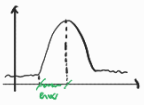
DATA PREPROCESSING

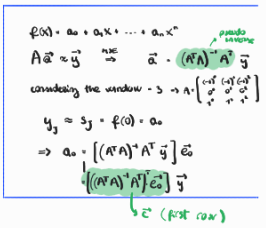
1) **Data Cleaning** → missing values (ignore, manually, global constant, central tendency, mean central tendency, most probable value (inference)) + noise handling + inconsistent data (eg. date format)

NOISE → binning (local smoothing) ⇝ sort data and partition into bins, then smooth by bean → considering a pool neighbours
- outliers analysis ⇝ via clustering method
- filtering
- regression

- **rectangular sliding average** → replace the signal with the average of m adjacent points
  - if the signal has a non-zero second-derivative ($y''(t) \neq 0$) ⇒ I will introduce a bias, but x-location of extremums are kept
- **triangular smooth** → correspond to two application of the rectangular filter (base 5 = 2 base 3) → more effective in reducing high frequency noise
- **Savitzky-Golay Filter** → replace a point with a linear combination of adjacents $s_j = \sum_{n=-n_L}^{+n_R} C_n y_{j+n}$ (higher order polynomial)
  - preserve higher moments of the signal
  - since least-squares involve matrix inversion (linear operation) ⇒ the coefficients C_n are themselfs linear
  - **smoothing ratio** = $\frac{\text{window size}}{\text{half width}}$ — increasing the sr will increase SNR of the signal; sr < 0.2 if I want to measure height and width of a peak

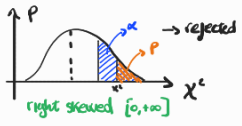


→ smoothing for cosmetic reason or if the algorithm will be penalyzed (maximum location)
- but not for increasing accuracy, it will underestimate errors and smoothed noise may be mistaken for signal



2) **Data Integration** combining data from more sources → solve conflicts, redundancies (correlation analysis)

- **CHI-SQUARED TEST** ⇝ correlation between two categorical variables via the contingency table ($df = (c-1)(r-1)$)
  - non parametric — H_0 → two var are statistically independent
  - $\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$ ($e_{ij} = \frac{n_{A=a_i} \cdot n_{B=b_j}}{n_{tot}}$)
  - α (significance level) → probability of rejecting H_0 when is actually true (concluding a dependency when there isn't)



- **PEARSON PRODUCT** (corr coefficient) $r_{A,B} = \frac{\sum(a_i - \bar{a})(b_i - \bar{b})}{n\sigma_a\sigma_b} = \frac{\sum(a_i b_i) - n\bar{a}\bar{b}}{n\sigma_a\sigma_b}$ — if $r_{A,B} > 0$ ⇒ A,B are positively correlated (linear relationship)
- **CORRELATION** $a_k' = \frac{a_k - \bar{a}}{\sigma_A}$ $b_k' = \frac{b_k - \bar{b}}{\sigma_B}$ ⇒ $corr(A,B) = \vec{a}\cdot\vec{b} = r_{AB} \cdot n$
- **COVARIANCE** $Cov(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$
  $= \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n} = r_{A,B}\cdot\sigma_A\cdot\sigma_B$ — X,Y independent ⇒ $Cov(X,Y) = 0$ — contrary only if $X,Y \sim N$

3) **DATA REDUCTION** find a new data representation with volume much smaller but comparable analytical results (reduce disk usage and computation time) also noise and irrelevant features

- **DIMENSIONALITY REDUCTION** → reduce the # of features of D → curse of dimensionality → data increasingly sparse (d loss meaningful + # of data sample to cover the space grows exponentially)

→ **PCA**: finding a projection that captures the largest variation in data → unsupervised
normalize the dataset $A = D - \overline{D_{mean}}$ (mean of dimension i) ⇒ $Cov(A,A) = \frac{AA^T}{n-1}$ (Bessel correction) — symmetric ⇒ diagonalizzabile + autovettori ortogonali
- formula is valid since mean is null

Variance of projected data: $\frac{1}{n}\sum_{i=1}^{n}(\vec{u}^T\vec{x_i})^2 = \vec{u}^T\Sigma\vec{u}$

$\frac{\partial V}{\partial u} = \frac{\partial}{\partial u}\left[u^T\Sigma u + \lambda(1 - \vec{u}^T\vec{u})\right] = 2\Sigma\vec{u} - 2\lambda\vec{u} = 0 \Rightarrow \Sigma\vec{u} = \lambda\vec{u} \Rightarrow \vec{u}^T\Sigma\vec{u} = \lambda$ (unitary constrain)

then compute the eigen-vectors of $Cov(A,A)$ and discard some with lower eigen-values ⇝ put them inside E
$F = E^T A$ (new representation)
- unsupervised + computationally heavy (matrix inversion)

→ **Heuristic Search Attributes**: 2^d possible attributes combinations ⇒ greedy approach

1) best k attribute selection (given some metrics)
2) best step wise attribute selection (select the first, then the second conditioned to the first)
3) step wise attribute elimination (repeatedly remove the worst attribute)
4) branch and bound (selection + elimination)

Can use Chi test with the attribute and the output or using the **normalized mutual information feature selection**

mutual info: $I(X,Y) = \sum_x\sum_y p(x,y)\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$ → $I(X,Y) = 0$ when independent

$NI(f_i, f_j) = \frac{I(f_i,f_j)}{\min\{H(f_i), H(f_j)\}} \in [0,1]$ (compensates to MI bias towards multivalued features)

$G = I(c, f_i) - \frac{1}{|S|}\sum_{f_j \in S} NI(f_i, f_j)$ (output, attributes already selected) — redundancy penalization
- supervised method
- not just linear dependencies

→ **ATTRIBUTE CREATION** in order to get more informative/effective representations like with FFT

predominare $\min \|X\vec{w} - \vec{y}\|_2^2$

$\frac{\partial}{\partial w} f(w) = X^T(X\vec{w} - \vec{y}) = X^T X\vec{w} - X^T\vec{y} = 0$

$\vec{w} = (X^TX)^{-1}X^T\vec{y}$

· NUMEROSITY REDUCTION : reduce the # of data records or semplify their representation

→ PARAMETRIC METHODS ⇝ assume data fits some model, then store the parameters, discarding data

· Regression analysis : least squared error to get the best fit of a dependent variable against one or more independent one

$w_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$ $w_0 = \bar{y} - w_1\bar{x}$ or with more variables : $\vec{w} = (X^TX)^{-1}X^T\vec{y}$

· I can get polynomial checking X and adding the powers

→ NON PARAMETRIC METHODS ⇝ group in major families

· HISTOGRAM ANALYSIS → divide data in bucket and store avg (same width or same frequency / depth)
· CLUSTERING → store cluster representation (diameter and centroids)
· SAMPLING → choose a representative subset, simple random have poor performance if skew distribution · reduce the complexity of alg
- w/wo replacement or stratified sampling (partition dataset and sample accordingly) → eg classes

· DATA CUBE AGGREGATION → each dimension store an attribute, cell of the cube store aggregated measures
eg. date, location, product → store aggregated sales

· DATA COMPRESSION → lossy vs. lossless

3) DATA TRANSFORMATION → map the entire set values of one attribute with a replacement set, where every point got mapped

· smoothing, attribute construction, aggregation, normalization, discretization

min-max normalization → $v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$ (↳ new range to be mapped)

Z-score normalization → $v' = \frac{v - \mu_A}{\sigma_A}$

decimal scaling normalization → $v' = \frac{v}{10^j}$ j' smallest integer that $\max(|v'|) < 1$ $C = [-1, +1]$

· DISCRETIZATION attribute types : nominale (unordered set), ordinal (ordered), numeric ($\mathbb{R}$) can be merge (bottom-up) or split (top-down)

→ BINNING (unsupervised, split) → equal width - equal depth (intervals with the same # of samples)
→ BY CLASSIFICATION (supervised, split) → using decision tree to find the optimal cut point → entropy of partitions
→ BY CORRELATION (supervised, merge) → using a chi-merge algorithm → chi-squared between two neighboring intervals and the classes
· if the class is independent, I should merge the intervals df = #classes - 1
→ BY CLUSTERING (unsupervised split/merge)

→ FAYYAD and IRANI approach. optimal cuts point → between two examples of different classes in the sorted values of A
· I can take the midpoint

class entropy ← $H(S) = -\sum_{i=1}^{k} p(C_i, S) \cdot \log(p(C_i, S))$ (dataset, output classes)

· $H(S) = 0$ when I have only one class
· I want intervals with low class entropy → entropy as partitions quality measure

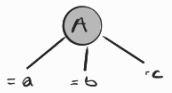
entropy of partitions ← $EP(A, T, S) = \frac{|S_1|}{N} H(S_1) + \frac{|S_2|}{N} H(S_2)$ (attribute, cut point)

1) Select the T_A with lowest EP → $S = S_1 \cup S_2$
2) partition recursively S_1 and S_2 until a termination condition is met (no thresholds defined)

· HIERARCHY GENERATION → from low level concept (eg. age) to higher level (eg. young, adult)
· need to define a partial / total order (for numeric data I need to discretize first)

1. street < city < state < country → schema level
2. {Urbana, Chicago} < Illinois → explicit data grouping
3. Automatic generation by occurrence counting → exception (day - week - month - year)

CLASSIFICATION → two step process: model construction + model usage / model testing (Eager Classifier)
· supervised or unsupervised (went to understand if ∃ classes)



DECISION TREE → internal node does a test on an attribute, branches are the different outcomes and leaf hold a class values
· no parameter setting, domain knowledge → perform automatically the feature selection process · initially all attributes are at the root
· termination when 1) no more sample 2) pure partition 3) no more attribute
· very interpretable · induction complexity grows exponentially with the height

· Greedy algorithm to find at each node the purest partition → different metrics

1) Information Gain $I(D) = -\sum p_i \cdot \log_2(p_i)$ (# bit to represent a class; p_i: class probability)

$I_A(D) = \sum \frac{|D_j|}{|D|} I(D_j)$ (mean # bits to represent a class after spitting A)

$Gain(A) = I(D) - I_A(D)$ (reduction in info requirements by knowing the value of A)

· I can use this for discretization → sort A, then do a binary split for the midpoint with the lowest $I_A(D)$ (lower # of classes in the splits)
· biased towards attribute with high # of values → eg. ID · the attribute with biggest gain will be selected · infeasible for big dataset

2) Gain Ratio $SplitInfo_A(D) = -\sum \frac{|D_j|}{|D|} \log_2 \frac{|D_j|}{|D|}$ (# of bits to express a partition) → if I have a lot of partition is bigger

$GainRatio_A(D) = \frac{Gain(A)}{SplitInfo(A)}$ select the highest · now bias towards unbalanced split

3) Gini index $gini(D) = 1 - \sum p_i^2$ · consider just binary splits $2^m - 2$

$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$

$\Delta gini(A) = gini(D) - gini_A(D)$ (impurity's reduction)

· Overfitting when I have too many branches → poor generalization $A_{test} \ll A_{train}$ → overfitting

· Generalization Error
1) optimistic $e_{TS} = e_{TR}$ (errors in D / |D|) 2) pessimistic $e_{TS} = e_{TR} + \frac{0.5 \cdot N}{|D|}$ (N: # of leaf in decision tree; add half sample error for each leaf)
3) use test set to get the overall 4) use prune set to get generalization error of a leaf or a sub-tree

↳ OCCAM RAZOR: for models with same A I should prefer the simpler one, since the other probabely captured also noise and outliers → more robust

· Repetition (attribute repeatedly tested, continuous) or Replication (duplicate subtree) → can be reduced by attribute construction from sparsely represented ones

· Pre-pruning → early stopping, but I need some threshold
· if number of instances are less then k
· if class is independent of available features (chi-squared test)
· if expanding the node doesn't improve the impurity measures (info gain or gini index ...)

· Post-pruning → removing branches and substituting with a leaf of the most frequent class

REDUCED ERROR PRUNING (REP) $gain = \#misclass(T_v) - \#misclass(v)$ (on prune set; if pruned in v; calculated on all non-leaf nodes) → prune the largest gains until only negative ones remain
· bottom-up restriction: T can be pruned ⟺ doesn't contain sub-tree with lower error than v

COST COMPLEXITY PRUNING $cost = resubstituting\text{-}error + \beta \cdot \#leaves$ (training) · start from the bottom, if the cost of the pruned is less ⇒ prune

1) bottom-up pruning if cost is less 2) order the trees at different steps by resubstitution error
3) Pick the tree with the smallest # of leaves within one std of error from the lowest cost tree

PESSIMISTIC PRUNING like the above but with training error + some value derived from statistic

entropies computations very efficient ←

RAIN FOREST → memory efficient algorithm · if training set doesn't fit in memory, with decision trees I need a lot of swaps

AVC-set (attribute, value, class label) is the projection of the training dataset to the attribute X and class values. The individual count is aggregated
· mantain one at each node to describe the tuples at that node

AVC-group: set of AVC-set for all the attributes at node n

AGE	Class	
	1	2
youth	4	2
middle	0	1
old	3	7

BOAT → use bootstrap sampling, where each fits into memory (↳ uniform with replacement) → get a set of trees that I use to get a unique T' that is very similar to what I would get normally

· Only two DB scan 1) small subset in order to get the underlying distribution
2) compute the final split with the info got from 1)
↳ read partition by partition the DB

· I can use it for incremental updates

BAYESIAN CLASSIFICATION

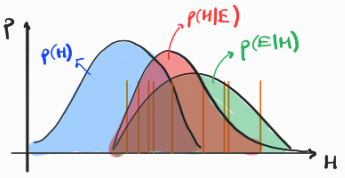
$$P(H|E) = \frac{P(E|H)\cdot P(H)}{P(E)} = \frac{P(E|H)\cdot P(H)}{P(E|H)\cdot P(H) + P(E|\neg H)\cdot P(\neg H)}$$

(posterior, likelihood, prior; product of two gaussian is gaussian (conjugate prior); P(E) = value since constant)

prior can be estimated from data $P(C_i) = \frac{|C_{i,D}|}{|D|}$

I want to find the max $P(E|H)\,P(H)$ — $x_k|c_i$, c_i

- assuming independence of attributes (losing accuracy because dependencies exists) $P(X|C_j) = \prod_{k=1}^{n} P(x_k|C_j)$
- If A_k is continous I will use a gaussian distribution $P(x_k|C_j) = g(x_k, \mu_j, \sigma_j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- I need to use the Laplacian Correction (adding 1 to each case)
- P(H) weights the likelihood, If we do not want to assume anything I can use non informative priors (uniform)

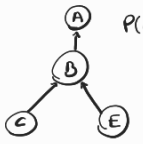


BAYESIAN BELIF NETWORKS → represent attribute dependencies via a DAG. I can also add hidden variables to the model

CPT (conditional probability table) → that maps input probabilities to the one of the node

$P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(Y_i))$ (values x; variable associated with x_i observation)

- A node is selected as an output with the class label



$P(A,B,C,E) = P(A|B)\cdot P(B|C,E)\cdot P(C)\cdot P(E)$

i	C,E	C,¬E	¬C,E	¬C,¬E
B	0.8	x	y	1-z
¬B	0.2	1-x	1-y	z

J ↓ K →

1) Given attributes and dependencies compute CPT
2) Given networks but assuming some vars are hidden → GD methods
   - randomly initialized weights (CPT entries)

$P_w(D) = \prod_{d=1}^{|D|} P_w(X_d) = \prod_{d=1}^{|D|} \prod_i P(x_i^d \mid \text{parents}(x_i^d))$ (i-th attribute; $\ln P_w(D) = \sum_{d=1}^{|D|} \sum_i \ln P(x_i^d \mid \text{parents}(x_i^d))$)

in the summation $P(x_i^d = v_{ij} \mid \text{parents}(x_i^d) = w_{ik})$ — when var is v_{ij} and has parents with value w_{ik} $\Rightarrow$ $\frac{\partial \ln P_w(D)}{\partial w_{ijk}} = \sum_{d=1}^{|D|} P(y_i = g_{ij} \wedge U_i = w_{ik} \mid x_d)\cdot \frac{1}{w_{ijk}}$ (variable; parents; is not 0 or 1 since some variables are hidden)

$w_{ijk} \leftarrow w_{ijk} - \alpha \frac{\partial \ln P_w(D)}{\partial w_{ijk}}$ $\qquad \sum_j w_{ijk} = 1$ (normalize)

DIFFERENCES
- trees are by design exclusive and exhaustive
- rules are stand-alone, trees I need to follow from root to leaf

3) attributes but not dependencies → search through the model space
4) no attribute nor dependencies → not doable

RULE BASED CLASSIFICATION → IF antecedent THEN consequent

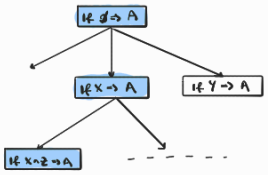
n_{covers} = # of tuples covered by a rule
$n_{correct}$ = # of tuples correctly classified by a rule
$coverage(R) = \frac{n_{covers}}{|D|}$ $\qquad accuracy(R) = \frac{n_{correct}}{n_{covers}}$

- CONFLICT RESOLUTION
  1) Size ordering → assign priority to the toughest rule with the most attribute
  2) Rule ordering → manages a list of rules and test it in order (majority class first) (implies the negation of the rules that come before)
     a) class-based ordering: decreasing order of prevalence or misclassification cost (no need to order inside a class)
     b) rule-based ordering: by some quality measures (accuracy, cover, size) or after advice of domain expert

- RULE MINING
  - from DECISION TREE → one rule per leaf, they are mutually exclusive (no conflict) and exhaustive (cover all possible cases) (one rule for attr-val)
    - pruned (estimated eg. by pessimistic approach)
      1) condition that doesn't improve the accuracy of the rule can be pruned
      2) any rule that doesn't improve the overall accuracy can be pruned

    After pruning the rules will no longer be mutually exclusive and exhaustive
    - I can class group the rules and find an order that minimize false positives → putting the set with lowest FP first
    - The default class can be the one with most uncovered tuples

  - HEURISTIC APPROACH → sequential covering algorithm (different from decision trees): rules learned one at the time, remove the tuples covered, until no more training example or returned rule has quality under a specific threshold



I use greedy depth-first search → from the empty one then consider each attributes I could test

optimistic evaluation — $FOIL\_GAIN = pos' \left(\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg} \right)$ (pos' correct; tuples of the class we are learning the rules; A accuracy)

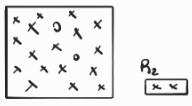
- favour rules that have high accuracy and cover many positive tuples
- until foil-gain > t ⇒ add predicate

To prune the rule I could use $FOIL\_PRUNE = \frac{pos - neg}{pos + neg}$ (evaluation of prune set)

- increase with the accuracy of R on Dprune
- if the pruned version have higher foil ⇒ prune it

- conflict creation that needs handling

cannot use accuracy for rule quality, because I could have perfect accuracy, but very small coverage



pos' = positive tuples that are covered by the new rule (correct)
neg' = negative tuples that are covered by the new rule (wrong) → new rule means with a predicate more

LEAZY LEARNERS → less time training, more in inference · use a richer hypothesis space that lead to better accuracy

- **K-NN** → instance based learning (use locally weighted regression)
  - for discreate values K-NN return the most frequent value across the K nearest training example from $\vec{x_q}$
  - voronoi diagram to visualize decision bounderies of 1-NN
  - $T(n,d,K) = O(n \cdot d \cdot K)$
  - with categorical values I can use the distance 0 if they are equal, 1 if they are different
  - if a value is missing from a tuple with all the others give the maximum distance
  - sorting and saving the training dataset to search-trees can reduce the # of comparison to $O(\log(n))$
  - can be parallelized · use editing methods to remove useless training tuples · use a subset of attributes and if above t I stop the computation
  - curse of dimensionality → less meaningful attributes must be eliminated
  - prediction with K-NN returning the means of the neighbours, I can also weight by the distance $w_i = \frac{1}{d(x_q, x_i)^2}$

  1) **WILSON EDITING** → clean overlap interclass regions → smother bounderies
     $\forall o_i \in O$ label it with the K-NN and remove o_i misclassified
  2) **MULTI-EDIT** → repeat wilson to N random subsets
     partition O in $N \geq 3$ random subset, classify S_i (subset) with $S_{(i+1) \bmod N}$ discard incorrect cases, repeat until no changes

     compression_rate = $(1 - \frac{r}{n}) \cdot 100$ (r = # of tuples after editing)
  3) **CITATION EDITING** → $\forall o_i \in O$ find K-NN and C nearest citers (where o_i is in their K-NN)
     classify o_i both with K-NN and citers and discard the misclassified
  4) **SUPERVISED CLUSTERING** → substitute O (set of objects) with O_r (set of cluster representative) that are cluster representatives (one for cluster)
     - strong complexity and space reduction, but also accuracy

- **CASE-BASED REASONING** → solve new problems with past solutions to similar one — ASSUMPTION: similar problems have similar solutions
  - symbolic representations (tuples, graphs not points) where I use knowledge based reasoning (inference) eg patient X has Y symtom, treated with Z
  - when a new solution is found I will update my case-base to increase its competence → learning

  a) retrieve → need to define a similarity function. When $|D|$ grown the efficacy of retrieve decrease · use particular data structures
  b) reuse → may need some adaptation — transformation using a function
     — generative approach (generate new from scratch)
  c) revise → get feedback as accuracy indicators
  d) retain → selective retention of new solutions, based on accuracy and the usefulness of using it (since it will grow the dataset)

ESEMBLE METHODS → use a combination of models in order to get a better accuracy

- **BAGGING**: averaging the prediction over a set of n classifiers
  - I get n dataset from bootstrap sampling each of which allow me to learn a different classifier M_i
  - → better accuracy since more detailed decision boundaries
    → computationally heavier, but highly preferable
  - by averaging I reduce the model variance make it more robust

  $\mathbb{E}_T\left[\left(\hat{f}(x) - \mathbb{E}_T[\hat{f}(x)]\right)^2\right]$ → over different training set from the same distribution $T_i \sim D$
  (estimated · mean prediction on T_i)

  - **RANDOM FOREST** where the M_i are decision trees (> 100 trees) — automatic estimation of attributes
    - create D_i dataset using bootstrap, then at each node randomly choose m attributes from the available → works well on big dataset
    - the tree is fully growned and not pruned → low probability of overfitting (majority of votes)

- **BOOSTING** assign weights to each tuple. Then learn sequentially the classifiers updating every time the weights, in order to weight more the one misclassified by M_i (M_{i+1} can focus). M^* combines the otputs weighted by M_i accuracy
  - better accuracy then bagging but also high risk of overfitting

  - **ADABOOST** initially $w_i = \frac{1}{d}$ At round K I get a bootstrap sample, learn M_i $\Rightarrow$ $err(M_i) = \sum_{j=1}^{d} w_j \, err(X_j)^{\in\{0,1\}}$
    - if a tuple is correctly classified I update the weight to $w_j' = w_j \cdot \frac{err(M_i)}{1 - err(M_i)}$ (< 0.5 otherwise random), then $w_j'' = w_j' \cdot \frac{\sum w_j}{\sum w_j'} = \frac{w_j'}{\sum w_j'}$ → keep the same sum as before (=1)
    - I assign a classifier weight as $W_i = \log\left(\frac{1 - error(M_i)}{error(M_i)}\right)$ → class with highest sum of weight vote wins
    - reduce the model bias by sequentially focusing on wrong errors

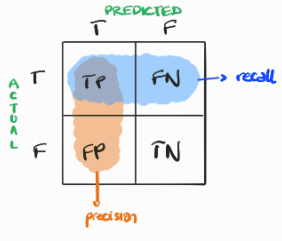
    $\mathbb{E}_T[\hat{f}(x)] - f(x)$; variance-bias trade off
    (estimated · true)

MODEL EVALUATION → accuracy for unbalanced dataset doesn't work (predicting always majority class leads with high accuracy)

- I need to use the confusion matrix → I would like it diagonal

$precision = \frac{TP}{TP + FP}$ ⇝ % of tuples that the classier labelled as positive are actually positive

$recall = \frac{TP}{TP + FN}$ ⇝ % of positive tuples that the label classified as positive



- I can also create a multi-class confusion matrix by considering all the other attributes as negative

F-MEASURE $F_\beta = \frac{(1+\beta^2) \cdot precision \cdot recall}{\beta^2 precision + recall}$ $\beta = 2$ weights recall twice as much

DATASET GENERATION for statistical evidence for comparing results of two models

1) **Hold-out** method: partition D in two randomly partitions K times, train K models, then calculate the metrics

2) **Cross-Validation**: partition dataset (training set) in K parts use D_i as test, all the others as train
- maybe it's better to perform pre-processing only on training so I should avoid to do it to all dataset
- LEAVE-ONE-OUT K folds with $K = \#$ of tuples (for small size dataset) $\rightsquigarrow$ test only on one tuple
- STRATIFIED cross-validation → maintain the same rate of majority/minority classes (better than the standard)

3) **Boot Strap**: sample uniformly with replacement maintaining the original dataset cardinality $P(x \notin D_i) = (1-\frac{1}{d})^d \approx \frac{1}{e} = 0.368$ (d = |D|) · get K dataset
$D \setminus D_i$ → test set (training)

$$Acc(M) = \frac{1}{K}\sum_i \left(0.632 \cdot Acc(M_i)_{test} + 0.368\, Acc(M_i)_{training}\right)$$

if measured only on test is negatively biased since only .632 is seen $\Rightarrow$ define this new estimator to correct the bias

M1 vs M2

1) **T-TEST** (parametric test, assume two gaussian with similar variance) → need to verify the assumptions before testing

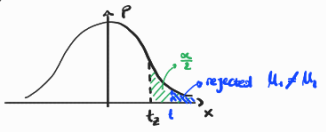
H_0: the two distribution for $err(M_1)$ and $err(M_2)$ are the same (equivalent to $\mu_1 - \mu_2$ since $\sigma_1 = \sigma_2$ for hypotesis)

$K-1$ degrees of freedom, with $K = \#$ of measurements

Use K folds and evaluate M_1 and M_2 on the same partitions

$$var(M_1 - M_2) = \frac{1}{K}\sum_{i=1}^{K}\left[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))\right]^2$$

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/K}}$$



If the two models are compared on different K-folds: $var(M_1 - M_2) = \sqrt{\frac{var(M_1)}{K_1} + \frac{var(M_2)}{K_2}}$ $df = \min\{K_1, K_2\}$

- If $|t| > t_z$ we lies in the rejection zone $z = \frac{\alpha}{2}$ → significance level because it is symmetric

2) **WILCOXON SIGNED RANK SUM TEST** (non-parametric, but variances must be similar)

H_0: the medians of the two distribution are equals

$d_i = x_i - y_i$ (samples from K-cross val) → then rank the $|d_i|$ with 1 for the minimum and so on

Label its rank with the sign of d_i → calculate W^+ and W^- $\left(W^+ + W^- = \frac{n(n+1)}{2}\right)$

- for H_0 the d_i will be centered around zero randomly: $W = \min(W^+, W^-)$ ($W^+ \approx W^-$) → use a table of critical values to find p of getting a more extreme value

If $\frac{n(n+1)}{2} > 20$ → normal approximation $\mu_W = \frac{W^+ + W^- - \frac{n(n+1)}{2}}{2} = \frac{n(n+1)}{4}$ $\sigma_W = \sqrt{\frac{n(n+1)(2n+1)}{24}}$

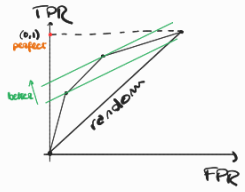
then I need to calculate the **z-score** $z = \frac{W - \mu_W}{\sigma_W}$ (know many std I'm outside the mean)

- If I have d_i that are equals → give the same rank as the mean (eg. 5.5) and reduce variance for each group of t tied ranks $\frac{t^3 - t}{48}$
- If $d_i = 0$ ignore that sample and adjust n accordingly

3) **AREA under the ROC CURVE (AUC)** → alternative to F-measures for imbalanced dataset

$$FPR = \frac{FP}{FP+TN} \qquad TPR = \frac{TP}{TP+FN} \text{ (recall)}$$

→ area below ROC curve (convex hull) is a measure of the model accuracy independently from model thresholds → single number



- the diagonal is like random choice because $TPR = FPR$

	predicted	
actual	0.8	0.2
	0.8	0.2

$$cost = \underbrace{FPR}_{\text{prob of FP in } D} \cdot \underbrace{P(neg)}_{\text{a priori prob}} \cdot \underbrace{c(Y,n)}_{\text{cost of FP (class)}} + \underbrace{FNR}_{\text{prob of FN in } D} \cdot P(pos) \cdot c(N,P) \quad (FNR = 1 - TPR)$$

cost of a classifier

iso-cost lines $\Rightarrow 0 = dFPR \cdot P(neg) \cdot c(Y,n) - dTPR \cdot P(pos) \cdot c(N,P)$

$$\rightarrow m = \frac{dTPR}{dFPR} = \frac{P(neg)\, c(Y,n)}{P(pos) \cdot c(N,P)}$$

→ useful to get the min cost point of ROC and select right thresholds

- I can compute the AUC for each trial and then perform a statistic test on the result of AUC on the two models comparing single point ROC curves and comparing them statistically instead of using AUC of the full curve

IMBALANCED DATASET → models will be biased towards the majority class

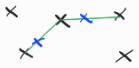
→ I need to rebalance just the training, not the test, otherwise the metrics will be biased

1) oversampling 2) under-sampling → loose information 3) threshold moving - from 0.5, so that rare class tuples are easier to classify (less FN errors)
4) ensamble methods (boosting) 5) class weight, that modify the loss function

- **SMOTE** (synthetic minority over-sampling technique) → create synthetic data instead of sampling (for minority class, typically improve AUC)

$\forall$ point from the minority class compute the K-NN (from K points select the right number based on the oversampling required, like 2 out of 5 for a +200%)
- then compute the difference between that point and one selected from K-NN and multiply by $\alpha \sim U(0,1)$ $p' = p + \alpha\, d_{difference}$ ($diff = p'_{knn} - p$)

- pay attention to test it on original data



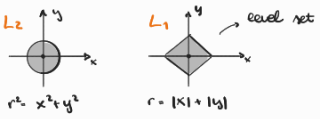
CLUSTERING

CLUSTER → set of objects that are similar within the cluster and dissimilar with other objects
CLUSTER ANALYSIS → process of finding similarities between data and grouping similar data objects into clusters → unsupervised learning
Can be used as **preprocess** (data compression, outliers) or **stand-alone** tool

Similarity is expressed with a distance function $d(i,j)$ → change for the type of attribute
- high intra-class sim. and low inter-class sim.

• L_2 I can get only **circular** clusters (rotationally symmetric)
can be **weighted** based on application data semantics → weighted L_2 I can get **ellipsoid** clusters

Quality of clustering → separate function that measure the quality. It's highly subjective



- single level vs. hierarchical partitioning → useful to have different clustering's level
- distance-based vs. connectivity-based similarity → euclidian or density
- exclusive vs. non-exclusive → sample can be associated with more clusters

Assessing clusters tendencies → evaluate if data set contains meaningful clusters

HOPKINS STATISTIC → spatial test to understand if a variable is randomly distributed across a space (uniformly distributed)

1) sample n points $\{p_1, \dots, p_n\} \subseteq D$ for each find it's nearest neighbour in D $\quad x_i = \min_{v \in D,\, v \neq p_i} dist(p_i, v)$
2) draw n points $\{q_1, \dots, q_n\} \sim U(D_{max}, D_{min})$ → same variation as D $\quad y_i = \min_{v \in D} dist(q_i, v)$

→ $H = \frac{\sum y_i}{\sum y_i + \sum x_i}$ If uniform $H \approx 0.5$, otherwise $\sum x_i \ll \sum y_i \Rightarrow H \approx 1$

H_0: data is uniformly distributed
- compute H for several $\{p_1, \dots, p_n\}$ samples and average it
If $H > 0.75$ => cluster tendency with 90% of confidence

1) PARTITIONING METHODS → construct partitions and evaluate them, then iterative approach to relocating object from one partition to another

- I want to find K cluster such that they minimize the sum of squared distances with the cluster centroid/medoid $\quad E = \sum_{i=1}^{K} \sum_{p \in C_i} (p - c_i)^2$ (c_i: centroid of C_i; C_i: cluster) $\quad d(p, c_i) = L_2$

→ I'm forcing the **compactness** of the clusters, hope to have **separation** as side effect $\quad$ (when $K = |D| \Rightarrow E = 0$)
→ cannot use this function to select K because it will always **decrease**

• K-MEANS

c_i → centroid of the cluster $\quad c_i = \frac{1}{|C_i|} \sum_{p \in C_i} p$

1) randomly assign K objects from D as centroid
2) reassign each object to the K clusters
3) recompute the centroid of the cluster

for each point n find the nearest cluster $O(Kn)$ recompute cluster centers $O(n)$, repeat at most t iterations

- $O(tKn)$ and since $t, K \ll n \Rightarrow$ linear with the number of objects
- unfortunately terminates at local optimum → multi-start approach
- need to specify K
- not suitable for discovering non-convex shapes (only spherical) → since clusters centers computed with the mean → L_2
- can be used only for continuous $\mathbb{R}^n$ data (I would not know how find the center c_i) → I can use **K-modes** for categorical data
  - sensitive to outliers
  - category with highest frequency
  - change distance function for categorical
  - use mode to find new centroids (most frequent)

ELBOW METHODS → determine the optimal K and if there are clusters
Increasing K will decrease the intra-class variance of each cluster. I found the turning point when split cohesives clusters will not result in a big decrease $\quad J = \sum_{i=1}^{K} var(C_i)$

• K-MEDOIDS

medoid is the most centrally located object in a cluster
- can be used with more **d-type** and is more robust to **outliers** respect to K-means

1) randomly choose K representatives $\{o_1, \dots, o_K\}$
2) assign each object to the nearest representative o_i eg. with L_1 similarity
3) for each o_j randomly choose o_{random} and compute the cost of swap them $\quad S = E'_{new} - E_{old} = \sum_{i=1}^{K} \sum_{p \in C_i} |p - o_i'| - \sum_{i=1}^{K} \sum_{p \in C_i} |p - o_i|$
when $S < 0$ => replace them
(o_{random}: one in another cluster; possible reassignment)

PAM (original algorithm)

o_1 has the smallest $\sum_{i=1}^{n} d(i, o_1)$ → still local optimum because it cannot search through all possibilities (depends on initialization, I could be two swap from optimum)
o_j, $j > 1$ decreases the $\sum_{i=1}^{n} \min_{t=1,\dots,j} d(i, o_t)$ as much as possible → BUILD PHASE: greedy choice (we do not explore every possibility)

SWAP PHASE: then consider all pairs (i, j) : $i \in \{o_1, \dots, o_K\} \wedge j \notin \{o_1, \dots, o_K\}$ (non-medoid) $\Rightarrow$ swap $i \leftrightarrow j$ that decrease the objective the most

- more robust to outliers then K-means
- doesn't scale well for large dataset $\quad O(K(n-K)^2) \approx O(n^2)$
  consider for each medoid the $n-K$ possible swaps and recompute for each S that cost $O(n)$
  $O(K(n-K)n) \approx O(K(n-K)^2)$ ($n \gg K$) for each step

CLARA (improvement from PAM) → apply on initial sample of the dataset and apply PAM ↗ the size will affect accuracy and complexity

- if it is representative the medoids of the sample should be the one of the dataset
  → cannot find the best medoids if they are not in the sample
- to improve the approximation I can draw more samples and returning the best one

1) for m times draw a data sample and use PAM to get the K meroids
2) assign each object on the entire D to $\{O_1, \dots, O_K\}$
3) If the average dissimilarity on D is less then the current minimum ⇒ new medoid found

$O(m[Ks^2 + K(n-K)])$ → can be used for larger dataset (s = sample size; assign each n-K remaining objects to the K medoids; PAM $O(K(s-K)s) \simeq O(Ks^2)$)

- I can introduce bias and accuracy loss for the sample

CLARANS (randomized clara) → clustering as searching a graph where each node represent the set of K meroids

- Two nodes are neighbour if they differ by just one medoid
- each node is associated with a cost that is the total dissimilarity → search a minimum
- At each step I will search all the neighbours and select the one with highest descend → K(n-K) connections per node, evaluate only m (constant)
  ↳ every medoid could be swapped by n-K possibilities
- At each step I draw sample from the neighbours to examine → this is better then CLARA since I will not limit the search area
- I can search sequentially more local optimum, is a parameter of the algorithm
- more efficient and scalable then PAM/Clara + better quality $O(mn) = O(n)$ ↗ for each step compute m times S and swap

- All three alg. can be seen as a graph search where PAM examins all the neighbours, CLARA draws an initial sample and search it all and CLARANS draw a random neighbours samples

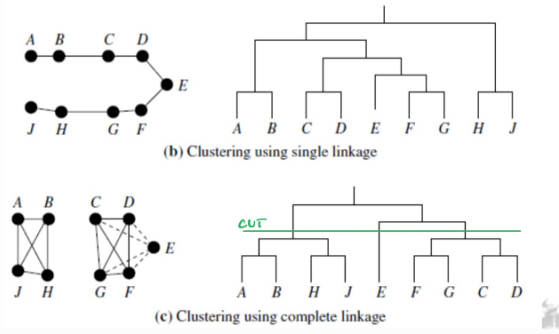
2) HIERARCHICAL METHODS → create an hierarchical decomposition of the set

- once a step is taken it cannot be undone

- Use distance matrix as clustering criteria → no need for K but a termination condition
  ↳ connectivity matrix
- I need to define the distance between subset that can be constructed from the connectivity matrix, such distance is called LINKAGE METRIC
  → will affect the resulting clusters · defined as: $d(C_1, C_2) =$ operation $\{ d(x,y) : x \in C_1, y \in C_2 \}$

1) minimum → single link (NN) distance of two clusters is the distance of their closest object → represent long chains } susceptible to outliers
2) maximum → complete link (furthest neighbour) : perform well with natural clumps, not with chains
   · tends to minimize the diameter increasing of the cluster at each iteration
3) average → pair-group average ; works well with clumps or elongated clusters

DENDOGRAM is a tree structure commonly used to represent the process of hierarchical clustering
→ A cluster is obtained by cutting the dendogram at the desired height
⇒ each connected component is a cluster



1) Single linkage algorithm terminates when the minimum distance between nearest clusters is above t

If I see data as nodes ⇒ merge two cluster is adding an edge between the nearest objects
· the algorithm is also called minimal spaning tree algorithm because they are equivalent

2) Complete linkage terminate when the maximum distance between nearest clusters exceed t →
each cluster is a complete subgraph, with edges all the node in the cluster

→ have at least $O(n^2)$ but in general gives better quality clusters

AGNES (agglomerative nesting) → uses single link methods (merge)

1) Each point in a different cluster
2) with distance matrix found the nearest tuples and merge the clusters (semplify the dissimilarity matrix)

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

d=1 →

	AB	CD
AB	0	2
CD	2	0

DIANA (divisive analysis) → uses average dissimilarity, inverse order of AGNES (split)

· look the example on how to do it with dissimilarity matrix

1) start with a large cluster containing all the points
2) Select the cluster with the highest intra-dissimilarity (distance between two points) → largest diameter
3) find the most disparate sample (largest average dissimilarity) → create a splinter group and populate the new cluster with points of old cluster that are closer to the splinter
   → row mean in the dissimilarity matrix

$\forall i \notin$ splinter $\quad D_i = \text{avg}_{j \notin \text{splinter}} d(i,j) - \text{avg}_{j \in \text{splinter}} d(i,j)$

· find the highest and if $D_i > 0$ ⇒ move it to the splinter group
· repeat until all $D_i \leq 0$ → recalculate since D_i will change (new point splinter)

4) repeat 2-3 until all clusters contain one point

BIRCH (balanced iterative reducing and clustering with hierarchies)

Agglomerative clustering → type of hierarchical clustering that proceed by merging

for large amount of data → differently to other algorithms tries to minimize I/O because data can possibly not fit in memory

$O(n)$

CLUSTERING FEATURES (CF) → summary of 0-th, 1-st and 2-nd moment of a cluster $CF = (N, LS, SS)$

$N \rightarrow$ # of points in the cluster
$\vec{LS} \rightarrow$ linear sum of the points in the cluster $\sum_i \vec{x}_i \in \mathbb{R}^p$
$SS \rightarrow$ squared sum $\sum_i |\vec{x}_i|_2^2 \in \mathbb{R}$

I can use it to compute:

1) centroid of the cluster $\vec{x}_0 = \frac{\sum \vec{x}_i}{n} = \frac{\vec{LS}}{N}$

2) radious (avg sqr root of squared distances from centroid) $R = \sqrt{\frac{\sum |\vec{x}_i - \vec{x}_0|_2^2}{N}} = \sqrt{\frac{2|\vec{x}_i|_2^2 - 2\vec{x}_0 \cdot 2\vec{x}_i + 2|\vec{x}_0|_2^2}{N}} = \sqrt{\frac{1}{N}\left[SS - \frac{\vec{LS}}{N}\cdot\vec{LS} + N\frac{|\vec{LS}|_2^2}{N^2}\right]} = \sqrt{\frac{N \cdot SS - |\vec{LS}|_2^2}{N^2}}$

3) diameter $D = \sqrt{\frac{\sum_i \sum_j |\vec{x}_i - \vec{x}_j|_2^2}{n(n-1)}} = \sqrt{\frac{\sum_i \sum_j (|\vec{x}_i|_2^2 + |\vec{x}_j|_2^2 - 2\vec{x}_i \cdot \vec{x}_j)}{N(N-1)}} = \sqrt{\frac{2NSS - 2|\vec{LS}|_2^2}{N(N-1)}}$

→ since they are all summations I can just update CF by adding the new elements or another CF when merging

CF-TREE is an height balanced tree (hierarchical representation of clusters) (where $|h(T_L) - h(T_R)| \leq 1$)

- Each node maintains the CF for each entry

I need the following parameters:

1) B (branching factor) → max # of children per non-leaf node → regulates the tree's height
2) T (threshold parameter) → maximum diameter of the clusters in the leaf node (no subcluster can have it higher)
3) L → max # of entries in a leaf

Phase 1: build an in-memory CF-tree incrementally. For each point that I consider:

1) If T violated (it will increase too much the diameter of the closest cluster)
   a) If there is space → insert it as a new cluster in the leaf
   b) If L violated ⇒ take the two further entries and create two nodes, put the remaining entries to the nearest leaf
      - update the entries and CF of the parent
      - we may split the parent as well (B violated). Splitting the root will increase the tree's height
2) otherwise insert it into the closest cluster (update CF)

→ If during tree construction CF tree exceed memory available I can reconstruct the partially built tree in a smaller one from old tree leafs (no need to rescan points) (increase the value of T threshold) · each leaf CF entry is treated as a pseudo-point

Phase 2 (optional): builds smaller CF-tree by increasing T from leafs

Phase 3: use an arbitrary cluster method for the CF-tree's leafs entries, that are treated as points (representative of micro clusters)
→ merge microclusters together · example I want to find K clusters

Phase 4 (optional): scan the entire dB to label point
→ I may read some outlier handling that doesn't fit in any cluster

- handles only numeric values and is sensitive to their order insertion
- cluster may not be so natural for the threshold given
- tends to be spherical (cannot exceed the given diameter)

CHAMALEON → measures similarity with a dynamic model

- two clusters are merged if the interconnectivity (internal strenght) and closeness (proximity) are high relative to their internal measures

Construct the graph K-NN of the data → partition the graph into sub-clusters (→ graph-partitioning algorithm) → agglomerative clustering of the sub-clusters (merge phase)
(→ agglomerative hierarchical clustering algorithm)

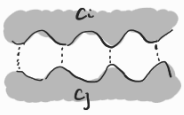
1) K-NN captures the concept of neighborhood dynamically → important to set K correctly

- the radius of the neighborhood is inversely proportionate to the density of the region
- the density is calculate from edge weight → each edge get a weight based on how close the two nodes are
  - used to calculate inter-connectivity

2) Partition phase → a cluster is partitioned into subclusters C_i and C_j to minimize the sum of weights (edge cut) of the edges that would have been cut → absolute interconnectivity $EC_{\{C_i,C_j\}}$

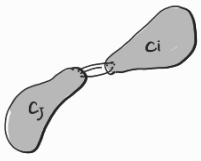
- both $C_i \wedge C_j$ must contain at least 25% of C nodes
- repeatedly select the largest sub-cluster (graph) and use the partition algorithm to bisect it until the largest contains fewer vertices than a specified number (1% - 5% of overall data)

3) Agglomerative hierarchical clustering based on similarity that takes into consideration RI and RC



$$RI(C_i,C_j) = \frac{|EC_{\{C_i,C_j\}}|}{\frac{1}{2}\left[|EC_{C_i}| + |EC_{C_j}|\right]}$$

- $|EC_{\{C_i,C_j\}}|$: absolute inter-connectivity (sum of weights of vertices between C_i and C_j)
- relative interconnectivity
- strenght of the structure
- $|EC_{C_i}|$, $|EC_{C_j}|$: internal inter-connectivity → minimum sum of weights of edges which would be cut by partitioning C_i into two roughly equal parts



$$RC(C_i,C_j) = \frac{\bar{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\bar{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\bar{S}_{EC_{C_j}}}$$

- $\bar{S}_{EC_{\{C_i,C_j\}}}$: avg weight between C_i and C_j
- $\bar{S}_{EC_{C_i}}$, $\bar{S}_{EC_{C_j}}$: avg weight of the edges that belongs to the min-cut bisector of C_j (same as for RI)
- measures the compactness

→ Merge the clusters that has RI and RC above T_{RI} and T_{RC}

- Visit all C_i and merge C_j with the highest absolute inter-connectivity $EC_{\{C_i,C_j\}}$ that satisfies $RI(C_i,C_j) \geq T_{RI} \wedge RC(C_i,C_j) \geq T_{RC}$

- The time complexity is $O(n^2)$ in the worst case → arbitrarily shaped clusters with high accuracy

DENSITY-BASED METHODS → clustering based on densities

- discover clusters of all shapes
- handle well noise
- one scan
- needs density parameters as termination conditions

Needs two parameters
- Eps: max radious of neighbour (hypersphere)
- MinPts: min # of points inside an ε-neighbour to consider it dense → the point is called CORE OBJECT

- directly density-reachable: an object p is dir from q wrt ε, MinPts ⟷ 1) $p \in N_{eps}(q)$ 2) q is a core object $|N_{eps}(q)| \geq MinPts$
- density reachable (not symmetric (if p not core object)): an object p is dr from q wrt ε, MinPts ⟷ $\exists (p_1,\dots,p_n)$ with $p_1 = q$, $p_n = p$: p_{i+1} is dir from p_i
- density connected: an object p is dc from q wrt ε, MinPts ⟷ $\exists o$: p and q are dr from o
  - equivalence relation (a-a, transitive, simmetrica)

DBSCAN → cluster is a maximum set of density-connected points

- no problem with the noise/outliers
- custom shape
- $O(n^2)$

- search the ε-neighbourhood of each point in the DB if above MinPts ⇒ create a new cluster and iteratively add the neighbourhood until no other point can be added
  - otherwise noise

→ the alg collects iteratively directly density reachable points

- noise → not core points (could be border region of a cluster)
- outliers → noise that is not in any neighbourhood (not in clusters)

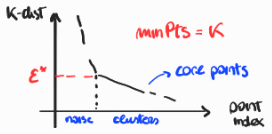
- **Closure of density-connectedness** to form clusters → initially just a point, then applying the binary relation of density-connection I create the maximum closed set that is called cluster

  - if I would use the dr for the cluster I would have missed points if going in this sequential manner I would have need merge

A subset $C \subseteq D$ is a cluster $\iff$ 1) internally connected $\forall o_1, o_2 \in C$ o_1 and o_2 are density connected
2) maximality $\nexists\, o_1 \in (D \setminus C),\ o_2 \in C$: o_1 and o_2 are density connected

The parameters (ε and minPts) are valid for all the space, but I could have clusters with different density

- we define a function called **K-dist (p)** = distance with the K^{th} NN
  → sort the point in descending order and plot it



OPTICS (ordering points to identify clustering structure)

- global density parameter cannot characterized the intrinsic cluster structure

  → I could use an hierarchical approach
  - single link can merge two clusters just for a single connection chain
  - dendograms when I have a lot of points are difficult to understand

  → I could use density based algorithm but the # of parameters are infinite
  → if I discretize still the results would be hard to analyze

Density clusters are **monotonic** wrt their ε → with fixed minPts higher density clusters (lower ε) are completely contained by lower densities clusters (higher ε)

Given ε and MinPts I can calculate the following properties of a point:

- **core distance**: smallest ε' such that p is a core-object → distance from P to the MinPts NN
  - if p is not a core-object ⇒ c-d is undefined

- **reachability distance** of object q from p is the minimum radious that makes q directly density-reachable from p

  $r\text{-}d = \max\{ c\text{-}d(p),\ dist(p,q) \}$ ← q must be on it's neighbourhood
  ! p must be a core obj

  not a real distance because not symmetric

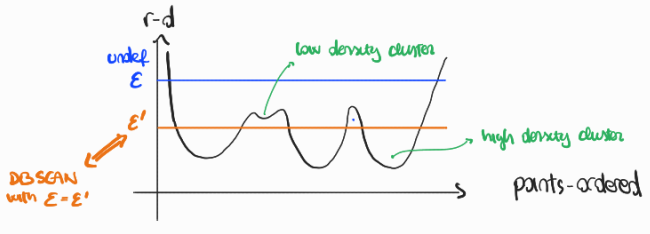
  - can be undefined

1) Select a point from the input DB (or OrderSeed if not empty). Retrive its ε-neighbourhood, calculate c-d and set its r-d to undefined

   - if p is not a core object ⇒ move with next element of OrderSeed (or DB if empty)
   - if p is a core object ⇒ $\forall q \in \varepsilon$-neigh(p) update their reachability distance from p and insert q into OrderSeed if not already processed

2) The object inside the **OrderSeed** are sorted by their reachability distance to the closest core obj
   → The next step of the while will retrieve the first element of OrderSeed
   → once a point is examined is saved to a file with core-distance and its current minimum r-d
   - The program output is a list of object (file) in the order the objects have been processed
   - This is not a clustering algorithm since I don't assign objects but is just an ordering

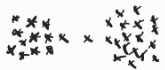
$O(n^2)$



→ I can get the clusters with different densities parameters without re-executing the algorithm but just looking at the order

- wider valley correspond to more significant clusters
- peaks are threated like noise

- with minPts = 2 ⇔ single link clustering
  -chaining effects problem with low minPts



DENCLUE → density algorithm can be highly sensitive to the value of radius used

- It is a clustering methods based on a set of **density distribution functions**
  - each observed object is treated as an indicator of high-probability density in the surroundings (depends on distance)

Kernel density approximation of the probability density function: given $\{x_1, \dots, x_n\}$ iid from f random variable $\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$

KERNEL: $K : \mathbb{R} \to \mathbb{R}^+$ must satisfy $\int_{-\infty}^{+\infty} K(u)\,du = 1 \;\wedge\; K(-u) = K(u)$

an example $K\left(\frac{x-x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}$ or $K_{square}\left(\frac{x-x_i}{h}\right) = \begin{cases} 0 & \text{if } \frac{|x-x_i|}{h} > 1 \\ 1 & \text{otherwise} \end{cases}$

I can determine the clusters mathematically by identifying **clusters attractors** (local maxima of $\hat{f}$)

- to avoid trivial local maxima I use a threshold where I consider x^* : $\hat{f}(x^*) \geq \xi$
- x^* are centers of clusters → to assign points to x_i^* I will use a **step-wise hill-climbing**

$$x^0 \leftarrow x \qquad x^{j+1} \leftarrow x^j + \delta \frac{\nabla \hat{f}(x^j)}{|\nabla \hat{f}(x^j)|} \qquad \nabla \hat{f}(x) = \frac{1}{nh} \sum K'\left(\frac{x - x_i}{h}\right) \cdot \frac{1}{h}$$

- It will stop when $\hat{f}(x^{k+1}) < \hat{f}(x^k)$ and assign x to attractor x^k
- I keep a list with points that have $d(x^j, x^k) \leq \frac{h}{2}$ and will assign these point to $x^* = x^k$ without re-applying the hill-climbing
  ↳ for each step of the climb $1 \leq j \leq k$

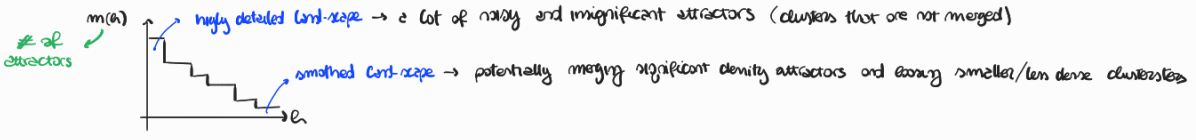
Noise or outliers are object x that converge to x^* : $\hat{f}(x^*) < \xi$

In order to get arbitrarely shaped clusters I need to merge density attractors

- A cluster $C \subseteq D$: $\forall x \in D,\ \exists x^* \in X$ (attractors set) : $\hat{f}(x^*) \geq \xi$ and the rule below $(\forall x_1^*, x_2^* \in X)$
- I can merge two density attractors $\iff$ $\exists$ a path $P \in F^d$ (features space in d dimension) from x_1^* to x_2^* where $\forall p \in P$: $\hat{f}(p) \geq \xi$

→ solid and compact math foundations. Extremely faster then existing density algorithms $O(n \log n)$

But need an accurate choice of h (determines the influence of the point in its neighbourhood) and ξ (avoid noisy and insignificant attractors)



I should choose $h : [h_{min}, h_{max}]$ for constant m(h) is larger → stable structure

- If the dB is noise-free all density attractors of D are significant → $0 < \xi \leq \min_{x^* \in X} \{\hat{f}(x^*)\}$

1) The minimal bounding hyper-rectangle of dataset is divided into d-dimensionse cubes, with edge lenght of 2h (grid computations to speed up the algorithm)

- numbered according to relative position from the origin : these keys will be stored in search-trees for easy retrive
  - for each populated cube (that has points inside) store key, # of points inside (N_c), pointers to those points and $\sum_{x \in C} x$
  - connect neighbours cube $c_1, c_2 \in C_p \iff d(mean(c_1), mean(c_2)) < 4\sigma$ → if K is gaussian $\sigma = h$
    ↳ efficiently computed by the linear sum

2) Only highly populated cubes (C_{sp}) and the are connected to them are considered in determine clustering

$$C_{sp} = \{c \in C_p \mid N_c \geq \xi_c\} \qquad C_r = C_{sp} \cup \{c \in C_p : \exists c_s \in C_{sp} \text{ connected to } c\}$$

$x \in C$ and $c, c_i \in C_i$ we set $near(x) = \{ x_i \in c_i \mid d(mean(c_i), x) < k\sigma \wedge \exists \text{ connection}(c_i, c) \}$ (eg. 4 allow to neglet marginal influences)

$\hat{f}(x) = \sum_{x_i \in near(x)} e^{-\frac{d(x, x_i)^2}{2\sigma^2}}$ (local density) $\Rightarrow$ take the hill-climbing with $\hat{f}$

Should'nt I consider all distances to connected cubes?

CONSIDERATIONS for DENSITY-BASED CLUSTERING

- not effective when high dimension for clustering data → sparse space
  - unlikely that points are closer to each other rather then the avg distance of points

GRID-BASED CLUSTERING

→ use multi-resolution grid data structure, quantized space → fast processing time but loose in Accuracy

STING (statistical information grid)

- each cell is partitioned into a # of smaller cells → at the lowest level are computed from db
- parameters of higher cells can be easily derived from smaller cell (μ, σ, min, max, #, type of distribution) (tested with a stat. test)
- top-down approach to answere spatial queries → find regions with particular constraints or return attribute of a region
  ↳ I can start also from the non-top layer
- for each cell I compute the confidence interval reflecting the relevance of the cell in the query
  - calculated using the statistical parameters of each cell
  - label them as relevant or irrelevant → then proceed to the lower level until the bottom is reached returning the relevants squares
- simple to parallelize, generation of clusters $O(n)$, query processing time $O(g)$ (# of cell at the bottom)
- Only vertical or horizontal boundaries of clusters → no diagonal
- Is an approximation of region returned by DBSCAN as granularity → 0

CLIQUE

→ identifies sub-spaces of the original high dimension space in which to perform a better clustering

- density-based + grid-based
- partitions each dimension by the same # of equal lenght intervals

- a unit is dense if the points in the unit are more then a threshold
- A cluster is a maximal set of connected dense units within a subspace

Identify dense units by the APRIORI principle

K-th dimensional unit is dense => its projection in (K-1)-dimension are dense.
Therefore the candidates for being dense units are the ones generated by the dense (K-1) dense units

1) Identify in every dimension the dense intervals
2) Iterattly join two K-dimensional dense cells C_1, C_2 (the join defined in this way avoid duplicates)
   with $C_1 \leq < D_{i_1}, \dots, D_{i_K} >$, $C_2 \leq < D_{i_1}, \dots, D_{i_{K-1}}, D_{j_K} >$ (share same K-1 dimensions and interval in these) => generate a (K+1) dimensional unit $< D_{i_1}, \dots, D_{i_K}, D_{j_K} >$
3) Check if I new has the necessary # of points specified by threshold → terminates when I cannot join anything or no dense results (→ and also if the other K-subspaces are dense?)
4) Assembly dense cells in each subspace to find clusters → find a convenient representation
   - Minimum Description Length principle (MDL) → use maximal regions to cover connected dense cells
     ↳ hyper-rectangle containing only dense cells, and cannot be expanded in any dimension
   - the best description of a cluster is NP-hard, CLIQUE use a greedy approach, where it starts from a cell and then try to expand the rectangle in any dimension. Then add a new rectangle

   → terminates when all the dense cells are covered

- scales linearly with the # of data in input and also with the dimensions of the data
- needs proper tuning of grid-size and density threashold → I get get low accuracy for the simplicity of the method
- automatically find sub-spaces of highest dimensions in which the clusters exist (the projections)

EVALUATING CLUSTERS → assessing cluster tendencies ($x \neq U$), # of clusters and clustering quality

CROSS VALIDATION METHOD → measure the # of clusters → use a part to perform evaluation (eg. sum of squared distances from nearest centroid)
$\forall k > 0$ that is interesting I can take the avg of the m results and comparing the overall quality

CLUSTERING QUALITY MEASURES

1) EXTRINSIC → supervised $Q(C, C_g)$ → quality of cluster C compared to the ground-truth C_g

must satisfy:
a) homogeneity → the purer the better
b) completeness → assign all equal class obj to the same C_i
c) Rag Bag → putting wrong obj to a pure cluster should be penalized more then putting inside a miscellaneous cluster
d) Small cluster preservation: splitting a small category to pieces is more harmful then splitting a larger one

BCUBED → evaluates precision and recall for every object

precision → indicates the purenes of that category inside my cluster
recall → indicates the completeness of the category into the real cluster

Let $D = \{o_1, \dots, o_n\}$ a set of object and C a cluster on D

Let $L(o_i)$ the object category and $C(o_i)$ what the clustering assigned (a cluster ID)

$$\Rightarrow \text{correctness}(o_i, o_j) = \begin{cases} 1 & \text{if } L(o_i) = L(o_j) \iff C(o_i) = C(o_j) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Precision} = \frac{1}{n}\sum_{i=1}^{n} \frac{\sum_{o_j : i \neq j,\, C(o_i) = C(o_j)} \text{correctness}(o_i, o_j)}{|\{o_j : i \neq j,\, C(o_i) = C(o_j)\}|}$$

→ # of correct clustering inside of o_i's cluster
→ # of object on the cluster of o_i

- hope they tends to one

$$\text{Recall} = \frac{1}{n}\sum_{i=1}^{n} \frac{\sum_{o_j : i \neq j,\, L(o_i) = L(o_j)} \text{correctness}(o_i, o_j)}{|\{o_j : i \neq j,\, L(o_i) = L(o_j)\}|}$$

→ # of correct clustering inside o_i real cluster
→ # of object on the true cluster of o_i

2) INTRINSIC when ground true is not available → evaluate clustering based on compactness and separation

SILHOUETTE coefficient $\forall o \in D$ compute:
1) $a(o)$ = avg distance between obj and it's cluster obj
2) $b(o)$ = min avg distance to a cluster where o doesn't belong to

$$Q(o) = \frac{\sum_{o' \in C_i, o \neq o'} dist(o, o')}{|C_i| - 1}$$ (low better)

$$b(o) = \min_{C_j : j \neq i} \left\{ \frac{\sum_{o' \in C_j} dist(o, o')}{|C_j|} \right\}$$ (high better)

$$S(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}$$

$S(o) \in [-1, +1]$ (+1 preferable)

when $s(o) < 0$ ⇒ a different cluster is in avg nearer → should be avoided

- then I can take the avg of all points
- usable ⇔ the clusters are convex

MINING FREQUENT PATTERN

PATTERN: - a set of items $\{milk, bread\}$
- a subsequence $(PC, monitor)$ → the order matters
- a substructure pattern → like a subgraph or a sub-tree

FREQUENT PATTERN is a pattern that occurs frequently in a data set
→ finding them is essential for mining associations and find correlations

Itemset $I = \{I_1, \dots, I_m\}$ **K-itemset** $X = \{I_1, \dots, I_k\}$

Absolute Support of X → frequency of occurrence of an itemset (how many times it shown up in D)

Relative Support s = fraction of transactions that contain X → probability of X in being on a transaction

- An itemset X is frequent when its support is bigger then a threshold

Transaction T is a set of items: $T \subseteq I$

- an **association rule** is an implication $X \Rightarrow Y$ where $X \subset I$, $Y \subset I$, $X \cap Y = \emptyset$

- **Support** of $X \Rightarrow Y$ on D (transaction DB) → $supp(X \Rightarrow Y) = P(X \cup Y)$ → probability that a $T \in D$ is $T \supseteq X \cup Y$
  - reflects usefulness of the rule (in 2% of transactions are purchased together)

- **conf** $(X \Rightarrow Y) = P(Y|X) = P(X \cap Y) / P(X) = \frac{supp(X \cup Y)}{supp(X)}$
  - reflects certainty on the rule (70% of who bought a PC also bought a TV)

I can associate each item a boolean variable and create a **boolean vector** to keep track of what do I have in a transaction (not how many)

In general to mine rules:

1) Identify all frequent itemset (at least min-supp occurences) — computationally heavy phase
   - if an item is not frequent, useless to add more to it because will be more unfrequent
   $\{Beer\}: 3$, $\{Diaper\}: 4$, $\{Beer, Diaper\}: 2$
2) Generate strong association rules that satisfy minsup and minconf
   $\{Beer\} \Rightarrow \{Diaper\}$, $\{Diaper\} \Rightarrow \{Beer\}$
   → same support, different confidence

$\{a_1, \dots, a_n\}$ contains $\sum_{k=1}^{n} \binom{n}{k} = \sum_{k=1}^{n} \frac{n!}{k!(n-k)!} = 2^n - 1$ sub-itemsets

→ mine closed itemset and max-itemset instead

X is **closed** in a dataset D if X is frequent and $\nexists$ a super-itemset $Y \supset X$ with same support as X
X is a **max-itemset** in D if X is frequent and $\nexists$ a super-itemset $Y \supset X$ that is frequent

C → set of all closed FI in D, satisfying minsup
- contains complete information to create the whole set of frequent itemsets (C and its count information) → Data Compression
  ↳ since all subsets have the same count

M → set of all max-frequent-itemset in D, satisfying minsup

- in the worst case M^N frequent item-set can be generated (M: # distinct elements, N: max len transaction)
- depends strongly on minsup threshold

A-PRIORI ALGORITHM

Downward Closure property → any non-empty subset of a frequent itemset must be frequent
- anti-monotonicity property: If C fail a test also all its superset will fail

A-priori principle: If an itemset is infrequent ⇒ I should not test any of its superset

1. scan to get the frequent 1-itemset
2. get (k+1)-itemsets from combining k-itemsets
3. test candidate on D
   ↳ one DB test after each C_k

How to construct L_k (frequent itemset of lenght k) from L_{k-1} with the a-priori principle

1. **join step** $C_k \leftarrow L_{k-1} \bowtie L_{k-1}$
   - assuming that transactions and itemsets are sorted in lexicographic order
   - $l_i[k-2]$ refers to the $k-2^{th}$ term in the l_i itemset
   - $l_i[1] < l_i[2] < \dots < l_i[k-1]$ $\quad l_i \rightarrow$ is a $(k-1)$-itemset

   → since they are ordered $L_{k-1} \bowtie L_{k-1}$ is joinable on two sets $l_1, l_2 \iff$ the first $k-2$ elements are equal (since all the subsets must be preserved in L_{k-1}, then also the one that has the first $k-2$ elements in common) and $l_1[k-1] < l_2[k-1]$ → avoid duplication
2. **prune step** check that each $(k-1)$ subset of a candidate k-itemset is present in L_{k-1}
   done for $k > 2$ (1-itemset are frequent)

In order to generate association rules I can: $\forall l \in$ frequent_itemset $\forall s \subset l$ (non empty subset), $s \neq \emptyset$

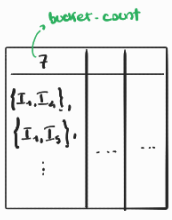
output $s \Rightarrow l - s \iff \frac{\text{supp\_count}(l)}{\text{supp\_count}(s)} \geq \text{min\_conf}$

RARE ITEM PROBLEM → items that occurs really infrequently and are pruned, but would generate interesting rules
- happens in strongly unbalanced dataset

- a problem with confidence is that is sensitive to the consequent (what comes after the arrow)
  → consequent with higher support will automatically produce higher confidence even though no association exists

IMPROVEMENT of the APRIORI

1) If I partitions D → an itemset must be frequent in at least one of the DB equal partition
   $\text{supp\_count}_i(X) \geq \text{min\_sup} \cdot |D_i|$
   - I can find the frequent itemsets in the local partitions in parallel
   - Combine all local frequent itemsets to form candidate itemsets → check it scanning the DB
   - just two scan since all the local partitions fits in memory



2) Hash-Based-Technique to reduce the size of the candidate $\in C_k$. When scanning for 1-itemsets I can create an hash-table in which I map all 2-itemsets and associate for each bucket a counter, not consider the one with counter lower then a threshold

- Generation of candidate itemsets is expensive → a lot of possible combinations (exponentially to the max-lenght of the frequent itemset)
  generate-and-test approach → BREADTH-FIRST SEARCH
- A lot of DB scan (I/O operations)

ECLAT → exploring vertical data format (equivalence class transformation)

$\{\text{TID} : \text{itemset}\}$ (TID: transaction ID) → horizontal format

$t(AB) = \{T_{12}, T_{25}, \dots\}$ ($\{I2, I7, I5\}$)

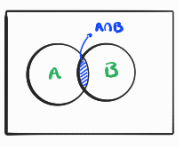
$\{\text{item} : \text{TID\_set}\}$ (TID_set: set of transaction IDs) → vertical format

$t(X) = t(Y)$ always happen together $\quad t(X) \subset t(Y)$ transaction having X always has Y

- Use of a-priori property to generate $(k+1)$-itemsets
- No need to scan the DB to find k-itemset support $(k > 1)$. However TID list can be quite long and expensive to intersect

EVALUATION $P(\text{videos}) = 75\%$ $\quad r$: computer games → videos $\quad \text{sup}(r) = 40\% \quad \text{conf}(r) = 66\%$

→ negatively correlated, buying PC games decreases the probability of buying videos



CORRELATION LIFT (from association analysis to correlation analysis)

$$\text{lift}(A \Rightarrow B) = \text{lift}(B \Rightarrow A) = \frac{\text{conf}(A \rightarrow B)}{\text{supp}(B)} = \frac{\text{conf}(B \rightarrow A)}{\text{supp}(A)} = \frac{P(A \text{ and } B)}{P(A)P(B)}$$

we solve the problem if B is always present

- correlation is symmetric instead an association rule is directional

- Lift measures how many times more often X and Y occur together respect than expected
  If lift < 1 -> negatively correlated, the presence of one reduce the other
  · If lift = 1 => A,B are independent

-> it is not down-ward closed (it's not monotonic, like sup_count, because probabilities can change in unpredicted ways) and doesn't suffer from the rare item problem

- possible problem when two rare itemsets occur together by chance and produce an enormous lift value

Another metrics that I could use is the χ^2

$$all\_conf(A,B) = \frac{sup(A \cup B)}{max\{sup(A), sup(B)\}} = min\{P(A|B), P(B|A)\}$$

$$max\_conf(A,B) = max\{P(A|B), P(B|A)\}$$

$$Kulczynski(A,B) = \frac{1}{2}[P(A|B) + P(A|\bar{B})]$$

→ can be 0.5 for complete irrelevance or just one side is important (dataset imbalance) -> I need another number

0.5 for imbalance or ...

$\in [0,1]$ closer to 1 stronger the relation

null-invariant

A null transaction is a transaction that does not contain any of the item-sets being examined

· Lift and χ are not null-invariant, since adding null transactions increase the lift value

$$\frac{\frac{n_{A\cup B}}{n_T+x}}{\frac{n_A}{n_T+x}\cdot\frac{n_B}{n_T+x}} = \frac{n_{A\cup B}}{n_A \cdot n_B}\cdot(n_T+x) = lift\_old + x\,\frac{n_{A\cup B}}{n_A \cdot n_B}$$

· important property especially in big dataset

We can measures the imbalance in two itemset A and B $IR(A,B) = \frac{|sup(A) - sup(B)|}{sup(A) + sup(B) - sup(A\cup B)}$

Tends to one when complets imbalanced (eg $sup(B) \to 0$ -> $sup(A\cup B) \to 0$))

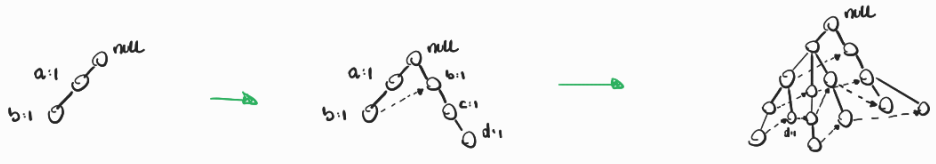
• generally use IR and Kulc to get a good picture of a rule

FP-GROWTH -> avoid candidate generation that is the apriori bottleneck
·builds a compact data structure of D that fits into memory (in two scan) and then find the FI

1) FP-TREE construction

a) scan for each single item, discard the unfrequent, then sort elements in a transaction by decreasing order, based on their support -> more overlapping (heuristic, not always true)

2) for each transaction, map it to a path in the tree : each node is an item and has an associated counter
- if T1 and T2 has the same prefix, they will have overlapping path -> DATA COMPRESSION
- pointers are maintained between nodes containing the same items

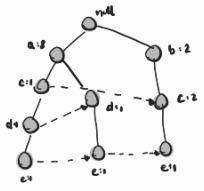


independent of T order

- best case : 1 path FP-tree
- worst case : every transaction doesn't have common prefixes
  · heavier than the original D (need pointers and counters)

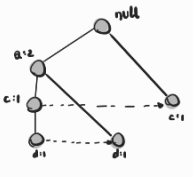
2) FI generation - bottom-up generation (from leaf to root) -> divide and conquer approach

sub tree containing e :
-> FI ending in e

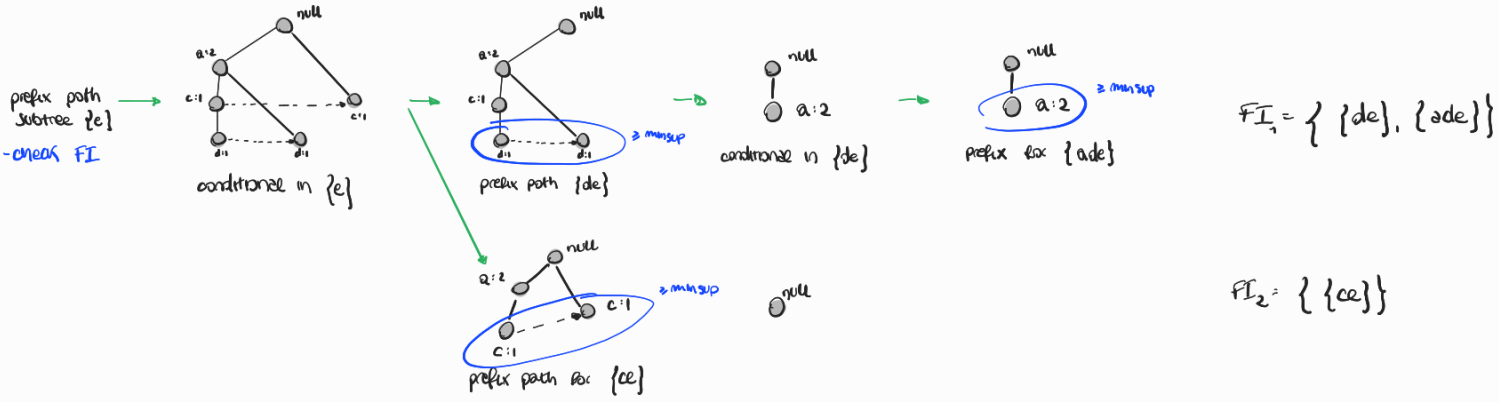


- proces recursively the trees to extract FI that ends with e - de - cde bde ce be
· divide and conquer

conditional sub-tree containing e : FP-Tree that contains only T with e (then remove it)
- remove items that cannot be Frequent - update counters



1. Obtain the prefix path sub-tree for e
2. Check if e is frequent (adding counts along the linked list) → if yes add {e} to FI
3. If {e} ∈ FI → build the e conditional tree and check for {ce}, ~~{ae}~~ (not in cond tree), {de} and {ae} recursevely



$FI_1 = \{ \{de\}, \{ade\} \}$

$FI_2 = \{ \{ce\} \}$

- FP tree is heavy to build, but I compress D (could discard it and save just FP if it doesn't grow)
  - completeness (preserve complete infos)
  - compactness (reduce irrelevant infos + common path)
- Order of magnitude faster than the apriori, especially for small min_sup

APRIORI → BFS, generate and test candidate itemset, level by level
FP-GROWTH → DFS, explore one branch of FP-tree completly before moving on

ADVANCED FREQUENT PATTERN ANALISYS

- in **multi-level association rule** I should reduce minsup accordingly for lower level item (soy milk)

  milk ⇒ bread
  soy milk ⇒ bread

  $\frac{supp(milk)}{supp(soy\ milk)} \approx \frac{supp(r1)}{supp(r2)}$, $conf(r1) \approx conf(r2)$ then r2 is **redundant** because of the ancestor

  · I should use non-uniform, group based min sup (more valuable items are less frequent)

- multi dimensional rules (≥ 2 predicates)
  - inter-dimensional rule: $age(X, "19-25") \wedge occupation(X, "student") \Rightarrow buys(X, "coke")$ → no repetition
  - hybrid-dimensional rule: $age(X, "19-25") \wedge buys(X, "popcorn") \Rightarrow buys(X, "coke")$

- for quantitive attributes I can discretize / cluster

MINE NEGATIVELY CORRELATED PATTERNS

· negatively correlated patterns infrequent tends to be more interesting than frequent one

def 1: If X and Y are both frequent but rarely occurs together $P(X \cup B) < sup(X) \cdot sup(Y)$
→ X and Y are negatively correlated · it's not null invariant (depends on the # of null transaction)

def 2: $sup(x \cup y) \cdot sup(\bar{x} \cup \bar{y}) \gg sup(x \cup y) \cdot sup(\bar{x} \cup \bar{y})$ - still suffer from null invariant

def 3: X and Y frequent $\frac{1}{2}[P(x|y) + P(y|x)] < \varepsilon$ → are negatively correlated

CONSTRAINED BASED MINING

- knowledge constraint: classification we want the posterior to be the class label
- data constraint: products sold in Chicago
- dimension constraint: filter for relevance to a specific hierarchy of data (eg. soy milk)
- pattern constraint: specify the form of conditions to be mined (predicates, # of conditions, ...) → metarules

META-RULES: form an hypotesis regarding the relations the user is trying to confirm
$P_1 \wedge P_2 \wedge \dots \wedge P_l \Rightarrow Q_1 \wedge \dots \wedge Q_r$

I can use the rule constraints to prune the research space:

· pruning **pattern** search space (not consider some candidate pattern (itemsets) in the generate and test phase)
- anti-monotonic: if constraint c is violated, it's further mining is terminated (it's supersets will fail) sum(I.price) ≤ 100 $
- monotonic: if c is satisfied, no need to check c again (supersets will hold) sum(I.price) ≥ 100 $
- succint: c must be satisfied, so I can start with dataset where c holds → avoid to enumerate everything min(I.price) ≥ 50 $ → avoid generate and test (no support counting)
  · sum(I.price) ≥ v not succint, because that result on algorithm that generate precisely only the ones that satisfy the constraint
- convertible: c not monotonic or antimonotonic, but it can be converted into if items in the transaction are ordered → avg price ≥ x
  - strongly convertable when monotonic or anti-monotonic according to the order
  - may exist conflicts on the order of items required → prune with one, then the other

· pruning **data** research space (consider only useful part of the dataset) → remove transaction
- data succint: remove some transaction before patter mining
- data anti-monotonic: if a transaction t doesn't satisfy c, t can be pruned from further mining

MINING COLOSSAL FREQUENT PATTERN → itemset of cardinality ≥ 50

- curse of down-ward closure property → any subpattern of a frequent pattern is frequent
  → the number of subitemset that are frequent are exponential

- considering closed itemsets may partially alleviate the problem but not really solve it, also them could be exponential
· colossal patterns are usually attached with greater importance than those of small pattern and only a small # of patterns are colossal

→ Apriori or FP-Growth will get stuck in the huge number of **middle sizes patterns**
is stuck in huge # of subtrees

PATTER-FUSION traverse the tree in a **bounded-breath way** → approximation that quickly find most of them
1. only a subset of patterns are used as starting points → avoid exponential search space
2. at each step it agglomerates more items together → shortcut to reach massive size

Core Patterns → for a frequent pattern α, a **subpattern** β is τ-core pattern of α ⇔ β shares a similar support set

$\frac{|D_\alpha|}{|D_\beta|} \geq \tau$ (τ: core ratio) $\tau \in (0,1]$ $|D_\alpha|$ → # of patterns containing α

A pattern α is (d,τ)-robust if d is the max # of items that can be removed from α for the remaining pattern to stay a τ-core pattern of α

$$d = \max_{\beta} \{ |\alpha| - |\beta| : \beta \subseteq \alpha \wedge \beta \text{ is a } \tau\text{-core of } \alpha \} \qquad d \in \Omega(2^d)$$

- massive patterns have far more core-patterns than smaller ones → remove some items and maintain similar support

$$Dist(\alpha,\beta) = 1 - \frac{|D_\alpha \cap D_\beta|}{|D_\alpha \cup D_\beta|}$$

If two pattern α, β are core patterns of the same pattern → they have their distance bounded by a ball

$$Dist(\alpha,\beta) \leq 1 - \frac{1}{\frac{2}{\tau} - 1}$$

- given one pattern from it's core pattern subset I can get all of them, just by examining the bounding ball
- If massive pattern, the ball will be denser

If we draw randomly from the set of patterns of size c we would be more likely to pick a core descendant (lower level core patterns) of a colossal pattern than of a small pattern
- merging more core patterns to get the massive one

1) Initial Pool → use an existing algorithm to mine all frequent patterns up to a small size

2) Iterative Pattern Fusion
- K seeds pattern are randomly picked from the current pattern pool → high probability of being core-descendent of a colossal pattern
- for each one I get those within a bounding ball centered in the pattern (from the current pool)
  → all the pattern founds are merged together to generate a set of super-patterns. This set will be the new pool for the next iteration
  ↳ avoids mid-sized pattern, more than one for each K
- If the pool exceed a threshold I will sample it
- check that the set returned is frequent ?

3) Termination when the current pool contains no more than K patterns at the beginning

→ The longer the pattern, the greater the chance to be generated

MINING COMPRESSED PATTERNS $\quad D(P_1,P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}$ → distance between two pattern

Given two pattern A and B, A can be expressed by B if $O(B) \subset O(A)$
- where $O(A)$ is the itemset of pattern A

In a cluster r: $\bigcup_{i=1}^{n} O(P_i) \subseteq O(r)$

A pattern P is δ-covered $(0 \leq \delta \leq 1)$ by pattern P' if $O(P) \subseteq O(P') \wedge D(P,P') \leq \delta$
- could belong to multiple clusters

Finding the minimum set of representative pattern is NP-hard

REDUNDANCY AWARE TOP-K PATTERNS → not only high frequency, but also small redundancies

$S(p)$ is the degree of usefulness of the pattern P → can be based on subjective or objective measures (supp, conf ...)
↳ significance

$R(p,q) = S(p) + S(q) - S(p,q)$ → we can approximate it with distance between patterns
↳ redundancy

I want to find the K patterns that maximise the marginal significance → relevant and not redundant to the already picked

OUTLIER ANALYSIS

outlier → object that deviates significantly from the normal object as if they where generated by a different mechanism
- universe credit card purchase - novelty detection is different (once points are merged in the model)

noise → random error in measuring
- should be removed before outlier detection → may hide outliers

GLOBAL OUTLIER → points significantly deviates from the rest of the data

CONTEXTUAL OUTLIER → deviates significantly based on a selected context (form contextual attributes) → based on behavioural attributes (characteristics of the object)
- 35°C in summer or winter

COLLECTIVE OUTLIERS → subset of data collectively deviate
- intrusion detection in IT systems (multiple DoS)

· I could have multiple types of outlier, one object may belong to more then one type
· domain specific distance measures

1. Based on user labelled outlier (supervised, semi-supervised, unsupervised)

· supervised → classification problem
- imbalanced classes → boost outliers
- recall is more important (not mislabelling outliers for normal data) → catch as many outliers as possible

· unsupervised → assume objects are clustered into multiple groups. A cluster is distant from all groups
· cannot detect collective outliers effectively · high FP rate and still miss some outlier
· could use clustering algorithm directly → no cluster ≈ outliers, but difficult from noise and costly since I need to cluster everything

· semi-supervised → when the # of labelled data is small
- depends if I have label for normal or outliers - use proximity measures and unsupervised help

2. Based on assumptions about normal data (statistical, proximity, clustering)

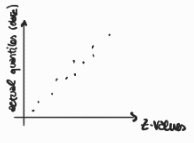
· STATISTICAL TECHNIQUES → effectiveness highly depends on whether the assumption of the statistical model holds or not
model-based methods (assume underlying distribution)

Assumes data are generated by a stochastic process → generative model
· Learn a generative model and identify objects in low probability regions as outliers

parametric → data generated by a parametric distribution $f(x,\theta)$ (fixed in advance, learnable) the smaller this value, the more likely x is an outlier

NORMAL DISTRIBUTION → assume univariate data (one attribute)

1. Using Q-Q plots → useful in general to understand if a sample comes from a continuous distribution
- plot quantiles from the sample against the quantile of the normal distribution
· divide the N into n+1 segments, where n is the number of values → then calculate the z-values (cut-off for each segment) that is the # of std from the mean
· if straight line ⇒ data comes from normal (the slope depends on the θ of the data)



2. Maximum likelihood to estimate μ, θ $\quad P(x_i|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \quad \Rightarrow \quad P(X|\mu,\sigma^2) = \prod_{i=1}^{n} P(x_i|\mu,\sigma^2)$

Find μ,σ^2 maximizing the likelihood $\quad N(\mu_0,\sigma_0^2) = \text{argmax}\{P(X|\mu,\sigma^2),\ \mu,\sigma^2 \in \mathbb{R}\}$

$$\ln P(X|\mu,\sigma^2) = \sum_{i=1}^{n} \ln P(x_i|\mu,\sigma^2) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum(x_i-\mu)^2$$

- $\frac{\partial \ln P(X|\mu,\sigma^2)}{\partial \mu} = -\frac{1}{\sigma^2}\sum(x_i-\hat{\mu}) = 0 \quad \Rightarrow \quad \hat{\mu} = \frac{1}{n}\sum x_i$

- $\frac{\partial \ln P(X|\mu,\sigma^2)}{\partial \sigma^2} = -\frac{n\sigma}{\sigma^2} + \frac{1}{\sigma^3}\sum(x_i-\mu)^2 = 0 \quad \Rightarrow \quad \frac{1}{\sigma^3}\left[\sum(x_i-\mu)^2 - n\sigma^2\right] = 0 \quad \Rightarrow \quad \sigma^2 = \frac{1}{n}\sum(x_i-\mu)^2$

$\mu \pm 3\sigma$ contains 99.7 % of data, everything outside can be considered an outliers

3. **Grubb's Test** → find wether a min/max value is an outlier

- find **G test statistics** order and find mean ($\bar{Y}$) and std (s) of dataset

$$G = \frac{\max_{i=1,\dots,N} |Y_i - \bar{Y}|}{s}$$ → this is because only max or min

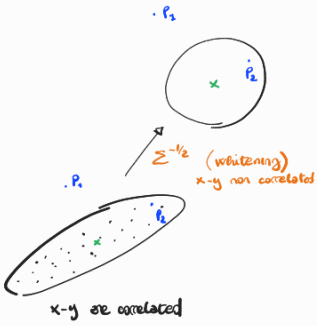
- find G critical value from a table (given α and N)
- if G test < G critical keep the sample, not an outlier, otherwise reject

MULTIVARIATE DATA → transform multivariate outlier decision to univariate

PARAMETRIC METHODS

1. **Mahalanobis distance**: between a point and a distribution

- euclidian distance works well when dimension are equally weighted and independent ← z score $\left(\frac{x-\bar{x}}{\sigma}\right)$ or min-max interval



The steps are:
- transform the columns into uncorrelated variables
- scale to have variance $\sigma = 1$
- calculate the euclidian distance

$$MDist^2(\bar{o}, \bar{o}) = (o-\bar{o})^T \Sigma^{-1} (o-\bar{o})$$

covariance matrix → like in z score divide by σ

→ If the variables are strongly correlated the covariance will be high and the distance reduced

Use the distance to perform univariate outlier analysis → Grubb's test

- computationally heavy since I need to find S and S^{-1}, also store them

2. **X^2 statistic** $X^2 = \sum_{i=1}^{n} \frac{(X_i - \bar{X}_i)^2}{\bar{X}_i}$ ($\bar{x}_i = E[x_i]$) n = # of variables

- when n is big enough (>30) $\Rightarrow X_c^2 \sim N$ (central limit theorem)
  - when X is really large → outlier
  - set the upper limit as: $\bar{X}^2 + 3S_{X^2}$

NON PARAMETRIC METHODS → not assume any a-priori statistical model and determine the model from the input data (not only the parameters)

- parameters are flexible and learned by the data, not fixed in advance

1. **Histogram** → hard to choose appropriate bin size:
   - too large then outliers in frequent bin (false negative)
   - too small then normal objects in rare bins (false positive)

→ better to use kernel methods

2. Kernel Density Estimation $\hat{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(x - x_i) = \frac{1}{n\alpha}\sum_{i=1}^{n} K\left(\frac{x-x_i}{\alpha}\right)$ (symmetric, not necessarily ≥ 0, but integrates to one; α = smoothing parameter)

The **complexity** of statistical methods depends on the model

- gaussian → linear
- for kernel density estimation, the model learning can cost up to quadratic → for each point I need to compute all the distances

PROXIMITY BASED → outliers when NNs are far away (eg. 3-NN)

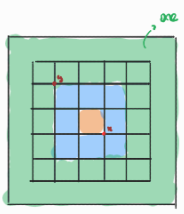
- effectiveness relies on the quality of the proximity measure → **distance** based or **density** based
- assumes that an outlier has a significantly higher proximity than the other data

1) DISTANCE BASED: for each object $o \in D$ examine the # of objects in the r-neighbourhood (r = distance threshold)

- o is an $DB(r, \pi)$ outlier (same for all the space) if $\frac{|\{o' | dist(o,o') \leq r\}|}{|D|} \leq \pi$ or k-th NN has dist $(o, o_k) > r$ with $k = \lceil \pi \cdot |D| \rceil$

Complexity is $O(n^2)$ but I can stop when I find $\pi \cdot |D|$ objects with distance $\leq r$ → linear average complexity

2) distance + grid based → Data space is partitioned in hyper-cubes, each with a diagonal of $\frac{r}{2}$



level 1: for any point x in cell C and y in a level 1 cell → dist $(x,y) \leq r$
level 2: for any point x in C and y: dist $(x,y) \geq r$ → y is in a level 2 cell

not all made r-neighbourhood
- b_1 = total # of objects in L_1
- b_2 = total # of objects in L_2
- a = # of objects in C

1) pruning by L_1: if $a + b_1 > \lceil \pi \cdot |D| \rceil \Rightarrow$ all objects in C are not $DB(r, \pi)$ outliers
2) pruning by L_2: if $a + b_1 + b_2 < \lceil \pi \cdot |D| \rceil + 1 \Rightarrow$ all objects in C are outliers

The other objects I need to check manually

3) Local distance based → useful to handle cluster with different densities → identify local outliers

$D_K(x_i) = \frac{1}{K} \sum_{j \in N_K(x_i)} \|x_i - x_j\|$ (avg dist KNN; $N_K(x_i)$: K-NN of x_i)

when $x \in \mathbb{R}^n$ with n very large $\|\cdot\|_2$ doesn't work well

outlierness: $O_K(x_i) = \frac{D_K(x_i)}{\frac{1}{K}\sum_{j \in N_K(x_i)} D_K(x_j)}$

$O_K(x_i) > 1$ → x_i is further away from neighbours than expected

$O_K(x_i) \in [0, +\infty)$

- problem: when I have two close clusters with different densities (one sparse, the other dense), then the boundary of the dense one is considered outliers
- two parameters are needed (K and threshold for O_K)
- time complexity $\in O(n^2 \cdot \#\text{dimensions})$

4) Density based outlier detection → density around an outlier is significantly different from the one around it's neighbours

$dist_K(o)$ = distance of the Kth NN $N_K(o) = \{ o' \mid o' \in D, dist(o,o') \leq dist_K(o) \}$ → $|N_K(o)| \geq K$ more object with the same distance $dist_K(o)$

$reachdist_K(o' \leftarrow o) = \max \{ dist_K(o'), dist(o,o') \}$

local reachability density $lrd_K(o) = \frac{|N_K(o)|}{\sum_{o' \in N_K(o)} reachdist_K(o' \leftarrow o)}$ ($\approx K$)

K = # of neighbours to take into account for density

local outlier factor $LOF_K(o) = \frac{\sum_{o' \in N_K(o)} \frac{lrd_K(o')}{lrd_K(o)}}{|N_K(o)|} = \sum_{o' \in N_K(o)} lrd_K(o') \cdot \sum_{o' \in N_K(o)} reachdist_K(o' \leftarrow o)$ → N^2

avg of the ratios $LOF_K(o) \approx 1$ → ok

→ compare local densities of the K-NN

- same problem for the boundary as before

- CLUSTERING BASED → normal data belongs to large and dense cluster, whereas outliers to small or sparse clusters or do not belong to any cluster
  - clustering is expensive

1. x do not belong to any cluster → DBSCAN but two params are needed (ε, # points) → common to all the space (no local)
2. Far from its closest cluster → using K-means if $\frac{dist(o, c_o)}{avg.dist(c_o)} \gg 1$ → likely an outlier (only spherical clusters)
3. In small clusters (typically in large D, outliers makes cluster) eg. intrusion detection

   b is the boundary between large and small clusters if one of the formulas holds:

   $\sum_{1 \leq i \leq b} |D_i| \geq \alpha |D|$

   $\frac{|C_b|}{|C_{b+1}|} \geq \beta$ ($|C_b|$: least large cluster; $|C_{b+1}|$: first small cluster)

   large cluster $LC = \{ C_i \mid i \leq b \}$ and small as $SC = \{ C_j \mid j > b \}$

CBLOF (cluster based local outlier factor) is associated with each object

$$CBLOF(o) = \begin{cases} |C_i| \cdot \min \{ dist(o, C_j) \} & \text{when } C_i \in SC: o \in C_i, C_j \in LC \\ |C_i| \cdot dist(o, C_i) & \text{when } C_i \in LC: o \in C_i \end{cases}$$

($|C_i|$: small; $dist(o, C_j)$: large)

- first I need to cluster the data, then I need to find the CBLOF for $\forall o \in D$ → high computational cost
- larger CBLOF(o) more likely it is that o is an outlier → small C_i for from LC or o_i for away from centroid

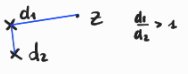
- CLASSIFICATION BASED → need to face the high imbalance set (supervised methods)

1) ONE CLASS MODELS → learn the decision boundary of the normal class.
   - can identify outliers that are not similar to those in the D (unseen anomaly)

a) Can be used to learn the decision boundary SVDD → construct an hypersphere around almost all positive class points with the minimum radious
   - Outlier if it falls outside the sphere

b) Using NN-Description (NN-d) $f_{NN^{tr}}(z) = I\left(\frac{\|z - NN^{tr}(z)\|}{\|NN^{tr}(z) - NN^{tr}(NN^{tr}(z))\|} \right)$ (z: test obj; $NN^{tr}(z)$: NN of z on training)

$$I(x) = \begin{cases} 1 & \text{when } x \leq 1 \rightarrow \text{accepted} \\ 0 & \text{when } x > 1 \rightarrow \text{outlier} \end{cases}$$



2) SEMI-SUPERVISED LEARNING → combining classification and clustering

- find a large cluster C and a small C_1 - since same obj in C carry the lable "normal" => C is normal
- use a one-class model to identify outliers from C - since some obj in C_1 carry the label "outlier" => C_1 out
  → obj that doesn't belong to C or C_1

→ quality is based on the quality and representativity of the training set

- MINING CONTEXTUAL OUTLIERS

- If the context is clearly identified I can find outliers in the limited dataset using conventional techniques.

- If $D_{context}$ is not representative, then we can assume that similar context has similar behaviours (similar ages or locations)
  Learn a mixture model on U → contextual variables and on V → behavioural attributes
  U has clusters $U_1, \dots, U_n$ (on context attr.) — V has cluster $V_1, \dots, V_m$
  outlierness score (based on context): $S(o) = \sum_{U_j} p(o \in U_j) \sum_{V_i} p(o \in V_i)\, p(V_i \mid U_j)$ — if $o \in U_j$ what p that $o \in V_i$

- I can model the normal behaviour based on the context → I get an outlier when the prediction and the behaviour differs significantly
  - useful when I cannot identify clearly the context since I do not need to specify, but the model understands it autonomously → Markov Chains
  eg. # of articles browsed in the past of one customer to be considered to understand if a new purchase is an outlier

- MINING COLLECTIVE OUTLIERS → group of objects that deviates significantly from the entire dataset (more dense)

Individual points may seem normal and not outliers, but this change when I consider the entire group together
→ time series may have a strange shape in an interval, but I couldn't understand it by a single point

- I need to check the relations between data objects → more difficult than other outliers (more computational complexity)
  bigger units than individual object/points

Depends on the type of data:
1) temporal data need to check smaller structured in time
2) spatial data explore local areas
3) graph data explore subgraphs

- In contextual outlier analysis the structures to be analyzed are the context, here is not specified (needs to be learned/modeled)
  Two methods:
  1) Reducing the problem to classical outlier analysis → identify structure units (time segments, local areas, subgraphs)
     Treat every structure units as a data object and extract features → outlier analysis
  2) Models the expected structural units directly → eg. time series and Markov model, outlier when deviates
     - do not need to define structure units

- MINING OUTLIERS IN HIGH DIMENSIONS

problems: 1) Difficult to interpret since too many attributes (which subset particularly differs?)
2) Data sparsity (curse of dimensionality) → distance is dominated by noise

1) Extends conventional outlier detection
   HilOut Alg.
   $w(o) = \sum_{i=1}^{k} dist(o, NN_i(o))$ (i-th NN) → top e object with higher weight are outlier
   - I'm using ranks between distances and not absolute distances to make the outlier detection
   - Not good for interpretability

2) Dimensionality Reduction → search outliers in sub-spaces (PCA)

- GRID BASED → project data on different subspaces and find areas where densities are much lower than the average

1) discretize data into grids with φ equi-depth (helps to cope with different distributions across attributes) (constant number of points $f = \frac{1}{\varphi}$) not equal width (fixed sized intervals)
2) search for cubes extremely sparse in k dimension (subspace)

If n objects in D, then the expected # of pts into a k-dimensional cube is: $\left(\frac{1}{\varphi}\right)^k \cdot n = f^k n$ → Binomial random variable (or in or not)
The std of the # of pts in a k-cube is $\sigma = \sqrt{f^k (1-f^k) n}$ — $\mu_x = np$ — $\sigma^2 = p(1-p)n$

sparsity coefficient: $S(C) = \frac{n(C) - f^k n}{\sqrt{f^k (1-f^k) \cdot n}}$ (# of obj inside the cube)

- If $S(C) < 0$ ⇒ less object than expected (more negative, more sparse)
- assuming $S(C) \sim N$ we can get the significance level

→ Find the m cells with the lowest sparsity coefficient
- brute force algorithm (check every cube in every subspace) $O(\varphi^k)$ for the $\binom{k}{k}$ subspaces (φ intervals per k dimension → φ^k cubes)
- Quality depends on grid resolution and position → adopts a global approach on the expected # of points

3) Angle based methods → angle measurements do not degrade with dimensions

$o \in D$ is an outlier if most objects are located in the same direction
- use the distance-weighted angle variance as outlier score

ABOF (angle based outlier factor): $ABOF(o) = VAR_{x,y \in D,\ x,y \neq o} \frac{\vec{ox} \cdot \vec{oy}}{\|o-x\|^2 \cdot \|o-y\|^2}$ → high ABOF means outlier

Time complexity $O(n^3)$

CLUSTERING HIGH DIMENSIONAL DATA → text documents or DNA arrays

Clustering should not only consider dimensions but also features (attributes)
- I can perform **feature transformation** to remove redundancies (PCA or SVD)
- Or **feature selection** to find a subspace with relevant clusters

1) SUB-SPACE CLUSTERING → like in CLIQUE algorithm

- clusters may exist only in some subspaces
They can be **similarity based** (bottom-up or top-down subspace search), **correlation based** (PCA) or BI-CLUSTERING

BI-CLUSTERING: cluster attributes and objects simultaneously

It has four requirements: 1) only small set of objects participate in a cluster, 2) A cluster only involve a small # of attributes (influenced by δ), 3) objects can participate in more clusters or no cluster, 4) attribute can participate in more clusters or no cluster

Example: customers and products purchased (attributes)

A bi-cluster is a sub-matrix with some consistant pattern. We have four types:
1) constant values $\forall i \in I, j \in J \Rightarrow e_{ij} = c$ (constant sub-matrix)
2) constant values on rows $\Rightarrow e_{ij} = c + a_i$
3) coherent values $\rightarrow e_{ij} = c + a_i + \beta_j \Rightarrow e_{ij_1} - e_{ij_2} = e_{ij_1} - e_{ij_2}$ on columns but also on rows
4) coherent evolutions on rows $\rightarrow (e_{ij_1} - e_{ij_2})(e_{i_2j_1} - e_{i_2j_2}) \geq 0$

Real word data is noisy. Two techniques to approximate bi-clusters:

a) Optimization-Based Techniques → find one sub-matrix at the time with a greedy approach (local optimum)

δ-Bi-Clustering

mean of the i-th row $e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{ij}$ mean of the j-th column: $e_{Ij} = \frac{1}{|I|} \sum_{i \in I} e_{ij}$ all elements mean $e_{IJ} = \frac{1}{|I| \cdot |J|} \sum e_{ij}$

The quality of a bi-clustering is: $H(I \times J) = \frac{1}{|I| \cdot |J|} \sum_{ij} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2$ → how much it deviates from a perfect coherent bi-cluster

mean squared residue

if perfect: $(c + a_i + b_j - c - a_i - b_j - c - a_i - b_j + c + a_i + b_j)^2 = 0$

A sub-matrix $I \times J$ is a δ-bi-cluster if $H(I \times J) \leq \delta$ ($\delta = 0$ is a perfect bi-cluster with coherent values)
- A **maximal δ-bi-cluster** when $\nexists$ another δ-bi-cluster that contains the first one

Since computation is costly, use **greedy search** → approximate solution:

1: Deletion Phase → iteratively removes rows-columns while $H(I \times J) > \delta$
- to understand which row-column to remove at each step calculate the mean-squared-residue and remove the highest
$d(i) = \frac{1}{|J|} \sum_j (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2$ and $d(j) = \frac{1}{|I|} \sum_i (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2$

2: Addition Phase → extend the cluster until δ constraint is kept → add the smallest $d(i)$ / $d(j)$

In this way I can found only one δ-bi-cluster (deterministic) => replace the elements of the cluster founded with random values (cannot be in more than one cluster)

b) Enumeration-Based → use a tolerance threshold to specify the degree of noise allowed. Then try to enumerate all submatrices

δ-pCluster: for a 2×2 submatrix of $I \times J$ **p-score** $\begin{pmatrix} e_{ij_1} & e_{ij_2} \\ e_{i_2j_1} & e_{i_2j_2} \end{pmatrix} = |(e_{ij_1} - e_{ij_2}) - (e_{i_2j_1} - e_{i_2j_2})|$ if perfect p-score = 0

- Allow a tolerance (noise) **p-score $\leq \delta$** for every 2×2 sub-matrix of $I \times J$
p-score controls local noise in each point, while mean-squared residue is on average noise
- **Monotonicity property**: if $I \times J$ is a δ-pCluster => all the sub-matrices are δ-pClusters

→ Use similar techniques as FI analysis: for each combination J (features) (obj can be in more subsets) find the maximal subset of objects I, such that $I \times J$ is a δ-pCluster.
- If $I \times J$ is not a sub-matrix of any found δ-pCluster it is maximal

2) DIMENSIONALITY REDUCTION METHODS → sometimes is more efficient to use new features space rather than using original subspaces

SPECTRAL CLUSTERING (eigenvalues): combine feature extraction and clustering → use the similarity matrix to perform dimensionality reduction

NJW Algorithm
1) calculate the affinity matrix $W_{ij} = e^{-\frac{dist(o_i, o_j)}{\sigma^2}}$ → scaling parameter (how fast it decrease with the distance), $W_{ii} = 0$
2) define a diagonal matrix $D_{ii} = \sum_{j=1}^{n} W_{ij}$ → $A = D^{-1/2} W D^{-1/2}$ → transformed sub space $A = f(W)$
3) and k leading eigenvectors of A, project original data and cluster it with ordinary methods (costly to compute)

CONSTRAINED CLUSTER ANALYSIS → constrained like environment impedition that modify the distance between two points (a wall)

Constrained on instances:
1) must-link (x,y) → x and y on the same cluster
2) cannot-link (x,y) → on two different clusters eg. $dist(x,y) > d$

Constrained on clusters. ewe min # of objects, max diameter, shape (convex), # of clusters (K)

- **δ constrained** (minimum separation) $\forall s_p, s_q \in S_i \times S_j$, $S_i \neq S_j \Rightarrow dist(s_p, s_q) \geq \delta$
  → equivalent to a must-link (x,y) : $dist(x,y) < \delta$ → conjunction of constraints
- **ε constrained** $\forall S_i : |S_i| > 1$, $\forall s_p \in S_i$, $\exists s_q \neq s_p$, $s_q \in S_i$: $dist(s_p, s_q) \geq \varepsilon$ → constrained on the **compactness**
  → every point x must-link with a point y : $dist(x,y) \leq \varepsilon$ → disjunction of constraints

Constrained on similarity measures → requirement that the similarity calculation must respect
- euclidian distance for walking in a city is not good (walls)

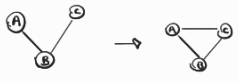
Constraints can be **hard** (cannot be violated) or **soft** (better not to violate) (also called PREFERENCES) → want to find clustering that maximally respects constraints

→ It needs to handle **conflicting** constraints (must-link and cannot-link of same x-y)

→ Measure the **usefulness** of a set of constrains :
1) **Informativeness** : infos carried by the constrained beyond the clustering models
D(dataset), A (algorithm), C(constraints) → $I_A(C,D)$ = # of constraints violated during the clustering of D by A
2) **Coherence** : degree of redundancy among C

- helps to filter out useless constrained

- Adding constraints limits the search space of the algorithm → quicker



1. COP-K-means → handle hard constraints (given as part of the dataset labelled)
   - for must-link compute the transitive-closure (add implied constraints). Then replace these objects with their mean → super-instance
   - modify the original K-means to compute the K means on the possible objects (don't consider cannot-link)

2. Handle soft constraints : add a **penalty** when a constraint is violated
   - then the objective will be to minimize an objective → sum of distances (ewe K-mean), adjusted by the penalties
   - when must-link (x,y) but they are assigned different clusters, add $dist(c_1, c_2)$ ↳ cluster of y
   - when cannot-link (x,y) but assigned on the same cluster, then add $dist(c,c')$ ↳ closest cluster to C that can accomodate x or y

Can be complex to find these clusters => optimization methods based on the domain of the application
- eg. group closed points in micro-clusters → reduce complexity instead of handling single points

GRAPH CLUSTERING

Network → a collection of entities that are interconnected with links (eg. social networks, communication networks, biological networks)

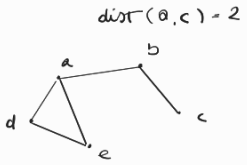
Graph → mathematical representation of a network (nodes and edges) $G(N,E)$

- larger and larger: visual inspection doesn't work

To cluster graphs we can adopt two techniques:

1) Introduce a similarity metrics → standard cluster algorithms on dissimilarity matrix (high dimensional clustering if large graphs)
2) Algorithms that works directly on graphs

SIMILARITY METRICS



1) **Geodesics distance** $dist(A,B)$ = # of edges of the shortest path between A and B

Eccentricity of a node $eccen(v) = \max_{u \in V-\{v\}} dist(v,u)$

Radious of graph $r = \min_{v \in V} eccen(v)$ → distance between the most central point and the furthest border

Diameter of graph $d = \max_{v \in V} eccen(v)$ → largest distance of any pair of vertexes

- A **peripheral vertex** is a vertex that achives the diameter (d,e,c)

The geodesics distance doesn't measure well the closeness of vertexes in particular domains

- Bob and Alice are vertexes in a network, the shortest distance doesn't account for all the other possible interactions between the two

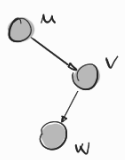
The similarity in a social network can have two meanings

1. **structural context-based similarity**: two customers are similar if common neighbors
   - people that receive common suggestion from multiple friends often make the similar decision
2. **Random walk similarity**: a company sends promotional mails to both Ada and Bob. They randomly forward to their friends. The closeness is the likelihood that other customers simultaneously receive the promotion

2) **SimRank**: context based similarity. Directed graph

$I(v) = \{u \mid (u,v) \in E\}$ (IN-NEIGHBOURHOOD)

$O(v) = \{w \mid (v,w) \in E\}$ (OUT-NEIGHBOURHOOD)



$$s(u,v) = \frac{c}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s(x,y) \qquad c \in (0,1)$$

- If no neighbors $s(u,v) = 0$
- recursive formula

Compute it iteretively with a matrix $S_i(*,*) \in [0,1]^{n \times n}$

$$S_0(u,v) = \begin{cases} 0 & \text{if } u \neq v \\ 1 & \text{if } u = v \end{cases}$$ → A lower bound on the actual value

$$\begin{cases} S_{i+1}(u,v) = \frac{c}{|I(u)||I(u)|} \sum_{x \in I(u)} \sum_{y \in I(v)} S_i(x,y) & \text{if } u \neq v \\ S_{i+1}(u,v) = 1 & \text{if } u = v \end{cases}$$

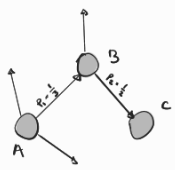
(how much similarity spreads over the network) → non decreasing

Fix a number of iteration to get the approximation $O(k n^2 d_2^2)$ (# of iter; avg of $|I(u)| \cdot |I(v)|$)

3) **Random Walk**

$d(u,v) = \sum_{t: u \rightsquigarrow v} P[t]\, \ell(t)$ (expected distance; path lenght)

$P(t) = \begin{cases} \prod_{i=1}^{k-1} \frac{1}{|O(w_i)|} & \text{if } \ell(t) > 0 \\ 0 & \text{if } \ell(t) = 0 \end{cases}$ ($= \langle w_1, \dots, w_k \rangle$)

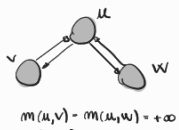


$d(u,v)$ is the number of steps a random surfer needs to take to go from u to v

- not symmetric

Expected Meeting Distance (EMD) is the expected # of steps two surfers needs to take before meeting each other - $m(u,v)$

- symmetric



$m(u,v) = m(u,w) = +\infty$

$m(v,w) = 1$

To formally define EMD we use the derived G^2 of node-pairs

- each node $(u,v) \in V^2$ can be thought as the present state of two random surfers in G
- an edge in E^2 $((u,v),(t,w))$ means that $u \to t$ and $v \to w$, possible steps of two surfers
- A tour in G^2 having length n also in G has length n

EMD is the expected distance in G^2 from (u,v) to any node $(x,x) \in V^2$ (meeting point)

$$m(u,v) = \sum_{t:(u,v) \rightsquigarrow (x,x)} P[t]\, \ell(t)$$

- If G is strongly connected, G^2 may not be → no path from $(u,v) \rightsquigarrow (x,x) \in V^2$, then $m(u,v) = +\infty$
- If from (u,v) there is another path to (u,v) there is a problem on the definition if passes through a meeting point $(x,x) \in V^2$

EXPECTED f-MEETING distance : Map all distances to a finite interval, instead of computing $\ell(t)$, we can compute $f(\ell(t))$

f: non-negative, monotone, bounded $[0,+\infty) \to [a,b]$ — solution to infinite EMD (when G^t is not strongly connected)

$$M(u,v) = \sum_{t:(u,v)\rightsquigarrow(x,x)} P[t]\, c^{\ell(t)} \qquad c \in (0,1)$$

$s'(a,b) = 0 \Rightarrow$ no path between a and b

$s'(a,b) = 1 \Rightarrow a = b$

Equivalence to simrank with parameter c

$$s'(a,b) = \sum_{x=1}^{|O(a)||O(b)|} \sum_{t':O_x(a,b)\rightsquigarrow(y,y)} P[t']\, c^{\ell(t')+2} \cdot \frac{1}{|O(a)|\,|O(b)|} \cdot \frac{c}{|O(a)|\,|O(b)|} \sum_{i=1}^{|O(a)|} \sum_{j=1}^{|O(b)|} s'(O_i(a), O_j(b))$$

(t': removed first element of the t)

- it's a random similarity but also a structural context-based similarity (also sim-rank)

GRAPH CUT: → inside a cluster points should be well connected, instead in different clusters connected in a much weaker way

A cut $C(S,T)$ is a partitioning, where $V = S \cup T \wedge S \cap T = \emptyset$

- The cut set is $\{(u,v) \in E \mid u \in S, v \in T\}$
- Size of the cut is the number of edges in the cut set, or if it is a weighted graph, the sum of weights

1) MINIMUM CUT → $C(S,T)$, where the cut size of T is not greater than any other cut's size
   - it takes polynomial time to compute
   - is not that good because it will find small clusters (even a single point)

2) SPARSEST CUT → Every node $n \in$ cut set has most of the edges belonging to one cluster

Given a cut $C(T,S)$ $\Rightarrow$ $\phi = \frac{\text{cut size}}{\min\{|S|,|T|\}}$ (sparsity) → the sparsest is the cut with the minimum ϕ

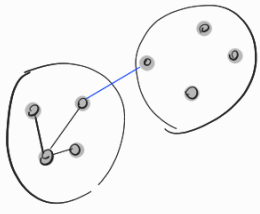
- favours balanced (bisection) and sparse clusters (thanks to min)
- the problem is NP hard but $O(\sqrt{\log n})$ approximation

Theoretically graph clustering problems can be regarded as optimal cut problems but they are computationally expensive

MODULARITY of CLUSTERING

$$Q = \sum_{i=1}^{n} \left[\frac{l_i}{|E|} - \left(\frac{d_i}{2|E|} \right)^2 \right]$$

l_i: # of edges in the i-th cluster → complexity inside C_i; d_i: sum of degrees of i-th cluster vertices; $\frac{l_i}{|E|}$: probability a random edge falls in C_i; $\left(\frac{d_i}{2|E|}\right)^2$: probability a entirely in C



- High Q means more edges in clusters than you expect by chance (randomly cutting the graph)
- Optimal cut has the highest modularity

SCAN ALGORITHM Define $\Gamma(v)$ as the immediate neighbourhood of a vertex $v \in V$ $\quad \Gamma(v) = \{u \mid (v,u) \in E\} \cup \{u\}$

$$\sigma(v,w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \cdot |\Gamma(w)|}}$$ → structural similarity : high for members of a clique, small for hubs and outliers

ε-Neighbour of v : $N_\varepsilon(v) = \{w \in \Gamma(v) \mid \sigma(w,v) \geq \varepsilon\}$

v is a core vertex $\iff$ $|N_\varepsilon(v)| \geq \mu$

- problem: select the two variables

$\text{DirREACH}_{\varepsilon,\mu}(v,w) \iff \text{CORE}_{\varepsilon,\mu}(v) \wedge w \in N_\varepsilon(v)$ (w directly reachable from v)

If a vertex $v \in N_\varepsilon(u)$ and u is a core vertex → v is assigned to the same cluster of u

- growing process until no cluster can be enlarged

Structure Reachable → transitive closure of direct reachability. A vertex v can be structure reachable from u $\iff$ $\exists \langle w_1, \dots, w_n \rangle$, w_1 reachable from u, w_i reachable from w_{i-1}, v reachable from w_n. $\text{REACH}(u,v)$

Structure Connected → v,w are connected if $\exists u$ core object : v and w can be reached from u

$\text{CONNECT}_{\varepsilon,\mu}(v,w) \iff \exists u \in V : \text{REACH}_{\varepsilon,\mu}(u,v) \wedge \text{REACH}_{\varepsilon,\mu}(u,v)$

A vertex is an hub if in $\Gamma(v)$ the vertexes are in different clusters and v doesn't belong to any cluster

All other point that are not in any cluster and are not hubs are outliers

- Clique is a subset of V where each v_1, v_2 are adjacent → tight social group (here used in a slightly relaxed way)

A cluster C is connected if $\forall v,w \in C$ → $\text{CONNECT}_{\varepsilon,\mu}(v,w)$

A cluster C is maximal if $\forall v,w \in V : v \in C \wedge \text{REACH}_{\varepsilon,\mu}(v,w) \Rightarrow w \in C$

SEQUENTIAL PATTERN MINING → discovering of subsequences in a sequence DB

(all records are sequences of ordered events (itemsets))

- format [transactionID, <Ordered sequence of Events>] eg. [T1, <(bread, milk), (bread, milk, sugar)>] (lexicographic order inside the itemset) → order between tuples of the sequence
  [T2, <(bread), (milk)>]

- **Web usage mining** → each event is a single item (1-itemset) since a user can access one page at the time
  - I want to discover users navigational patterns
    1. server-side: multiple users - single server. Good for recommendation systems (eg. Amazon). The cache is a problem
    2. client-side: single user - multiple servers. Web content personalization. Solves cache problems
    3. Proxy-Server: multiple users - multiple servers

Sequential DB is a **set** of sequential records (ordered sequences).
We want to find the set of all frequent sequences S in D, of items I at the given **min_sup**

A **sequence** $S = \langle e_1 e_2 \dots e_q \rangle$ (q-sequence) with e_j an itemset
- e_j is a **lexicographically ordered** list of items

eg (abc) < (abcd) and (ab) < (aba)

$t = \{i_1, \dots i_k\}$ (itemset), $t' = \{j_1, \dots j_r\}$ → $t < t' \iff (\exists 0 < k \leq \min\{k,r\} : i_k = j_k \ \forall j < k \ \wedge \ i_k < j_k)$ $\vee$ $(k < r \wedge i_k = j_k \ \forall j \leq k)$

- **TOTAL ORDER** for lexicographically ordered itemsets

- A item can appear only once inside an itemset, but more times across itemsets of a sequence
- A sequence $\alpha = \langle e_{i_1} \dots e_{i_m} \rangle$ is a **subsequence** of another sequence $\beta = \langle e_1 \dots e_n \rangle$ ($\alpha \leq \beta$) if $i_1 < \dots < i_m$ and $i_1 \geq 1 \wedge i_m \leq n$: $e_{i_j} \subseteq e_i$
  $\langle e(d)(ae) \rangle \not\leq \langle (a(bd)bc(de) \rangle$
  - a sequence is frequent if it is a subsequence of at least min_sup sequences in D
  - a subsequence is **maximal** if it's not a subsequence of any other sequence
  - S_a is frequent **closed** sequence if $\nexists S_b : S_a \leq S_b \wedge \sigma(S_a) = \sigma(S_b)$

$\sigma(S) = \frac{|\{s' : S \leq s'\}|}{|D|}$ (support)

ALGORITHM

1. **Sort Phase** → converts the original transaction DB to a sequential DB

2. **Litemset Phase** (large itemset) → find only the large itemsets (Apriori property, otherwise the sequence containing that itemset cannot be frequent)
   - has more than min_sup occurrencies, counting at most one per customer (different from traditional methods)

   Then **map** each large itemset to a set of contiguous integers
   - constant time compare of two itemsets and reduce the time to check if $S_a \leq S_b$

LITEMSET	MAP
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

3. **Transformation Phase** (events ∈ sequence)
   a) Replace each transaction with all the litemsets contained in the transaction
   b) empty transactions (with no litemset) are dropped, but considered for support counting ?

CUSTID	ORIGINAL SEQ	TRANSFORMED	MAPPED
1	<(10 20) (30) (40 60 70)>	<{30}, {(40), (70), (40 70)}>	<{1}, {2, 3, 4}>

(10 20) → dropped
(40 60 70) → {(40), (70), (40 70)} → {2, 3, 4}

4. **Sequence Phase** → Similar approach as Apriori
   Different algorithms: 1) Count-all: count all large sequences, including non maximal (AprioriALL)
   2) Count-some: try to avoid counting non-maximal S (AprioriSome)

5. **Maximal Phase** → find maximal sequences among large sequences

APRIORI ALL

1) $\forall S_1, S_2 \in L_{k-1}$: S_1, S_2 have in common the first $[1, k-2]$ elements, join them (add the last element of S_2 to S_1)

$L_3 = \{ \langle 123 \rangle, \langle 234 \rangle, \langle 124 \rangle, \langle 134 \rangle, \langle 135 \rangle \}$
$C_4 = \{ \langle 1234 \rangle, \langle 1243 \rangle, \langle 1354 \rangle, \langle 1345 \rangle \}$

- Important to add both of them, since I could join the two subsequences in both ways (the order is important for a sequence)

2) Delete sequence $S \in C_k$ if $\exists S' : S' \not\subseteq S$ (Apriori property)

$C_4 = \{ \langle 1234 \rangle, \langle 1243 \rangle, \langle 1354 \rangle, \langle 1345 \rangle \} \Rightarrow C_4' = \{ \langle 1234 \rangle \}$

APRIORI SOME → two phases 1) **forward phase**: Find all large sequences of some lengths, determined by the **next()** function (trade-off)

eg. AprioriAll next(k) = k+1

- skip some lengths if I think the corresponding sequences are frequent and non-maximal
  (I could use $hit_k = \frac{|L_k|}{|C_k|}$ as an index)

- In candidate generation, when L_{k-1} is not available I will use C_{k-1} to generate C_k
  - not a problem in terms of correctness, since $L_{k-1} \in C_{k-1}$

2) **Backward phase**: For all lengths that we skipped we delete the sequences $\in C_{k'}$ that are contained in some large sequence

- count the remaining sequences and keep the large
- Also delete, from the sequences found in the first phase, the one that are not maximal

APRIORI DYNAMIC SOME

1) **initialization phase** → All S : $|S| \leq$ step are counted (there is not any next(k) as in Apriori Some)
2) **forward phase** → all sequences whose lengths are multiples of steps are counted
   - We can directly find the C_7 sequences by joining $L_6 \bowtie L_3$ in a different way ← ?
3) **Intermediate phase** → generate the skipped candidate set
4) **Backward phase** → count the candidate set and remove the non maximal

(for each step)

: classic Apriori generation $C_k = L_{k-1} \bowtie L_{k-1}$

: on the fly generation (otf) $C_{k+step} = L_k \bowtie L_{step}$

Example step = 3 ⇒ assuming L_9 is found empty I have $C_1, L_1 - C_2, L_2 - C_3, L_3 - C_6, L_6 - C_9, L_9 = \{\emptyset\}$

3a) Generating C_7 from L_6 and C_8 from C_7

4a) Count C_8 and find L_8, remove non maximal from C_7, then count and find L_7

Repeat for C_4 and C_5

OTF generation → takes as argument L_k, L_{step} and a customer sequence c and returns the candidate $(k+step)$-sequences $\leq c$

$c = \langle c_1 \cdots c_n \rangle$ If $s_k \in L_k$ and $s_{step} \in L_{step}$ are contained in c, but do not overlap ⇒ possible candidate sub sequence of len $k+step$

$s_k.end = \min \{ j : s_k \not\subseteq \langle c_1 \cdots c_j \rangle \}$

$s_{step}.start = \max \{ j : s_{step} \not\subseteq \langle c_j \cdots c_n \rangle \}$

⇒ $\forall s_k, s_{step}$ join $s_k \bowtie s_{step}$ ⟺ $s_k.end < s_{step}.start$

$\langle s_k \; s_{step} \rangle$

→ It could be better to find C_{k+step} from L_k and L_{step}, rather then modify the apriori and found it by $L_k \bowtie L_k$ (with first k-step equal elements)

{ less candidate

- Anyway the performance test shows that Apriori Dynamic Some is the **slowest** because it generates too many candidates
  - there is no non-maximal pruning in the forward phase

APRIORI ALTERNATIVES

Bottlenecks of Apriori Methods

1) many DB scan
2) exponential # of short candidates when mining long pattern : a 100-sequential pattern requires $\sum_{i=1}^{100} \binom{100}{i} \approx 10^{30}$

f-list (frequent) : list of frequent items (1-itemset) in support descending order

f-list = b:5, c:4, a:3, d:3, e:3, f:2 ← sorted L_1, 1 DB scan to find it

PROJECTION RULE for SEQUENTIAL PATTERNS

i-projected db → set of sequential patterns containing i, but no items that followed i in the f-list

{ having i as subsequence

s is projected in the i-projected db as s_i, by removing infrequent items and any item following i in f-list

$\langle (ah)(bf)abf \rangle \xrightarrow{b\text{-proj}} \langle bb \rangle$

DB PARTITION

I can divide the dataset in 6 **non-overlapping** subsets

pattern sequencies containing f
pattern sequencies containing e, but not f
pattern sequencies containing d, but not f and e
⋮

Two way of projecting the db in order to achieve something more manageable (RAM resident) → final result is the **same**:

1) **Parallel projection**: generate all the projected db **at a time** but its result is a lot **larger** then originale dB
   - for each sequence, project it to all the elements of the f-list, following the rule → added to more db
   - the transaction can be projected in parallel in every db
   - Imagine to have on avg l frequent item per **transaction** ⇒ projected into $l-1$ dBs (excluding the first)
     The total size is $1 + \cdots + (l-1) = \frac{l(l-1)}{2}$ → size(DB') = $\frac{l(l-1)}{2}$ size(DB)
2) **Partition Projection** project a sequence to the projected db of the **least** frequent item in it, following the rule
   ↳ because a sequence is added to only one DB → partition the sequences (not / but serial)
   - After the processing of the db, remove the least frequent element and continue the projection
   - Size of the projection always smaller then the original dB - still one DB scan

projection level → size of the subsequence the sequence in db must have

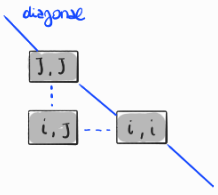
Two way of mining projected db

1) **Level-by-Level** → find f-list with one scan (L1), partition the db into f-list, mine L2, partition the projected dbs to L2, ... } both recursive
2) **Alternative level** → postpone the generation of projected db using annotations, free-span algorithm

b	4					
c	(4,3,0)	1				
a	(3,2,0)	(2,1,1)	2 <a,a>			
d	(2,2,2)	(2,2,0)	(1,2,1)	1		
e	(3,1,1)	(1,1,2)	(1,0,1)	(1,1,1)	1	
f	(2,2,2)	(1,1,0)	(1,1,0)	(0,0,0)	(1,1,0)	2
	b	c	a	d	e	f

FREE SPAN → similar to FP-growth
→ maximum 3 scans of the original dB

1) first scan get the f-list
2) construct the **Frequent Item Matrix** with another DB scan → triangular matrix $|L1| \times |L1|$
   - $F[i,j]$ → has one counter, reading the appearence of $\langle i_j, i_j \rangle$ in the dB
   - $F[j,k] = (A,B,C)$ with $A: \langle i_j, i_k \rangle$, $B: \langle i_k, i_j \rangle$, $C: \langle (i_j, i_k) \rangle$



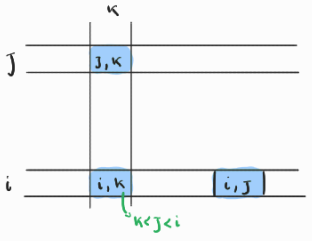
With this matrix we can:

a) generate L2 → If a counter is ≥ min_sup => add the corresponding sequence to L2

b) generate annotations for item repeating patterns
   - I. if $F[j,j] \geq$ min_sup → generate $\langle j j^+ \rangle$
   - II. for column $j \neq i$:
     - if $F[i,i] \geq$ min_sup => i^+
     - if $F[j,j] \geq$ min_sup => j^+
     - if only one of the tree counters of $F[i,j]$ is above min_sup => $\langle i^* j^* \rangle$ or $\langle j^* i^* \rangle$ otherwise generate $(i^* j^*)$

c) generate annotations for projected dbs of the form $\$ a_i a_j \$ \{b_p, \dots, b_q\}$ (only items to be included; subsequence to project on)

take a triangle with one vertex (i,j) $j<i$, then choose $K<j$. If $F(i,j), F(i,K), F[j,K]$ forms a **generating triple**
add $\$ ij \$: \{K\}$ → compute for all K and make the union



$F[e,f] = (1,9,0)$ $F[a,f] = (3,0,1)$ $F[a,e] = (1,1,3)$
→ it's not a generating sequence, because $\langle efa \rangle, \langle aef \rangle, \langle eaf \rangle, \dots$ cannot be frequent

But if $F[a,e] = (3,1,1)$ it could generate the pattern $\langle afe \rangle$
=> adding $\langle fe \rangle : \{a\}$

3) Discard F matrix, then scan the db the third and last time in order to find item-repeating sequences and projected dbs
4) Further mine the projected dbs → If they contains exactly tree items, then with one one scan we can generate all
   otherwise recursive, start from one

TIME SERIES → set of data points ordered in time

They can be **decomposed** in:
- trend: slow moving change
- seasonality
- residual: whatever cannot be explained by trend or seasonality

Forecasting → predicting the future using historical data
- It's different from regression tasks because here points have an order that could hide some pattern like seasonality

Mean Absolute Percentage Error: $MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \cdot 100$ (actual, forecast)

A **baseline model** (trivial solution to the forecasting problem, in order to make some benchmark) could be:
1) use the last known value → no mean or seasonality taken into account NAIVE METHOD
2) use the time serie mean
3) naive seasonal forecast → takes the last observed cycle and repeat it

Random Walk Process: process in which there is an equal probability of going up or down by a random number (ε_t)

$y_t = C + y_{t-1} + \varepsilon_t$ (C introduces a drift (trend)) with $\varepsilon_t \sim N(0, \sigma^2)$ also called white noise

- Formally a random walk is a time serie whose first difference $(C + \varepsilon_t)$ is stationary and uncorrelated

$\mathbb{E}[y_t] = \mathbb{E}[C] + \mathbb{E}[y_{t-1}] + \mathbb{E}[\varepsilon_t] = tC + y_0$
$Var[y_t] = t\sigma_\varepsilon^2$

To forecast random walk on the long period we cannot use statistical learning, since they change randomly
- I can use naive methods or baselines

Drift method: $\hat{y}_{t+k} = \frac{y_t - y_0}{t} \cdot k$ (forecast; first and last values in training set)

Stationary time serie: $\{y_t\} \Rightarrow \forall s,t$ the distribution of $(y_t, \dots, y_{t+n})$ does not change
- constant mean
- constant variance
- constant auto-correlation $cov(y_t, y_{t+n}) = \gamma(n)$ $\forall t$ → could be correlated, but independently from time

→ same distribution means relation among points doesn't change

- To make a time serie stationary we can implement some **trasformation** technique:
  - differentiating $y'_t = y_t - y_{t-1}$ helps stabilizing the mean
  - applying the log helps stabilizing the variance → e^x to untrasform

AUGMENTED DICKEY-FULLER TEST → assume the time serie is well described by $y_t = C + \alpha_1 y_{t-1} + \varepsilon_t$ AR(1)

α_1 is called the **root** of the time serie $\Rightarrow \{y_t\}$ will be stationary $\Leftrightarrow \alpha_1 \in (-1,1)$ → NO UNIT ROOT
- simple to see since $Var(y_t) = \alpha_1^2 Var(y_{t-1}) + \sigma_\varepsilon^2 \overset{?}{\Rightarrow} \sigma^2 = \alpha_1^2\sigma^2 + \sigma_\varepsilon^2 \overset{?}{\Rightarrow} \sigma^2 = \frac{\sigma_\varepsilon^2}{1-\alpha_1^2}$ (cannot be negative)

- **NH** (null hyp): a unit root is present (not stationary) → check the returned p-value with significance level
  - p-value < α_{signif} ⇒ reject

Auto Correlation $c_k = \frac{1}{T}\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})$ $r_k = \frac{c_k}{c_0} \leftarrow \frac{cov(y_t, y_{t-k})}{Var(y_t)}$ (autocorrelation)

AutoCorrelation Function (ACF) $f(k) = r_k$
- when we have trend, autocorrelation for small k tends to be large and positive
- with seasonality, we have peaks at multiples of the seasonal frequency

→ stationarity $\not\Rightarrow$ white noise wn is a special case with zero autocorrelation $\mathbb{E}[\varepsilon_t \varepsilon_{t-1}] = 0$

MOVING AVERAGE MODEL → stationary, but there is autocorrelation, past ACF coefficients and if they are not significant after lag q ⇒ MA(q)

$MA(q) = \mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \dots + \theta_q\varepsilon_{t-q}$
- depends linearly with mean, current and past error terms
- it is a stationary time series

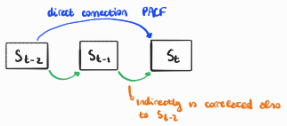
- to choose q, I need to plot ACF and see after which lag q, all coefficients are not significant
  - predicting after q steps in time simply returns the mean (no error to use) ⇒ use **rolling forecast** to predict up to q steps ahead, then retrain with new observation

AUTOREGRESSIVE MODEL → regression of a variable against itself

(when ACF is not conclusive, then plot PACF and if coeffients are abruptly non significant after lag p ⇒ AR(p))

- C is no longer the mean

$AR(p)$: $y_t = C + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \varepsilon_t$



- to find the order p I need to calculate the **partial autocorrelation function** (PACF)
  - PACF measures the relations after removing the effects of the observation in between
  - is actually a regression to find ϕ_p

AUTOREGRESSIVE MOVING AVG → when both ACF and PACF doesn't rapidly decay

$ARMA(p,q)$: $y_t = C + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$

- $ARMA(0,q)$ is equivalent to $MA(q)$, $ARMA(p,0) = AR(p)$

- To determine p and q we use the **Akaike Information Criterion** (AIC)

$$AIC(p,q) = 2k - 2\ln(\hat{L}) = p+q - 2\ln P(y_t \mid ARMA(p,q))$$

(explainability; goodness on fitting data)

1) for every tuple (p_i, q_i) evaluate AIC and select the best (the lowest)
2) Analyze the model residual which is the difference of $\Delta y_i = |y_i - \hat{y}_i|$

If they are **white noise**:
- uncorrelated → Ljung-Box test
- $\sim N(0,\sigma^2)$ → Q-Q plot is a straight line

⇒ use them to forecast, otherwise change (p', q')

Ljung-Box test → statistical test to determine if the autocorrelation is significant different from zero
- NH: data is independently distributed (no auto-correlation)
- p-value > 0.05 → cannot reject → white noise

AUTOREGRESSIVE INTEGRATED MOVING AVG → when the series is not stationary I will apply d times the differentiation and then ARMA

(d: lowest # to make it stationary)

$$y'_t = C + \phi_1 y'_{t-1} + \dots + \phi_p y'_{t-p} + \varepsilon_t + \theta_1 \varepsilon'_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

y': differenced time series; ε': differenced error

SEASONAL ARIMA → adds seasonality parameters to $ARIMA(p,d,q)$ → $SARIMA(p,d,q)(P,D,Q)_m$

(m: seasonal frequency = # of observation per cycle)

P: order of seasonal AR(P) process, D: order of seasonal integration, Q: order of seasonal MA(Q) process

- $SARIMA(p,d,q)(0,0,0)_m = ARIMA(p,d,q)$

m = 12 if 1 year cycle, with data every 1 month

P = 2 including two past value, multiple of m → y_{t-12}, y_{t-24}

- Now I need to fit (p,q,P,Q) tuples
- d and D are selected before

SARIMAX MODELS → it adds a linear combination of exogenous variable to SARIMA

$$y_t = SARIMA(p,d,q)(P,D,Q)_m + \sum \beta_i X^i_t$$

(X^i_t: need to know the variables at time t)

- to predict at $t' > t+1$ we need to forecast also the exogenous variables (may be easier to predict)

ML MODELS for time series

- statistical approaches works well when we have small dataset (less then 10k points) otherwise they are very slow
- ML when we have large datasets or residuals are not white noise

1) Single-step model: predicts the single value for the next time step → single step
2) Multi-step model: predicts the single value but for many time steps into the future
3) Multi-output model: when predicting for more then one variable (temp and wind speed) → multiple single steps

TIME ENCODING → since we want to encode infos for cyclical behaviour (if it increase always is not useful)

If I want to map the 24h cycle: $\vec{t} = (t_1, t_2)$

$$t_1 = \sin\left(t_s \frac{2\pi}{24 \cdot 60 \cdot 60}\right) \qquad t_2 = \cos\left(t_s \frac{2\pi}{24 \cdot 60 \cdot 60}\right)$$

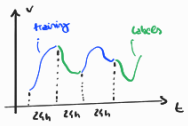
(t_s: timestamp in seconds)

(t_1: not sufficient 12am = 12pm)

Data windowing process → we define a sequence of points that are training and those that are labels

ex. 24h period to predict the next 24h period

- huge data waste

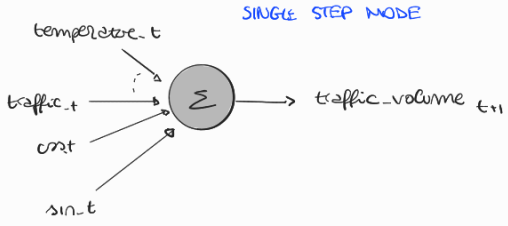


- **Moving windows with step K** : shift of K positions the training/labels windows

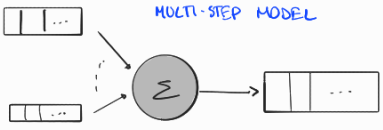
Baseline model for multistep process : input width = N, label width = N, shift (K) = N

1) last known value for the next N steps
2) repeat the last N steps to the next

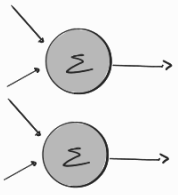
SINGLE STEP MODE



MULTI-STEP MODEL



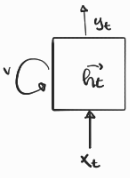
MULTIVARIATE MODEL



- simple multivariate linear regression

Often **RNN** are used → but they soffer of short-term memory

- previous values stop having influence in the for future (seasonality)



LSTM try to solve this problem introducing input and forget gate to select the new infos to store and the one to forget

FOUNDATIONAL MODELS → big pre-trained models

- normally good performances if they are trained with something similar
- nevertheless they are not explainable and may not be the most efficient solution

DATA STREAMING

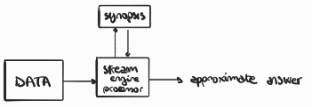
need to compute the model as data arrives

Stream → potentially unbounded, ordered sequence of instances $S = \{x_1, \dots, x_N\}$ $x_i \in X^d$, $N \to +\infty$

- traditional techniques → entire dataset that needs to be scanned multiple times and randomly accessed. Computationally heavy learning phase
- data streams techniques → impractical/impossible to store the entire dB. multiple scans or random access

Typically we need to handle massive volumes of data and their rapid production frequency (low complexity training, impractible for each instance to re-execute the algorithm)

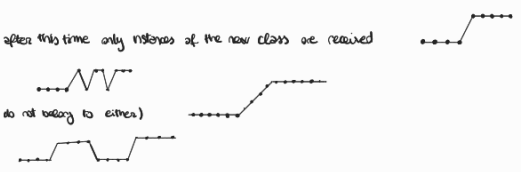
Requirements (algorithms usable also for massive dB)
1) single pass: record examined at most once
2) bounded storage: limited memory for storing synopsis (summarization of data)
3) Real-time



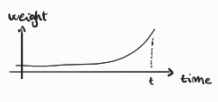
1) approximate answers
2) e.g. with deterministic bounds
3) e.g with probabilistic bounds (ε-δ) (with $1-\delta$ probability; ε: distance from correct ans)

SYNOPSIS → particular data structure enable to incrementally summarize data
1) feature vectors → summary of data instances by features extraction
2) prototype arrays → keep only representative instances
3) coreset trees → keep the summary in a tree structure (like BIRCH clustering algorithm)
4) grids → keeps data feature space in a quantized grid

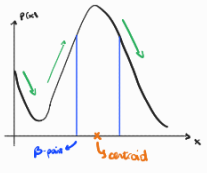
CONCEPT DRIFTS: unforseen change in statistical properties of data stream instances over time
1) sudden concept drift: sudden change in two consecutives instances at once, after this time only instances of the new class are received
2) gradual concept drift: # of one class decrease gradually, the other increasing
3) incremental concept drift: evolving to the new class gradually (in the meantime do not belong to either)
4) recurring concept drift: data instances change between two classes in terms



WINDOW MODEL (helps in reducing concept drift effects) → how to weight recent and past data
1) damped window model: recent have more weight then older one (importance decreases over time)
   - decay function to model the feature importance like: $f(t) = 2^{-\lambda(t_{now} - t)}$ (λ: decay rate)
2) Land-mark window model: locate data in a window between two landmarks with same importance
   - consecutive windows do not intersect $W_m = [x_{m \cdot w}, \dots, x_{m \cdot w + w - 1}]$ (m: m^{th} window; w: window size) x_i is in the $m = \lfloor \frac{i}{w} \rfloor$ windows
3) Sliding window model: window shift one position at each time (FIFO style)
   - consecutives windows mostly overlap $W_m = [x_m, \dots, x_{m+w-1}]$



CLUSTERING → since no y_{true} is needed is easily used with data streams



ADAPTIVE STREAMING K-MEANS

1) collect e data instances
2) keep best clustering parameters (K, centroids) by evaluating silhouette coefficients (for each point e, calculate all distances $\in O(dK)$ and repeat for cs different centroids sets) $\in O(d \cdot e \cdot K \cdot cs)$
   - determine the distributions (pdf) of each variable independently → may follow different distributions
   - to find K_i identify directional change in the pdf: $\in O(e)$ $K \in [K_{min}, K_{min} + K_{max}]$ for all features $\in O(d \cdot e)$
   - split the pdf in K equiprobable areas (boundaries are called β-points), assign candidate centroids to the center of those regions (typically high data density)
3) continous clustering with K means of the data stream, if a conceptual drift is detected, the restart the initialization phase (assign a single point) $\in O(dK)$ (↳ predicted by comparing μ and σ stored)

- The worst case scenario to cluster a point, involving the computation of initialization phase is $\in O(de) + O(d \cdot e \cdot K \cdot cs) + O(dK)$