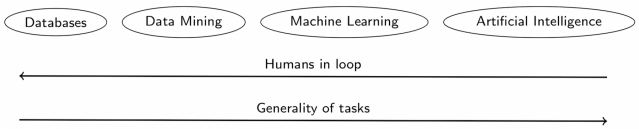
Machine Learning

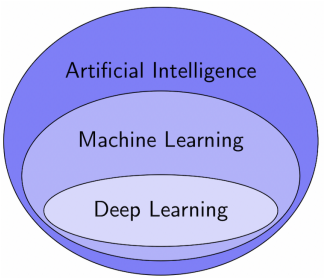
Wednesday, 3 March 2021 10:50 AM

Introduction

Data Mining vs. Machine Learning



- Data Mining often viewed as closer to databases: automatically extract useful knowledge from large data sets
- Machine Learning often viewed as closer to AI: use computers to automatically detect patterns in data



Fundamentals

- Basic assumptions
  - Training and test data need to be related in some way
  - Training and test data are (approximately) independent and identically distributed (IID)
- Memorization vs learning:
  - You can do well on training data by memorizing it (practice on the exam)
  - You have only learned if you can do well in new situations (the actual exam)
- Parameters vs hyper-parameters
  - We estimate parameters by training a model
  - We tune hyper-parameters using a validation score
- Golden Rule of Machine Learning - the test data cannot influence the training phase in any way, e.g.
  - After tuning hyperparameters on training/validation sets, the selected model is usually retrained on the whole training/validation data, with the test data untouched all the time
  - Imputation on training and test/validation sets are done separately
- No free lunch theorem - there is no best model achieving the best generalization error for every problem

Preprocessing

Data cleaning

- Missing/Incomplete values lacking attribute values, certain attributes, or containing only aggregate data
  - Problems
    - Missing completely at random (MCAR)
      - Completely unrelated to the data
      - Potential problem? Small sample size
    - Missing at random (MAR)
      - The fact the data are missing is related not to the missing attribute, but to some other data in the data set
      - Potential problem? Bias due to row-wise deletion
    - Missing not at random (MNAR)
      - There is a reason the data are missing and it is related to the attribute itself
      - Potential problem? Bias due to row-wise deletion
  - Solutions (Imputation)
    - Usually ignore the record if the class label is missing
    - A global constant

X

sunny	warm	Mon	May	...
cloudy	?	?	July	...
sunny	cold	?	?	...
...	...	...	...	...
overcast	cold	Sat	June	...

→

X'

sunny	warm	Mon	May	...
cloudy	missing	missing	July	...
sunny	cold	missing	missing	...
...	...	...	...	...
overcast	cold	Sat	June	...

    - The attribute mean changes relationship with other variables -> bias in data

X

12	2	22	38	...
11	?	?	90	...
2	23	?	?	...
...	...	...	...	...
9	11	54	23	...

→

X'

12	2	22	37	...
11	12	38	90	...
2	23	38	30	...
...	...	...	...	...
9	11	54	23	...

    - The attribute mean of the samples belonging to the same class might change relationship with other variables in other classes -> bias in data

$X|Y$

12	2	22	38	...	1
11	?	?	90	...	0
2	23	?	?	...	1
...	...	...	...	...	...
9	11	54	23	...	0

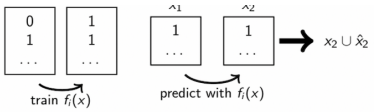
→

$X'|Y$

12	2	22	38	...	1
11	11	54	90	...	0
2	23	22	38	...	1
...	...	...	...	...	...
9	11	54	23	...	0

    - The most probable value can be inferred by machine learning algorithms

x_1 x_2 $\hat{x}_3$

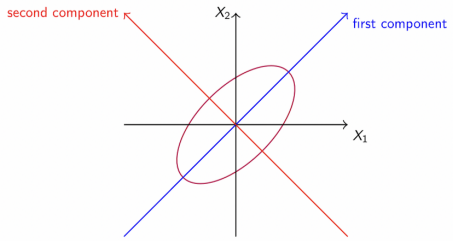


- Matrix factorization is a collaborative filtering method to predict the missing values using the latent features, e.g. singular value decomposition (SVD)
  - Decompose the data matrix M such that M = UΣV∗
  - Create imputed matrix M ′ by multiplying U × Σ × V ∗
- Expectation Maximization
  - Use other variables to impute the values (Expectation)
  - Check if value is most probable (Maximization)

- Noisy data containing noise, errors, or outliers
  - Binning: first sort data and partition into (equal-frequency) bins, then one can smooth by bin means, smooth by bin median, smooth by bin boundaries, etc.
  - Clustering
- Inconsistent data containing discrepancies in codes or names
- Intentionally wrong data, e.g. there are a lot of pictures with a GPS location just a bit west of Africa

Data reduction

- Dimensionality reduction - "curse of dimensionality": density and distance between points, which is critical to clustering, outlier analysis, classification, regression becomes less meaningful
  - Principal Component Analysis – PCA
    - Find a projection that captures the largest amount of variation in data
    - The original data are projected onto a much smaller space, resulting in dimensionality reduction
    - We find the eigenvectors of the covariance matrix, and these eigenvectors define the new space



    - Given N data vectors from n-dimensions, find k ≤ n orthogonal vectors (principal components) that can be best used to represent data
      - Normalize input data: Each attribute falls within the same range
      - Compute k orthonormal (unit) vectors, i.e. principal components
      - Each input data (vector) is a linear combination of the k principal component vectors
      - The principal components are sorted in order of decreasing “significance” or strength Since the components are sorted, the size of the data can be reduced by eliminating the weak components, i.e. those with low variance (i.e. using the strongest principal components, it is possible to reconstruct a good approximation of the original data)
  - Feature selection
    - Correlation
      - For nominal data, given two attributes A and B with values a1,...,ac and b1,...,br the correlation can be calculated using the χ2 test:

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$$

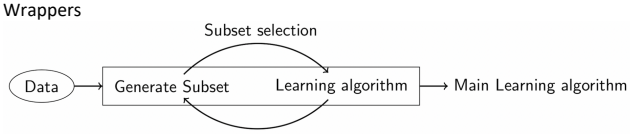
        - oij is the actual frequency of the event (ai , bj)
        - eij is the expected frequency (n is the number of instances)
      - Numerical data can be compared using Pearson’s correlation coefficient:

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(a_i b_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}$$

        where means are $\bar{A}$ and $\bar{B}$, number of instances is n, and standard deviations are σA and σB
    - Relief is a feature selection algorithm that rewards the differences and penalize the similarities

Input: Data set with N_a attributes and N_i instances that bellong to one of two classes, and parameter $N_r < N_i$
First normalize the data
Create a weight vector W with one weight $w_i \in W$ for each attribute
Initialize the weights to 0
for $j \in 1 \ldots N_r$ **do**
  Randomly select instance $X = [x_1, \ldots, x_n]$
  Choose instance $H = [h_1, \ldots, h_n]$ as the closest neighbour of X in the same class (*nearHit*)
  Choose instance $M = [m_1, \ldots, m_n]$ as the closest neighbour of X in the other class (*nearMiss*)
  for $i \in 1 \ldots N_a$ **do**
    $w_i = w_i - (x_i - h_i)^2 + (x_i - m_i)^2$
  end
end
for $i \in 1 \ldots N_a$ **do**
  $w_i = \frac{w_i}{N_r}$
end

    - Wrappers



      - generate a subset of the features and evaluate the performance of the classifier on the subset
      - Add or remove attributes from the subset and see if the performance of the classifier improves
- Numerosity reduction

    - Regression and Log-Linear Models
    - Histograms, clustering, sampling
    - Data cube aggregation Data compression
- Data compression

Transformation and discretization

- Normalization
    - Min-max normalization to the range of (new_minA, new_maxA)

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

    - Z-score normalization – mean μ, standard deviation σ

$$v' = \frac{v - \mu_A}{\sigma_A}$$

    - Normalization by decimal scaling

$$v' = \frac{v}{10^j}$$

Where j is the smallest integer such that $Max(|v'|) < 1$

- Imbalanced data
    - Randomly under-sample the majority class
    - Randomly over-sample the minority class
- Hierarchy generation
    - Problems
        - Nominal – values from an unordered set, e.g. colour
        - Ordinal – values from an ordered set, e.g. rank
        - Numeric – real numbers, e.g. integers or reals
    - Solutions
        - Equal-width (distance) partitioning, sensitive to outliers and skewed data
        - Equal-depth (frequency) partitioning, tricky handling categorical data

Classification

Decision Trees

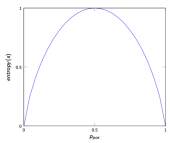
Models where a nested sequence of “if-else” decisions based on the features (splitting rules) and a class label as a return value at the end of each sequence

- Train a decision stump - a simple decision tree with one splitting rule based on thresholding one feature
    a. Define a 'score' for the rule, e.g. "Information Gain" is the most common score in practice to decrease the entropy (the degree of disorder or randomness in the system)

$$information\ gain = \underbrace{entropy(y)}_{\text{entropy before split}} - \underbrace{\frac{\overbrace{n_{yes}}^{\text{number of examples satisfying rule}}}{n}}_{\text{entropy examples satisfying rule}} entropy(y_{yes}) - \frac{n_{no}}{n} entropy(y_{no})$$

with

$$entropy(s) = -p_{s_{pos}} \log_2 p_{s_{pos}} - p_{s_{neg}} \log_2 p_{s_{neg}}$$



    b. Search for the rule with the best scores
- Greedy Recursive Splitting
    a. Fit a decision stump to each leaf's data
    b. Add these stumps to the tree
- Stop when
    - Clean split (leaves) with only one class label
    - User-defined maximum depth
- Prune when the information gain is low for several levels and then becomes high again

Reduced Error Pruning

Input: decision Tree T; labelled data D
Output: Pruned tree T'
for *every internal node N of T, starting from the bottom* **do**
  $T_N \leftarrow$ subtree of T rooted at N;
  $D_N \leftarrow \{x \in D | x$ is covered by $N\}$;
  if *accuracy of T_N over D_N is worse than majority class in D_N* **then**
    replace T_N in T by a leaf labelled with the majority class in D_N;
  end
end
return *pruned version T*

Ensemble

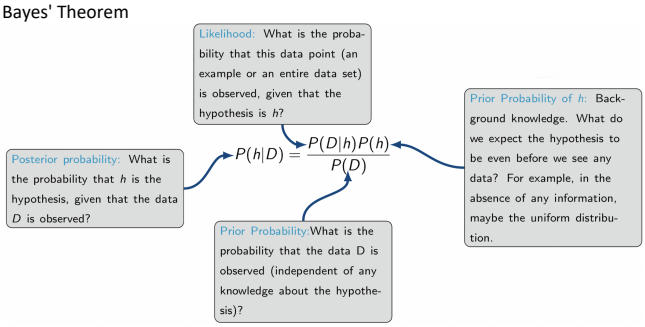
- Definition - an ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples
    - The individual classifiers disagree with one another
    - The individual classifiers make uncorrelated errors at rates less than random
- Bootstrapping to reduce the variance of an individual classifier
    - Construct ensembles
        - Manipulating the training set for unstable learning algorithms
            - Resample with replacement (bagging)
            - Resample without replacement (Cross-validated committees)
            - Resample with adjusted weights (boosting -> overfitting)
            - Resample with penalization (gradient to take steps in the opposite direction -> underfitting)
        - Manipulating the input features for correlated input features, e.g. only randomly choose m out of p features (random forest, e.g. $m \approx \sqrt{p}$)
        - Manipulating the output features, e.g. Error Correcting Output Coding (ECOC) for using binary classification models on multi-class classification tasks

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	15
A	0	0	0	1	0	0	1	0	1	1	0	1	1	1
B	0	0	1	0	0	1	0	1	0	1	1	0	1	1
C	0	1	0	0	1	0	0	1	1	0	1	1	0	1
D	1	0	0	0	1	1	1	0	0	0	1	1	1	0

        - Injecting randomness
            - Into decision tree split criteria so they chooses randomly among the best n tests at each node
            - Into ANN bootstrap sampling of training data to add Gaussian noise to the input features
    - Combine classifiers
        - Unweighted
        - Weighted
        - Stacking - use outputs of n classifiers as attributes for target
        - Gating = stacking + weighted/unweighted
- Random forest
    1. Sample N (size of training set) cases at random - with replacement, from the original data. This sample will be the training set for growing the tree.
    2. For M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
    3. Each tree is grown to the largest extent possible. There is no pruning
- Comparisons
    - Random Forest
        - Bagging and Random Forests can be easily run in parallel, but not for Boosting for there is no random sample drawn.
        - No need for cross-validation or a separate test set to get an unbiased estimate of the test set error for as it can be tested again the out of bag (OOB) samples during the run
        - A good m needs to be tuned to balance trade-offs between correlation and strength of each tree in the forest
        - Variable importance can be ranked by randomly permuting values of m variables in the OOB cases, and comparing them to the untouched OOB cases (only work if variables are independent)
    - AdaBoost: natively reweighted, so it's sensitive to class errors
    - Gradient Boost: can be penalised by L1 or L2 regularisation
    - XG Boost: improve the performance of Gradient Boost series

Bayesian Learning

Bayes' Theorem



- D: The event that we observed this particular data set
- h: The event that the hypothesis h is the true hypothesis

Applications

- Explicit manipulation of probabilities, e.g. among the most practical approaches to certain types of learning problems
- Useful framework for understanding learning methods that do not explicitly manipulate probabilities, e.g. determine conditions under which algorithms output the most probable hypothesis: justification of the error functions in ANNs, justification of the inductive bias of decision trees, etc

Maximum A Posteriori (MAP)
In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis h ∈H given the observed training data D

- The maximum a posteriori (MAP) hypothesis h_{MAP} where P(D) is dropped because it is a constant independent of h

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h|D) \\ &= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg\max_{h \in H} P(D|h)P(h) \end{aligned}$$

- Example
    - Consider a medical diagnosis problem in which there are two alternative hypotheses
        - The patient has a particular form of cancer (denoted by $cancer$)
        - The patient does not have a cancer (denoted by $\neg cancer$)
    - The available data is from a particular laboratory with two possible outcomes: $\oplus$ (positive) and $\ominus$(negative)

$$\begin{aligned} P(cancer) &= 0.008 & P(\neg cancer) &= 0.992 \\ P(\oplus|cancer) &= 0.98 & P(\ominus|cancer) &= 0.02 \\ P(\oplus|\neg cancer) &= 0.03 & P(\ominus|\neg cancer) &= 0.97 \end{aligned}$$

    - Suppose a new patient is observed for whom the lab test returns a positive ($\oplus$) result
    - Should we diagnose the patient as having cancer or not?

$$\begin{aligned} P(cancer|\oplus) &\sim P(\oplus|cancer)P(cancer) & &= 0.98 * 0.008 = 0.0078 \\ P(\neg cancer|\oplus) &\sim P(\oplus|\neg cancer)P(\neg cancer) & &= 0.03 * 0.992 = 0.0298 \end{aligned}$$

$\Rightarrow h_{MAP} = \neg cancer$

- The exact posterior probabilities can be determined by normalizing the above probabilities to 1

$$P(cancer|\oplus) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$
$$P(\neg cancer|\oplus) = \frac{0.0298}{0.0078 + 0.0298} = 0.79$$

Maximum Likelihood (ML)

- The equation above can be further simplified assuming that every hypothesis is equally probable a priori. This is called maximum likelihood (ML) hypothesis h_{ML}

$$h_{ML} = \arg\max_{h \in H} P(D|h)$$

- Least-Squared Error
  - Problem: learning continuous-valued target functions (e.g. neural networks, linear regression, etc.). Under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis and the training data, will output a ML hypothesis
    - $(\forall h \in H)[h : X \to \mathbb{R}]$ and training examples of the form $< x_i, d_i >$
    - unknown target function $f : X \to \mathbb{R}$
    - m training examples, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean $(d_i = f(x_i) + e_i)$
  - Solution:
    - It is common to maximize the less complicated logarithm, which is justified because of the monotonicity of this function

$$h_{ML} = \arg\max_{h \in H} \sum_{i=1}^{m} log \frac{1}{\sigma\sqrt{2\pi\sigma^2}} - \frac{(d_i - h(x_i))^2}{2\sigma^2}$$

    - The first term in this expression is a constant independent of h and can therefore be discarded.

$$h_{ML} = \arg\max_{h \in H} \sum_{i=1}^{m} - \frac{(d_i - h(x_i))^2}{2\sigma^2}$$

    - Maximizing this negative term is equivalent to minimizing the corresponding positive term.

$$h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} \frac{(d_i - h(x_i))^2}{2\sigma^2}$$

    - Finally, all constants independent of h can be discarded.

$$h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

    $\Rightarrow$ the h_{ML} is one that minimizes the sum of the squared errors

Minimum Description Length (MDL)

- Occam's razor: choose the shortest explanation for the observed data
- Interpret the definition of h_{MAP} in the light of information theory concepts, we can see that that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(D|h)P(h) \\ &= \arg\max_{h \in H} log_2 P(D|h) + log_2 P(h) \\ &= \arg\min_{h \in H} -log_2 P(D|h) - log_2 P(h) \end{aligned}$$

- A basic result of information theory
  - Problem: design a code C to transmit messages drawn at random
    - Probability of encountering message i is pi
    - Interested in the most compact code C
    - Shannon and Weaver (1949) showed that the optimal code assigns –log2pi bits to encode message I
    - LC (i) ≈ description length of message i with respect to C
  - Solution: Minimum description length principle

$$h_{MAP} = \arg\min_{h \in H} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

    - LCH (h) = –log2P(h), where CH is the optimal code for hypothesis space H
    - LCD|h (D|h) = –log2P(D|h), where CD|h is the optimal code for describing data D assuming that both the sender and receiver know hypothesis h
- MDL principle provides a way for trading off hypothesis complexity for the number of errors committed by the hypothesis, e.g. decision tree learning
  - C_H might be some obvious encoding, in which the description length grows with the number of nodes and with the number of edges
  - Choice of $C_{D|h}$?
    - Sequence of instances $< x_1, \ldots, x_m >$ is known to the transmitter and the receiver
    - We need only to transmit the classifications $< f(x_1), \ldots, f(x_m) >$
    - If h correctly predicts the classification, no transmission is necessary $(L_{C_{D|h}}(D|h) = 0)$
    - In case of missclassified examples, for each missclassification a message has to be sent that identifies this example (at most $log_2 m$ bits) as well as its correct classification (at most $log_2 k$ bits, where k is the number of possible classifications)

Bayes Optimal Classifier

- This is different from the MAP framework that seeks the most probable hypothesis (model). Instead, we are interested in making a specific prediction, so the most probable classification is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities

$$\arg\max_{v_j \in V} P(v_j|D) = \arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

$h_i \in H$

where P(vj|D) is the probability that the correct classification is vj

- Example

$V = \{\oplus, \ominus\}$

$$P(h_1|D) = 0.4, P(\ominus|h_1) = 0, P(\oplus|h_1) = 1$$
$$P(h_2|D) = 0.3, P(\ominus|h_2) = 1, P(\oplus|h_2) = 0$$
$$P(h_3|D) = 0.3, P(\ominus|h_3) = 1, P(\oplus|h_3) = 0$$

therefore

$$P(\oplus|D) = \sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = 0.4$$
$$P(\ominus|D) = \sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = 0.6$$

and

$$\arg\max_{v_j \in \{\oplus, \ominus\}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Naïve Bayes Classifier

- The computational cost of Bayes optimal classifier is to weigh in the predications of all hypotheses. In the case where each instance x is described by a conjunction of attribute values < a1, a2, . . . , an > and where the target function f (x) can take on any value from some finite set V, this can be simplified assuming attribute values are conditionally independent given the target value:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$$

  - P(vj) can be estimated by counting the frequency of vj in D
  - The number of terms is |distinct attribute values| × |V | + |V |
- Document Classification
  - For documents presented by features x1,x2,...,xn, let X be the set of unique feature values (e.g. unique words)

$$c_{MAP} = \arg\max_{c \in C} P(c) \prod_i P(x_i|c)$$

where document features can be based on word counts

$$P(x_i|c) = \frac{count(x_i, c)}{\sum_{x_j \in X} count(x_j, c)}$$

  - Laplace smoothing for unknown words, i.e. count(xi,c) = 0 for all classes, then a constant is added

$$P(x_i|c) = \frac{count(x_i, c) + 1}{\sum_{x_j \in X}(count(x_j, c) + 1)}$$

  - Example

	Doc_ID	Words	class
Training	d1	Kiwi, Sheep, Kiwi	NZ
	d2	Kiwi,Kiwi,Bird	NZ
	d3	Kiwi,Auckland	NZ
	d4	Munich,Oktoberfest,Kiwi	DE
Test	d5	Kiwi,Kiwi,Kiwi,Munich,Oktoberfest	NZ

$$P(c) = \frac{N_c}{N}$$
$$P(x|c) = \frac{count(x, c) + 1}{\sum_{x_j \in X} count(x_j, c) + |X|}$$

■ Priors

$P(NZ) = 3/4$
$P(DE) = 1/4$

■ Conditional Probabilities

$P(Kiwi|NZ) = (5+1)/(8+6) = 3/7$
$P(Munich|NZ) = (0+1)/(8+6) = 1/14$
$P(Oktoberfest|NZ) = (0+1)/(8+6) = 1/14$
$P(Kiwi|DE) = (1+1)/(3+6) = 2/9$
$P(Munich|DE) = (1+1)/(3+6) = 2/9$
$P(Oktoberfest|DE) = (1+1)/(3+6) = 2/9$

■ Choosing a class for d5

$P(NZ|d5) \sim 3/4 * (3/7)^3 * 1/14 * 1/14 \approx 0.0003$
$P(DE|d5) \sim 1/4 * (2/9)^3 * 2/9 * 2/9 \approx 0.0001$

Instance-based Learning

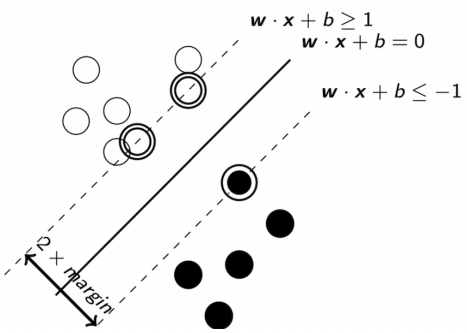
A class of algorithms that do not learn a function but rather directly compare to known examples

K-Nearest Neighbour (KNN)

- Compute the pair-wise distances (Euclidean, Mahalanobis distance, etc) to the labelled examples, and find the nearest k neighbours
- Choose the class label of the majority votes from the k nearest neighbours

Support Vector Machines (SVM)

- Hard margin: maximum margin hyperplane achieves maximal separation between the classes where the encircled training instances define the margin borders for each class -> maximal margin classifier



- Training data {(xk = [xk1,...xkm],yk)} yk ∈{−1,+1},k ∈[1,n]
- Classify instance xk as +1 if w·xk +b≥1, −1 if w·xk +b≤−1, where $wx_k = \Sigma_{i=1}^{m} w_i x_{ki}$
- The distance of point x to a hyperplane defined with parameters w and b is $\frac{|wx+b|}{\sqrt{w*w}}$ where the distance of x−/x+ to the decision boundary is reduce$\frac{1}{\sqrt{w*w}}$

- Soft margin allows misclassification errors -> support vector classifier/ soft margin classifier
  It maximizes using Lagrange multiplier

$$L(\alpha) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{q=1}^{n} \alpha_k \alpha_q y_k y_q \boldsymbol{x_k} \cdot \boldsymbol{x_q}$$

  with respect to α subject to the constraints 0 ≤ α ≤ C
  - Complexity parameter C > 0 trades off training error for model complexity (number of support vectors, i.e. the observations that violate the margin)
  - C → ∞ we get the maximal margin classifier
- Non-linear kernel transforms the dot product in the decision making process -> support vector machine

$$d(\boldsymbol{x}) = b + \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x}$$

  becomes

$$d(\boldsymbol{x}) = b + \sum_{i=1}^{l} \alpha_i y_i (\boldsymbol{x}_i \cdot \boldsymbol{x})^h$$

  - Number of support vectors l
  - Parameters αi , b (estimated by training / optimization)
  - Test instance vector x
  - Support vector xi
  - An instance can be classified by using $\hat{y}(x) = sign(d(x))$
  - Different kernels to encode expert knowledge
    - Polynomial kernels: K(xi,x) = $(\gamma x_i^T x + 1)^d$, γ > 0, d = 1, 2, ... -> image processing
    - Radial kernels: K(xi,x) = $\exp(-\gamma \|xi - x\|^2)$, γ > 0 -> general purpose
    - Sigmoid kernels: K(xi,x) = $\tanh(\gamma x_i^T x + 1)$, γ > 0 where $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$ -> neural network

Evolutionary Algorithms

Framework

1. Random generation of an initial population
2. Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.
3. Reproduction of the population based on fitness values.
4. If requirements are met, then stop. Otherwise go back to

Generic Algorithm (GA)

- Biological foundations
  - Phenotype is determined by genotype and learning/interaction with the environment
    - Genotype is the collection of genes of an organism.
    - Phenotype is the morphology, physiology and behaviour of an organism.
  - Evolution is a search process for phenotypes
    - Chromosomes (DNA molecules) can be interpreted as character strings in nature's base-4 alphabet - nucleotides: A (adenine), C (cytosine),G (guanine) and T (thymine)
    - The evolutionary process performs natural selection and genetic operations (mutation & crossover)
- Genetic Algorithms are highly parallel search algorithms inspired by evolutionary processes

GA($Fitness$, $Fitness\_threshold$, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.
Fitness_threshold: A threshold specifying the termination criterion.
p: The number of hypotheses to be included in the population.
r: The fraction of the population to be replaced by Crossover at each step.
m: The mutation rate.

- *Initialize population:* $P \leftarrow$ Generate p hypotheses at random
- *Evaluate:* For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$ do

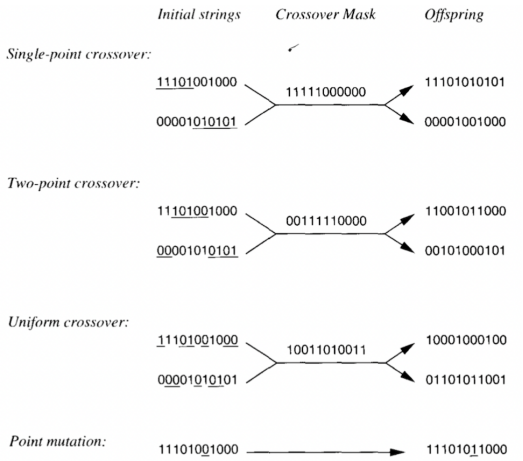
Create a new generation, P_S:

1. *Select:* Probabilistically select $(1-r)p$ members of P to add to P_S. The probability $\Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $\Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
3. *Mutate:* Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.
4. *Update:* $P \leftarrow P_s$.
5. *Evaluate:* for each h in P, compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.
  - Population models depending on the proportion of the population replaced (generation gap)
    - Generational Model (GGA) - Each individual survives 1 generation (1.0)
    - Steady State Model (SSGA) - One offspring generated per generation with one member replaced (1/pop_size)
  - Fitness function
    - Accuracy of the hypothesis over the training data
    - Number of games won by the individual when playing against other individuals in the current population
  - Generic Operators



- Applications - balancing the trade-off of changing variables, e.g. self-driving cars
  - Children of individuals with high fitness will have high fitness
  - Blocks of close genes - assuming uniform crossover is not used, the closer two Genes are to each other, the more likely they are to be passed together to the same child
- Limitation - early convergence to a local minima due to the overcrowding of the fittest gene
  - Sharing (fitness sharing causes individuals that are similar to many others to have their fitness reduced, making them less likely to be selected) for GA and crowding (the current population is sampled to find an individual that is "close" to the new offspring, then this closest individual is then replaced) for SSGA
  - Tournament Selection - pick k members at random, then select the best of these, can also use ρ to determine whether the fittest gene wins

Swarm Intelligence

- Biological foundations - collective behaviours result from the local interactions of the individuals with
  each other and/or with their environment, e.g. colonies of ants and termites, schools of fish, flocks of birds, bacterial growth, herds of land animals.
- Ant Colony Optimisation (ACO) - initially, ants wander randomly and return to their colony upon finding food, laying down pheromone trails. The pheromone trails get reinforced if any other ant follows the trail, finds food and returns to the colony, so a short path gets marched over faster with the pheromone density remains high as it is laid on the path as fast as it can evaporate -> real-time adaptability, e.g. network routing, transportation systems, etc.
  - The choice of a solution component from N(sp) is done probabilistically at each construction step, e.g.

$$p(c_{ij} \mid s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, \forall c_{ij} \in N(s^p)$$

    where τ_{ij} and η_{ij} are the pheromone value and the heuristic value associated with the component .

    α and β are positive real parameters whose values determine the relative importance of pheromone versus heuristic information.
  - Pheromone update
    - Decrease all the pheromone values through pheromone evaporation associated with bad solutions
    - Increase the pheromone levels associated with a chosen set of good solutions Supd:

$$\mathrm{T}_{ij} \leftarrow (1-\rho) \cdot \mathrm{T}_{ij} + \sum_{k=0}^{M} (\Delta \mathrm{T}_{xy}^{k})$$

    where $\rho \in (0,1]$ is a parameter called evaporation rate, and $\Delta \mathrm{T}_{xy}^{k}$ is the amount of pheromone left by the kth ant on the edge between x and y
- Particle Swarm Optimization (PSO) - bird flocking or fish schooling
  - A problem is given, and some way to evaluate a proposed solution to it exists in the form of a fitness function.
  - A communication structure or social network is also defined, assigning neighbours for each individual to interact with.
  - Then a population of individuals defined as random guesses at the problem solutions is initialized – candidate solutions.
  - The swarm is typically modelled by particles in multidimensional space that have a position and a velocity

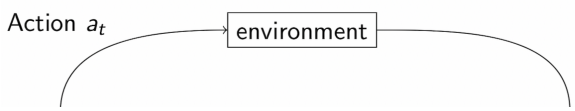
$$v_{i,j} \leftarrow c_0 v_{ij} + c_1 r_1 (globalbest_j - x_{i,j}) + c_2 r_2 (localbest_{i,j} - x_{i,j}) + c_3 r_3 (neighborhoodbest_j - x_{i,j})$$

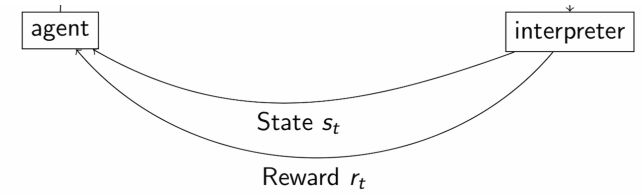
$$x_{i,j} \leftarrow x_{i,j} + v_{i,j}$$

    i is the particle and *j* is the dimension

Reinforcement Learning

to choose actions that maximize the total rewards -> a sequential decision making

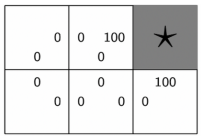




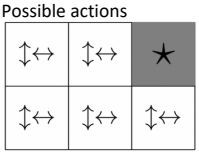
Markov Decision Process

- Mathematical formulation
  - S Set of possible states st ∈S
  - A Set of possible actions at ∈A
  - R Distribution of the current reward given (state, action) pair
  - P Distribution of the next state given (state, action) pair
  - γ Discount factor with 0 ≤ γ < 1 weighting the impact of future rewards
- Assumptions
  - The next state is only related to the current state and action:
    $P(s_{t+1}|s_t,\ a_t,\ s_{t-1},\ a_{t-1},\ \ldots) = P(s_{t+1}|s_t,\ a_t)$
  - The reward is only related to current state and action:
    $R(r_t|s_t,\ a_t,\ s_{t-1},\ a_{t-1},\ \ldots) = R(r_t|s_t,\ a_t)$
- The task is learning a policy π : S → A for choosing actions that maximizes
  $E\left[r + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$ for every possible starting state s:
  - Value function
    $$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t\right]$$
  - Optimal policy
    $$\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \in S$$
- Example: grid world where one reaches the terminal state (greyed out) in least number of actions.

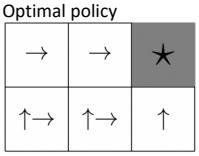
1. Rewards



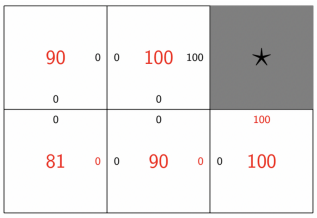
2. Possible actions



3. Optimal policy



4. Suppose $\gamma = 0.9$ and optimal policy



Q-Learning

- Value iteration for $V^{\pi^*}(s)$ (can be abbreviated as $V^*(s)$)
  - Assuming $P(S_{t+1}|S_t, A)$ is known

```
Initialize V(s) arbitrarily
while Not Done do
    foreach s ∈ S do
        foreach a ∈ A do
            Q(s, a) ← r(s, a) + γ Σ_{s'∈S} P(s'|s, a) V(s')
        end
        V(s) ← max_a Q(s, a)
    end
end
```

- $V(s)$ converges to $V^*(s)$
  - Interestingly, value iteration works even if we randomly traverse the environment instead of looping through each state and action methodically, but we must still visit each state infinitely often on an infinite run -> online learning as agent randomly roams
  - If max (over states) difference between two successive value function estimates is less than ε, then the value of the greedy policy differs from the optimal policy by no more than 2εγ/(1 − γ)

$$\max_s |V_{i+1}(s) - V_i(s)| < \epsilon \Rightarrow \max_s |V^*(s) - V_i(s)| < \frac{2\epsilon\gamma}{1-\gamma}$$

- Q-learning when $P(s_{t+1}|s_t, a_t)$ is unknown

foreach s, a **do**
  initialize table entry $\hat{Q}(s, a) \leftarrow 0$
end
observe current state s
while *forever* **do**
  select action a and execute it
  receive immediate reward r
  observe new state s'
  update the table entry for $\hat{Q}(s, a)$:
  $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s'a')$
  $s \leftarrow s'$
end

  - Q and V^* are closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

  - $\widehat{Q}$ is to take into account the maximum reward for the next state after executing a action with the probability that's proportional to rewards (so there's still a chance to jump round to find the global optimum instead of being trapped of local optima) to approximate Q:

$$\hat{Q}(s, a) \leftarrow \mathbb{E}[r(s, a)] + \gamma \max_{a'} \hat{Q}(s', a')$$

  - $\widehat{Q}$ will always converge to the true Q function
    i. The system is a deterministic MDP
    ii. The immediate reward values are bounded: ∀s,∀a,∃c > 0 : |r(s,a)| < c
    iii. Agents select actions such that every state-action pair is visited infinitely often
- The training rule for the nondeterministic Q-learning

Q Learning generalizes to nodeterministic worlds
Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')$$

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)},$$

  - If αn is set to 1, same rule as the deterministic case
  - We can still prove convergence of $\widehat{Q}$ to Q [Watkins and Dayan, 1992]

Association Analysis

Association Rule Mining
Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

- Definitions
  - Frequent Itemset: an itemset whose support is greater than or equal to a minsup threshold
    - k-itemset: an itemset that contains k items
    - Support count (σ): frequency of occurrence of an itemset
    - Support: fraction of transactions that contain an itemset
  - Association Rule: a rule whose confidence is greater than or equal to minconf threshhold
    - An implication expression of the form X → Y, where X and Y are itemsets
    - Rule Evaluation Metrics
      - Support (s): fraction of transactions that contain both X and Y
      - Confidence (c): measures how often items in Y appear in transactions that contain X
  - Example

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example: $\{\text{Milk}, \text{Diaper}\} \Rightarrow \{\text{Beer}\}$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{} = \frac{2}{} = 0.4$$

$$|T| \qquad 5$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

- Mining Tasks
  - Frequent Itemset Generation, given d unique items, the total number of itemsets M:
    $M = 2^d$
  - Rule Generation, given d unique items, the total number of possible association rules R:

$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$

$$= 3^d - 2^{d+1} + 1$$

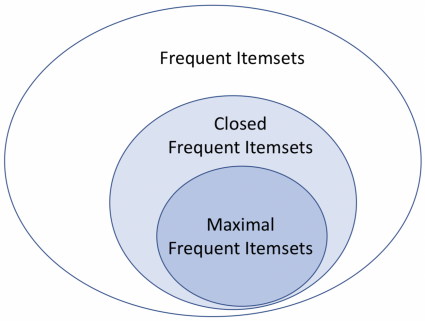
    - Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement.
    - If $|L| = k$, then there are 2k - 2 candidate association rules (ignoring $L \rightarrow \phi$ and $\phi \rightarrow L$)

Apriori Algorithm for Frequent Itemset Generation

- Apriori principle - If an itemset is frequent, then all of its subsets must also be frequent:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support
- Maximal Frequent vs Closed Frequent



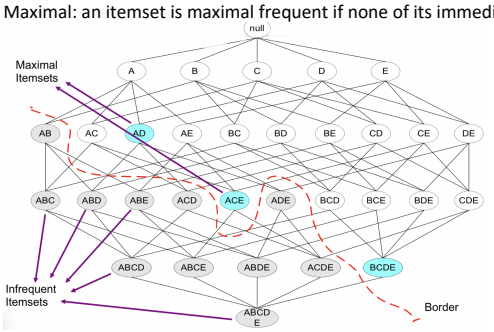
  - Closed: an itemset X is closed if none of its immediate supersets has the same support as the itemset X.

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

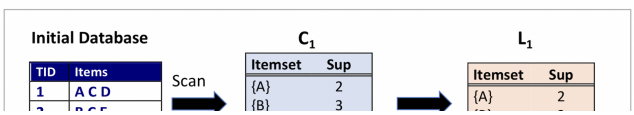
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	2
{A,B,C,D}	2

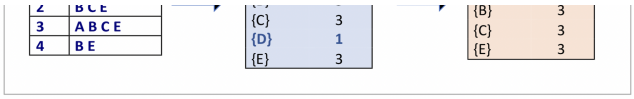
  - Maximal: an itemset is maximal frequent if none of its immediate supersets is frequent

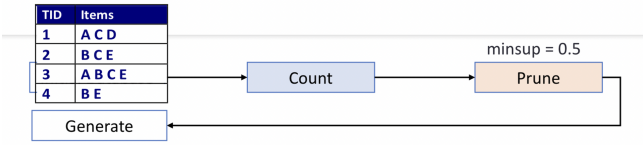


- Algorithm
  - Let k=1
  - Generate frequent itemset sof length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent
- Example (automatically):

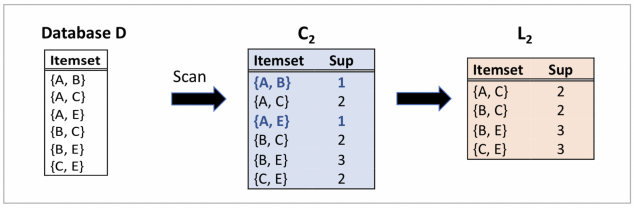
1.

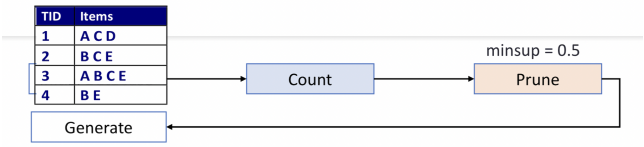




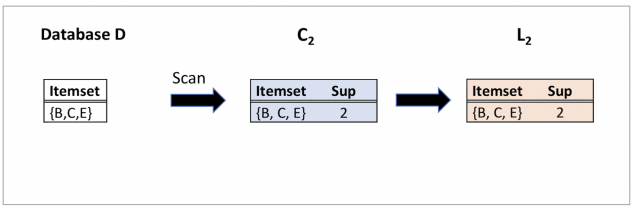


2.

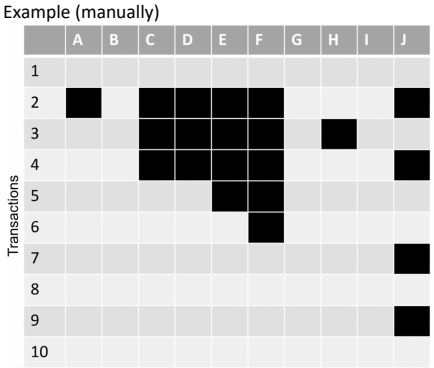




3.



- Example (manually)



Support threshold (by count) : 5
- Frequent itemsets: {F}
- Maximal itemsets: {F}

Support threshold (by count) : 4
- Frequent itemsets: {E}, {F}, {E,F}, {J}
- Maximal itemsets: {E,F}, {J}

Support threshold (by count): 3
- Frequent itemsets: All subsets of {C,D,E,F} + {J}
- Maximal itemsets: {C,D,E,F}, {J}

Pattern Evaluation for Rule Generation

- Confidence(X → Y) > support(Y), otherwise, rule will be misleading because having item X reduces the chance of having item Y in the same transaction

	$Coffee$	$\overline{Coffee}$	
Tea	15	5	20
$\overline{Tea}$	75	5	80
	90	10	100

Association Rule: Tea → Coffee

Confidence= P(Coffee|Tea) = 15/20 = 0.75

but P(Coffee) = 0.9, which means knowing that a person drinks tea reduces the probability that the person drinks coffee!

- Confidence of rules generated from the same itemset has an anti-monotone property, i.e. confidence is anti-monotone with reference to (w.r.t) items on the right hand side of the rule

e.g., L = {A,B,C,D},

conf(ABC → D) ≥ conf(AB → CD) ≥ conf(A → BCD)

$\frac{\sigma(ABCD)}{\sigma(ABC)}$ $\frac{\sigma(ABCD)}{\sigma(AB)}$ $\frac{\sigma(ABCD)}{\sigma(A)}$

- Given X → Y or {X,Y}, information needed to compute interestingness can be obtained from a contingency table

Contingency table

	Y	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

f_{11}: support of X and Y

f_{10}: support of X and $\overline{Y}$

f_{01}: support of X and Y

f_{00}: support of $\overline{X}$ and $\overline{Y}$

Used to define various measures

support, confidence, Gini, entropy, etc.

- X and Y are independent:
  - Confidence(X → Y) = support(Y)
  - P(Y|X) = P(Y)
  - P(X,Y) = P(X) × P(Y)
- X & Y are positively correlated: P(X,Y) > P(X) × P(Y)
- X & Y are negatively correlated: P(X,Y) < P(X) × P(Y)
- Those can be measured by Lift or Interest:

$$Lift = \frac{P(Y \mid X)}{P(Y)}$$

$$Interest = \frac{P(X,Y)}{P(X)P(Y)}$$

lift is used for rules while interest is used for itemsets

- Simpson's Paradox: observed relationship in data may be influenced by the presence of other confounding factors (hidden variables). Hidden variables may cause the observed relationship to disappear or reverse its direction, so proper stratification is needed to avoid generating spurious patterns, e.g.
  - Question

Hospital	$Recovered$	$\overline{Recovered}$	
A	99	81	180
B	54	66	120
	153	147	300

Conf({Hospital=A} → {Recovered}) = 99/180 = 55%

Conf({Hospital=B} → {Recovered}) = 54/120 = 45%

=> Patient who stay at hospital A are more likely to be recovered?

  - Answer:

Patient	Hospital	$Recovered$	$\overline{Recovered}$	Total
Young	A	1	9	10
	B	4	30	34
Old	A	98	72	170
	B	50	36	86

Young Patients

Conf({Hospital=A} → {Recovered}) = 1/10 = 10%

Conf({Hospital=B} → {Recovered}) = 4/34 = 11.8%

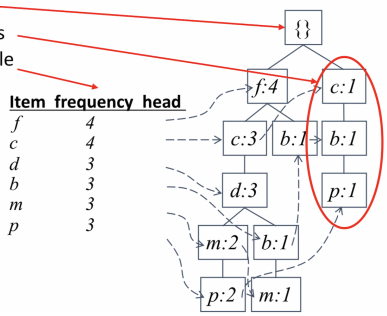
Old Patients

Conf({Hospital=A} → {Recovered}) = 98/170 = 57.7%

Conf({Hospital=B} → {Recovered}) = 50/86 = 58.1%

FP-Tree and FP-Growth for Data Structure

- FP-Tree
  - Components
    - One root: labeled as "null"
    - A set of item prefix subtrees
    - A frequent-item header table



  - Construction
    1. Sort items by the order in the header table

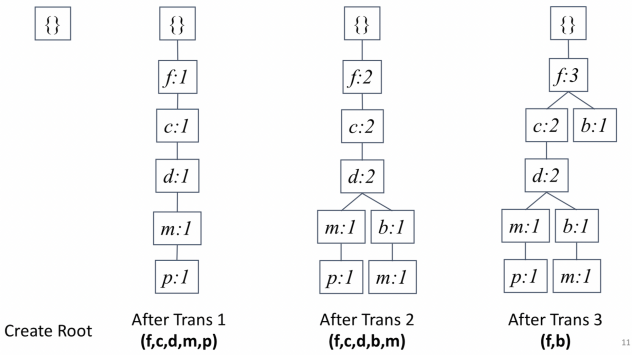
TID	Items Bought	(Ordered) Frequent Items
1	f,d,c,a,g,i,m,p	f,c,d,m,p
2	d,b,c,f,l,m,o	f,c,d,b,m
3	b,f,h,j,o	f,b
4	b,c,k,s,p	c,b,p
5	d,f,c,e,l,p,m,n	f,c,d,m,p

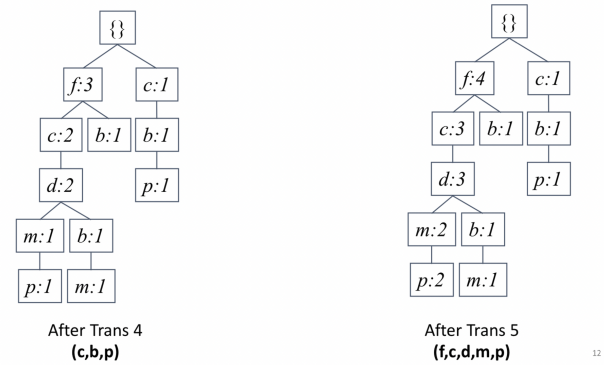
minimum support threshold (count) is set to 3

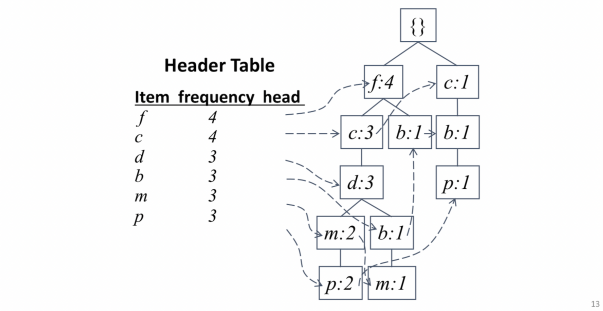
2. First Scan count and sort

$$L = \{(f:4), (c:4), (d:3), (b:3), (m:3), (p:3)\}$$

   1. count the frequencies of each item
   2. collect length 1 frequent items, then sort them in support/count descending order into L , frequent item list
3. Second Scan create the tree and header table



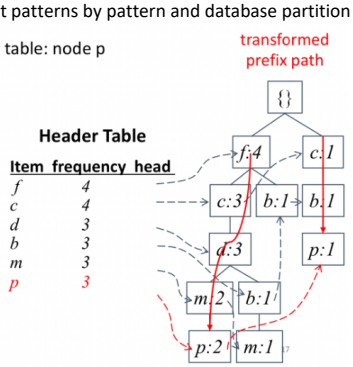




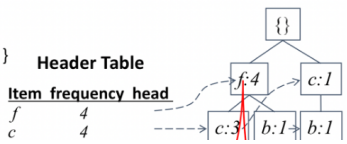
   1. Create the root, label it as "null"
   2. For each transaction
      a. select and sort the frequent items in Trans
      b. increase nodes count or create new node
         - If prefix nodes already exist, increase their counts by 1
         - If no prefix nodes, create it and set count to 1.
   3. build the item header table: nodes with the same item name are linked in sequence via node links

- The size of the FP tree depends on how the items are ordered
  - Best case scenario : all transactions contain the same set of 1 path in the FP tree
  - Worst case scenario : every transaction has a unique set of items (no items in common)

- FP-Growth: recursively grow frequent patterns by pattern and database partition
  - Start from the bottom of the header table: node p
  - Two paths
  - p's conditional pattern base
    - { (f:2, c:2, d:2, m:2), (c:1, b:1) }
  - p's conditional FP tree
    - Only one branch (c:3) → pattern (cp: 3)
  - Patterns: (p:3), (cp:3)

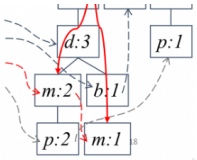


  - Continue with node m
  - Two paths
  - m's conditional pattern base
    - { (f:2, c:2, d:2), (f:1, c:1, d:1, b:1) }
  - m's conditional FP tree
    - (f:3, c:3, d:3)
  - Call mine(< f:3, c:3, d:3 > | m)



- Patterns:
  - (m:3)
  - See next slide

d	3
b	3
m	3
p	3



- node *d*:
  - *(dm:3)*
  - call mine(<*f:3, c:3*>|*dm*)
    - *(cdm:3)*
    - call(<*f:3*>|*cdm*) → *(fcdm:3), (fdm:3)*
- node *c*:
  - *(cm:3)*
  - call mine(<*f:3*>|*cm*)
    - *(fcm:3)*
- node *f*: *(fm:3)*
- All the patterns:*(m:3, dm:3, cm:3, fm:3, cdm:3, fdm:3, fcm:3, fcdm:3)*
- *A single path FP-Tree can be mined by outputting all the combination of the items in the path.*

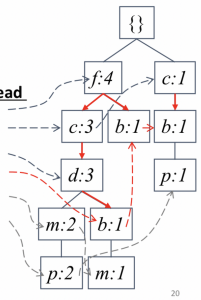
Header Table

Item	frequency	head
f	3	
c	3	
d	3	

- Continue with node b
- Three paths
- b 's conditional pattern base
  - { (f:1, c:1, d:1), (f:1), (c:1) }
- b 's conditional FP tree
  - Ø
- Patterns: (b:3)

Header Table

Item	frequency	head
f	4	
c	4	
d	3	
b	3	
m	3	
p	3	

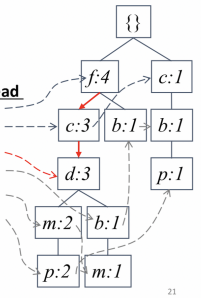


20

- Continue with node d
- One path
- d 's conditional pattern base
  - { (f:3, c:3) }
- d 's conditional FP tree
  - { (f:3, c:3) }
- Patterns:
  - (d:3)
  - (cd:3)
  - (fd:3)
  - (fcd:3)

Header Table

Item	frequency	head
f	4	
c	4	
d	3	
b	3	
m	3	
p	3	

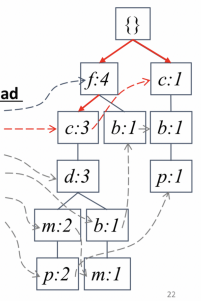


21

- Continue with node c
- Two paths
- c 's conditional pattern base
  - { (f:3) }
- c 's conditional FP tree
  - { (f:3) }
- Patterns:
  - (c:4)
  - (fc:3)

Header Table

Item	frequency	head
f	4	
c	4	
d	3	
b	3	
m	3	
p	3	

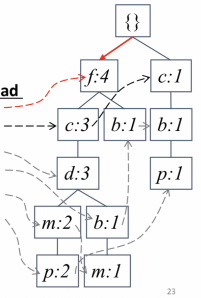


22

- Continue with node f
- One path
- f 's conditional pattern base
  - Ø
- f 's conditional FP tree
  - Ø
- Patterns:
  - (f:4)

Header Table

Item	frequency	head
f	4	
c	4	
d	3	
b	3	
m	3	
p	3	



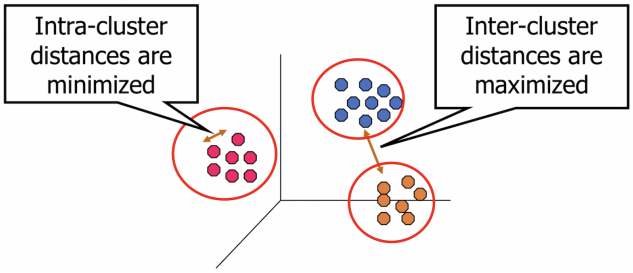
23

Item	Conditional Pattern Base	Conditional FP-Tree
p	{(f:2, c:2, d:2, m:2), (c:1, b:1)}	{(c:3)}| p
m	{((f:2, c:2, d:2), (f:1,c:1, d:1, b:1)}	{(f:3, c:3, d: 3)}| m
b	{(f:1, c:1, d:1), (f:1), (c:1)}	Ø
d	{(f;3, c:3)}	{(f:3, c:3}| d
c	{(f:3)}	{(f:3)}| c
f	Ø	Ø

1. For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
2. Repeat the process on each newly created conditional FP-tree
3. Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

Cluster Analysis

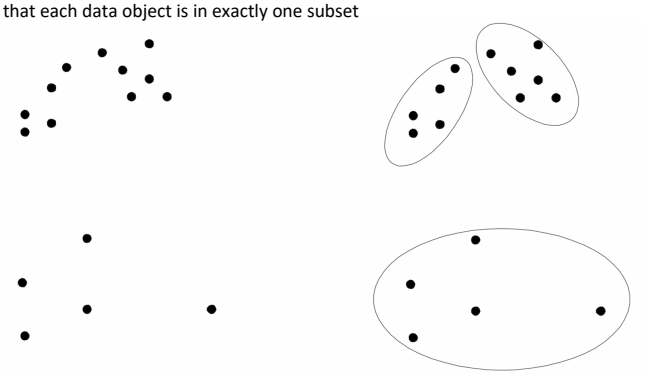
is finding groups of objects such that



1. the objects in a group will be similar (or related) to one another
2. different from (or unrelated to) the objects in other groups

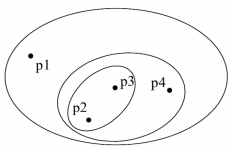
Types of Clustering (i.e. a set of clusters)

- Partitional versus Hierarchical
  - Partitional Clustering: a division of data objects into non overlapping subsets (clusters) such that each data object is in exactly one subset

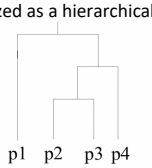


Original Points **A Partitional Clustering**

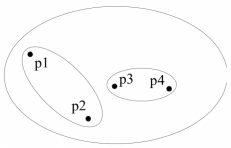
  - Hierarchical clustering: a set of nested clusters organized as a hierarchical tree



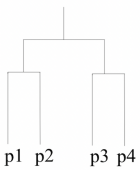
Traditional Hierarchical Clustering



Traditional Dendrogram



Non-traditional Hierarchical Clustering

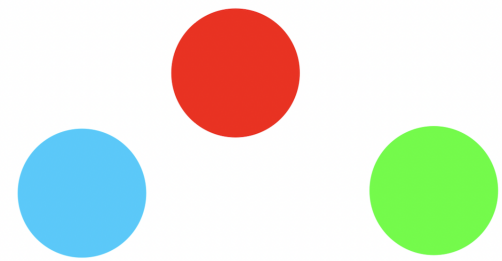


Non-traditional Dendrogram

    - Agglomerative: start with the points as individual clusters, and at each step, merge the closest pair of clusters until only one cluster (or k clusters) is left
    - Divisive: start with one, all-inclusive cluster, and at each step, split a cluster until each cluster contains an individual point (or there are k clusters)
- Exclusive versus non-exclusive: in non-exclusive clustering, points may belong to multiple clusters/multiple classes or could be 'border' points
- Fuzzy (one type of non-exclusive) versus non-fuzzy clustering: in fuzzy clustering, a point is with some weight between 0 and 1 indicating strength of belief in the event in question and weights must sum to 1 for that event, e.g. the point belongs to cluster a
- Partial versus complete: in partial clustering, we only want to cluster some of the data

Types of Cluster

- Well-separated: a cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster

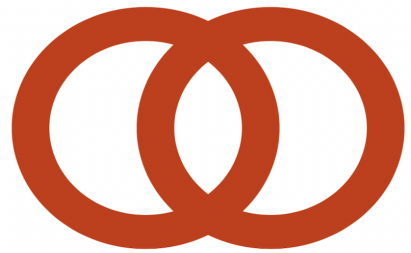


3 well-separated clusters

- Centre-based: a cluster is a set of objects such that an object in a cluster is closer (more similar) to the prototype or “centre” of a cluster, than to the centre of any other cluster, where the centre of a cluster is often
  - a centroid, the average of all the points in the cluster
  - a medoid, the most “representative” point of a cluster

4 center-based clusters

- Shared Property or Conceptual: clusters that share some common property or represent a particular concept

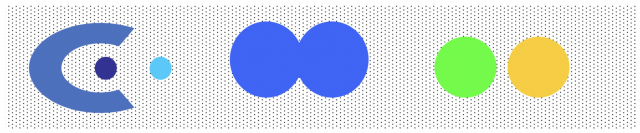


2 Overlapping Circles

- Contiguity-based (nearest neighbour or transitive): a cluster is a set of points such that a point in a cluster is closer (or more similar) to one or more other points in the cluster than to any point not in the cluster.

8 contiguous clusters

- Density-based: a cluster is a dense region of points, which is separated by low-density regions, from other regions of high density. It's used when the clusters are irregular or intertwined, and when noise and outliers are present.



6 density-based clusters

- Described by an Objective Function
  - Global vs. Local
    - Hierarchical clustering algorithms typically have local objectives
    - Partitional algorithms typically have global objectives
  - Parameterized vs. Mixture
    - Parameters for the model are determined from the data.
    - Mixture models assume that the data is a ‘mixture' of a number of statistical distributions.
  - Minimize vs. Maximize in the context of a weighted graph where the nodes are the points being clustered, and the weighted edges represent the proximities between points
    - minimize the edge weight between clusters and maximize the edge weight within clusters [if your objective function uses 1 to represent an exact match/similarity and 0 to represent no similarity]
    - maximize the edge weight between clusters and minimize the edge weight within clusters [if your objective function uses 0 to represent an exact match/similarity]
- Prototype-based clusters

Characteristics of Input Data

- Type of proximity or density measure
- Sparseness dictates type of similarity and efficiency
- Attribute type dictates type of similarity
- Type of Data dictates type of similarity and other characteristics, e.g., autocorrelation
- Noise and Outliers: often interfere with the operation of the clustering algorithm
- Clusters of differing sizes, densities, and shapes

K-means Clustering

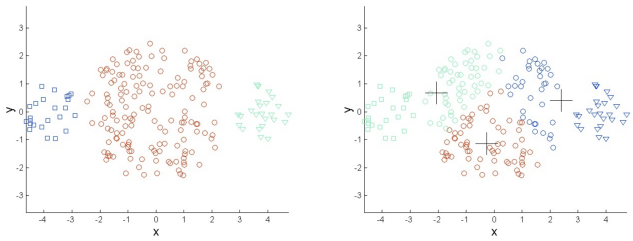
- Pre-processing
  - Normalize the data
  - Eliminate outliers

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

1. Initial centroids are often chosen randomly, so clusters produced vary from one to another
2. Time Complexity: O (I* n*K*d) where n = number of points, K = number of clusters, I = number of iterations, d = number of attributes

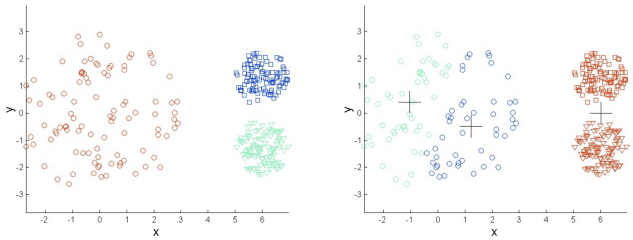
3. 'Closeness' is measured by Euclidean distance, cosine similarity, correlation, SSE etc.
4. The centroid is (typically) the mean of the points in the cluster
5. Most of the convergence happens in the first few iterations for the common similarity measures above, and often the stopping condition is changed to 'Until relatively few points change clusters'

- Post-processing
  - Eliminate empty clusters and small clusters that may represent outliers
  - Split 'loose' clusters, i.e., clusters with relatively high SSE
  - Merge clusters that are 'close' and that have relatively low SSE
  - These steps can be used multiple times during the clustering process
- K-means has problems when clusters are of differing
  - Sizes



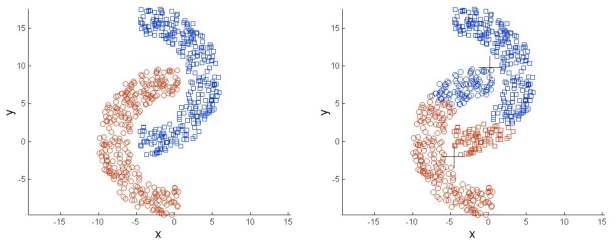
Original Points **K-means (3 Clusters)**

  - Densities



Original Points **K-means (3 Clusters)**

  - Non-globular shapes



Original Points **K-means (2 Clusters)**

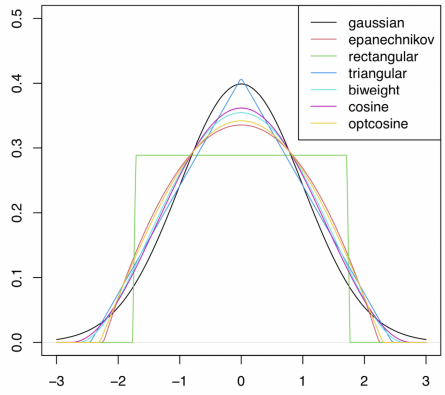
  - One solution is to use many clusters, find parts of clusters, and put them together

Mixture-based Clustering
is to fit a linear combination of probabilistic distributions, e.g. normal distributions,

$$f(x) = \pi_1 f_1(x) + \cdots + \pi_K f_K(x)$$

to the data, where each component represents a cluster, associated with h a mixing proportion πj , subject to πj > 0 and $\Sigma_{j=1}^{K} \pi_j = 1$.

- A kernel function K(x) (usually one that is symmetric about 0 and has a unit variance) is used to provide weights to observations in the neighbourhood of a point and a smoothing parameter h, known as the bandwidth, to control the size of the neighbourhood,



  - Let's denote

$$K_h(x) = \frac{1}{h} K\left(\frac{x}{h}\right)$$

if K(x) is the standard normal distribution, then Kh(x) is the normal distribution with mean 0 and variance h^2
    - The true distribution is denoted by f and transformed from the equation above
- For normal mixtures:
    - Equal variances (homoscadestic): All normal components share an identical variance σ2 (but may have different means μj).
    - Varying variances (heteroscadestic): Each normal component has its own variance σj2 (and mean μj).
- An observation x is of cluster j, if $\pi_j f_j(x)$ is the largest, i.e., the same rule we used for classification.
- Invariant of data scaling

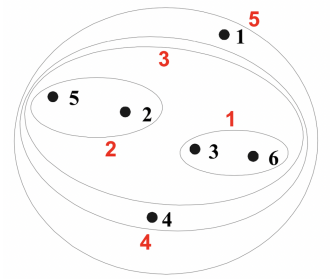
Agglomerative Clustering

1. Compute the proximity matrix
2. Let each data point be a cluster
3. **Repeat**
4. Merge the two closest clusters
5. Update the proximity matrix
6. **Until** only a single cluster remains

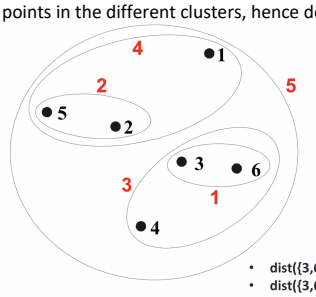
- Complexity
    - O(N2) space since it uses the proximity matrix. • N is the number of points.
    - O(N3) time in many cases
        - There are N steps and at each step the size, N2, proximity matrix must be updated and searched
        - Complexity can be reduced to O(N2 log(N)) time with some approaches
- The key operation is the computation of the proximity of two clusters, i.e. this depends how we prefer the clusters to be linked together, hence the notion of linkage, e.g.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

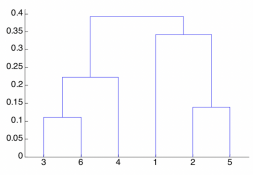
    - Single Linkage/Min: the proximity of two clusters is based on the two closest points in the different clusters, hence determined by one pair of points, i.e., by one link in the proximity graph



        - Pro: can handle non-elliptical shapes
        - Con: sensitive to noise and outliers
    - Complete Linkage/Max: the proximity of two clusters is based on the two most distant points in the different clusters, hence determined by all pairs of points in the two clusters



Nested Clusters



Dendrogram

- dist({3,6},{4})=max(dist(3,4),dist(6,4))= max(0.15,0.22)=0.22.
- dist({3,6},{2,5})=max(dist(3,2),dist(6,2),dist(3,5),dist(6,5))= max(0.15,0.25,0.28,0.39)=0.39.
- dist({3,6},{1})=max(dist(3,1),dist(6,1))= max(0.22,0.23)=0.23.

        - Pro: less susceptible to noise and outliers
        - Cons: tends to break large clusters; biased towards globular clusters
    - Average Linkage/Group Average ~ Ward's Method with the squared distance: the proximity of two clusters is the average of pairwise proximity between points in the two clusters:

$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| \times |\text{Cluster}_j|}$$

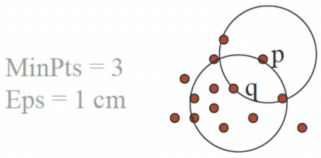
        - Pros: compromise between Single and Complete Link Strengths; less susceptible to noise and outliers
        - Cons: biased towards globular clusters

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Clustering

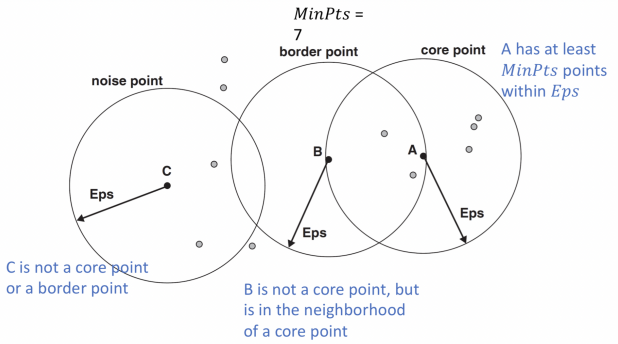
- Select a point p.
- Retrieve all points density reachable from p wrt. Eps and $MinPts$.

- If p is a core point, a cluster is formed.
- If p is a border point, no points are density reachable from p and DBSCAN visits the next point of the database.
- Continue the process until all the points have been processed.

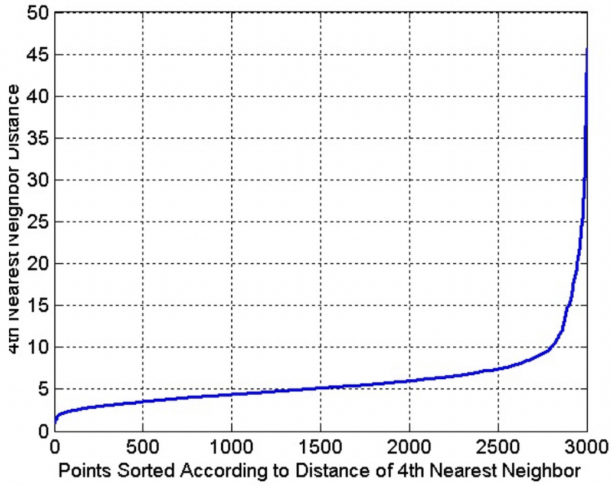
- Parameters
  - Eps: Maximum radius of the neighbourhood
  - $MinPts$: Minimum number of points in an Eps neighbourhood
  - NE(p): $\{q|\ \text{dist}(p, q) \leq Eps\}$ where the point q directly density reachable from p iff $q \in$NE(p) and p is a core object p, e.g.



- A point is, e.g.

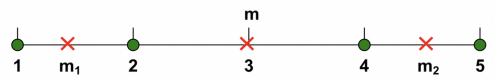


  - a core point if it has at least a specified number of points ($MinPts$) within Eps
    - these are points that are at the interior of a cluster
    - counts the point itself
  - a border point is not a core point, but is in the neighbourhood of a core point
  - a noise point is any point that is not a core point or a border point
- Determining EPS and $MinPts$ - for points in a cluster, their kth nearest neighbours are at close distance, and for noise points, they have the kth nearest neighbour at farther distance. So, plot sorted distance of every point to its kth nearest neighbour, e.g.



- Pros & Cons
  - Pros: is resistant to Noise and can handle clusters of different shapes and sizes
  - Cons: sensitive to varying densities and high-dimensional data

Cluster Validity
- External Index: Used to measure the extent to which cluster labels match externally supplied class labels, e.g. Entropy
- Relative Index: Used to compare two different clustering methods, e.g. SSE
- Internal Index: Used to measure the goodness of a clustering
  - WSS + BSS = constant $\quad WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2 \qquad BSS = \sum_i |C_i|(m - m_i)^2$



K=1 cluster: $WSS = (1-3)^2 + (2-3)^2 + (4-3)^2 + (5-3)^2 = 10$
$BSS = 4 \times (3-3)^2 = 0$
$Total = 10 + 0 = 10$

K=2 clusters: $WSE = (1-1.5)^2 + (2-1.5)^2 + (4-4.5)^2 + (5-4.5)^2 = 1$
$BSS = 2 \times (3-1.5)^2 + 2 \times (4.5-3)^2 = 9$
$Total = 1 + 9 = 10$

  - Cluster cohesion measures how closely related are objects in a cluster, which is measured by the within cluster sum of squares (WSS), i.e. SSE -> the lower, the better

$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

- $|C_i|$is the size of cluster *i*

- Cluster separation measures how distinct or well-separated a cluster is from other clusters, which is measured by the sum of the weights between nodes in the cluster and nodes outside the cluster (BSS) -> the higher, the better

$$BSS = \sum_i |C_i|(m - m_i)^2$$

- $|C_i|$is the size of cluster *i*