# Business Analytics
## Ensemble Methods and Clustering

Prof. Bichler

Decision Sciences & Systems

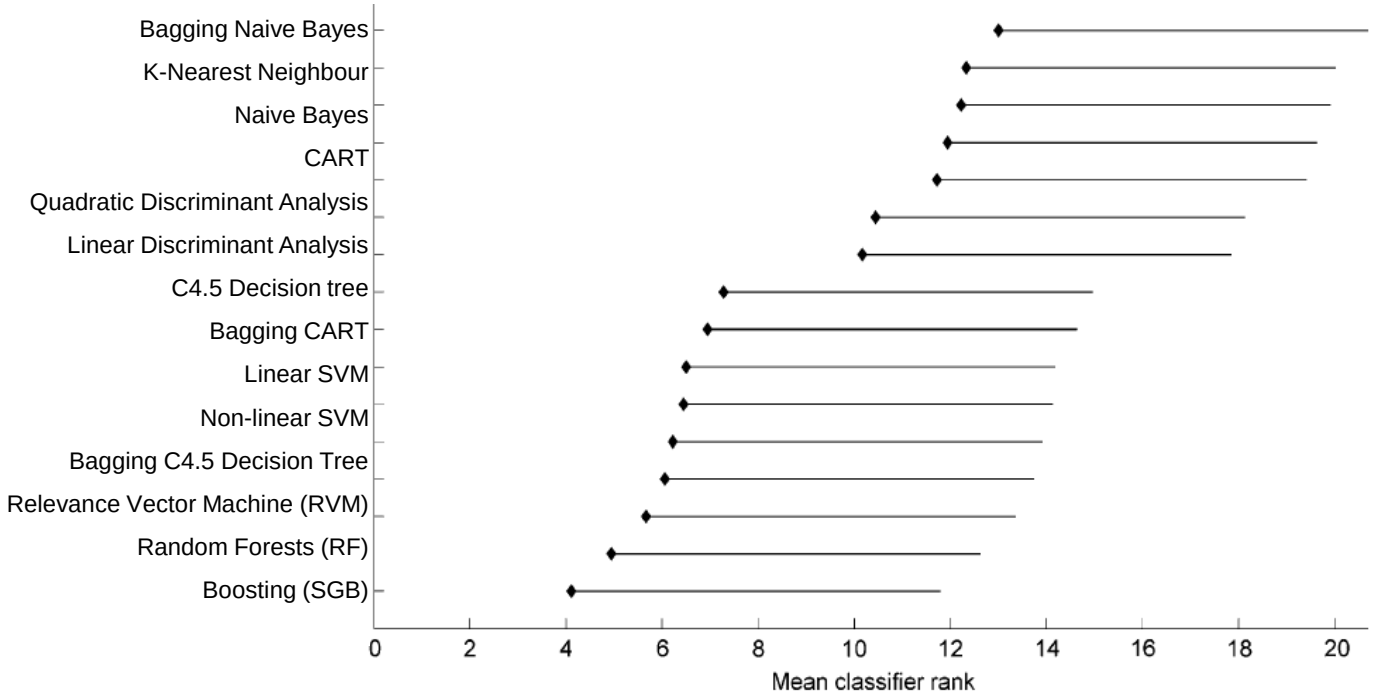
Department of Informatics

Technische Universität München

# 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**
- High-Dimensional Problems
- Association Rules and Recommenders
- Neural Networks

# An Empirical Study of Classifier Performance



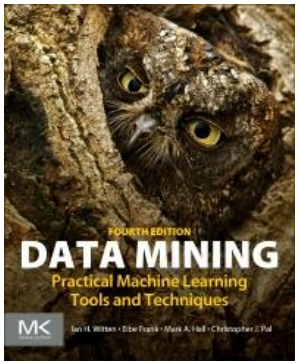

Lessmann, Stefan, and Stefan Voß. "Customer-centric decision support." *Business & Information Systems Engineering* 2.2 (2010): 79-93.

# Ensembles: Combining Multiple Models

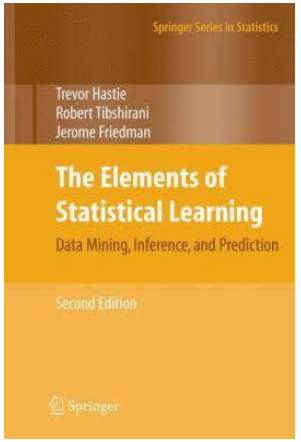
- Basic idea: build different “experts”, let them vote
- Advantage:
  - often improves predictive performance
- Disadvantage:
  - usually produces output that is very hard to analyze
  - but: there are approaches that aim to produce a single comprehensible structure

# Recommended Literature

- **Data Mining: Practical Machine Learning Tools and Techniques**
  - Ian H. Witten, Eibe Frank, Mark A. Hall, 2016
  - http://www.cs.waikato.ac.nz/ml/weka/book.html
  - Section: 12




- **The Elements of Statistical Learning**
  Trevor Hastie, Robert Tibshirani, Jerome Friedman, 2014
  - https://web.stanford.edu/~hastie/ElemStatLearn/
  - Section: 8.7, 8.8, 10, 15, 16



# Outline for today

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

- Clustering
  - Partitional clustering via $k$-means
  - 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)*
- 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 – Boostrap Aggregation

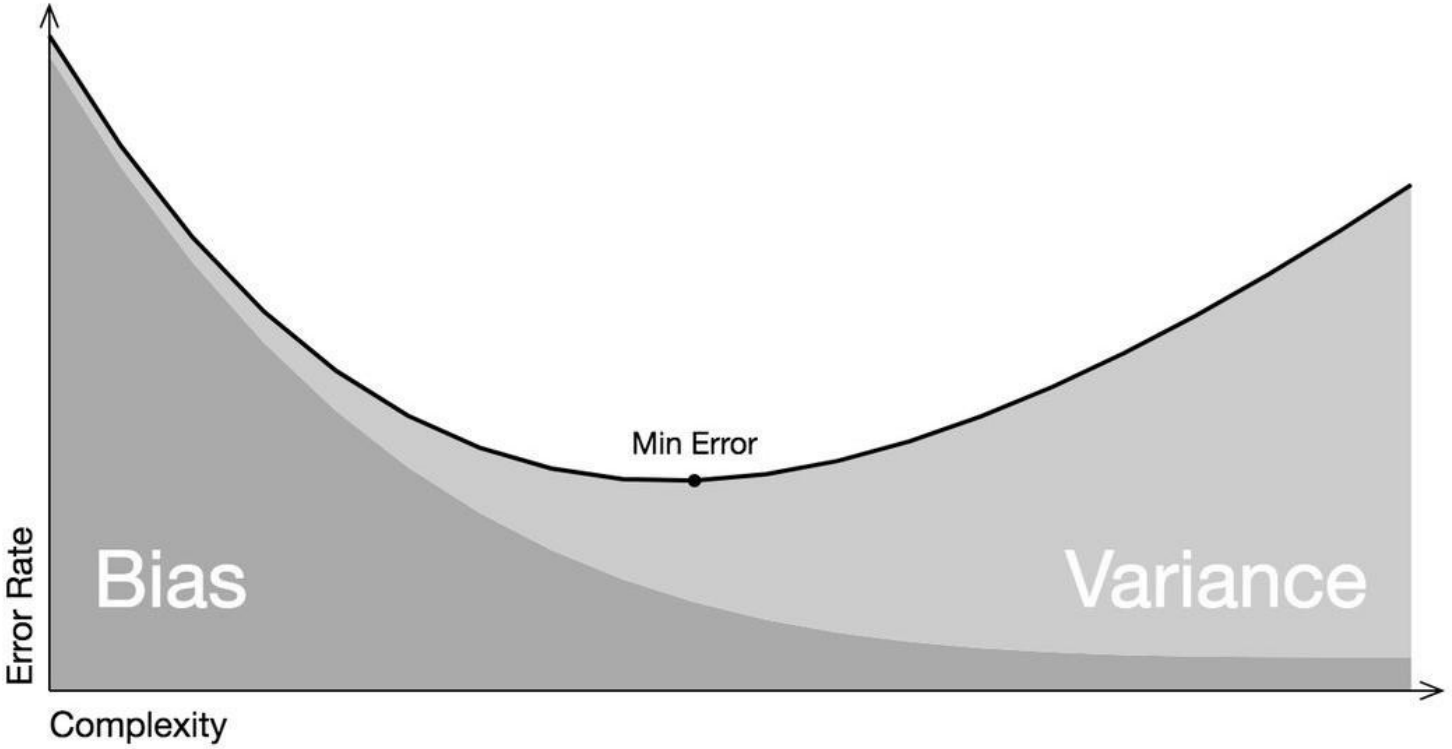
**Model generation**

```
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**

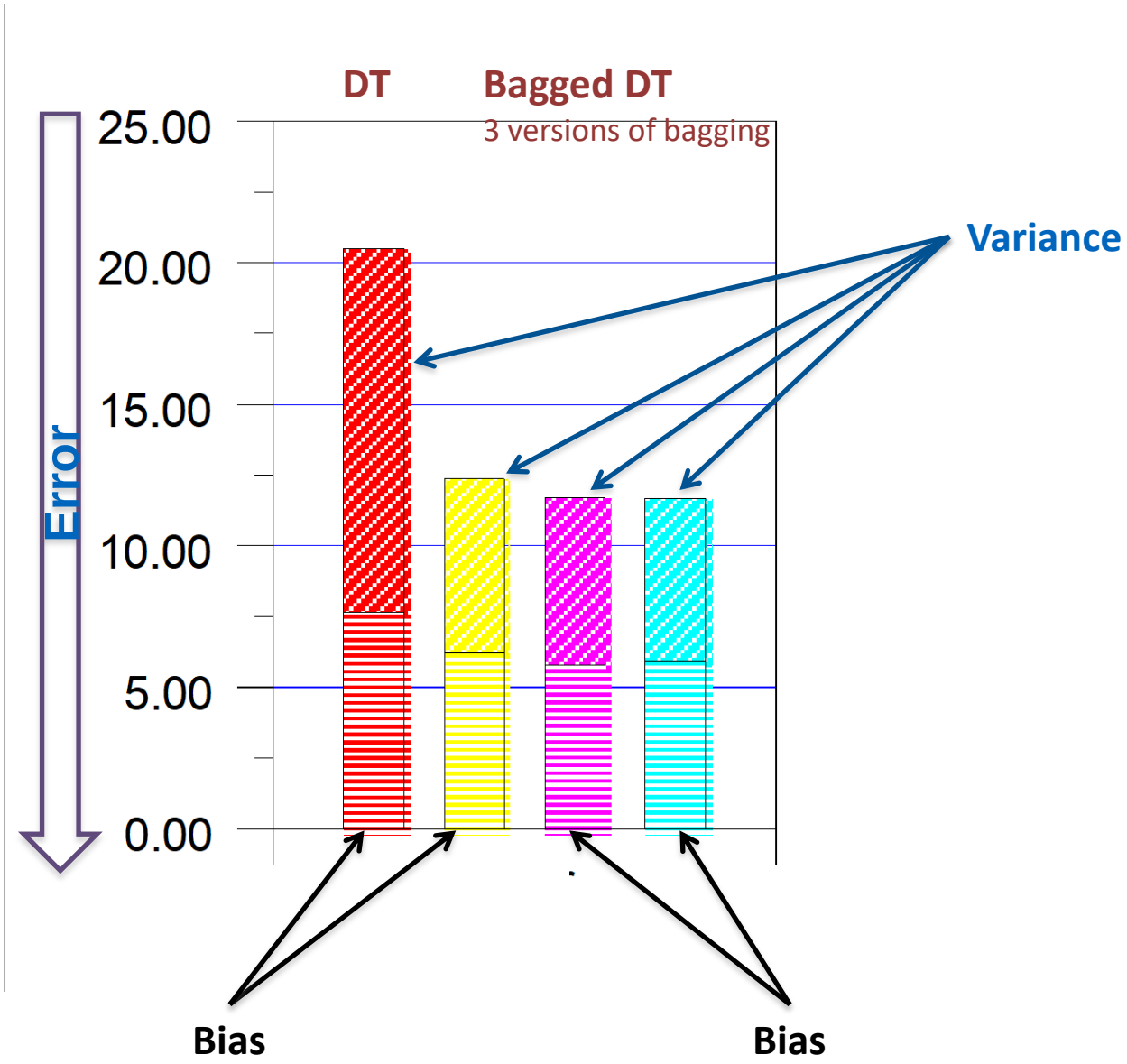
```
For each of the t models:
Predict class of instance using model
Return class that is predicted most often
```

# 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 built 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




**"An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants"**
Eric Bauer & Ron Kohavi, 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 R

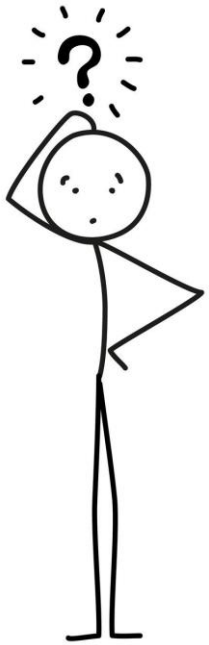
```
library(party)
library(randomForest)
head(readingSkills)

# Create the forest.
output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,
                              data = readingSkills)

# View the forest results.
print(output.forest)

# Importance of each predictor.
print(importance(output.forest,type = 2))
```

What is the difference between bagging and random forests?



# 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

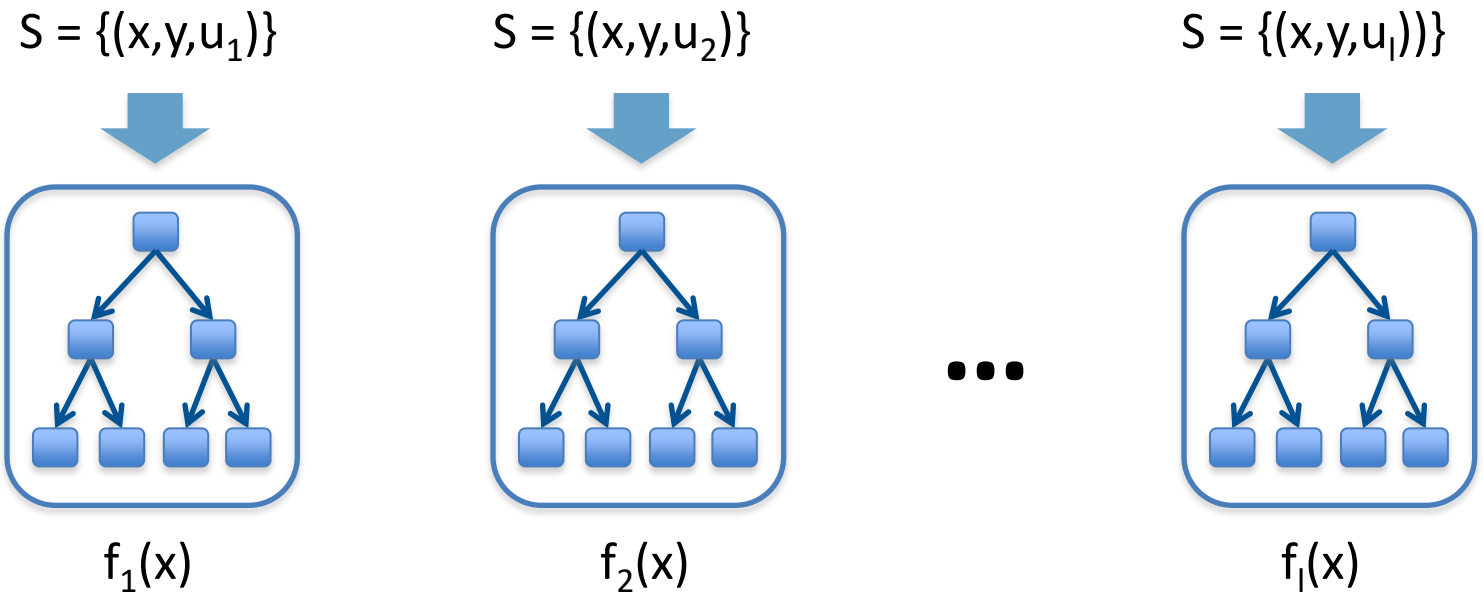
```
Assign equal weight to each training instance
For t iterations:
  Apply learning algorithm (e.g. C4.5) to weighted
  dataset, store resulting model
  Compute model's error e on weighted training dataset
  If e = 0 or e ≥ 0.5:
    Terminate model generation
  For each instance in dataset:
    If classified 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 you can apply boosting *without* weights:
  - Resample data with probability determined by weights
- In practice, boosting sometimes overfits if too many iterations are performed (in contrast to bagging) resulting in poor performance on test data.
- AdaBoost implements a more general idea that has been rediscovered in multiple fields with different names.
  - Multiplicative Weights Update Method
  - Additive Models

# Multiplicative Weights Update Method

A decision maker needs to iteratively decide whose expert advice to follow, say decide “up” or “down” in the prediction of stock performance. The decision maker wants to do as well as the best expert in retrospect.

- Initially, assign all experts same weight $w[i,t] = 1$
- The advice of expert *i* on day *t* results in a gain $w_i^t$ in $[-1, 1]$
- If the expert is wrong we set $w_i^{t+1} = (1-\varepsilon)w_i^t$
- The prediction for step t+1 is the weighted majority of experts, i.e. if the total weight of all experts is at least $\Sigma_i w_i^t/2$

The errors made by this *weighted majority* algorithm (a version of MWUM) are close to those of the best expert.
Weighted majority is a widely used algorithm design idea used in game theory, machine learning, and optimization.

# Additive Models for Numeric Prediciton

- In statistics, boosting is a greedy algorithm for fitting an *additive model*
  - Boosting implements *forward stagewise additive modeling*

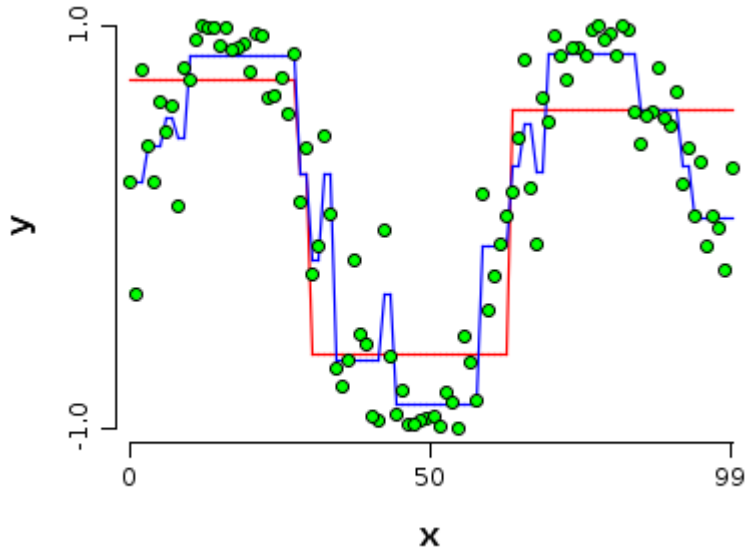
- A *forward stagewise additive model* is a well-known statistical technique.
  - For numeric prediction:
    1. Build simple regression model (e.g., regression tree or one-dimens. regression)
    2. Gather residuals, learn model predicting *residuals* (e.g. another regression tree), and repeat
  - To predict, sum up individual predictions from all regression models.

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

# Additive Models for Numeric Prediction

1. Start with a function $F_0(x) = 0$ and residual $r = y, m = 0$
2. $m = m + 1$
3. Fit a CART regression tree (or simple regression) to $r$ giving $g(x)$.
4. Set $f_m(x) = \epsilon g(x)$
5. Set $F_m(x) = F_{\{m-1\}}(x) + f_m(x),\ r = r - f_m(x)$ and repeat steps 2-5 many times.

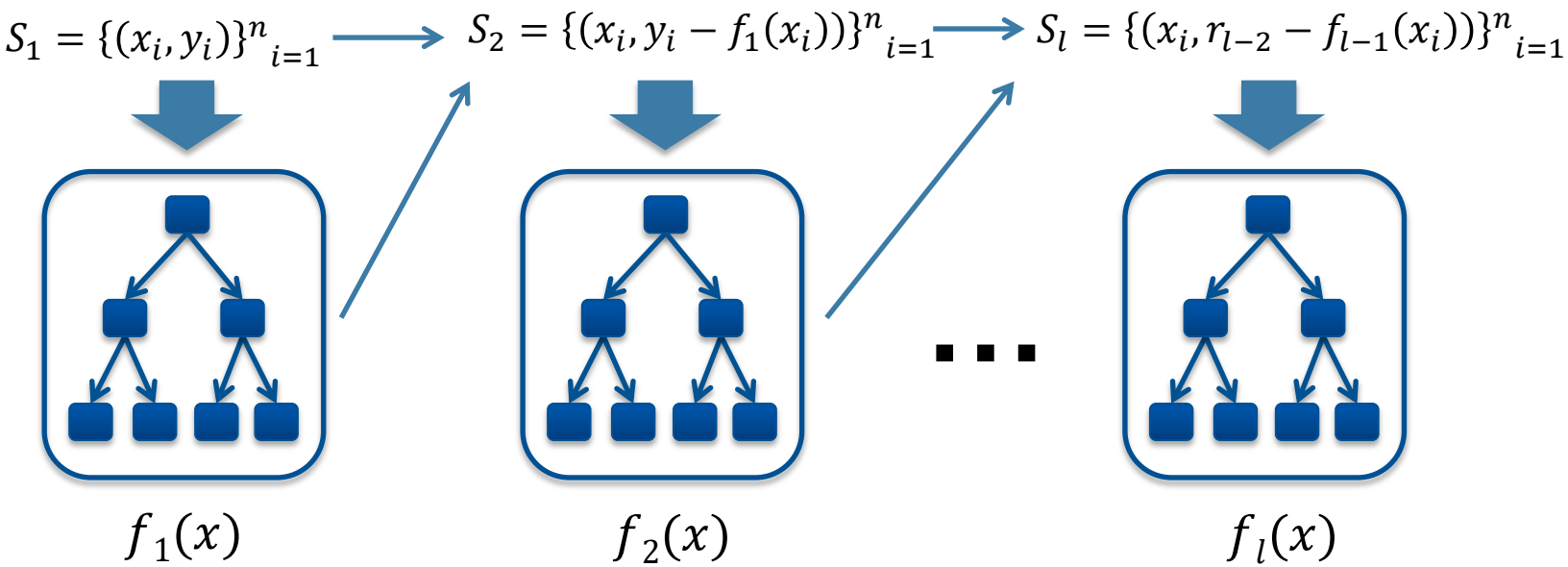
This is typically a shallow regression tree constrained to have only $k = 3$ splits, for example. $0 < \epsilon \leq 1$ is done to slow down the speed of overfitting, it modifies $F_m(x)$ to a smaller amount.



# Gradient Boosting

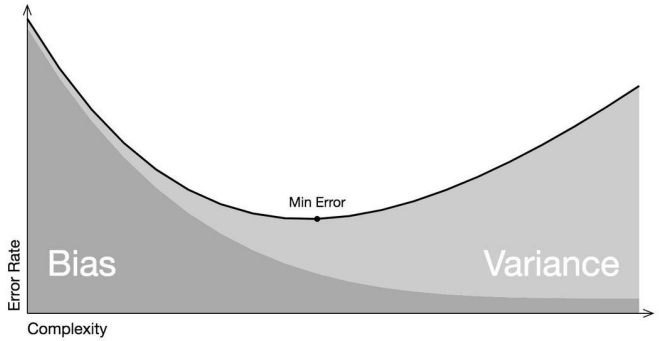
- Gradient Boosting: Instead of weighting the instances as in AdaBoost, the subsequent individual classifiers are trained on the *residual errors* made by the previous individual learners.
- There is a variety of related techniques and names (e.g.: Multiple Additive Regression Tree, Gradient Boosting Models, XGBoost)

$$F_l(x) = f_1(x) + \epsilon_2 f_2(x) + \ldots + \epsilon_l f_l(x)$$



# Summary

**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**

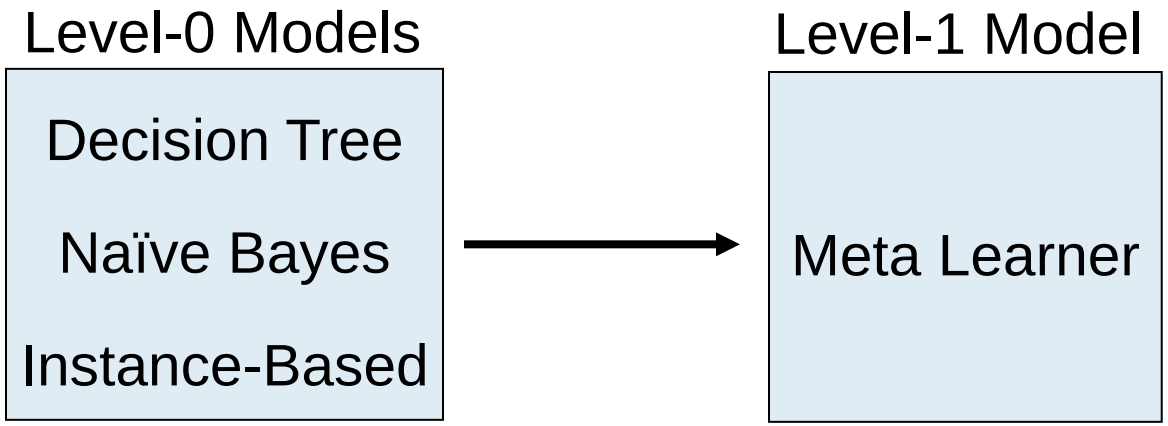
- Low-variance models are simple with high bias
- Boosting trains sequence of models on residual error ➔ sum of simple models is accurate

# Bagging vs. Boosting

| | Bagging | Boosting |
|---|---|---|
| **Sequent** | Two-step | Sequential |
| **Partitioning data into subsets** | Random | Give misclassified cases a heavier weight |
| **Sampling method** | Sampling with replacement | Sampling without replacement |
| **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** | Weighted average or majority vote | Majority vote |

# Stacking and Meta-Learning

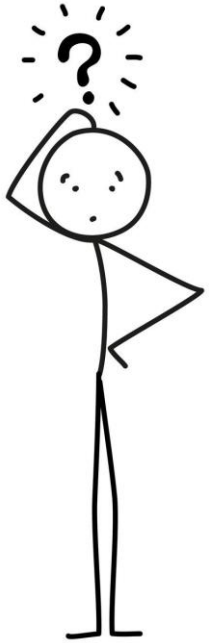
- In stacking, the predictions from heterogeneous classifiers are used as input into a meta-learner, which attempts to combine the predictions to create a final best predicted classification
  - Learn which learning algorithms are good (instead of voting)
  - Combine learning algorithms intelligently



# Meta Learning

- Holdout part of the training set
- Use remaining data for training level-0 methods
- Use holdout data to train level-1 learning
- Retrain level-0 algorithms with all the data
- 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

How does AdaBoost work and why does it reduce the bias?



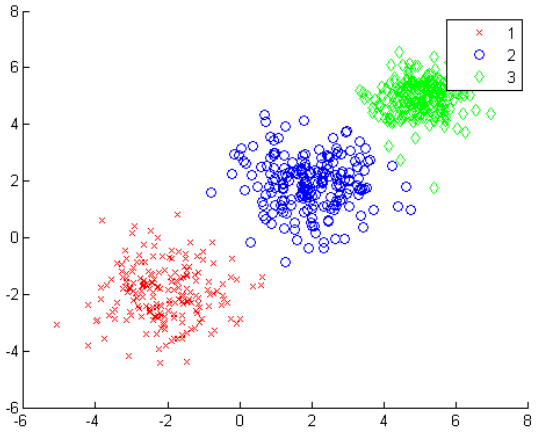
# Outline for today

- 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 $n$ $p$-dimensional data items.
- 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
      ➔intra-cluster similarity is maximized
    - items from different clusters are different
      ➔inter-cluster similarity is minimized

# Definition

- Given a database $D = \{t_1, t_2, \ldots, t_n\}$ of tuples and an integer value $k$, the clustering problem is to define a mapping $f: D \rightarrow \{1, .., k\}$ where each $t_i$ is assigned to one cluster

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

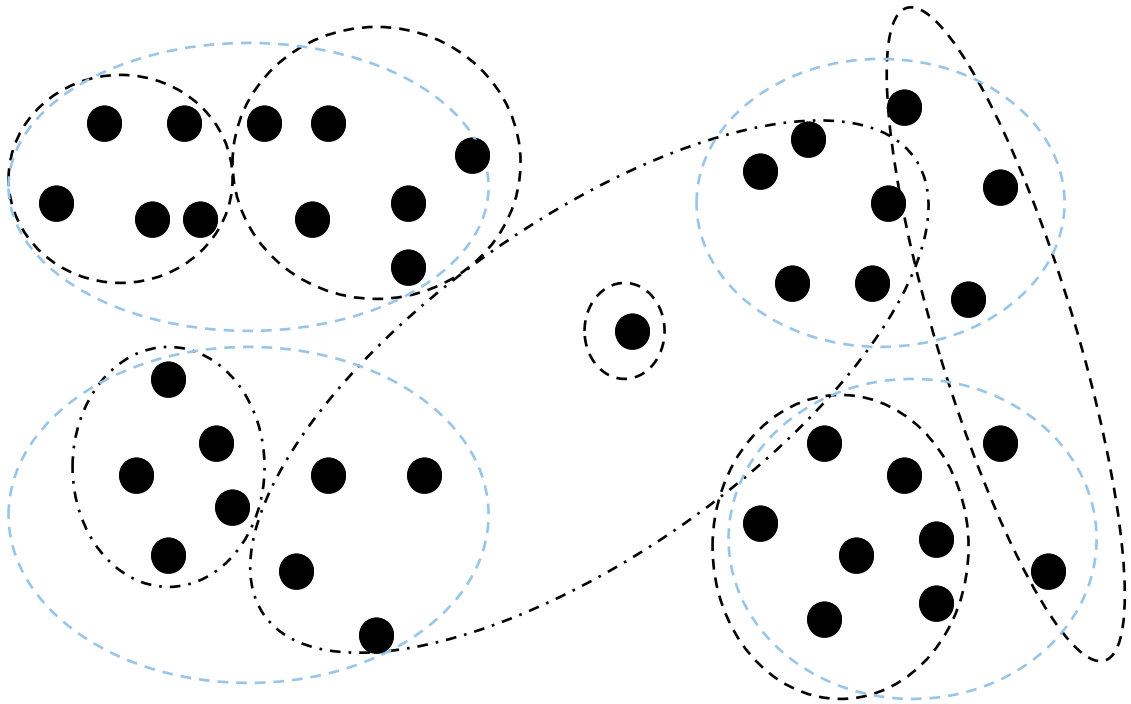
- A cluster, $K_j$, contains precisely those tuples mapped to it

- 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



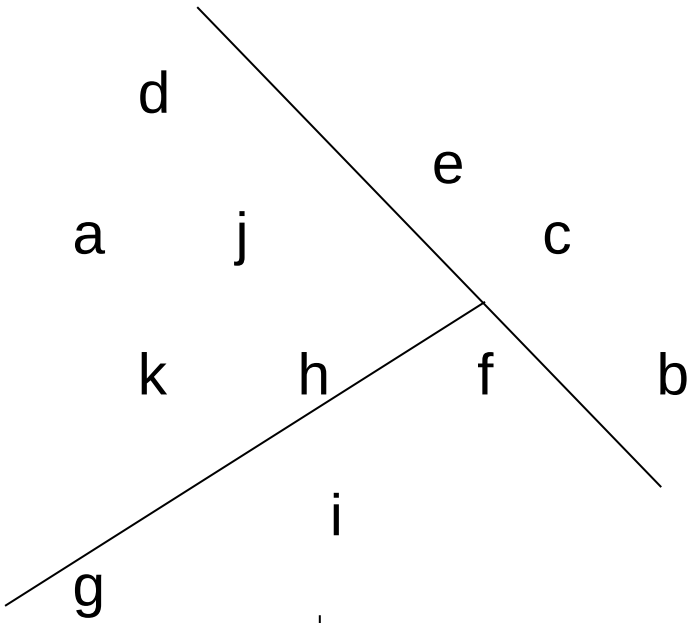
Geographic Distance Based

Size Based

# Clustering Methods

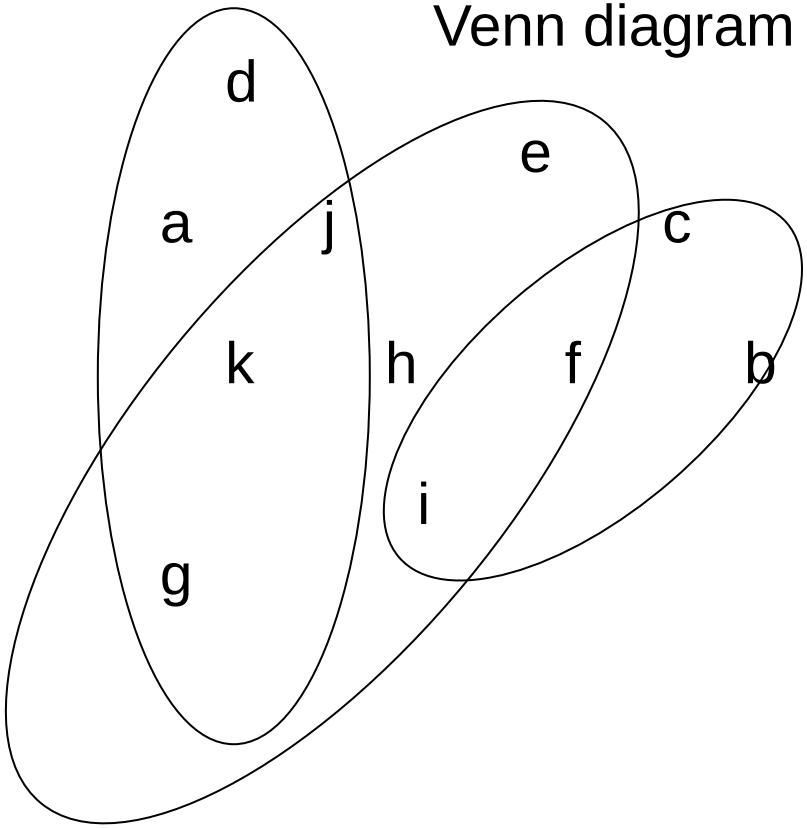
- **Many different methods and algorithms:**
  - For numeric and/or nominal data
  - Deterministic vs. probabilistic
  - Partitional vs. overlapping
  - Hierarchical vs. flat

# Clustering

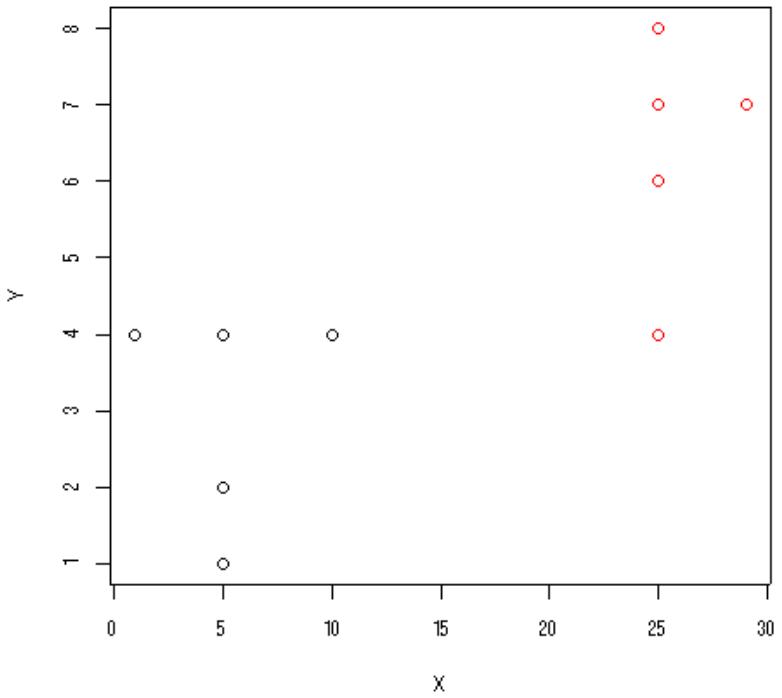


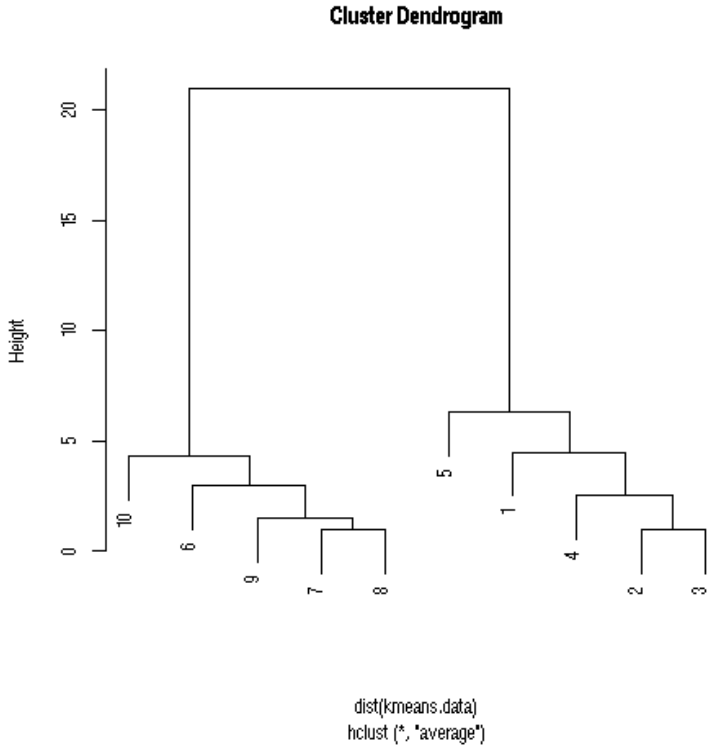

| | 1 | 2 | 3 |
|---|---|---|---|
| a | 0.4 | 0.1 | 0.5 |
| b | 0.1 | 0.8 | 0.1 |
| c | 0.3 | 0.3 | 0.4 |
| d | 0.7 | 0.2 | 0.1 |
| ... | | | |

Venn diagram



# Other Visualizations



# Clustering Issues

- Interpreting results
- Evaluating results
- Outlier handling
- Number of clusters
- Scalability of algorithms

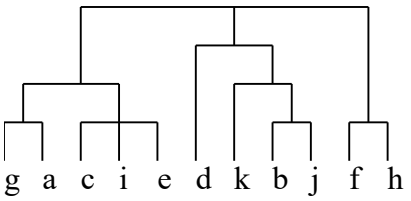
# Clustering Evaluation

- Manual inspection
- Benchmarking on existing labels
- Cluster quality measures
  - distance measures
  - high similarity within a cluster, low across clusters

# 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

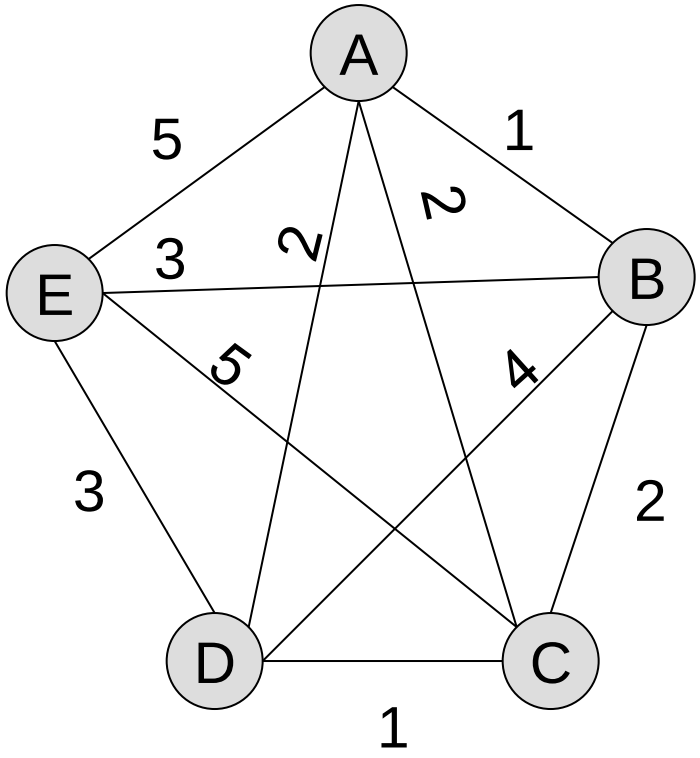


# Dissimilarity Matrices

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

| | **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 |

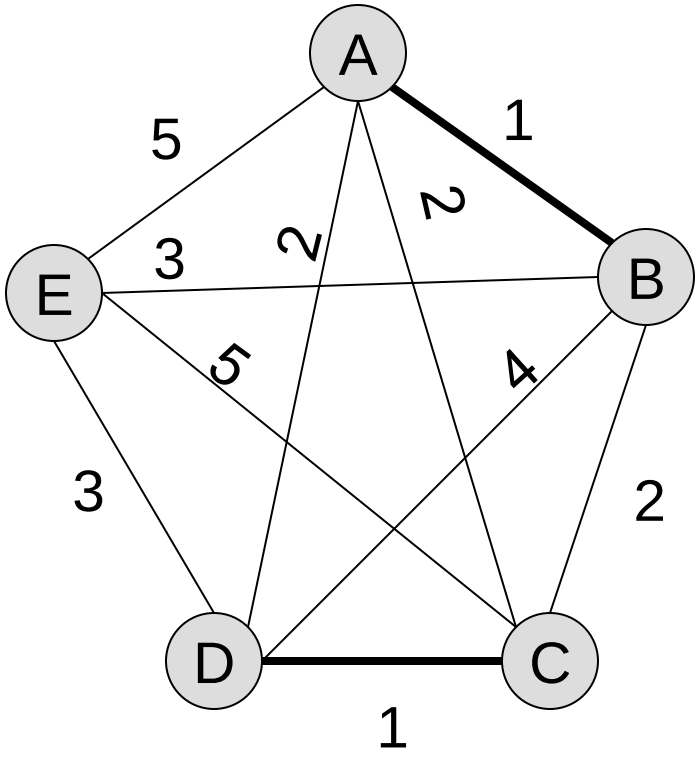
# Graph Perspective



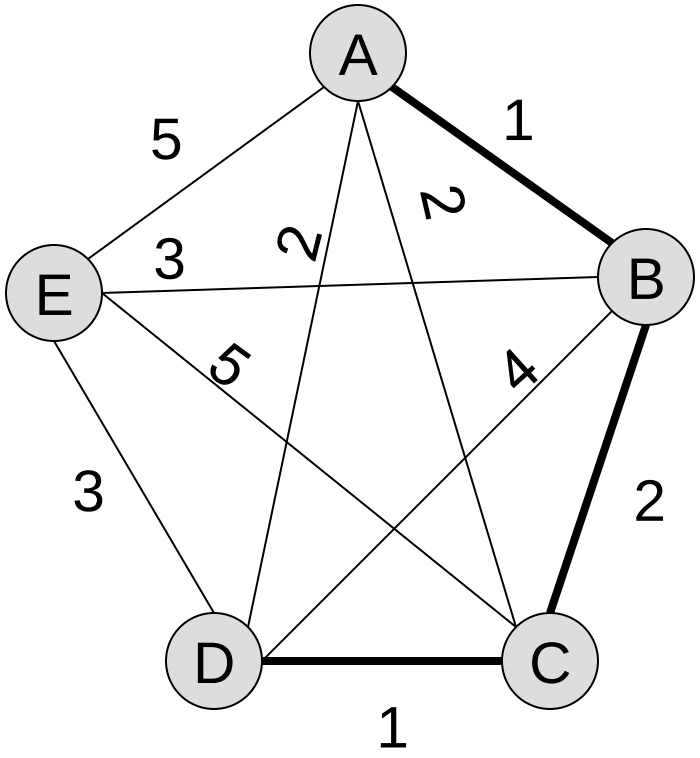
# MST Algorithm

- 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

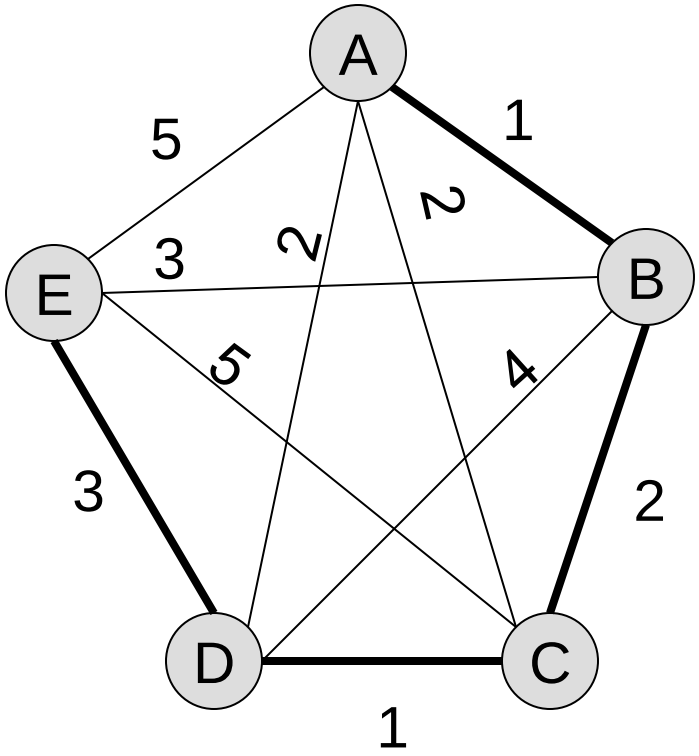
# Kruskal's Algorithm for Finding the MST



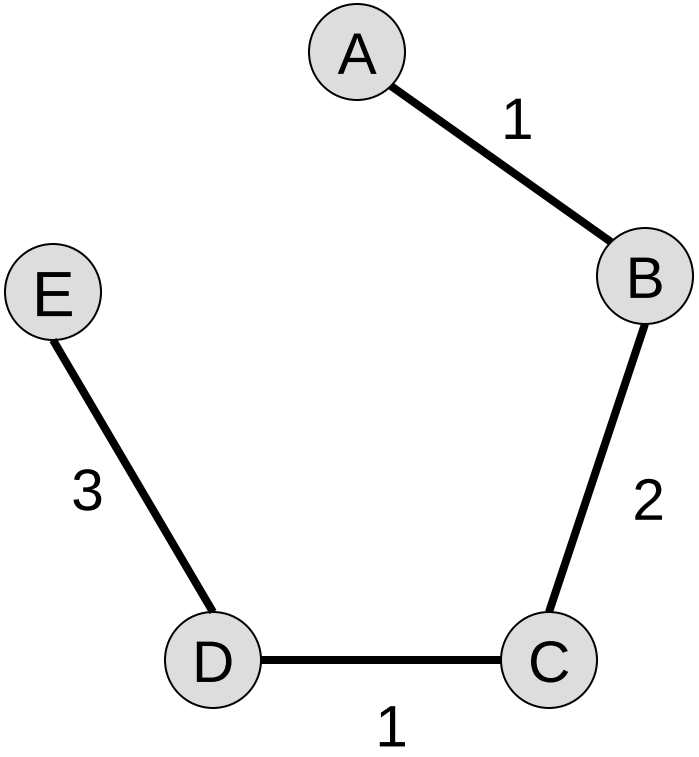
# Kruskal's Algorithm



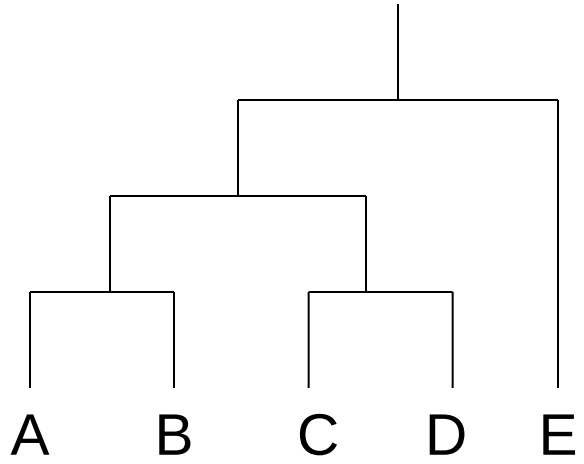
# Kruskal's Algorithm



# MST Clustering



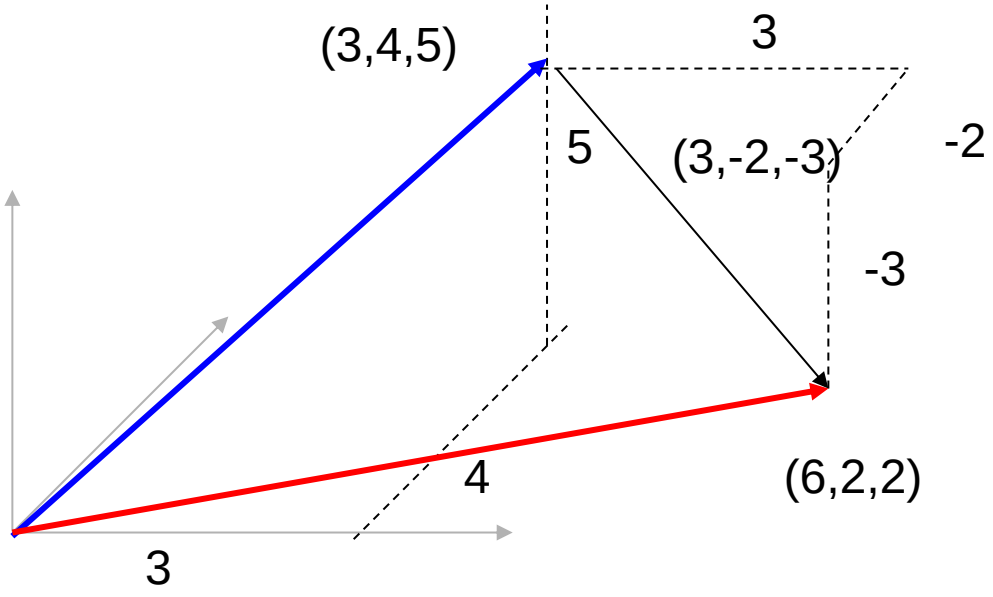

Dendogram



# Vectors in Euclidean Space

Operations tell us how to get from origin to the result of the vector operations

(3,4,5) + (3,-2,-3) = (6,2,2)



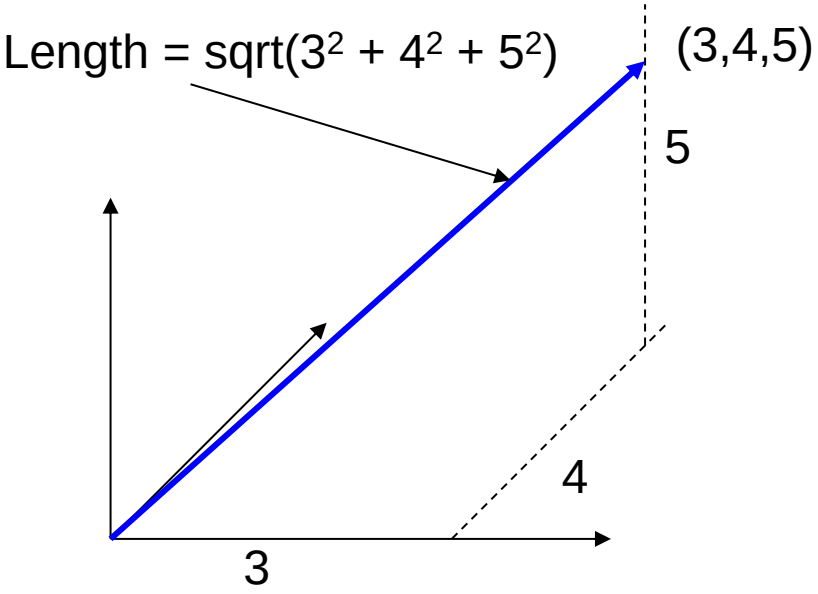
# Vector Norm

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

- Represented as

$$\left\|\begin{bmatrix} a & b & \dots \end{bmatrix}\right\| = \sqrt{a^2 + b^2 + \dots^2}$$

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



# Distance Measures as Vector Norms

- Each instance $i$ lives in $p$-dimensional space
  - $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$ describing the Cartesian coordinates

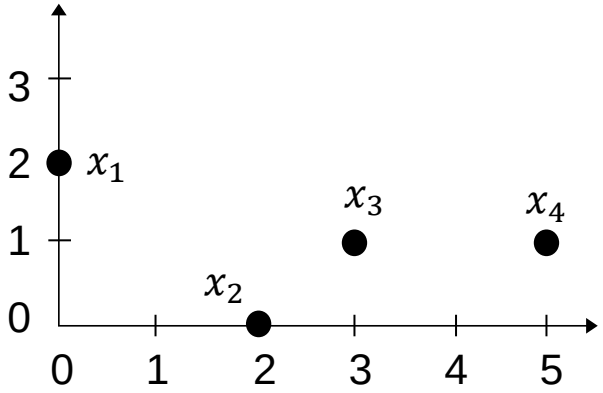
- Euclidean distance (or $L^2$ distance) between two instances

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

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

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

# 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

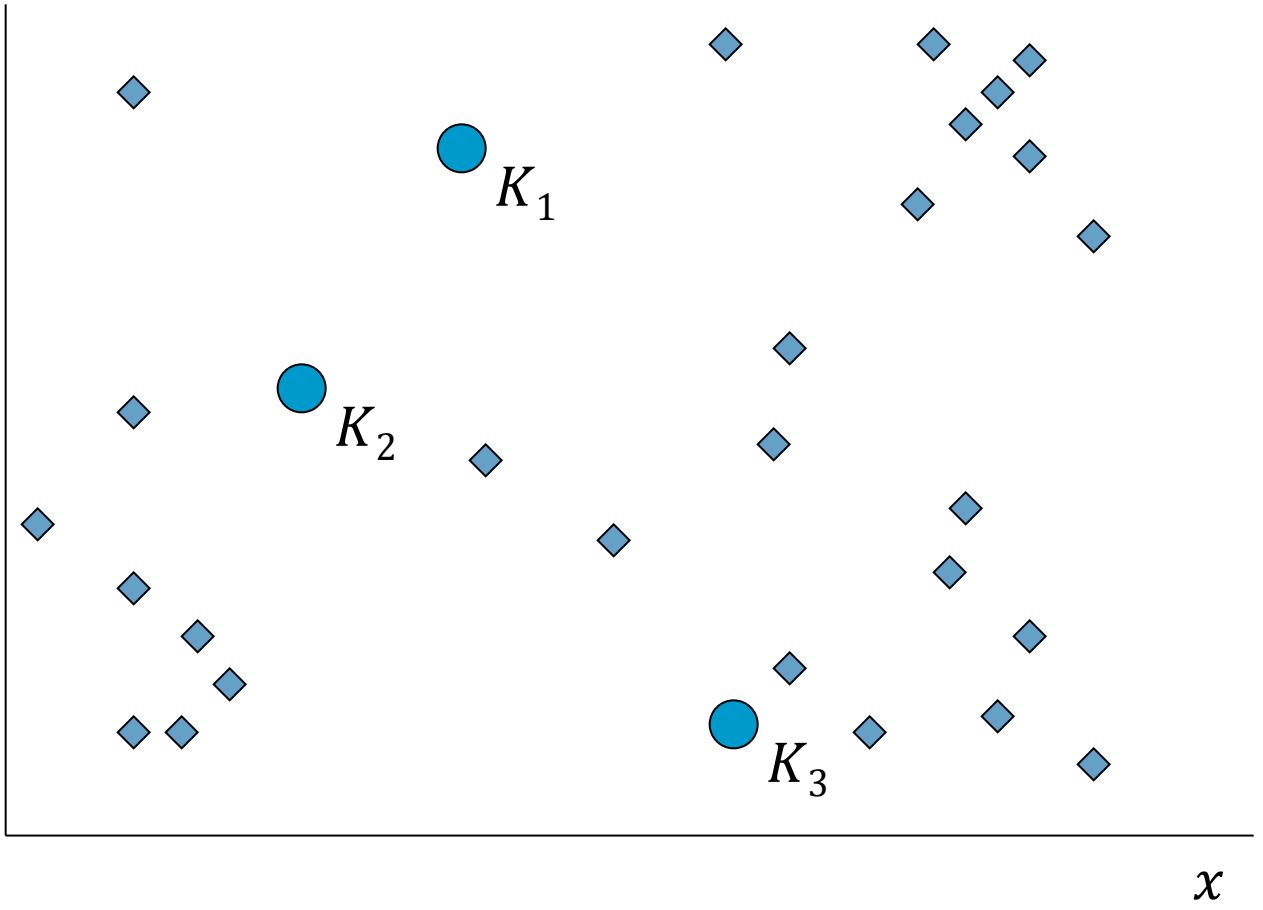
# $K$-Means Clustering

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)

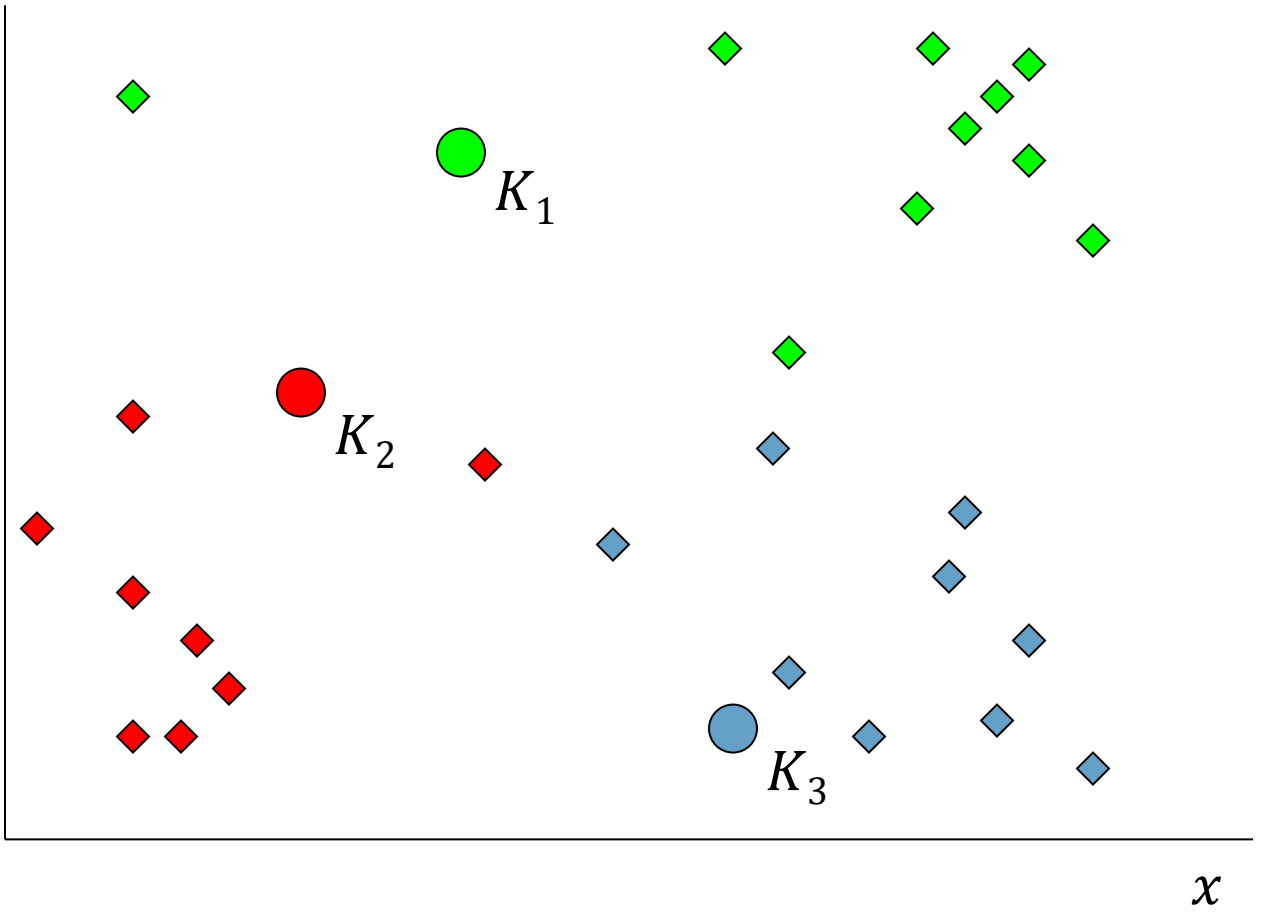
# $K$-Means: Example, Step 1 - Two Dimensions

Pick 3 initial cluster centers (randomly)



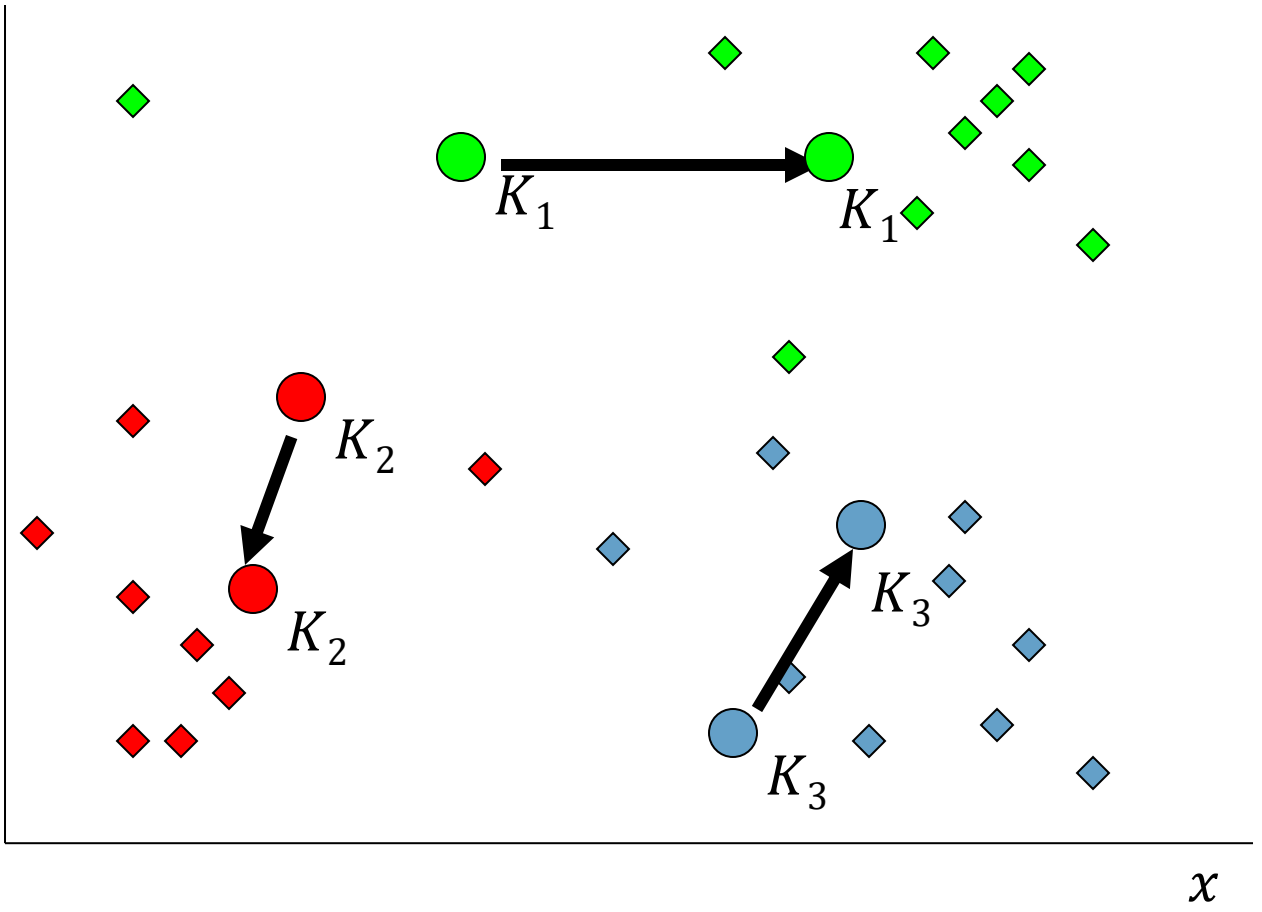
# $K$-Means: Example, Step 2

Assign each point to the closest cluster center



# $K$-Means: Example, Step 3

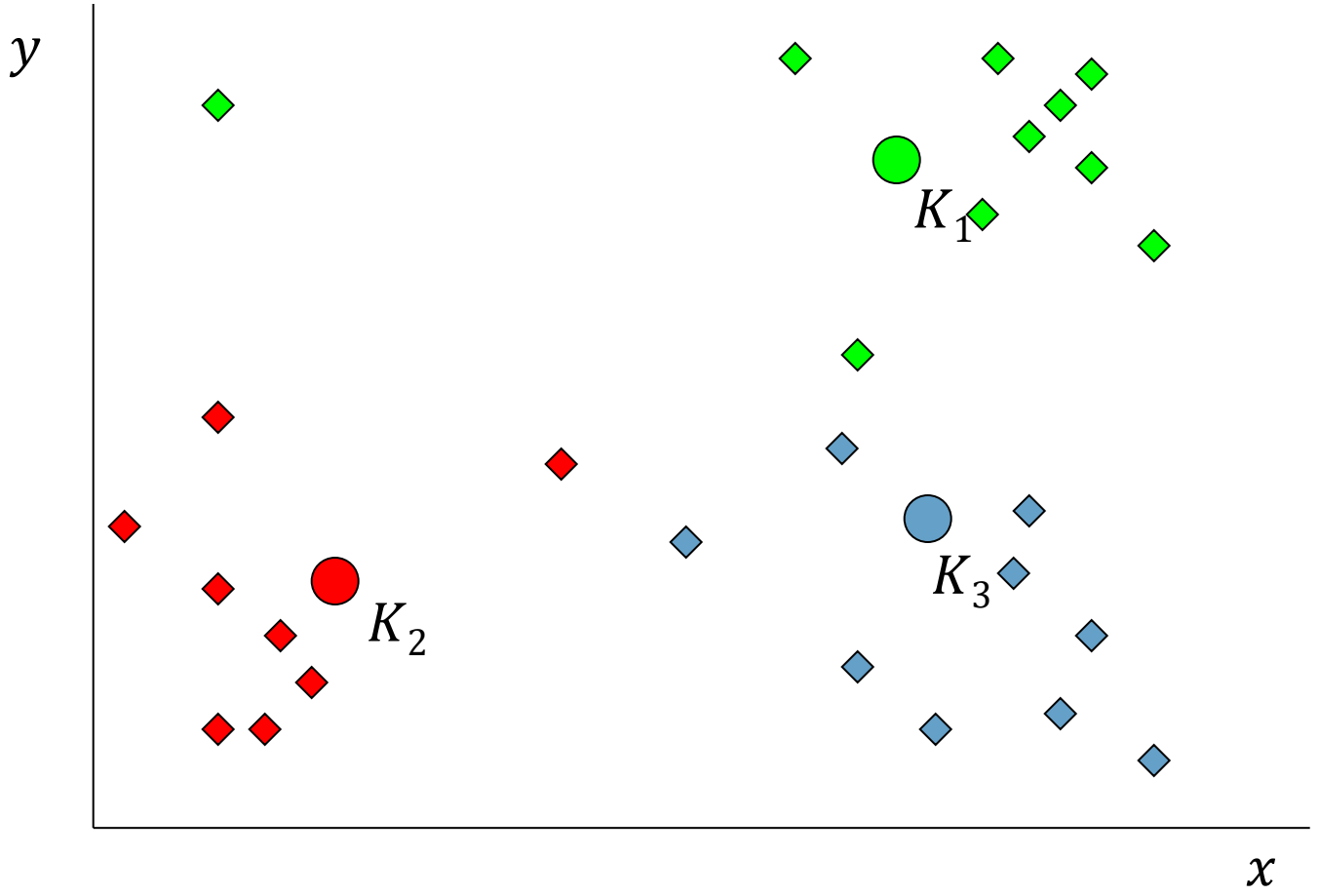
Move
each cluster
center
to the mean
of each cluster



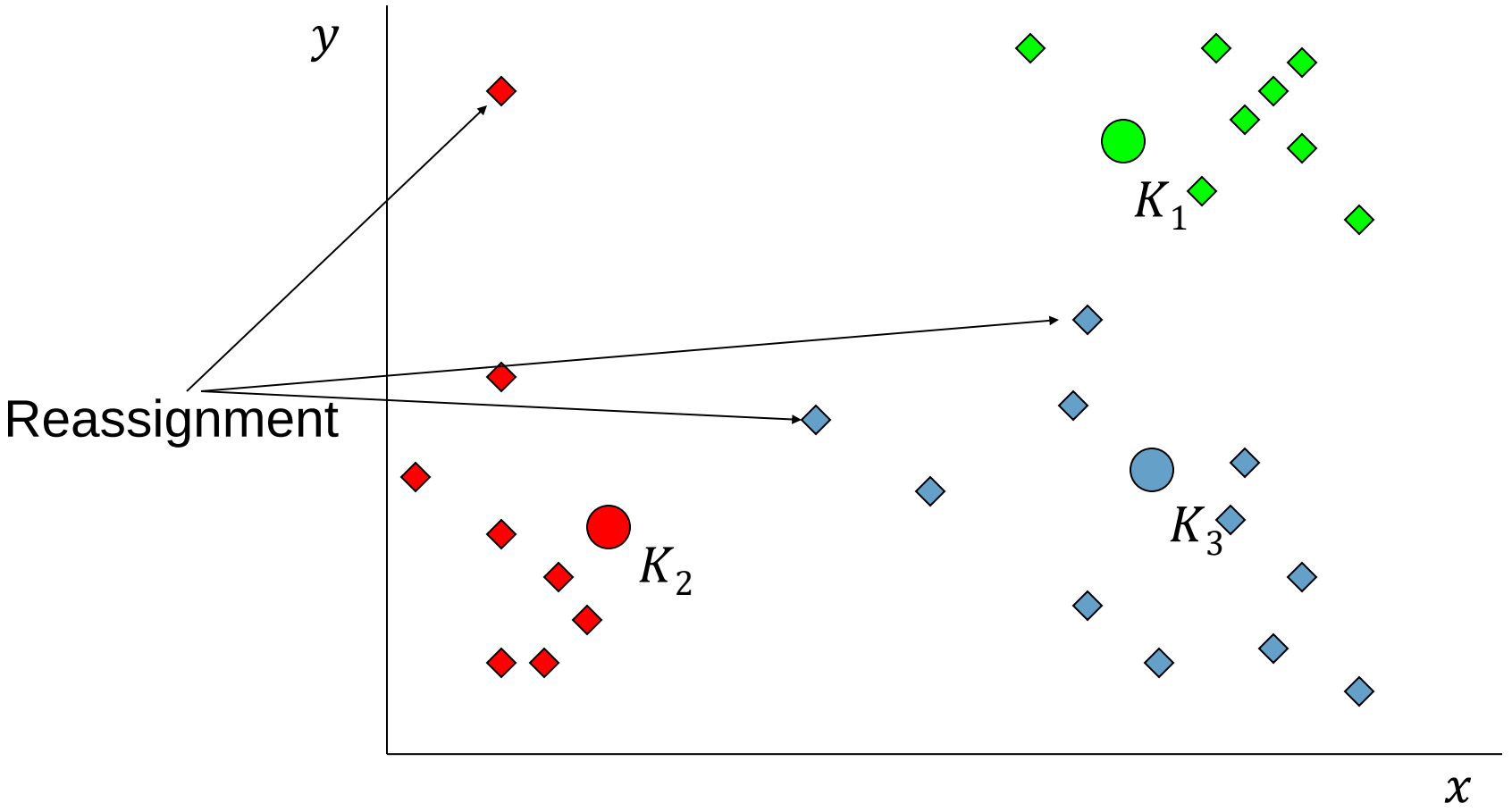
# $K$-Means: Example, Step 4

Reassign points closest to a different new cluster center

***Q: Which points are reassigned ?***

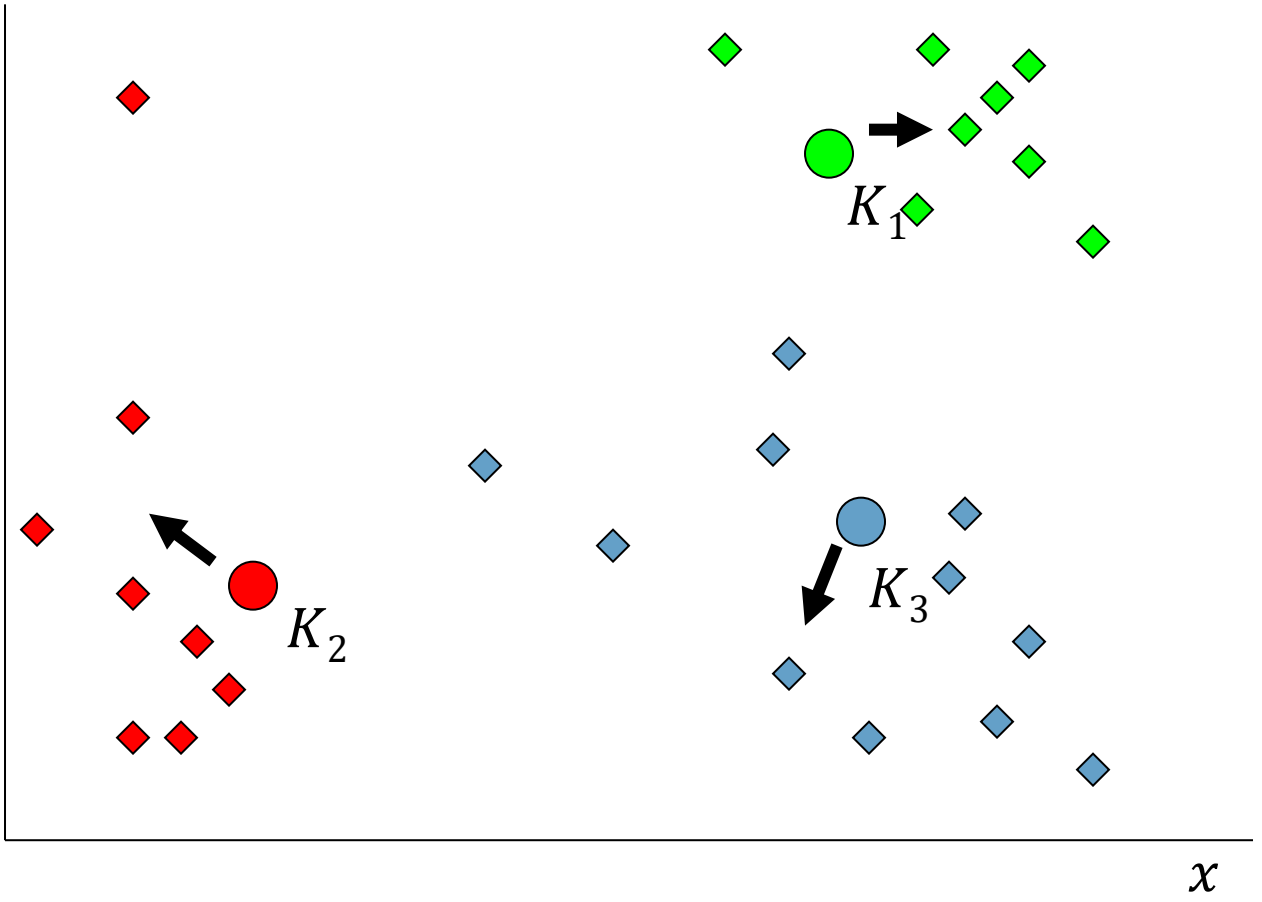


# $K$-Means: Example, Step 4 …



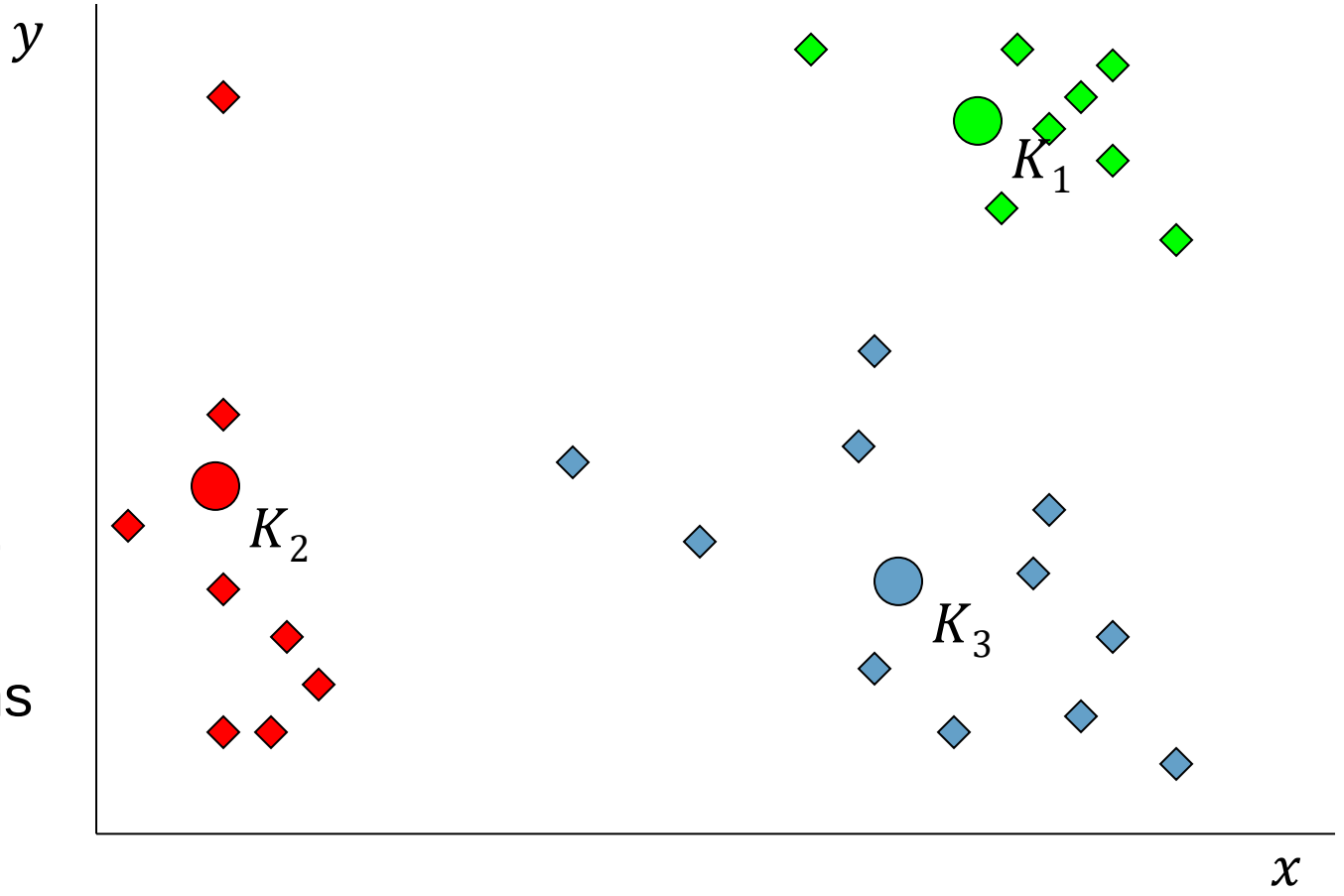
# $K$-Means: Example, Step 4b

re-compute
cluster means



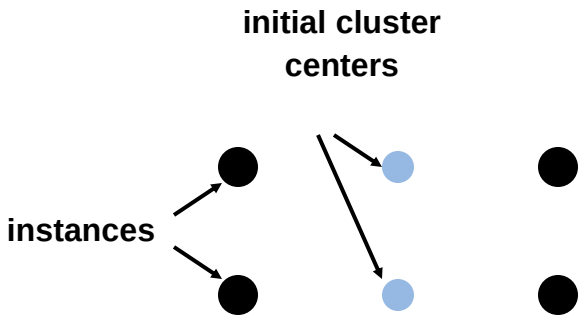
# $K$-Means: Example, Step 5

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

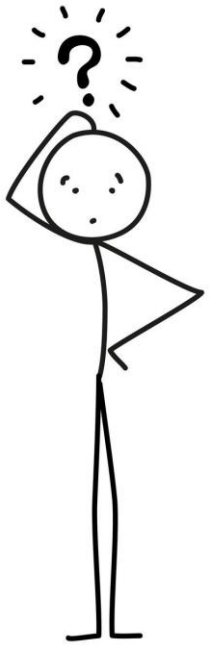
- Simple, understandable
- Items automatically assigned to clusters

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

Why is clustering described as unsupervised learning?



# 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.

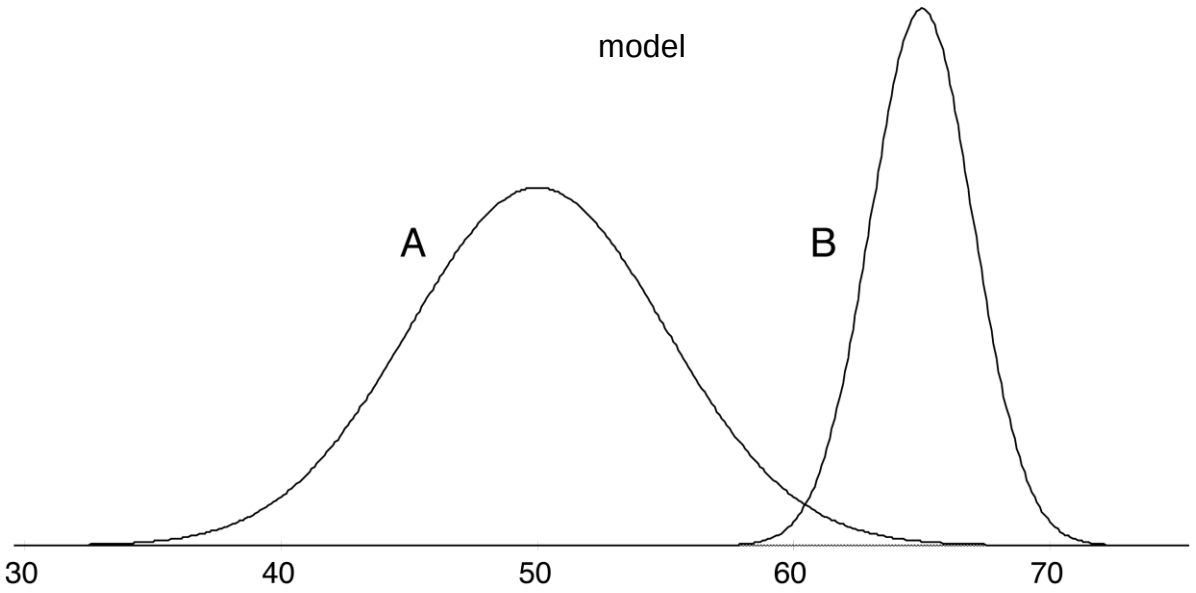
# Probability-Based Clustering: Basics

- Simplest case: A single numeric attribute and two clusters A and B each represented by a normal distribution
  - Parameters for A: $\mu_A$ - mean, $\sigma_A$ - standard dev.
  - Parameters for B: $\mu_B$ - mean, $\sigma_B$ - 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 | | |




$\mu_A$=50, $\sigma_A$ =5, $p_A$=0.6 $\mu_B$=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.
  - $\mu_A, \sigma_A, \mu_B, \sigma_B, \Pr[A]$
  - If data is labeled, easy
  - $\Pr[A]$ is just the proportion of instances in cluster $A$

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

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

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

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

- The denominator $\Pr[x]$ will disappear:
  - divide by $\Pr[A|x] + \Pr[B|x]$

# 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

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

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

# 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.