strong learner

- Also uses voting/averaging, but weights models according to performance

New models are influenced by performance of previously built ones

- New model to become an “expert” for instances misclassified by earlier models
- Intuitive justification: models should be experts that complement each other

Bias vs. variance

- **Boosting** tries to minimize the bias in terms of training performance of simple learners
- **Random forest** aggregates fully grown trees each of which has low bias but high variance: the random forest reduces the variance of the final predictor by aggregating such trees
- **Bagging** is aimed to reduce variance (error on test data) as well

Many variants of boosting exist

- AdaBoost, XGBoost, GBM/MART, SGB, etc.

AdaBoost.M1

Model generation:

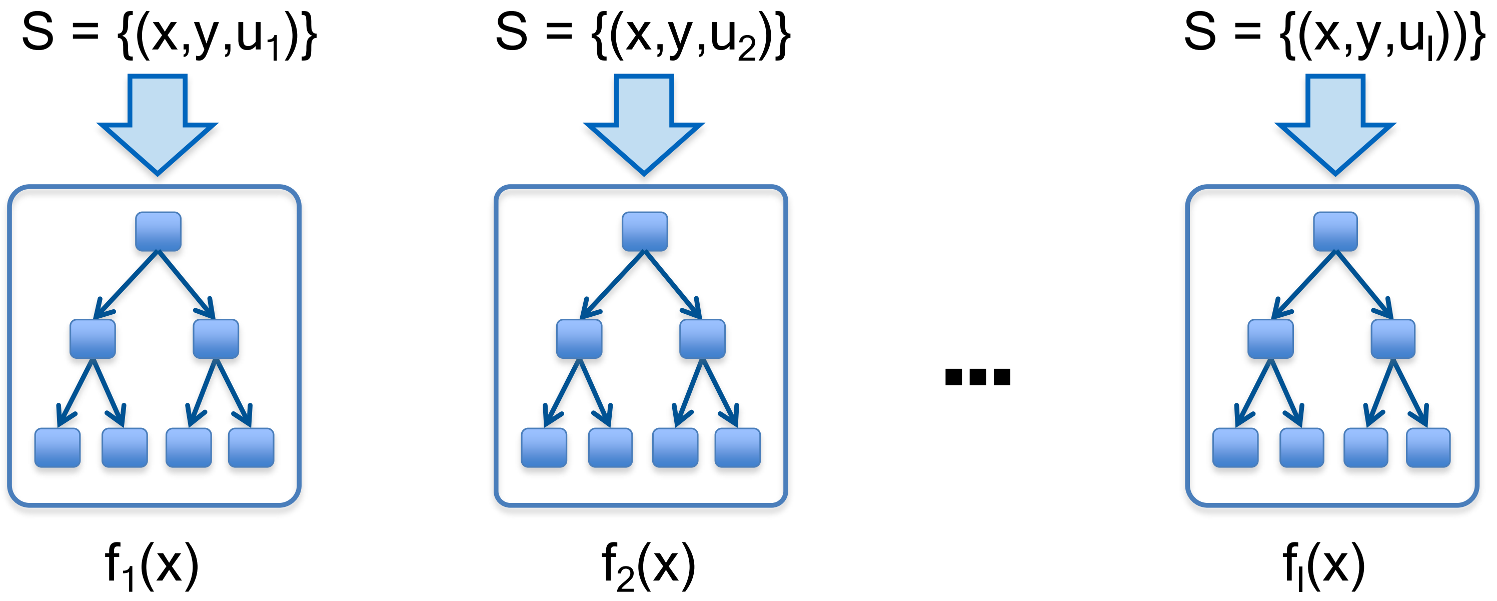
```
Assign equal weight to each training instance
For $t$ iterations:
  Apply learning algorithm (e.g. C4.5) to weighted
  dataset, store resulting model
  Compute model's error $e$ on weighted training dataset
  If $e = 0$ or $e \geq 0.5$:
    Terminate model generation
For each instance in dataset:
    If classified correctly by model, reduce weight by:
       Multiply instance's weight by $e/(1-e)$
  Normalize weight of all instances
```

Classification:

```
Assign weight = 0 to all classes
For each of the $t$ (or less) models:
    For the class this model predicts
       add $-\log e/(1-e)$ to this class's weight
Return class with highest weight
```

AdaBoost

$$F(x) = a_1 f_1(x) + a_2 f_2(x) + \ldots + a_l f_l(x)$$



u - weighting on data points
a - weight of linear combination

Stop when validation performance plateaus

see http://rob.schapire.net/papers/explaining-adaboost.pdf

More on Boosting

The training error of AdaBoost drops exponentially fast.

AdaBoost.M1 works well with so-called ***weak*** learners; only condition: error does not exceed 0.5.

- example of weak learner: decision stump (1-level decision trees)

Boosting needs weights, but boosting without weights can be applied:

- resample data with probability determined by weights

In practice, boosting can overfit if too many iterations are performed (in contrast to bagging) resulting in poor performance on test data.

Boosting is related a more general idea that has been rediscovered in multiple fields with different names such as the multiplicative weights update method or additive models.

Additive Models for Numeric Prediction

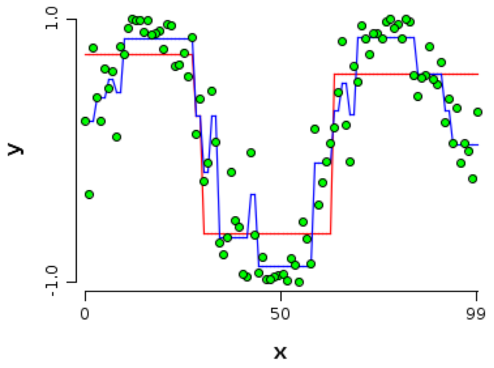
In statistics, boosting is a greedy algorithm for fitting an ***additive model***.

- Additive models can be built in parallel or sequentially and describe a large class of statistical models.

A ***forward stagewise additive model (FSAM)*** is a well-known statistical technique.

- For numeric prediction:
  1. Build simple regression model (e.g., regression tree or one-dimens. regression).
  2. Gather residuals, learn model predicting ***residuals*** (e.g., another regression tree), and repeat.
- To predict, sum up individual predictions from all regression models.

Additive regression greedily minimizes squared error of ensemble if base learner minimizes squared error.

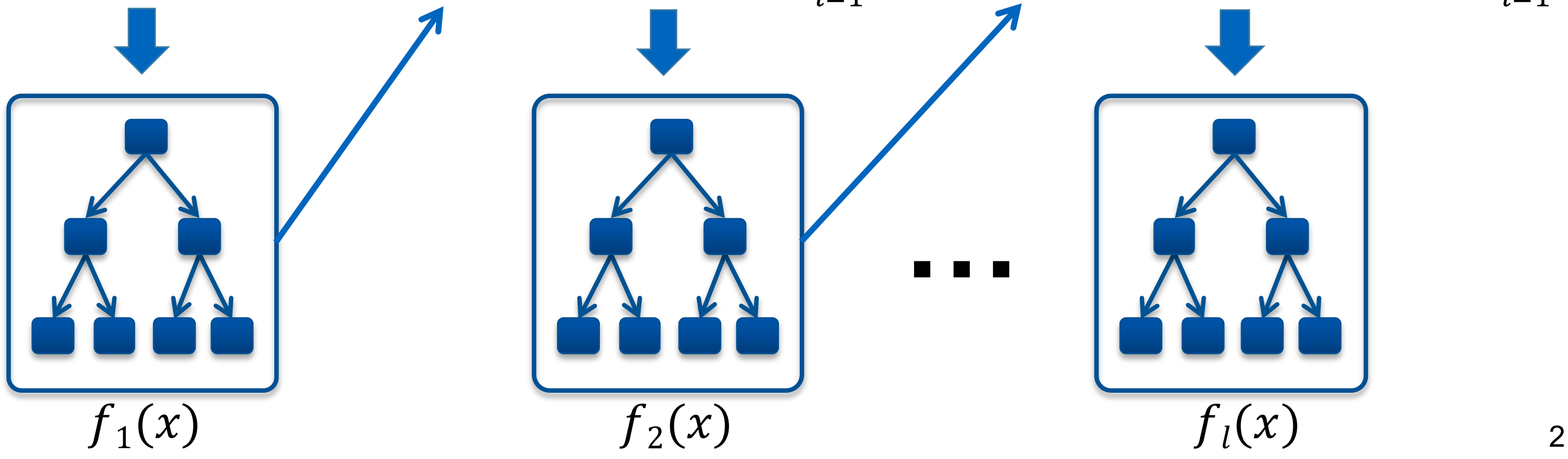


Forward Stagewise Additive Models (FSAM)

```
Start with a function $F_0(x) = mean(y),\ m = 0$
For m in 1 to M:
    $r = y - F_{m-1}(x)$
    $f_m(x) = fit\_model(X, r)$
    $F_m(x) = F_{m-1}(x) + \eta f_m(x)$
```

$$F_m(x) = f_1(x) + \eta_2 f_2(x) + \ldots + \eta_m f_m(x)$$

$$S_1 = \{(x_i, y_i)\}_{i=1}^{n} \longrightarrow S_2 = \{(x_i, y_i - f_1(x_i))\}_{i=1}^{n} \longrightarrow S_m = \{(x_i, r_{m-2} - f_{m-1}(x_i))\}_{i=1}^{n}$$



Gradient Boosted Trees (GBT)

- Gradient Boosted Trees (GBTs) is related to FSAM and uses the same pseudo code.
- The fundamental difference is in how residuals are computed and used. FSAM uses direct residuals (actual minus predicted values), while GBT uses pseudo-residuals derived from the gradient of a loss function.
- The **cross-entropy loss** $L(y,p) = -\sum_i y_i * \log(p_i)$ between the distribution of labels y and the predictions q can be used as a loss function to quantify the difference between these distributions in multi-class classification.
- **Pseudo-residuals** r for binary classification are the derivative of the loss function with respect to the predicted probability *p*:

$$r = -\frac{\partial L(y,p)}{\partial p} = -\left[\frac{-y}{p} + \frac{1-y}{1-p}\right] = \frac{y-p}{p(1-p)}$$

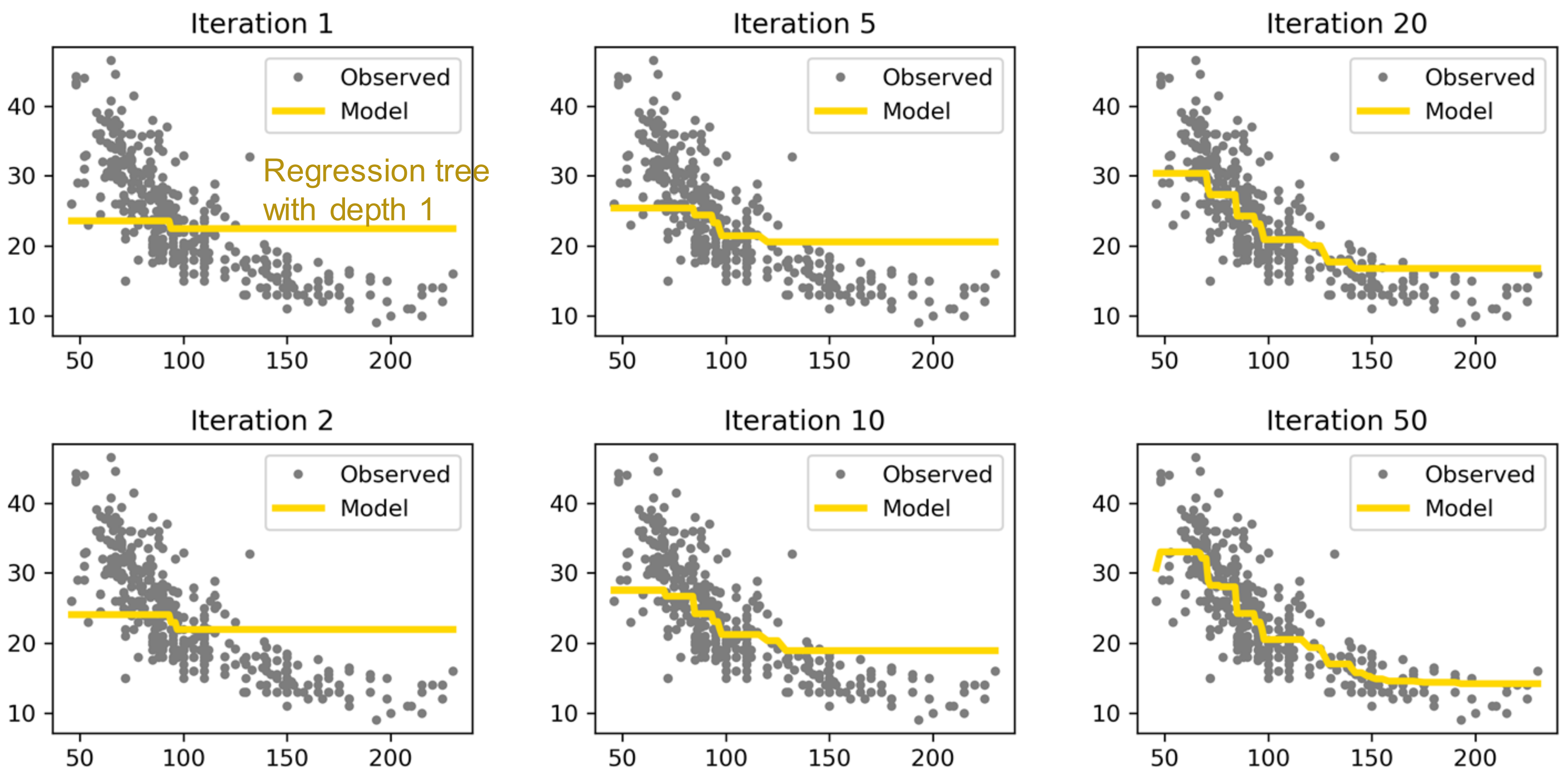
Outlook	**Temperature**	**Humidity**	**Windy**	y	p	r
sunny	hot	high	false	**1**	0.7	1.43
sunny	hot	high	true	**0**	0.7	-2.43
overcast	hot	high	false	**1**	0.3	3.33
rain	mild	high	false	**0**	0.3	-1.43

$$r = \frac{1-0.7}{0.7 \times (1-0.7)} \approx 1.43$$

$$r = \frac{0-0.7}{0.7 \times (1-0.7)} \approx -2.43$$

- Fit a model $f_m(x) = fit\_model(X, r)$ and add to the ensemble: $F_m(x) = F_{m-1}(x) + \eta f_m(x)$

Example of a Gradient Boosted Tree



FSAM vs. GBT

FSAM focuses on fitting new components to the residuals to directly correct errors, whereas GBT aims to minimize the overall loss function by moving in the direction indicated by the gradient of the loss.

FSAM may retain more interpretability due to its simpler components and approach, while GBRT often achieves higher predictive performance but can be less interpretable.

MART and XGBoost are specific implementations or variations of the basic GBT framework.

Hyperparameters such as the learning rate or the number of trees combat overfitting.

The sequential process can be slow to train in both cases.

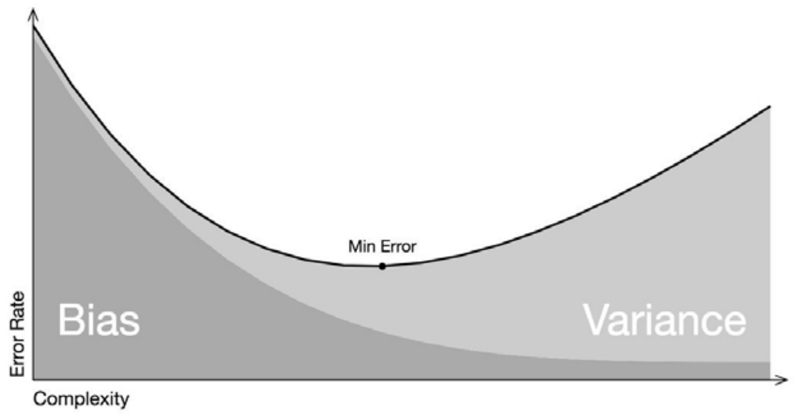
Bias-Variance Tradeoff

Bagging reduces variance of low-bias models

- Low-bias models are “complex” and unstable
- Bagging averages them together to create stability

Boosting reduces bias of low-variance models

- Low-variance models are simple with high bias
- Boosting trains sequence of models on residual error → sum of simple models is accurate



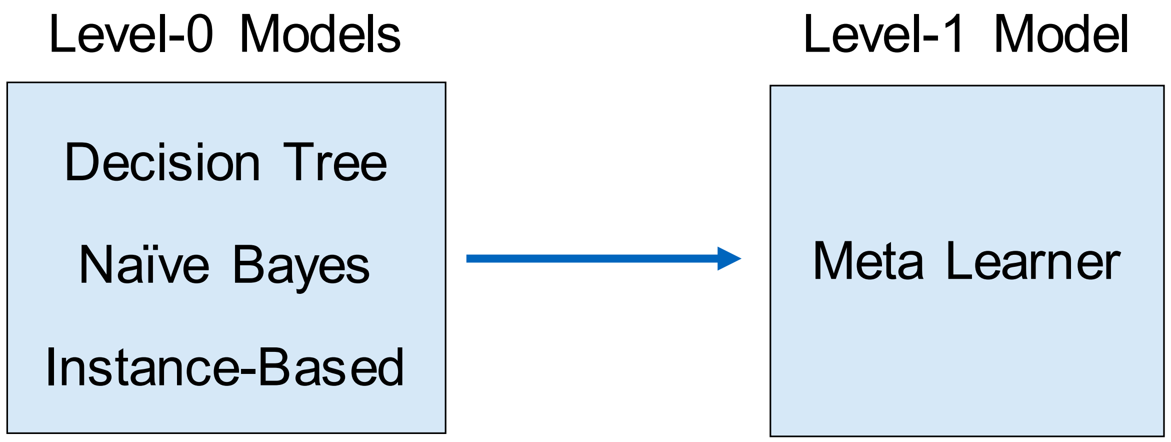
Bagging vs. Boosting

	Bagging	Boosting
Partitioning data into subsets	Random	Give misclassified cases a heavier weight
Sampling method	Sampling with replacement (elements are used multiple times)	Reweighting of training instances
Relations between models	Parallel ensemble: Each model is independent	Previous models inform subsequent models
Goal to achieve	Minimize variance	Minimize bias, improve predictive power
Method to combine models	Majority vote	Weighted average or majority vote

Stacking

In stacking, the predictions from heterogeneous classifiers are used as input into a meta-learner, which attempts to combine the predictions to create a final best predicted classification.

- Learn which learning algorithms are good (instead of voting).
- Combine learning algorithms intelligently.



Stacking

- Holdout part of the training set
- Use remaining data for training level-0 methods
- Use holdout data to train level-1 learning
- Level-1 learning: use very simple algorithm (e.g., linear model)
  - If the base learners can output class probabilities, use those as input to meta learner instead of plain classifications
  - Makes more information available to the level-1 learner
- Retrain level-0 algorithms with all the data

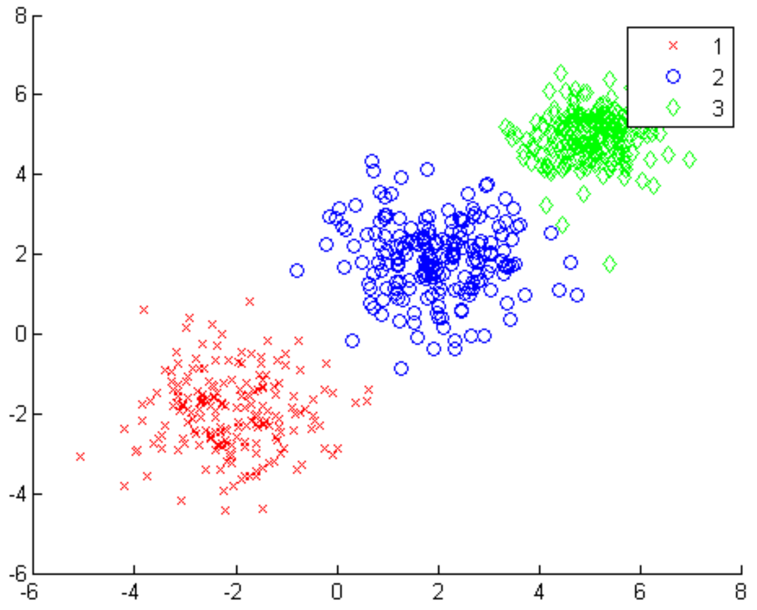
Outline for today

- Ensemble Methods
  - Bagging
  - Random Forests
  - Boosting
  - Stacking

- **Clustering**
  - partitional clustering via k-means
  - hierarchical clustering via MST
  - probabilistic clustering via Expectation Maximization (EM)

Selected Clustering Methods

- **Definitions**
- Partitional clustering via k-means
- Hierarchical clustering via MST
- Probabilistic clustering via Expectation Maximization (EM)



Clustering (Informal Definition)

Given:

- A data set with n data items of dimension p.

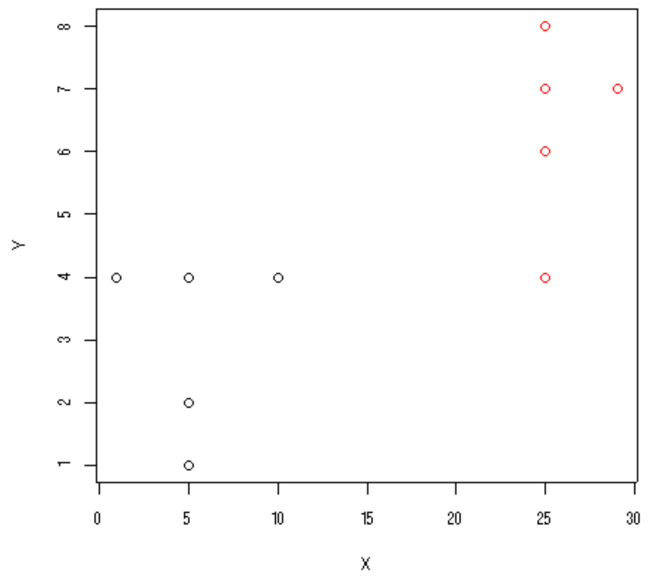
Find:

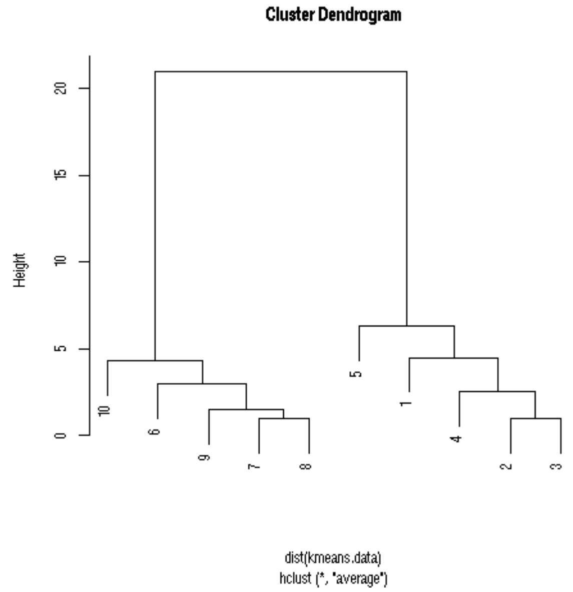
- A natural partitioning of the data set into a number of clusters (k) and noise.
- Clusters should be such that:
  - Items in same cluster are similar $\rightarrow$ Intra-cluster similarity is maximized.
  - Items from different clusters are different $\rightarrow$ Inter-cluster similarity is minimized.

Clustering Methods

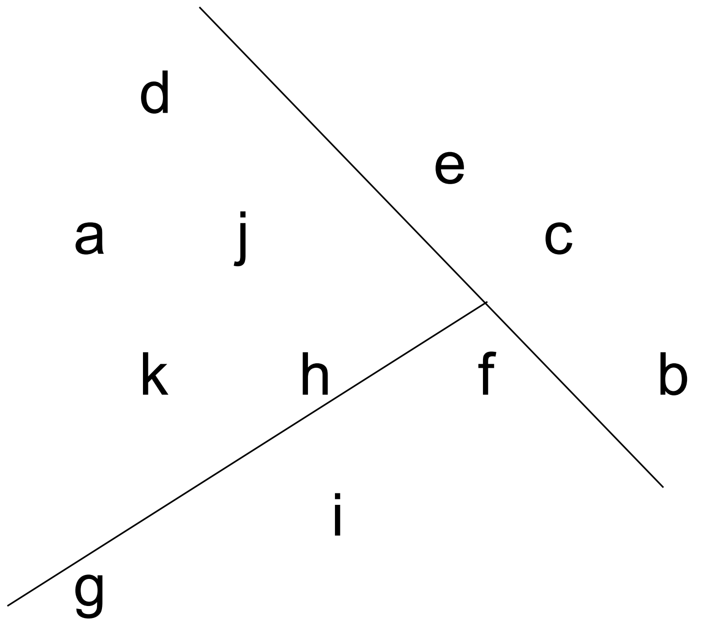
Many different methods and algorithms:

- for numeric and/or nominal data
- deterministic vs. probabilistic
- partitional vs. overlapping
- hierarchical vs. flat



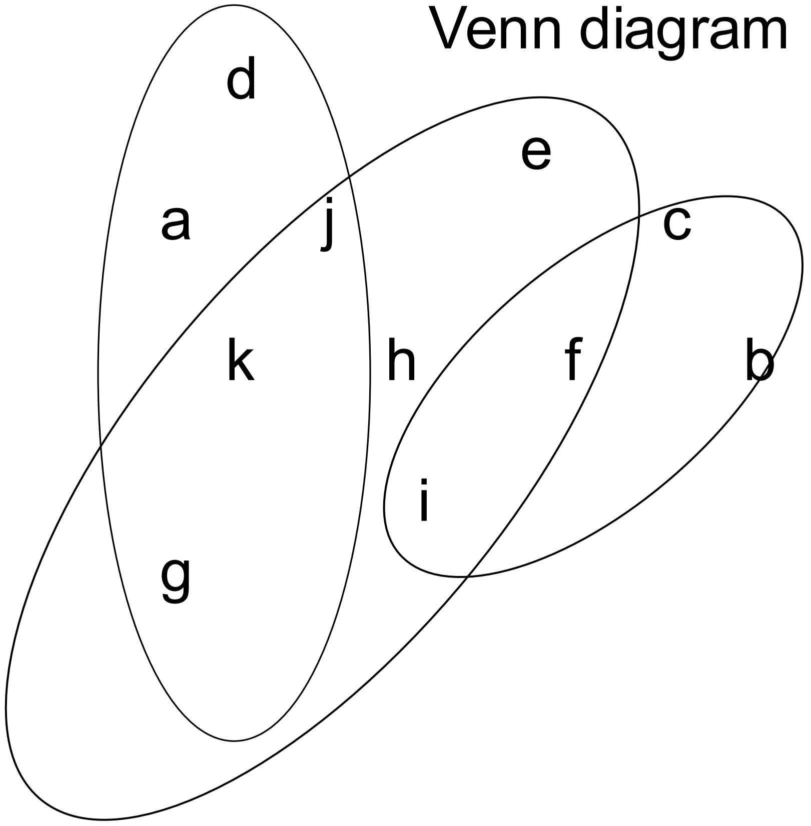


Clustering



	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.7	0.2	0.1
…			

Venn diagram



Definition of Partitional Clustering

Given a database $D = \{t_1, t_2, \dots, t_n\}$ of tuples and an integer value k, the clustering problem is to define a mapping:

$f\colon D \rightarrow \{1, .., k\}$ where each t_i is assigned to one cluster:

$$K_j, \qquad 1 \leq j \leq k$$

A cluster, K_j, contains precisely those tuples mapped to it.

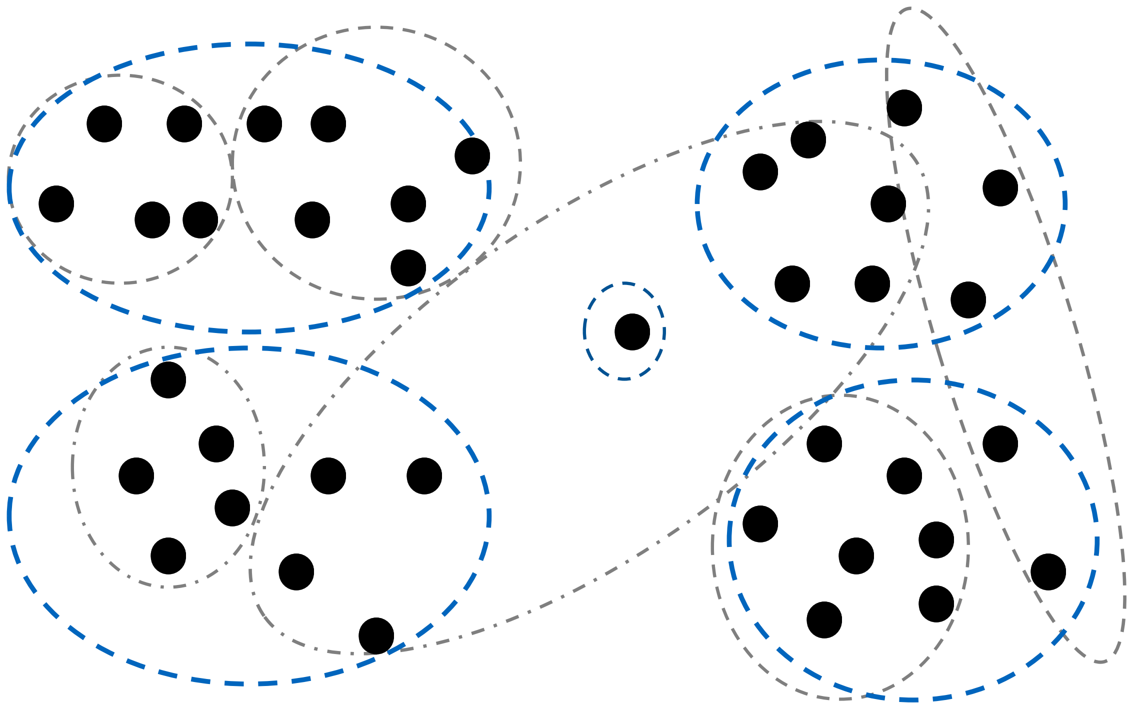
Unlike classification problems, clusters are not known a priori.

Clustering Applications

- Segment customer database based on similar buying patterns.
- Group houses in a town into neighborhoods based on similar features.
- Identify similar web usage patterns.
- Identify new plant species.

…

Clustering Houses



Geographic: Distance Based

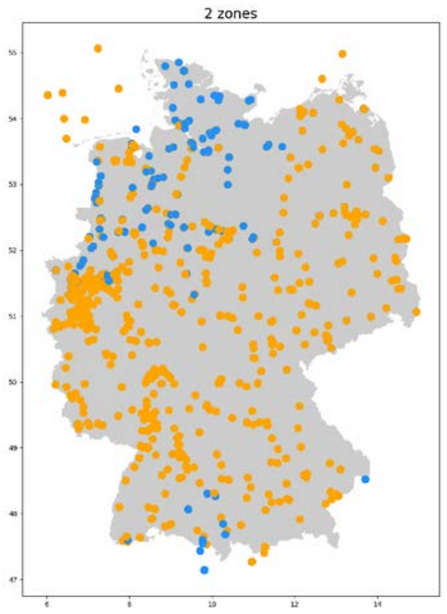
Size Based

Clustering Issues

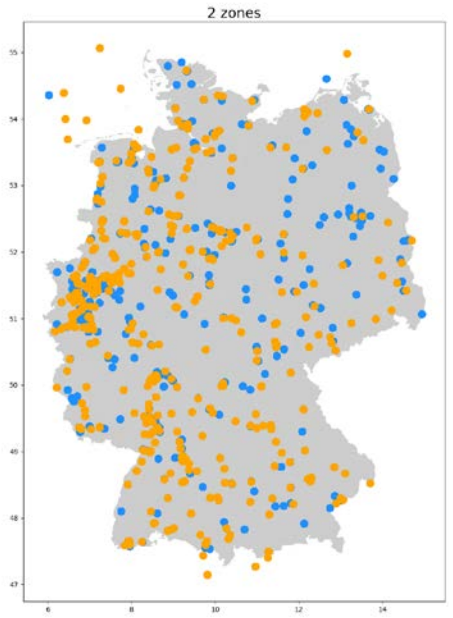
- Interpreting results
- Outlier handling
- Number of clusters
- Scalability of algorithms
- Evaluating results
  - Manual inspection
  - Benchmarking on existing labels
  - Cluster quality measures
    - distance measures
    - high similarity within a cluster, low across clusters

Application Example

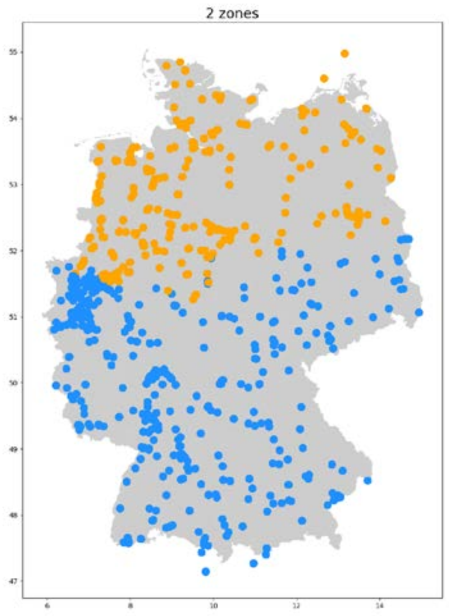
The Bidding Zone Review by the European Commission as an example.
Different methods and different inputs (prices only vs. prices, longitude and lattitude).
Different methods lead to different clusterings.



K-Means



Optimization-based



Spectral

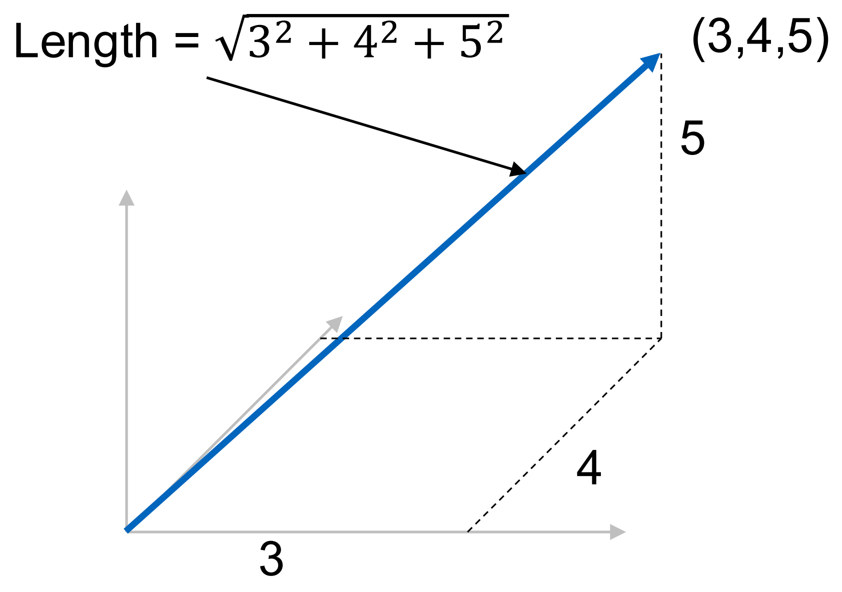
Vector Norms

Measure of how long a vector is: $\|\boldsymbol{x}\|$

- Euclidean norm is represented as

$$\|(a, b, c, \dots)\| = \sqrt{a^2 + b^2 + c^2 + \cdots}$$

Geometrically the shortest distance to travel from the origin to the destination.



Distance Measures as Vector Norms

The Euclidean norm measures the length (or magnitude) of the vector from the origin to the point represented by the vector in the Euclidean space.

Let’s now define **Euclidean distance**:
Each instance i lives in p-dimensional space.

- $x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,p})$ describing the Cartesian coordinates.

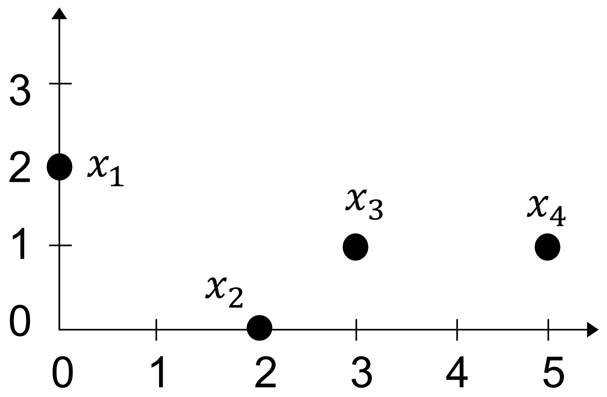
Euclidean distance (or L^2 distance) between two instances:

$$d_2(x, y) = \left(\sum_{j=1}^{p} \left| x_j - y_j \right|^2 \right)^{1/2}$$

Manhattan (or L^1 distance, or city block, or absolute) distance:

$$d_1(x, y) = \left(\sum_{j=1}^{p} \left| x_j - y_j \right| \right)$$

Euclidean Distance



Point	$x_{i,1}$	$x_{i,2}$
x_1	0	2
x_2	2	0
x_3	3	1
x_4	5	1

	x_1	x_2	x_3	x_4
x_1	0	2.828	3.162	5.099
x_2	2.828	0	1.414	3.162
x_3	3.162	1.414	0	2
x_4	5.099	3.162	2	0

Distance Matrix

Hierarchical Clustering

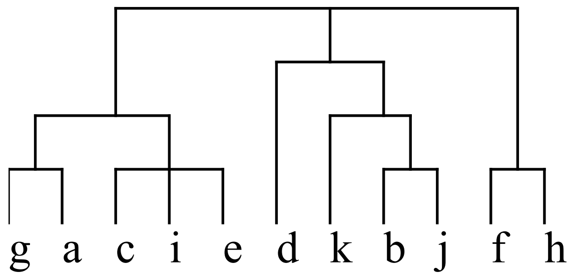
Bottom up

- start with single-instance clusters
- at each step, join the two closest clusters
- design decision: distance between clusters
  - e.g., two closest instances in clusters
    vs. distance between cluster means

Top down

- start with one universal cluster
- find two clusters
- proceed recursively on each subset
- can be very fast

Both methods produce a dendogram.



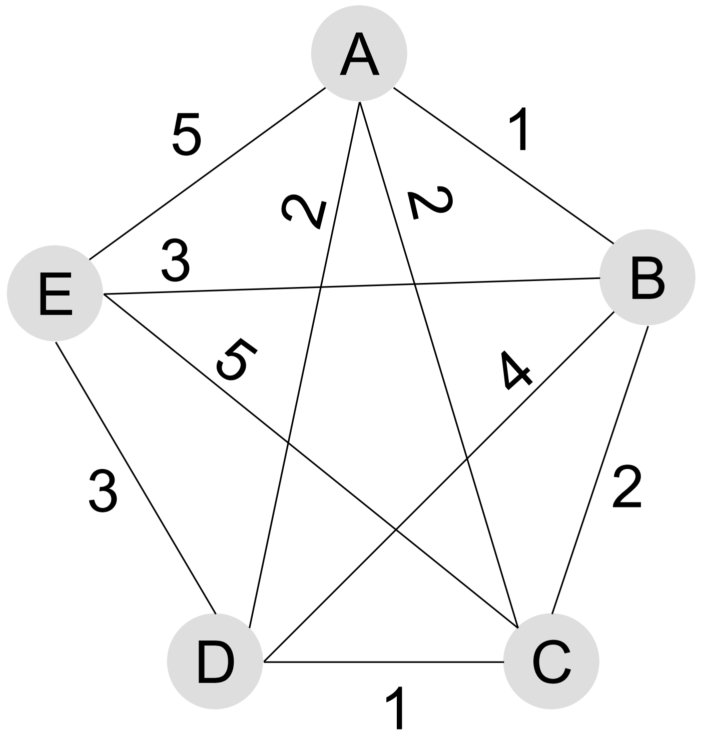
Dissimilarity Matrices

Many clustering algorithms use an **adjacency matrix** based on distances as input.

dist((A),(B))

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

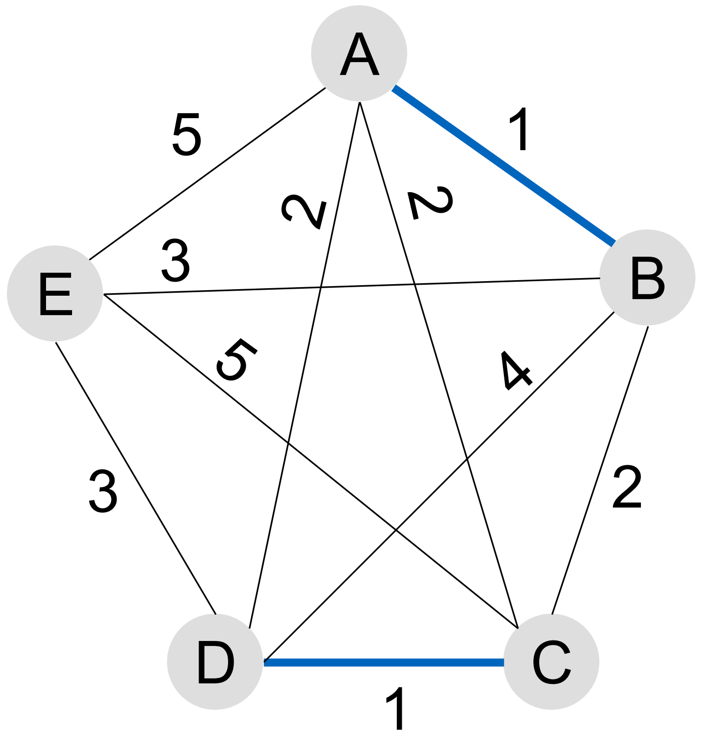
Graph Perspective



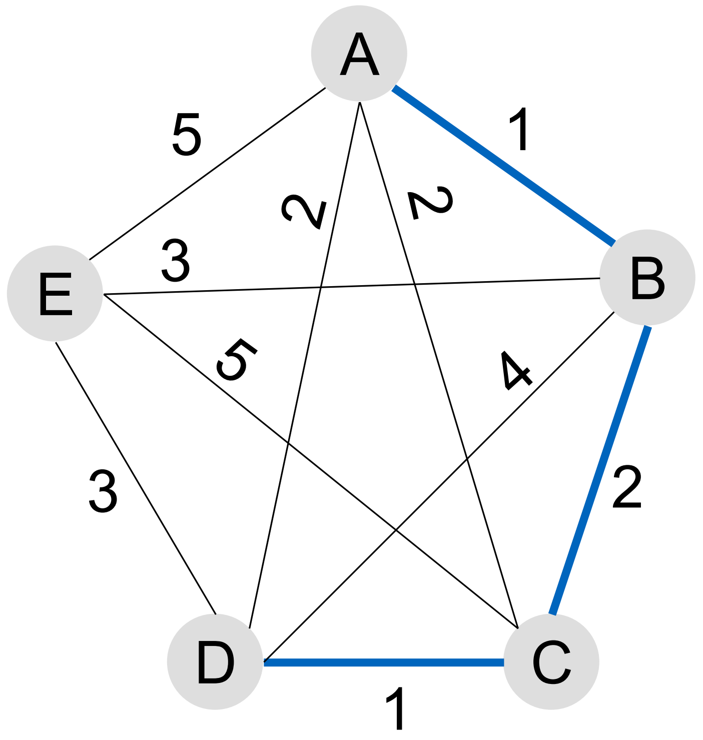
MST Algorithm

- Compute the minimal spanning tree of the graph
  - A minimum-weight tree in a weighted graph which contains all of the graph's vertices with minimal total weight
  - Kruskal – $O(d \log(d))$, with d being edges and n nodes
  - Prim – $O(d \log(n))$
- Connected graph components form clusters

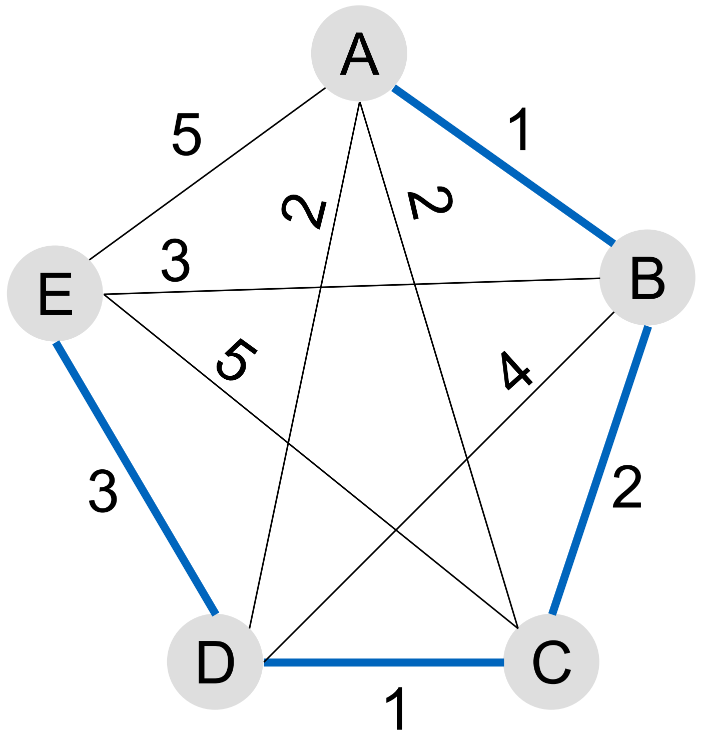
Kruskal’s Algorithm for Finding the MST



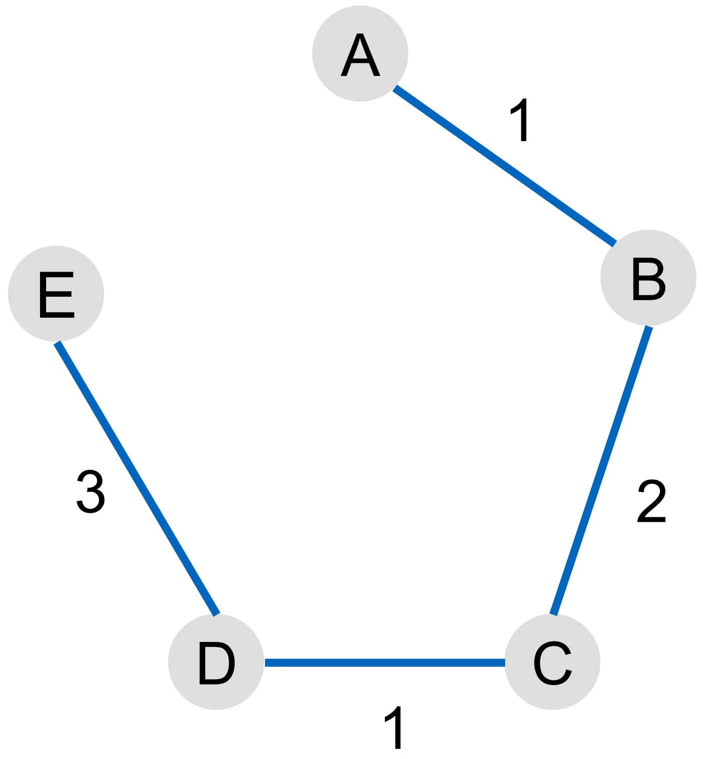
Kruskal's Algorithm



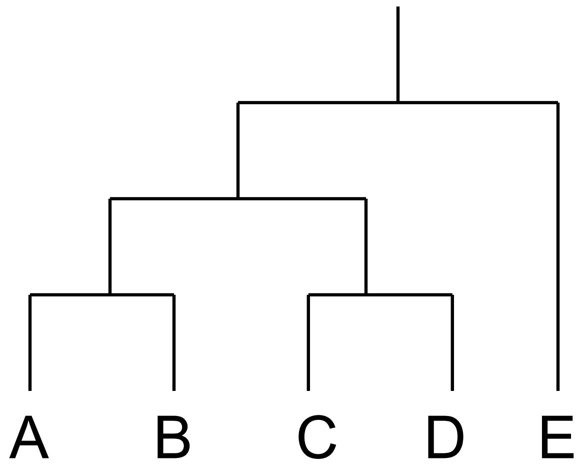
Kruskal’s Algorithm



MST Clustering



Dendogram

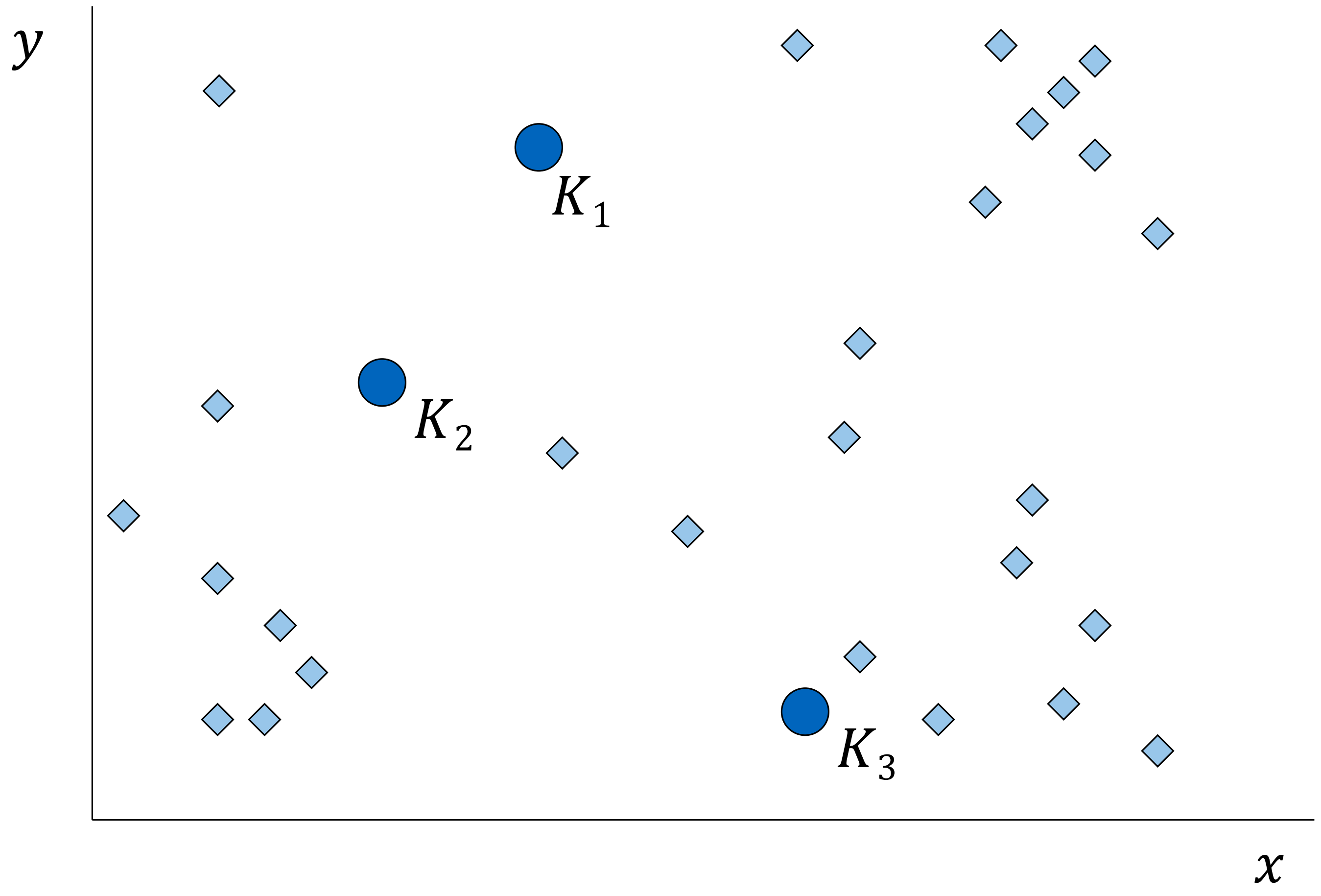


K-Means Clustering

Works with numeric data only.

1. Pick a number (k) of cluster centers (at random).
2. Assign every item to its nearest cluster center (e.g., using Euclidean distance).
3. Move each cluster center to the mean of its assigned items.
4. Repeat steps 2 and 3 until convergence (change in cluster assignments less than a threshold).

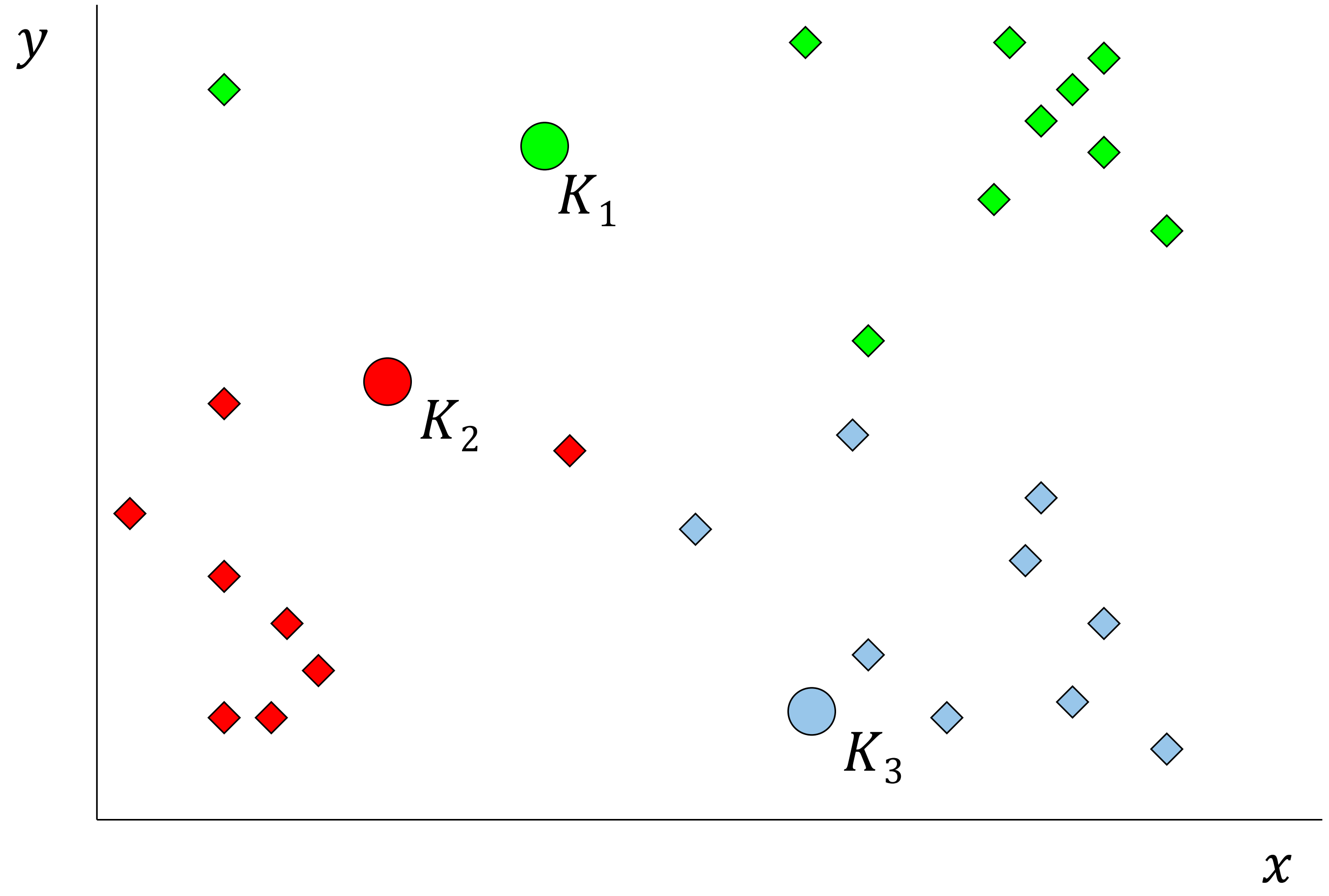
K-Means: Example, Step 1 - Two Dimensions



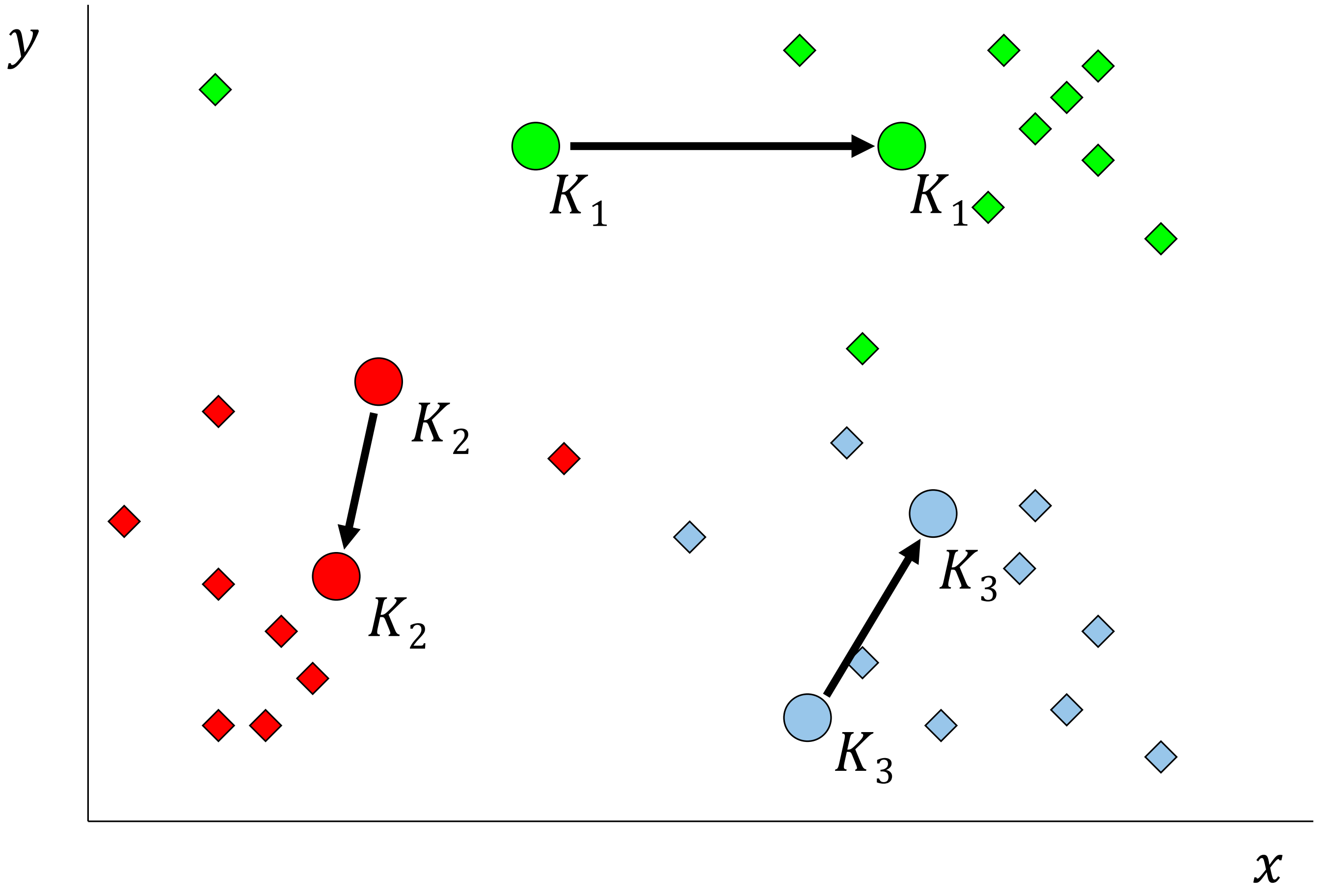
Pick 3 initial cluster centers (randomly)

K-Means: Example, Step 2

Assign each point to the closest cluster center



K-Means: Example, Step 3

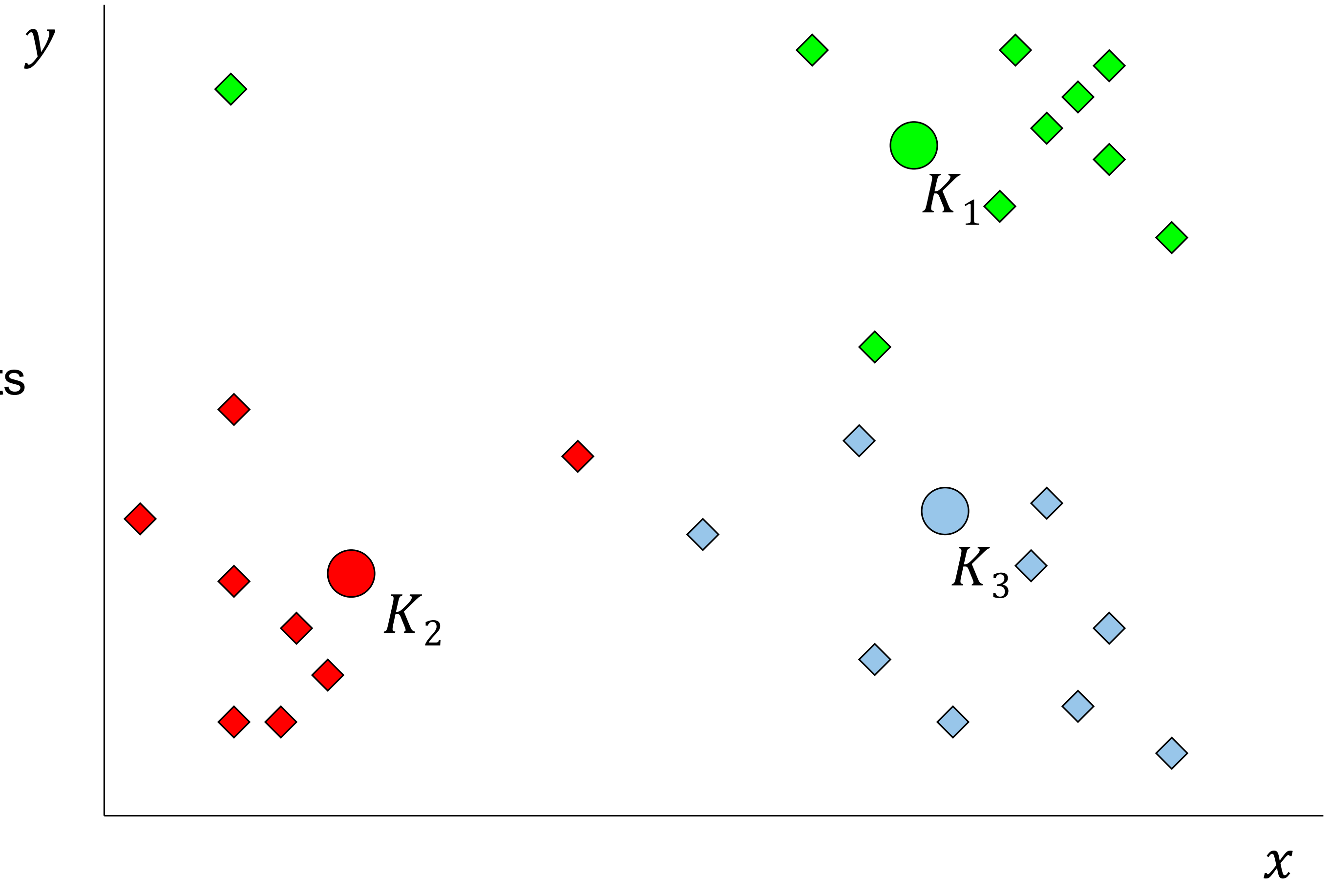


Move each cluster center to the mean of each cluster

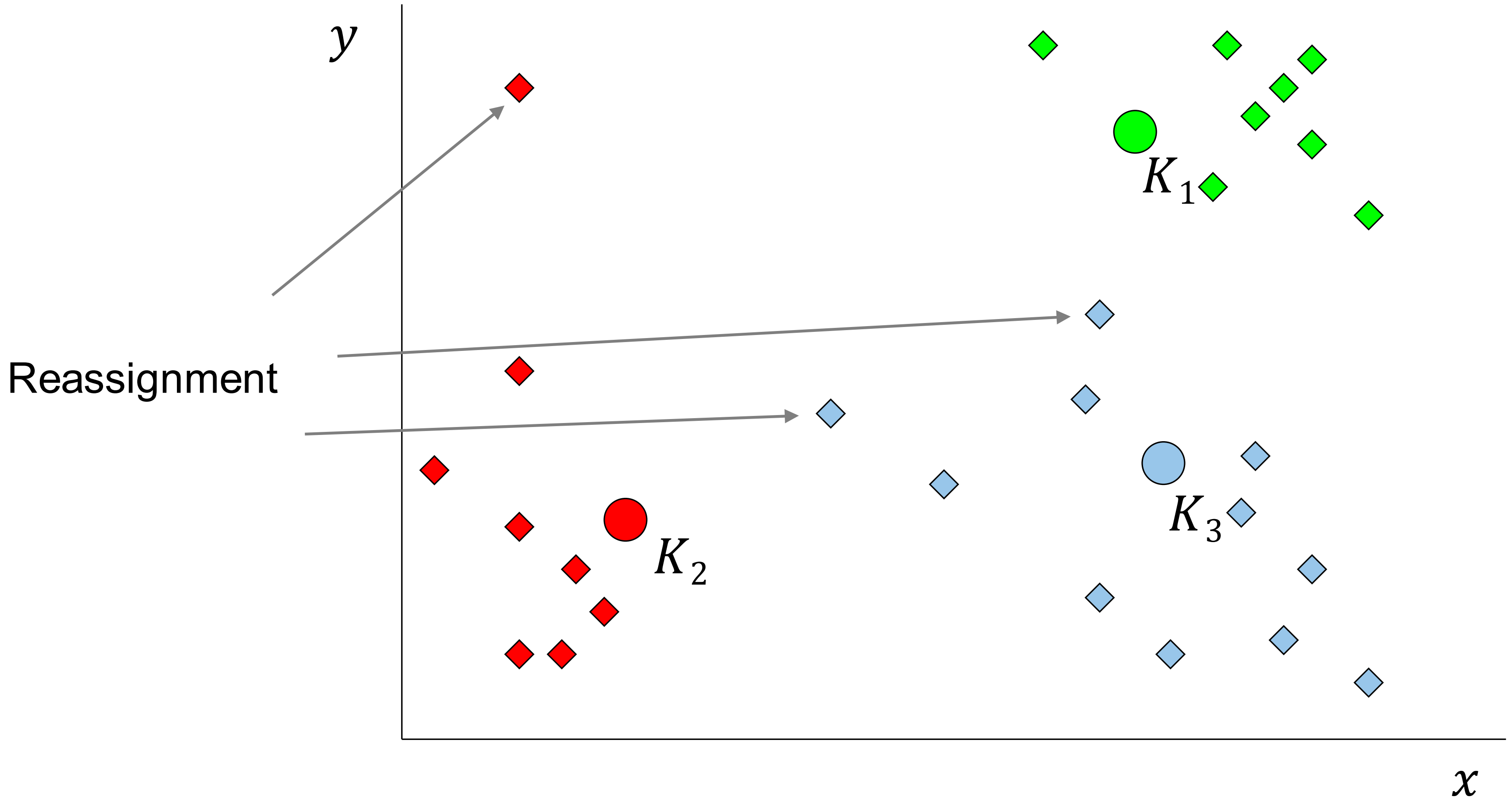
K-Means: Example, Step 4

Reassign points closest to a different new cluster center

Q: Which points are reassigned?

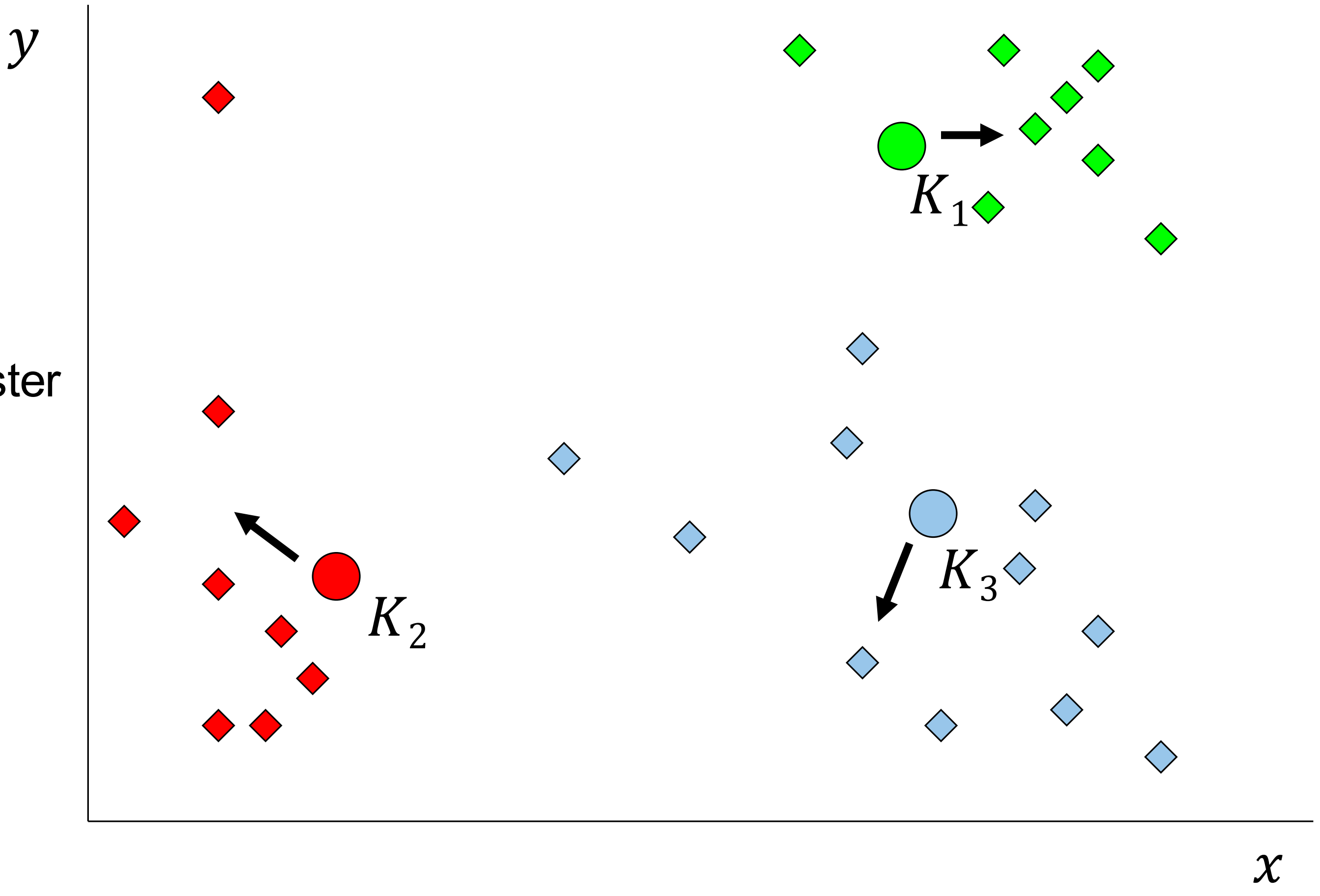


K-Means: Example, Step 4 …



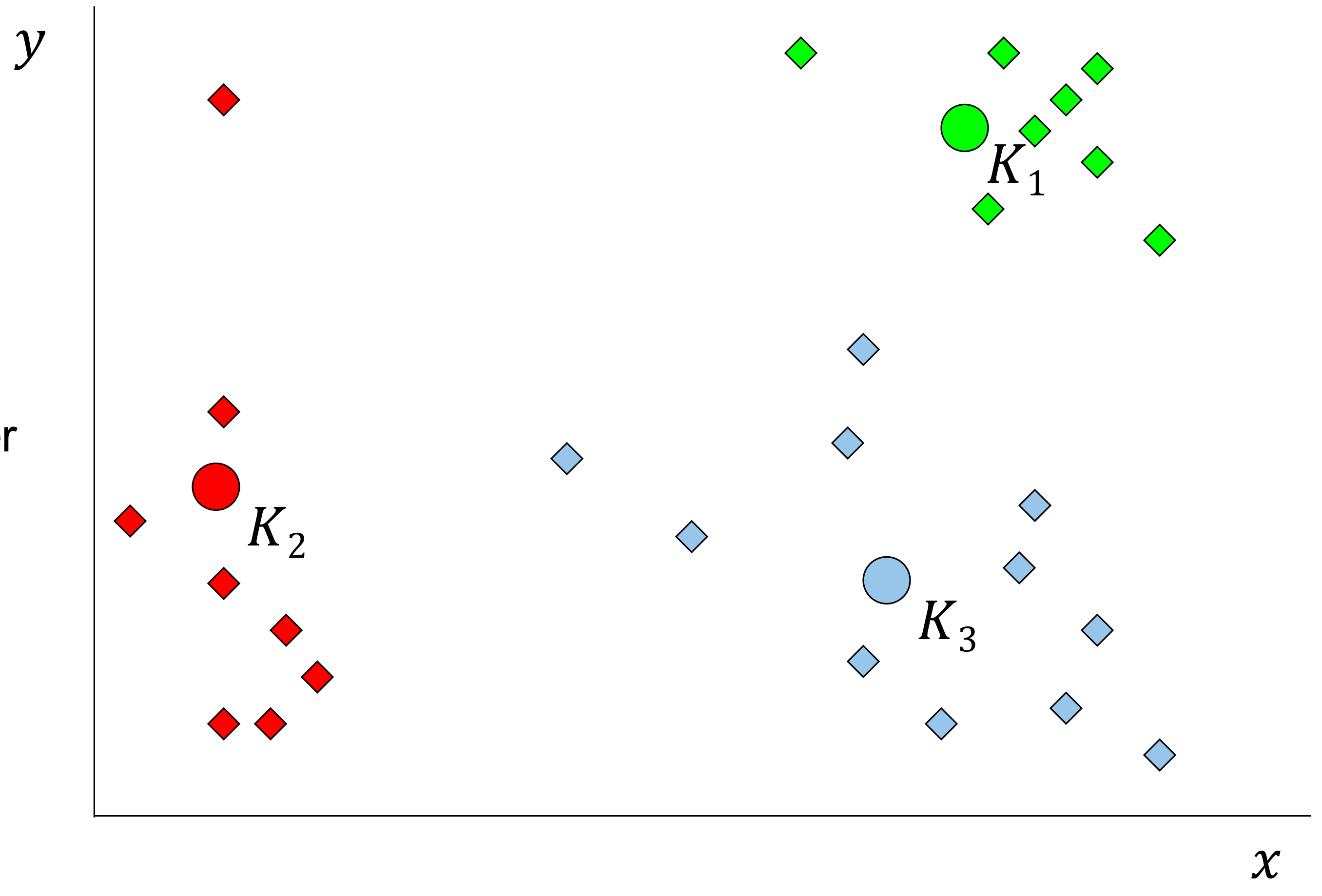
K-Means: Example, Step 4b

Re-compute cluster means



K-Means: Example, Step 5

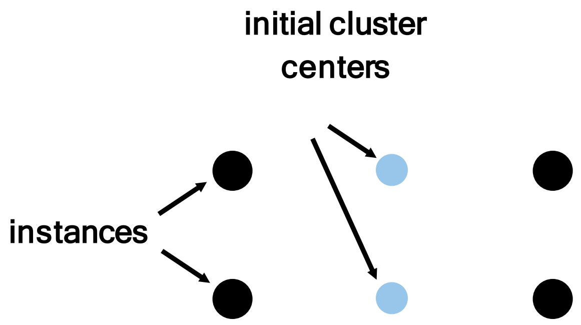
Move cluster centers to cluster means



Discussion

Result can vary significantly depending on initial choice of seeds.

→ Can get trapped in local minimum.



To increase chance of finding global optimum: restart with different random seeds.

The number of clusters needs to be determined a priori.

K-Means: Summary

Advantages:

- simple, understandable
- items automatically assigned to clusters

Disadvantages:

- must pick number of cluster before hand
- all items forced into a cluster
  - too sensitive to outliers

K-means is an example of
a **partitional** clustering algorithm.

Probability-Based Clustering: Expectation Maximization

Consider clustering data into k clusters.

Model each cluster with a probability distribution.

This set of k distributions is called a mixture, and the overall model is a finite ***mixture model***.

Each probability distribution gives the probability of an instance being in a given cluster.

Probability-Based Clustering: Basics

Simplest case:

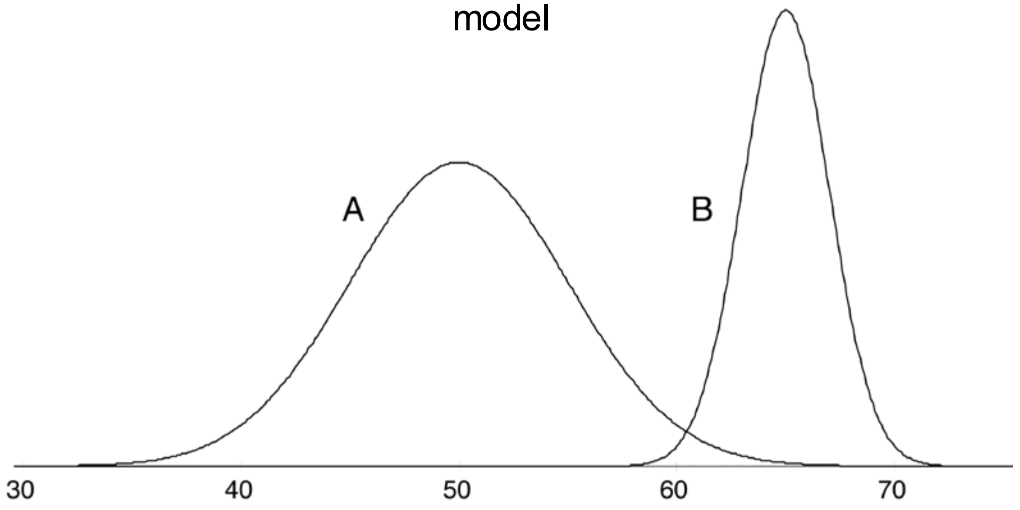
A single numeric attribute and two clusters A and B each represented by a normal distribution.

- Parameters for A: μ_A - mean, σ_A - standard dev.
- Parameters for B: μ_B - mean, σ_B - standard dev.
- And $\Pr[A]$, $\Pr[B] = 1 - \Pr[A]$, the prior probabilities of being in cluster A and B respectively

Probability-Based Clustering

data											
A	51	B	62	B	64	A	48	A	39	A	51
A	43	A	47	A	51	B	64	B	62	A	48
B	62	A	52	A	52	A	51	B	64	B	64
B	64	B	64	B	62	B	63	A	52	A	42
A	45	A	51	A	49	A	43	B	63	A	48
A	42	B	65	A	48	B	65	B	64	A	41
A	46	A	48	B	62	B	66	A	48		
A	45	A	49	A	43	B	65	B	64		
A	45	A	46	A	40	A	46	A	48		

model



$\mu_A = 50, \sigma_A = 5, p_A = 0.6 \qquad \mu_B = 65, \sigma_B = 2, p_B = 0.4$

Probability-Based Clustering

If we knew which instance came from each cluster, we could estimate these values.

$$\mu_A = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

- $\mu_A, \sigma_A, \mu_B, \sigma_B, \Pr[A]$

$$\sigma_A^2 = \frac{(x_1-\mu_A)^2+(x_2-\mu_A)^2+\cdots+(x_n-\mu_A)^2}{n-1}$$

- if data is labeled, easy
- $\Pr[A]$ is just the proportion of instances in cluster A

But clustering is typically used for unlabeled data.

Question is, how do we know the parameters for the mixture?

Use an iterative approach similar in spirit to the k-means algorithm.

Expectation Maximization (in a Nutshell)

Start with initial guesses for the parameters $\mu_A, \sigma_A, \mu_B, \sigma_B, \Pr[A]$.

Calculate cluster probabilities w_i for each instance.

- Expectation

Re-estimate the distribution parameters from probabilities.

- Maximization

Repeat.

Computing the Probability that an Instance Belongs to a Cluster

If we knew the parameters, we could compute the probability that an instance belongs to a cluster.

Given an instance x, the probability that it belongs to cluster A is

- Bayes Rule:

$$\Pr[A \mid x] = \frac{\Pr[x \mid A] \cdot \Pr[A]}{\Pr[x]} = \frac{f(x;\mu_A,\sigma_A)\Pr[A]}{\Pr[x]}$$

$$\underline{f(x;\mu_A,\sigma_A) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.}$$

The denominator $\Pr[x]$ will disappear:

- divide by $\Pr[A|x] + \Pr[B|x]$

$\nwarrow$ normalize

Maximization

Let w_i be the posterior probability of instance i belonging to cluster *A*, i.e., $w_i = \Pr[A|x]$

Recalculated parameters are

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}$$

$$\sigma_A{}^2 = \frac{w_1 (x_1 - \mu_A)^2 + w_2 (x_2 - \mu_A)^2 + \cdots + w_n (x_n - \mu_A)^2}{w_1 + w_2 + \cdots + w_n}$$

x_i now describes all instances, not just the ones of cluster A.
This is a **maximum likelihood estimator** for the variance.

Termination

The EM algorithm converges to a maximum.

Continue until overall likelihood growth is negligible.

- measure for finding the maximum likelihood

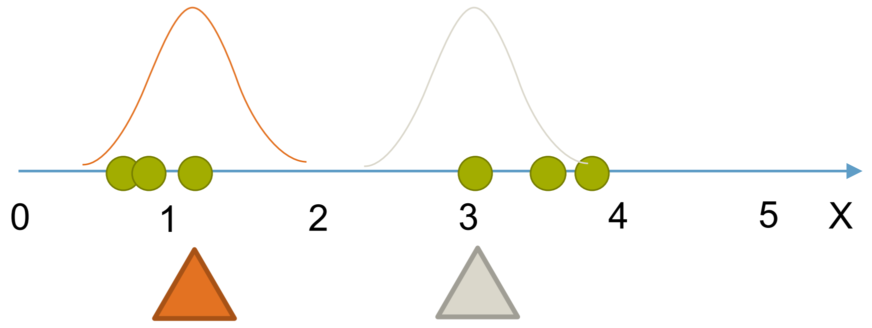
$$\boxed{\prod_{i=1}^{n}\left(\Pr[A]\Pr[x_i \mid A]+\Pr[B]\Pr[x_i \mid B]\right)}$$

The maximum found by EM could be a local optimum, so repeat several times with different initial values.

EM Clustering Example

Given $k = 2$, perform EM algorithm with the following instances

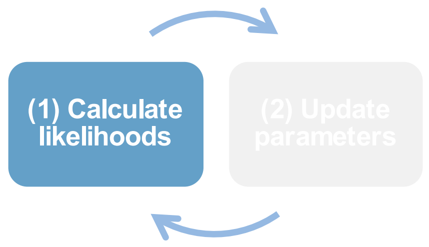
Instance	1	2	3	4	5	6
Value	0.76	0.86	1.12	3.05	3.51	3.75



Initialisation

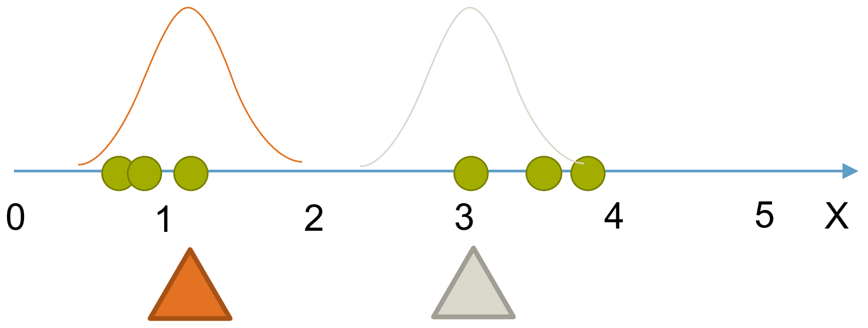
μ_A	1.12
σ_A	1.00
p_A	50%
μ_B	3.05
σ_B	1.00
p_B	50%

EM Clustering



Solution phase 1, step 1:
Calculate cluster probabilities

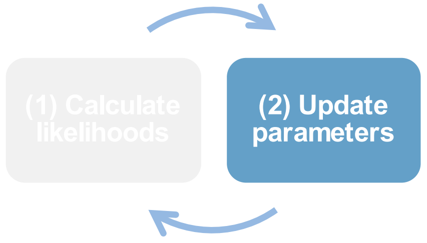
Instance	1	2	3	4	5	6
Value	0.76	0.86	1.12	3.05	3.51	3.75



▲	μ_A	1.12
	σ_A	1.00
	p_A	50%
△	μ_B	3.05
	σ_B	1.00
	p_B	50%

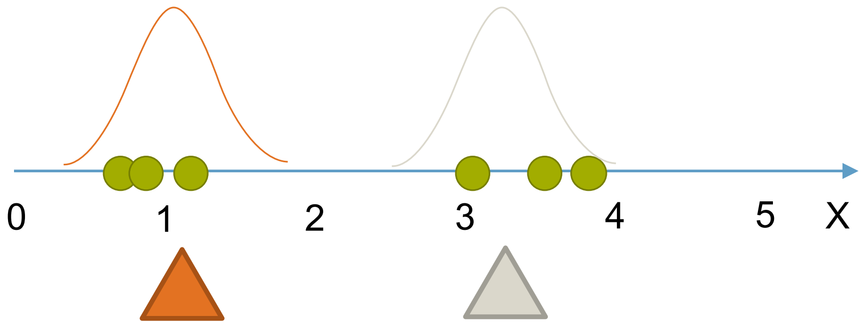
	1	2	3	4	5	6
Pr[A|x]	92.81%	91.41%	86.56%	13.44%	6.01%	3.87%
Pr[B|x]	7.19%	8.59%	13.44%	86.56%	93.99%	96.13%

EM Clustering



Solution phase 1, step 2:
Update distribution parameters

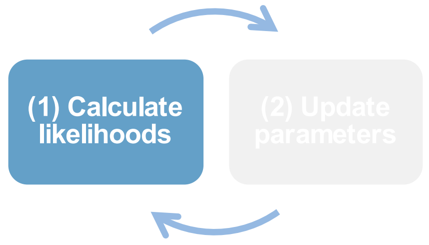
Instance	1	2	3	4	5	6
Value	0.76	0.86	1.12	3.05	3.51	3.75



μ_A	1.10
σ_A	0.66
p_A	49%
μ_B	3.21
σ_B	0.78
p_B	51%

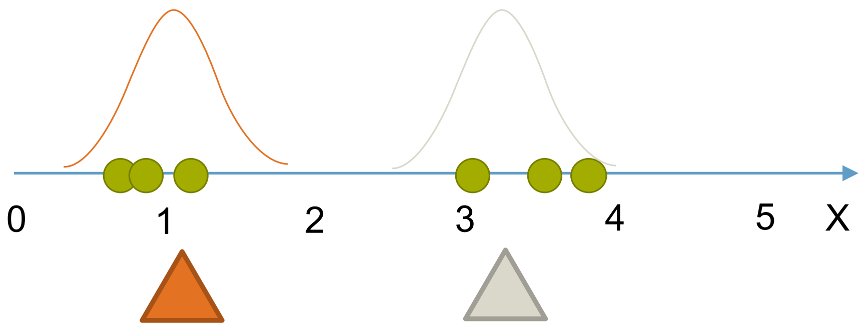
	1	2	3	4	5	6
Pr[A|x]	92.81%	91.41%	86.56%	13.44%	6.01%	3.87%
Pr[B|x]	7.19%	8.59%	13.44%	86.56%	93.99%	96.13%

EM Clustering



Solution phase 2, step 1:
Calculate cluster probabilities

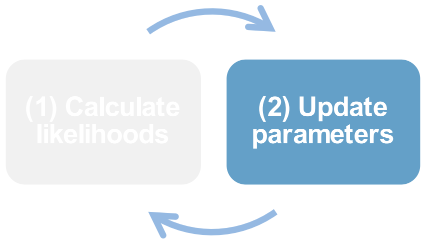
Instance	1	2	3	4	5	6
Value	0.76	0.86	1.12	3.05	3.51	3.75



▲		
▲	μ_A	1.10
▲	σ_A	0.66
▲	p_A	49%
△	μ_B	3.21
△	σ_B	0.78
△	p_B	51%

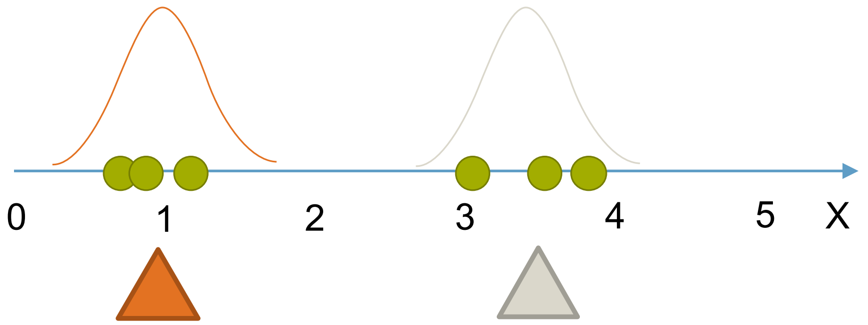
	1	2	3	4	5	6
Pr[A|x]	99.25%	98.97%	97.55%	1.49%	0.16%	0.05%
Pr[B|x]	0.75%	1.03%	2.45%	98.51%	99.84%	99.95%

EM Clustering



Solution phase 2, step 2:
Update distribution parameters

Instance	1	2	3	4	5	6
Value	0.76	0.86	1.12	3.05	3.51	3.75



μ_A	0.92
σ_A	0.22
p_A	49%
μ_B	3.40
σ_B	0.41
p_B	51%

	1	2	3	4	5	6
Pr[A|x]	99.25%	98.97%	97.55%	1.49%	0.16%	0.05%
Pr[B|x]	0.75%	1.03%	2.45%	98.51%	99.84%	99.95%

Extending the Model

Multiple clusters

- Extending to multiple clusters is straightforward, just use k normal distributions.

Multiple attributes

- Multiply the probabilities of all attributes to get the probability of an instance (assuming independence of the attributes).
- In case of correlation among attributes use multivariate normal distributions.

For nominal attributes

- You can't use normal distribution. Have to create probability distributions for the values.