Summary – Data Mining

V 1.2

Indice

This summary hasn't been revised by any professor

If you find any errors please contact me :)

By Flavio Primo

Introduction

Data Mining: process of identifying: valid, novel, actionable and understandable patterns in data.
Problems: noise, uninteresting relations, spurious relations → DM algorithms must be robust

Types of problems to solve:

- **Descriptive** → find how data is structured
- **Predictive** → predict future values given some premises

} **Prescriptive** → get insight on data and get action

Process:

1. **Selection**: select which data to use
2. **Cleaning**: clean the data from errors, inconsistencies and deal with missing values
3. **Transformation**: remove redundant features, create new features
4. **Mining**: select and apply mining approach (regression, classification, association)
5. **Validation**: check if results and patterns are sound and interesting
    - interestingness measure: confirm prior hypothesis
    - objective vs subjective: while objective are based on data and metrics, subjective are based on user's belief on data

Completeness and Optimization of patterns:

- **Completeness**: if all interesting patterns are found
- **Optimization**: ability to search only interesting patterns

Data Representation

How data is represented:

- **Instances**: rows of a dataset
- **Attributes**: columns of a dataset
- **Concepts**: relations and things that can be learned from data

Attribute types

Algorithms can check for validity and perform differently wrt specified data types.

Numeric

Domain: real (continuous), integer (discrete)

	Differences	Ratio	Zero point
Interval	x	x	
Ratio		x	x

Characteristics:

- measured in fixed and equal units
- sum and product don't make sense as operations

Categorical

Domain: set valued composed of a set of symbols

	Operations	Ordering	Distance measure	Can be binary
Ordinal	Equality/Inequality	x		
Nominal	Equality			x

Missing values

Categories of missing values:

	Depends on	
	Other observed data	Missing data
MCAR – Missing Completely At Random		
MAR – Missing At Random	x	
NMAR – Not Missing At Random		x

Strategies to deal with missing data:

- **Deletion**: involves deleting data related to the missing values
  - *Listwise*: delete rows with missing values → simplest but reduces data
  - *Pairwise*: delete cells with missing data → each analysis on all available data but can't compare analyses
- **Single imputation**: imputation based on statistical methods on the single feature.
  - *Mean/mode substitution*: but reduces data variability
  - *Dummy variable*: create indicator for missing value but biased estimate
  - *Regression*: replace missing values with a regression model
- **Model-based imputation**: imputation based on ML models and multiple features. Good for MCAR and less for MAR.

Inaccurate values

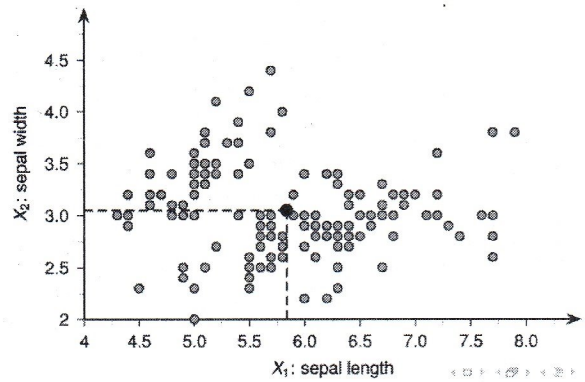
- Data in a **format** not suitable for data mining → reformat data

  data formats:
  - ARFF: supports data type, sparse attributes
  - DSPL: XML defines data type and CSV defines the data
- **errors and omissions** in dataset → check with domain expert and check for missing values

- **typos** in nominal attributes → check for consistency
- **measurements errors** → check for outliers

Geometric view

Consider row as a point with columns as dimensions.



Data Exploration

Preliminary exploration of data aimed at identifying most relevant characteristics.

- Visualization
- Summary statistics
- Clustering and anomaly detection

Summary statistics

Summary statistics: numbers that summarize data.

Given: • x an item of the dataset • n dataset length ($n=|\text{dataset}|$)

Frequency & Mode

Measure the percentage of time a value appears in a dataset.

- $Frequency(x)=\frac{|x|}{n}$
- $Mode=\arg\max_{x} Frequency(x)$

Mean & Median

Measure the location of a dataset.

- $Mean(x)=\bar{x}=\frac{1}{n}\sum_{i=1}^{n} x_i$ approximated and sensible to outliers

- $Median(x)=\begin{cases} x_{r+1} & if\ n=2r+1 \\ \frac{x_r+x_{r+1}}{2} & if\ n=2r \end{cases}$ gives the central value

Percentiles

Divides dataset values in percentaged sized sets.

- $Percentile(k)=(n+1)\cdot\frac{k}{100}$ dataset must be ascending ordered
- $TriMean=\frac{Percentile(25)+2\cdot Percentile(50)+Percentile(75)}{4}$ weighted mean of the 1°, 2° and 3° quartile
- $Truncated\ Mean(min_k, max_k)=\frac{2}{n}\sum_{i=min_k\cdot n+1}^{max_k\cdot n} x_i$ discards data above and below a certain percentile (IQM uses $min_k=25\%$, $max_k=75\%$)

Range & Variance

Measures the range and how the values are spread over the dataset.

- $Var(x)=s_x^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$ but sensitive to outliers
- $AAD(x)=\frac{1}{n}\sum_{i=1}^{n}|x_i-\bar{x}|$, $MAD(x)=Median(\{|x_1-\bar{x}|,\dots,|x_n-\bar{x}|\})$, $IQM\ range(x)=x_{75\%}-x_{25\%}$

Correlation Analysis

Given 2 features, it measures how strongly one implies the other (may be spurious!).

- **Numerical** → Pearson
- **Categorical**
  - Ordinal → Spearman Rank
  - Nominal → Chi-square
  - Binary → Point-biserial

Outliers

Outlier: instance which do not comply with the general behavior of data.

Detected by:

- **Statistical tests**
- **Distance** measures (far away instances from clusters are outlier)

- **Deviation**-based methods (analyze differences in the main characteristics of objects in a group)

If not the focus of analysis they are eliminated in various ways:

- **Winsorizing**: instances below 5% and above 95% percentiles are outliers
- **Trimming**: eliminate outliers instances

Normalization

Attributes that have vastly different scales must be normalized to a given range (ease comparison and models application).

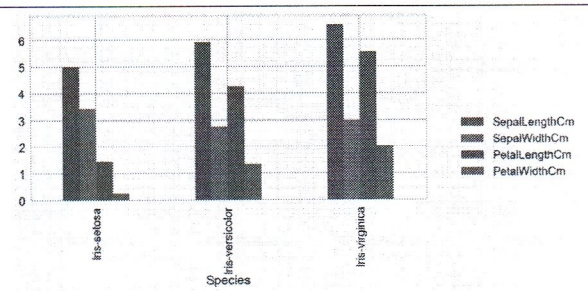
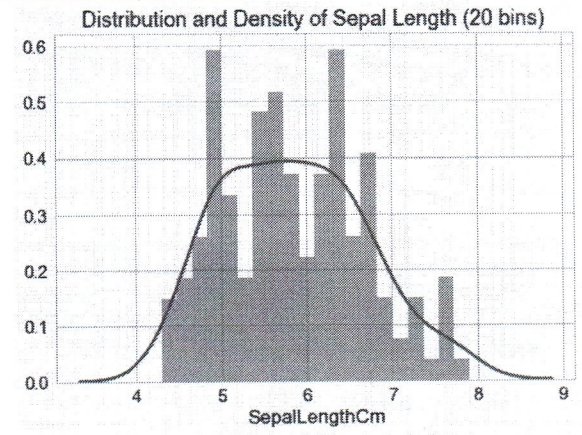
- **Range normalization** $x_i' = \frac{x_i - min(\text{dataset})}{max(\text{dataset}) - min(\text{dataset})}$
- **Standard Score normalization** $x_i' = \frac{x_i - \mu}{\sigma}$

Visualization

Visualization: conversion of data into visual/tabular format so humans can detect:

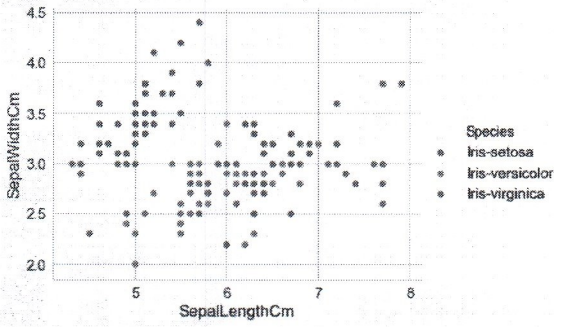
- general patterns and trends
- outliers and unusual patterns

Plots

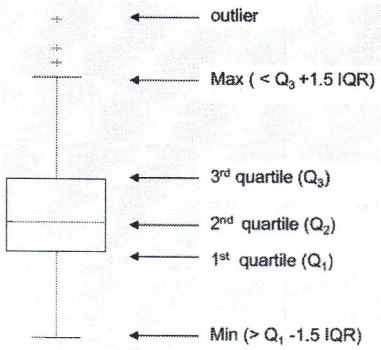
Bar plots: vertical (or horizontal) bars to compare categories. • x: compared categories (can also cluster them) • y: values	
Histograms: graphical representation of data distribution. • x: bins of values ascending ordered • y: frequency of each bin	

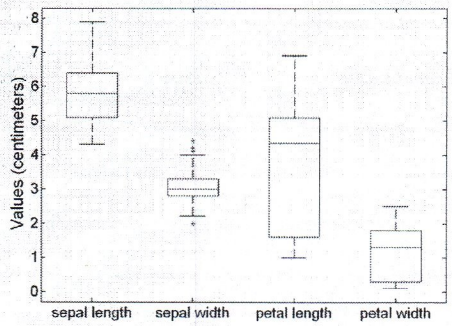
Scatter plots: let understand relation between 2 or more attributes.

- axes, color, shape, size: value of the attribute



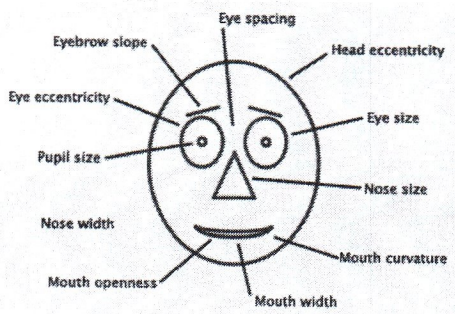
Box plots: compares various information about one attribute.

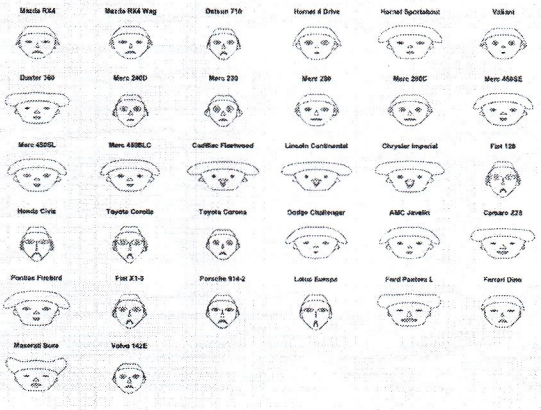


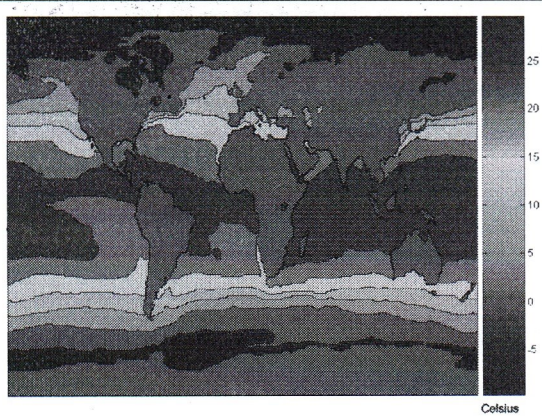


Chernoff faces: describes several attributes at once (star plot is similar).

- Face attribute: describe value attribute





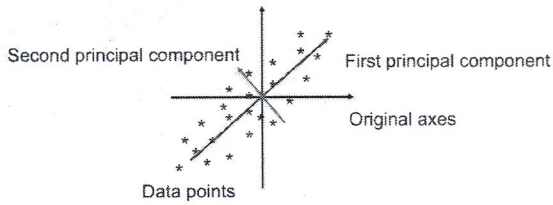
Contour plots: spatially partition similar valued data on a grid with colours.	

PCA (Principal Component Analysis)

Find projections that capture the largest amount of variation in data.

1. **Normalize** dataset (affected by scale)
2. compute **k orthonormal (unit) vectors**

   each input data (vector) is a linear combination of the k PC (principal component) vectors
3. **PC descendengly sorted**
4. **remove the least strong PC**



- only for numeric data
- good if high dimensionality of data

T Distributed Stochastic Neighbor Embedding

Non-linear dimensionality reduction technique that maps high-dimensional data to 2/3 dimensions. It uses "springs" that keep together similar data points and

1. **Define probability distribution over pairs** of high-dimensional data points so that:
   - similar data points have high prob to be picked
   - dissimilar data points have low prob to be picked
2. Define a similar distribution over the data points
   - minimize Kullback-Leibler divergence between the 2 distributions wrt to the location of the datapoints
   - minimize the score (gradient descent)

Problems:

- stochastic: different initializations lead to different results

- do not apply to too many dimensions

Association Rules

Association rule: frequent patterns in the form of $X \rightarrow Y$ implications where X, Y are itemsets taken from transactions of a database.

- **itemset**: collection of one or more items
- **k-itemset**: itemset of size k
- **Support count:** $\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}|$ number of transactions with itemset X
- **Support**: $s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$ how often X and Y appear together in transactions over the dataset
- **Confidence**: $c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$ how often items Y appear in trans that contain X

Goal: find all rules having thresholds $s \geq \text{minsup}$ and $c \geq \text{minconf}$

Find association rules

Brute force

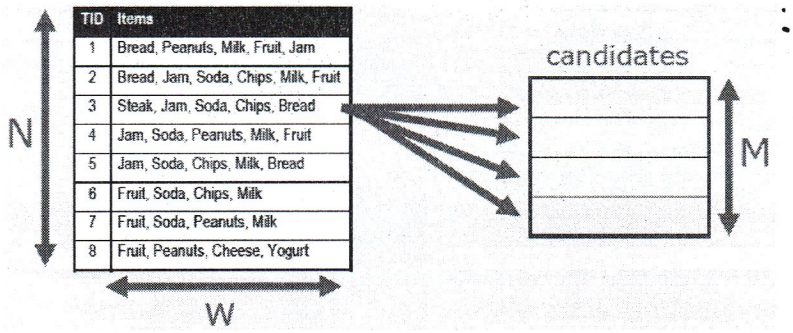
1. List all possible association rules
2. compute support and confidence for each rule
3. prune rules that fail "minsup" and "minconf" thresholds

Frequent Itemset Generation

Idea: Rules originating from the same itemset have identical support but can have different confidence.

1. Generate all itemsets with $s \geq \text{minsup}$
2. compute all confidence for each rule
3. prune rules that fail "minconf" thresholds

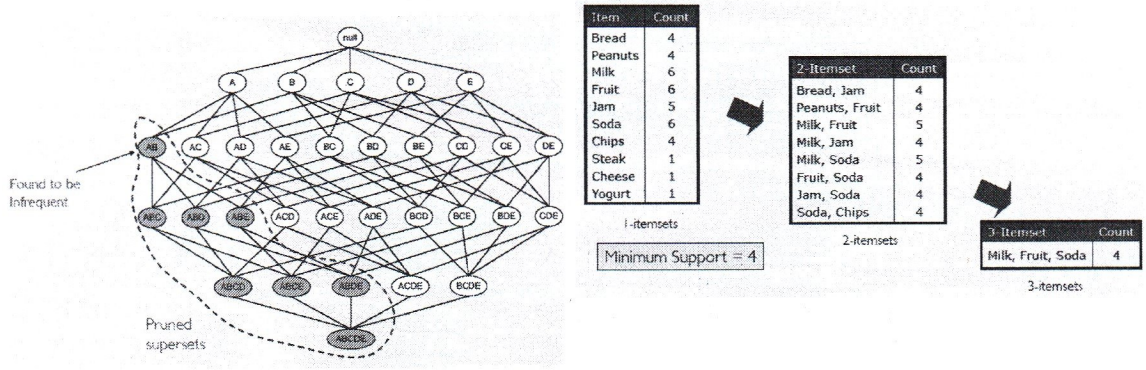
Cost: $O(NMw)$ better than brute-force but still expensive



Apriori

Reduces the number of candidates M wrt "Frequent Item Generation".

Idea: if itemset is frequent → all of its subsets must also be frequent
$\forall X, Y | (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$ (anti-monotome)



1. k=1
2. generate all k-itemsets (if k>1 generate all k-itemsets containing only the preceding survived (k-1)-itemsets)
3. compute support for all k-itemsets
4. prune itemsets with s<minsup
5. if k=kmax then end else k++ and goto (2)

Eclat

Reduces the number of comparisons NM wrt "Frequent Item Generation".

Idea: generate efficient vertical database data structure → comparisons for free without scanning whole db.

Algorithm is same as Apriori, but uses a different data structure.

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

t	i(*t*)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database

t(*x*)				
A	*B*	*C*	*D*	*E*
1	1	2	1	1
3	2	4	3	2
4	3	5	5	3
5	4	6	6	4
	5			5
	6			

Vertial Database

FP-Tree

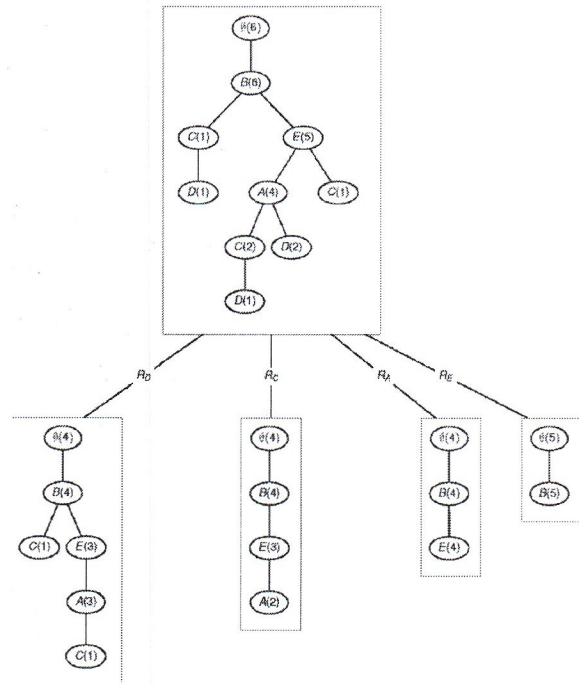
Mines frequent patterns without candidate generation.

Advantages:

- Completeness and compactness
- avoids costly db scans
- divide & conquer method

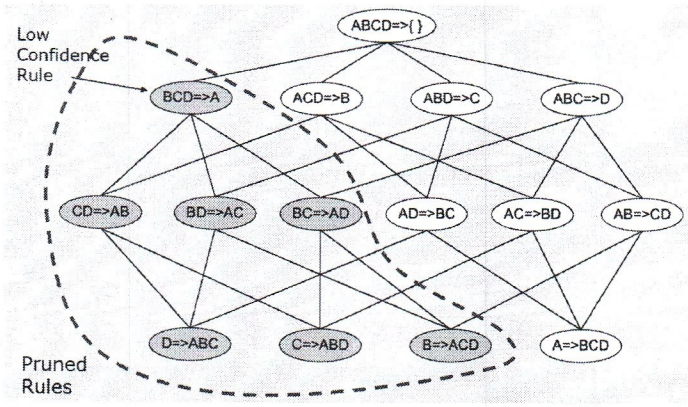
Steps:

1. generate 1-itemsets, remove the ones which not satisfy minsup and sort descendingly
2. reorder items in transactions
3. build fp-tree from transactions and explore it in a depth-first fashion
    1. find freq patterns (check minsup)
    2. if leaves reapply (3) on the same tree



Rule Generation

Confidence is antimonotone: $L=A,B,C,D | c(ABC \Rightarrow D) \geq c(AB \Rightarrow CD) \geq c(A \Rightarrow BCD)$



Rule Assessment measure

How to set appropriate $s \geq \text{minsup}$?

- High → miss itemsets with interesting rare items
- Low → computationally expensive and lots of itemsets

Surprise of a rule measures:

- $\textbf{\textit{Lift}}(X \rightarrow Y) = \dfrac{\sigma(X \cup Y)}{\sigma(X) \cdot \sigma(Y)} = \dfrac{c(X \Rightarrow Y)}{\sigma(Y)}$ ratio value

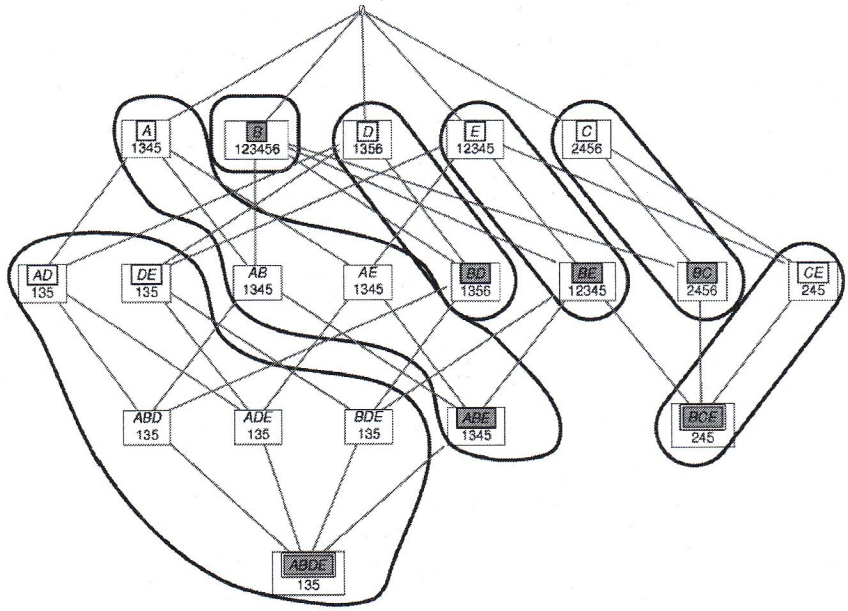
- ***Leverage*** $(X \rightarrow Y) = \sigma(X \cup Y) - \sigma(X) \cdot \sigma(Y)$ absolute value

Summarize itemsets

ABDE **Maximal:** $M = \{X | X \in F \wedge !\exists Y \supset X, such\ that\ Y \in F\}$ if X has no frequent supersets

BC **Closed:** $\sigma(X) > \sigma(Y) \forall Y \supset X$ if all supersets of X have strictly less support

CE **Minimal Generators:** $G = \{X | X \in F \wedge !\exists Y \subset X, such\ that\ \sigma(X) = \sigma(Y)\}$ if X has no subsets with same support



Sequential pattern mining

Association rules applied to ordered transactions.

- $s = \underbrace{s_1}_{s[1]}, \ldots, \underbrace{s_n}_{s[n]}$ sequence
- $r = r_1, \ldots, r_m$ subsequence of s ($r \subseteq s$) if all the elements of r are contained in s in the same order (it can also be a consecutive subsequence if the elements of r in s are consecutive).

Frequent subsequences (need to consider all permutations with sequences!):

- $support(r) = |\{s_i \in \text{dataset} | r \subseteq s_i\}|$ number of sequences in which subsequence r is present
- $relative\_support(r) = \frac{support(r)}{N}$

GSP Algorithm

Searches sequence prefix tree using a breadth-first search. Similar to Apriori.

Generate tree starting with k-sequence set and expand the node with k+1 sequenceset only if satisfied $s \geq \text{minsup}$. For each expansion consider all possible permutations!

Association rules for Classification

1. Map data into tabular format for association rules
2. apply association rule mining focusing the search for association rules such as $X \rightarrow \underbrace{c_i}_{\text{class label}}$
3. prune association rules using pessimistic error based method
4. sort and build the classifier

Linear Regression

Goal: predict a new value y given a never seen before x input.

Prediction is based on a linear model that tries to approximate inputs to outputs.

$$\underbrace{y_i}_{\text{predicted output}} = \underbrace{w_0}_{\text{bias}} + \underbrace{\sum_{j=1}^{D} \underbrace{w_j}_{\text{weight}} \cdot \underbrace{h_j(\vec{x}_i)}_{\text{input feature}}}_{\text{iterate over each input dimension}} + \underbrace{\epsilon}_{\text{error}}$$ linear prediction model.

$$\underbrace{RSS(w)}_{\text{Residual Sum of Square}} = \underbrace{\sum_{i=1}^{N} \left(y_i - w_0 - \sum_{j=1}^{D} w_j \cdot x_{ij} \right)^2}_{\text{iterate over each input}}$$ measures the model error for each prediction.

Weights are trained as follows:

- **Analitically**: put the gradient to 0 and solve. But unfeasible.
- **Gradient descent**: iteratively update the weights

$$w_j^{(t+1)} = w_j^{(t)} + 2 \cdot \underbrace{\eta}_{\text{learning rate}} \cdot \sum_{i=1}^{N} h_j(\vec{x}_i) \cdot \left(\underbrace{y_i}_{\text{actual value}} - \underbrace{\hat{y}_i(\vec{w}^{(t)})}_{\text{predicted value}} \right)$$

Model Evaluation

Metric

$$R^2 = 1 - \underbrace{\frac{RSS}{TSS}}_{=0 \text{ if perfect fit}}$$ with $$\underbrace{TSS}_{\text{Total Sum of Squares}} = \sum_{i=1}^{N} (y_i - \bar{y})^2$$

Measures how well the regression line approximates the real data points.

Overfitting

When the model has good performance on training set and bad on the test set.

Usually happens when weight are high → **idea: penalize high weights**

	RSS	*Weight update*
Ridge	$RSS_{Ridge}(w) = RSS(w) + \alpha \cdot \sum_{j=0}^{D} w_j^2$	$w_j^{(t+1)}{}_{\text{Ridge}} = w_j^{(t+1)} + 2\alpha w_j$

Lasso

Also feature selection by "zeroing" useless features

$$RSS_{Lasso}(w) = RSS(w) + \alpha \cdot \sum_{j=0}^{D} |w_j|$$

$$w_j^{(t+1)}{}_{\text{Lasso}} = \begin{cases} \dfrac{w_j^{(t+1)} + \frac{\alpha}{2}}{\sum_{i=1}^{N} h_j(\vec{x}_i)^2} & \text{if } w_j^{(t+1)} < -\frac{\alpha}{2} \\ 0 & \text{if } w_j^{(t+1)} < \left[-\frac{\alpha}{2}; \frac{\alpha}{2}\right] \\ \dfrac{w_j^{(t+1)} - \frac{\alpha}{2}}{\sum_{i=1}^{N} h_j(\vec{x}_i)^2} & \text{if } w_j^{(t+1)} > \frac{\alpha}{2} \end{cases}$$

Classification

Goal: assign a discrete class y given a never seen before x input.

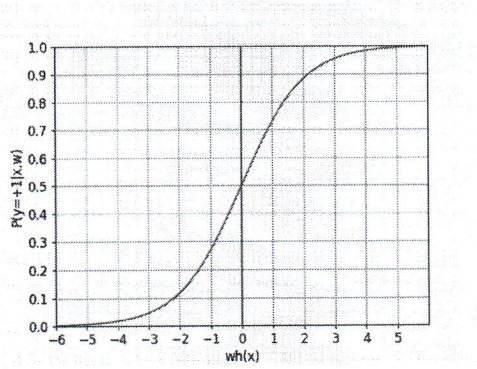
Prediction is based on a model which separates datapoints on the hyperplane, each of which represents a class.

Logistic Regression

2 class linear prediction model. Similar to regression then applies a sign function to discriminate class.

1. $Score(\vec{x}_i) = \sum_{j=0}^{D} w_j \cdot h_j(\vec{x}_i)$ score assigns a value to the given input

2. $P(\hat{y}_i = +1 | \vec{x}_i) = \frac{1}{1 + e^{-Score(\vec{x}_i)}}$ get probability of having class +1



3. $\hat{y}_i = \begin{cases} +1 & \text{if } P(\hat{y}_i = +1 | \vec{x}_i) \geq \lambda \\ -1 & \text{if } P(\hat{y}_i = +1 | \vec{x}_i) < \lambda \end{cases}$ assign class with the given probability

   usually $\lambda = 0.5$, but it can be modified to control precision and recall

   - $\lambda > 0.5$ → precision ↑, recall ↓
   - $\lambda < 0.5$ → precision ↓, recall ↑

Weights are trained as follows:

1. $l(\vec{w}) = \prod_{i=0}^{N} P(\hat{y}_i = +1 | \vec{x}_i, \vec{w})$ maximum likelihood
2. $ll(\vec{w}) = l(\vec{w})$ apply log likelihood to ease computation
3. $\frac{\delta ll}{\delta w_j}$ update weight

Overfitting

Apply regularization as for regression.

- Ridge $l(\vec{w}) - \alpha \cdot \|\vec{w}\|_2^2$
- Lasso $l(\vec{w}) - \alpha \cdot \|\vec{w}\|_1$

Multiclass

One vs Rest: create a classifier for each target class. Assigned class is the most probable.

One Hot Encoding

Map each categorical attribute with n values into n binary attributes, with each one describing one specific attribute value.

Credibility: Robust Evaluation of Models

How to evaluate a model?

- **Metrics**: how to evaluate a model
- **Methods**: how to obtain reliable estimates
- **Model comparison**: compare model performance
- **Model selection**: what kind of model to prefer

Metrics

How to evaluate a model.

Regression

- $\underset{\text{Residual Sum of Square}}{RSS(w)} = \underbrace{\sum_{i=1}^{N} \left(y_i - w_0 - \sum_{j=1}^{D} w_j \cdot x_{ij} \right)^2}_{\text{iterate over each input}}$
- $R^2 = 1 - \underbrace{\frac{RSS}{TSS}}_{=0 \text{ if perfect fit}}$ with $\underset{\text{Total Sum of Squares}}{TSS} = \sum_{i=1}^{N} (y_i - \bar{y})^2$
- $\underset{\text{Mean Square Error}}{MSE} = \frac{1}{N} \cdot (y_i - \hat{y}_i)^2$ dimensional error squared
- $\underset{\text{Root Mean Square Error}}{RMSE} = \sqrt{MSE}$ dimensional error

Classification

Confusion Matrix

		Predicted	
		Y	*N*
Actual	*Y*	C(TP)	C(FN)
	N	C(FP)	C(TN)

Can also associate a cost C which weights more or less a certain prediction type.

2 alternative metrics to evaluate a confusion matrix:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$ → $F1 = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$

Methods

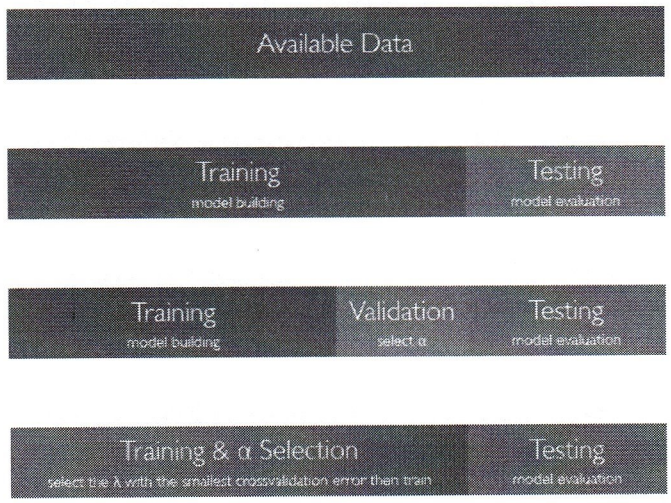
How to obtain reliable estimates.

Rule: training data must not be used to evaluate the model.

Stratified sampling: partition in a way that each class is represented in equal proportions

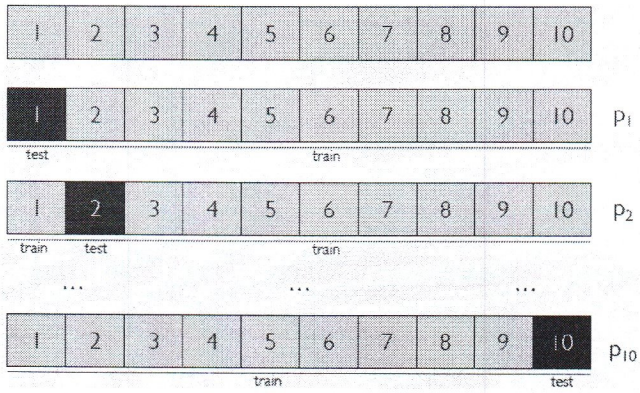
Model performance affected by: Class distribution, Cost of misclassification, Size of training and test sets

- **Holdout**: partition the dataset



- **Random subsampling:** repeated holdout
- **Cross Validation:** partition dataset into k disjoint sets. Usually k=10 or k=n (LOO)
  1. Split data into k subsets of equal size
  2. Test each subset while the remaining subsets are used for training
  3. After all estimations get the average error

After all estimations perform training on the whole dataset.



- **Bootstrap**: sampling with replacement. Same instance can be reused for other train or test set.
  1. Sample dataset of n instances n times to create the training set (also same instance!)
  2. use not used instances as test set
  3. compute the error $error = \underbrace{0.632}_{1-\left(1-\frac{1}{N}\right)^N} \cdot error_{test} + \underbrace{0.368}_{\left(1-\frac{1}{N}\right)^N} \cdot error_{train}$ train error is weighted less
  4. repeat several times

Model Comparison

Apply t-test and compute p-value which represents the probability that the reported difference is due to chance.

1. Generate **k folds for each model** A ($\theta_1^A, \ldots, \theta_k^A$) and B ($\theta_1^B, \ldots, \theta_k^B$)
2. Compute $\delta_i = \theta_i^A - \theta_i^B$, $\mu_\delta = \frac{1}{k} \cdot \sum_i \delta_i$, $\sigma_\delta = \sqrt{\frac{1}{k} \cdot \sum_i (\delta_i - \mu_\delta)^2}$
3. Set a **confidence level c** (for 95% → $c = 0.05$) and **test the hypothesis**:
   - $H_0 : \mu_\delta = 0$ if models have **similar** performance (p-value $\geq 1 - c$)
   - $H_a : \mu_\delta \neq 0$ if models have **different** performance (p-value $< 1 - c$)

t-test can be paired or unpaired (if estimates are from different datasets).

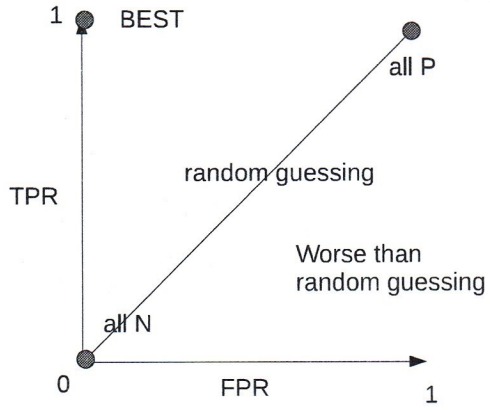
ROC (Receiver Operationg Characteristic) curves

Performance of each classifier is represented as a point on the curve.

- x: $FPR = \frac{FP}{FP+TN}$
- y: $TPR = \frac{TP}{TP+FN}$

ROC curve can be changed by:

- classification threshold
- sample distribution
- cost matrix



Model Selection

What kind of model to prefer.

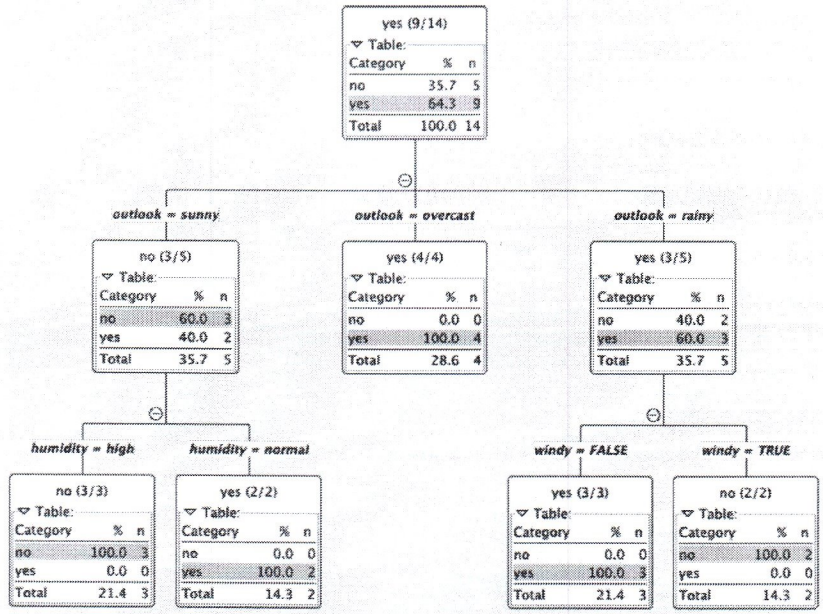
Occam's Razor: model with less complexity with highest possible accuracy

No Free Lunch Theorem: each problem is different and has a model that works better than others. That same model will perform worse on other problems.

Decision Trees

Tree which has as nodes tests on attributes. New cases find a path from root to leaf that suits them.

- Node: test on attribute that splits training samples
- Branch: outcome of test
- Leaf node: class label or class label probability

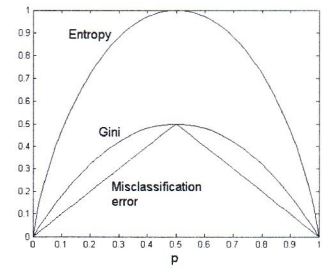


Tree construction

1. **Build tree (top-down)**: start with all training samples at the root. Then recursively partition them by choosing one attribute at a time.
2. **Prune tree (bottom-up)**: remove subtrees or branches starting from leaves to improve accuracy on new cases.

Splitting attribute

At each node find the available attribute which gives the purest partitioning.



Criterion =

- $Entropy(p_1,\dots,p_n) = -\sum p_i \cdot \log_2(p_i)$
- $Gini(p_1,\dots,p_n) = 1 - \sum_{i=1}^{n} p_i^2$

Thus we define:

$$Info(A) = Criterion\left(\frac{A_{\text{YES}}}{|A|}, \frac{A_{\text{NO}}}{|A|}\right) \qquad InfoAfter(A_1,\dots,A_n) = \sum_{i=1}^{n} \frac{|A_i|}{\sum_{j=1}^{n}|A_j|} \cdot Info(A_i)$$

Gain

- $Gain(A) = \underbrace{Info(A)}_{\text{info BEFORE split}} - \underbrace{InfoAfter(A_{\text{AFTER}})}_{\text{info AFTER split}}$ (the higher the better partition is)

  but biased towards attributes with a large number of values (examples: ID)

- $GainRatio(A) = \frac{Gain(A)}{InfoAfter(A_{AFTER})}$

Gini

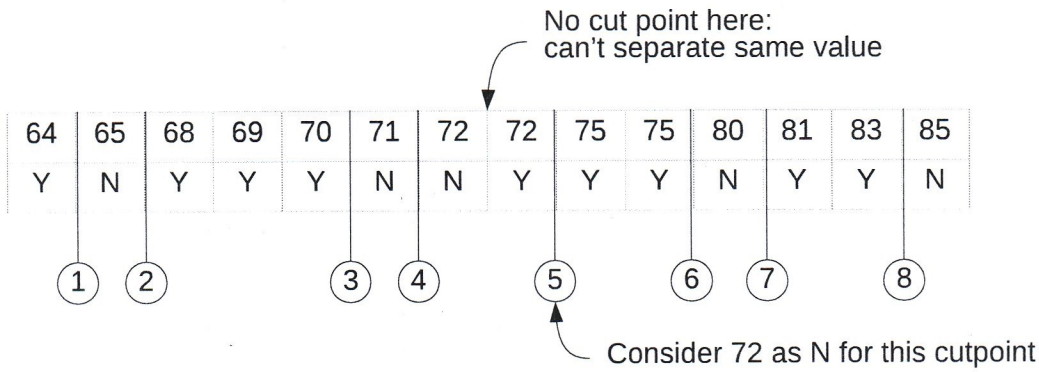
- $\Delta Gini(A) = \underbrace{Info(A)}_{\text{info BEFORE split}} - \underbrace{InfoAfter(A_{AFTER})}_{\text{info AFTER split}}$

not all leaves have to be pure, when to stop:

- all samples for a given node belong to the same class
- no samples left
- no remaining attributes for partitioning
- no gain in splitting

Numerical attributes

1. Sort attribute by value (keep class labels)
   - duplicates are considered once
2. check all the cutpoints (in one pass)



3. compute the information gain for all cutpoints and choose the best one

   example for cutpoint 3: $\begin{cases} A \geq 70.5 & Y=4, N=1 \\ A < 70.5 & Y=2, N=2 \end{cases} \rightarrow$

   $InfoAfter([4/1],[2/2]) = \frac{5}{14} Info([4,1]) + \frac{4}{14} Info([2,2]) = 0.544$

generally always 2 splits.

Overfitting Trees

Overfitting: usually when we have too many branches which represent noise and outliers

2 ways to avoid overfitting:

- **Prepruning**: halts tree construction early when node goodness measure is below a threshold (usually chi-squared)

- **Postpruning**: remove progressively branches from the tree and test it against a different dataset, keep the best.
    - Subtree replacement: bottom-up build the tree by considering all the substituting the nodes

[Estimating Error Rates]

Regression with Decision Trees

Decision trees can also be used for regression. 2 approaches:

- **Regression trees**: prediction is the average of the numerical target variable in the subspace
- **Model trees**: leaves contain a linear model to predict the target value of the corresponding subspace

Impurity measure: $SDR = \sigma(D) - \sum_i \frac{|D_i|}{|D|} \cdot \sigma(D_i)$

where: D original data, D_i data partitions, σ standard deviation of the target attribute in the set.

Decision Stumps

Decision stumps: one level decision trees, which are the simplest decision tree possible and building blocks for boosting methods.

- Categorical attributes: one branch for each attribute value or one branch for one value and one for the rest of the values
- Numerical attributes: 2 way split or multiple splits (rare)

Classification Rules

Classification rule: simple IF-THEN rules

- IF: states a condition over data (propositional with attribute-value comparison) or FO
- THEN: includes a class label

Metrics

$$Coverage(\underbrace{R}_{rule}) = \frac{\overbrace{n_{covers}}^{\text{number of examples covered by the rule}}}{|D|} \qquad Accuracy(\underbrace{R}_{rule}) = \frac{\overbrace{n_{correct}}^{\text{number of examples correctly classified by the rule}}}{n_{covers}}$$

Conflict resolution

If more rules are triggered on the same example we have a conflict → conflict resolution

- **Size ordering**: assign highest priority to the triggering rules that has the most attribute test
- **Class-based**: decrease order of prevalence or misclassification cost per class

- **Rule-based ordering**: rules organized into one long priority list

Rule Learning

- **Direct methods**: learn rules from data
- **Indirect methods**: learn rules from a model (decision tree → convert to rules, neural network → extract rules)

Direct Methods

1R classifier

Learns a simple rule involving one attribute and test all the values for this attribute.

1. Choose attribute
2. branch for each value
3. assign most frequent class per branch
4. measure attribute performance with error rate as the proportion of instances that don't belong to the majority class of that branch

missing are treated as separate value

1R discretization

1. Assign cutpoints as for decision trees
2. to limit overfitting: define threshold for majority class instances per interval
3. join intervals to satisfy threshold
4. join intervals with same majority class

Sequential Covering

1. **Learn best rule** given the examples:
   - accuracy
   - information gain
2. **remove** positive examples covered by this rule (otherwise would always find the same rule)
3. **repeat** until all examples are covered

Rule pruning

Stratification is good.

- **Reduced-error pruning**: build full rule set and then prune
- **Incremental reduced-error pruning**: after each rule is built prune rules

Indirect methods

	Rules	Trees

Model readability	x	
Replication problem		Subtrees replicated
Model type	Local	Whole domain

Naive Bayes, KNN, others

Naive Bayes

Predict class of an input given the prior and the posterior probabilities from the training set.

$$P(y|\vec{x}) = \frac{P(\vec{x}|y) \cdot P(y)}{P(\vec{x})} \rightarrow$$

$$class = \arg\max_y P(y|\vec{x}) = \arg\max_y \frac{p(x_1|y) \cdot \ldots \cdot p(x_n|y) \cdot p(y)}{P(\vec{x})} = \arg\max_y p(x_1|y) \cdot \ldots \cdot p(x_n|y) \cdot p(y)$$

- $P(y|\vec{x}) = \frac{P(\vec{x} \wedge y)}{P(\vec{x})}$ a posteriori probability (prob that an event after evidence is seen)
- $P(y)$, $P(\vec{x})$ a priori probability (prob of event before evidence is seen)

Assumptions:

- attributes **all equally important**
- attributes are **statistically independent**: knowing value of one attribute says nothing about the value of another (naive but it often works)

zero-frequency problem: probability can never be zero

- add a constant to the count (usually 1)
- different weigths to each count

missing value: omit attribute from calculation in testing and training

numeric attributes: assume it has a normal prob distribution $f(x) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

KNN

Predict the class of a given input based on the class frequency of the k most similar data examples.

The model is the data itself.

Components

- training dataset
- similarity functions (need to invert the distance)
  - $euclidean_{distance}(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

- $manhattan_{distance}(\vec{x}, \vec{y}) = \sum_{i=1}^{n} |x_i - y_i|$
- $cosine_{distance}(\vec{x}, \vec{y}) = \arccos \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$

- k: number of nearest neighbor to consider for the prediction
  - k small → noise
  - k big → include dissimilar examples

Algorithm

1. compute distance to other training records
2. identify k nearest neighbors
   - linear scan (inefficient)
   - KD-Trees: split space hierarchically, need to find good split point (mean or median) and direction (greatest variance).
3. determine class based on the frequency of the class of the nearest neighbors
   - majority vote
   - weights the vote based on the euclidean distance

Issues

- **Normalization**: needed since must compute distance between different features
  - $x'_i = \frac{x_i - min_i x_i}{max_i x_i - min_i x_i}$, $x'_i = \frac{x_i - \mu}{\sigma}$
- **nominal attribute**: distance is either 0 or 1
- **missing values**: maximally distant

Bayesian Belief Networks

Similar to naive bayes, but let specify also probabilistic relationships between features.

Let capture causality between features. Robust to overfitting.

Components

- Acyclic graph which encodes relationships among variables
- Probability table which associates each node to its parents node
  - X does not have parents → $P(X)$
  - X has 1 parent → $P(X|Y)$

- X has >1 parents → $P(X|Y_1,\ldots,Y_k)$

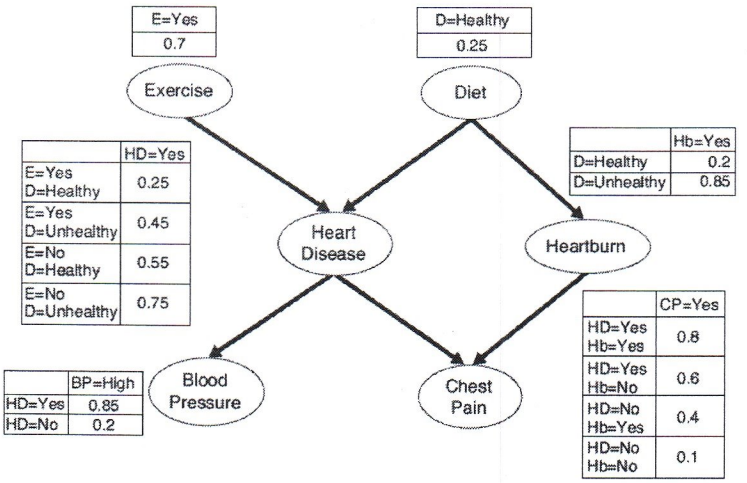
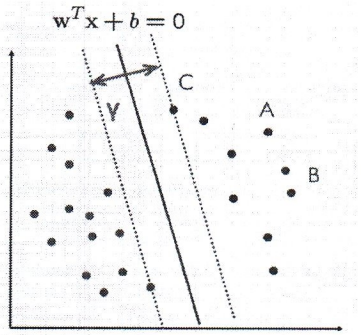


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

SVM

Search for the hyperplane that maximizes the margin or the largest gamma st:

$\forall i, y_i(wx_i+b)\geq\gamma$



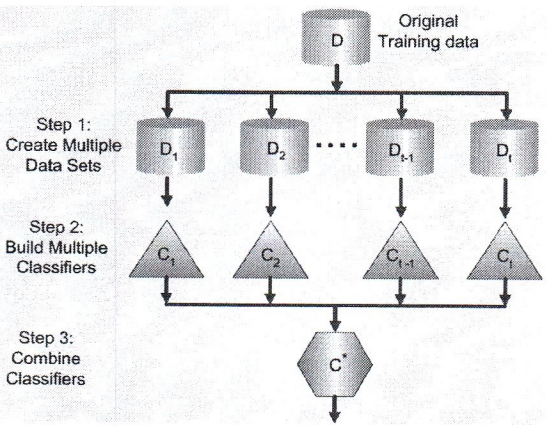
Classification Ensembles

Ensemble methods **aggregate predictions from many classifiers.**

- Improves accuracy
- output hard to analyze

Probability that ensemble makes a wrong prediction: $\sum_{i=\frac{M}{2}+1}^{M}\binom{M}{i}\cdot\epsilon^i\cdot(1-\epsilon)^{M-i}$

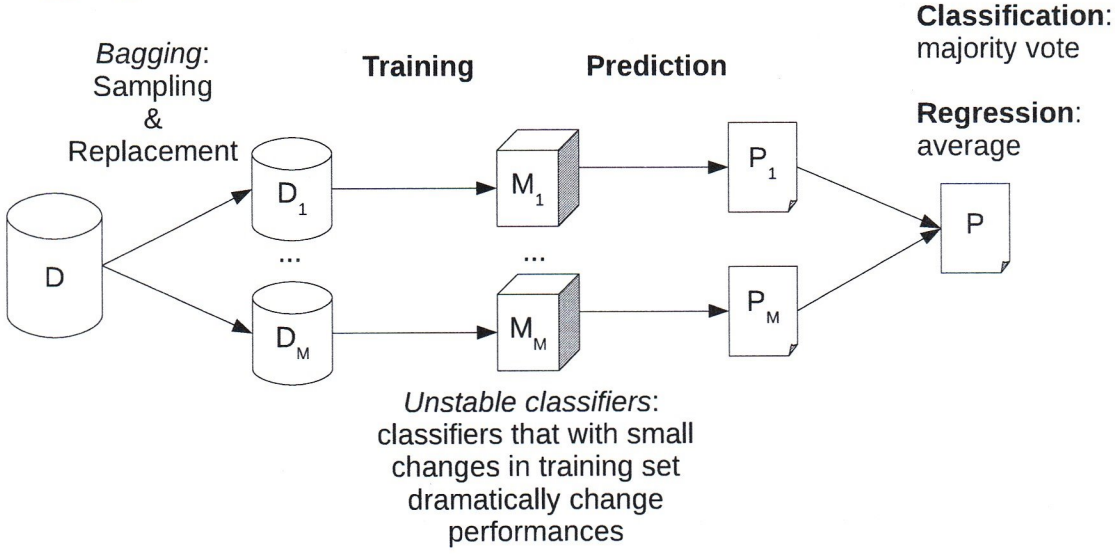
- M is the number of models
- ϵ error rate



- $\binom{M}{i} = \frac{M!}{i! \cdot (M-i)!}$

(in classification the majority of the classifiers must make the wrong prediction to fail)

Bagging

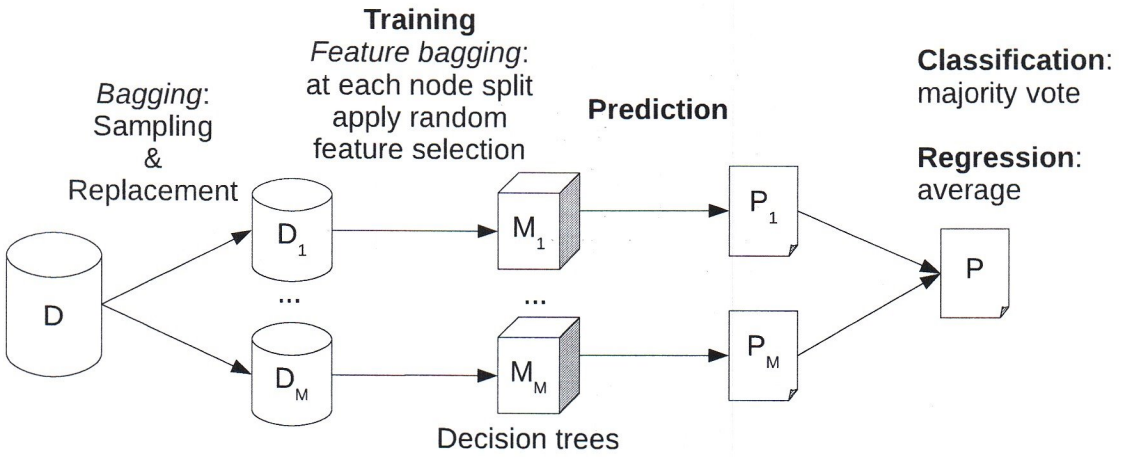


Pros:

- reduces variance
- the more classifiers the better
- good if noisy database

Randomize algorithms: can also randomize algorithm parameters (other than data)

Random forests



Pros:

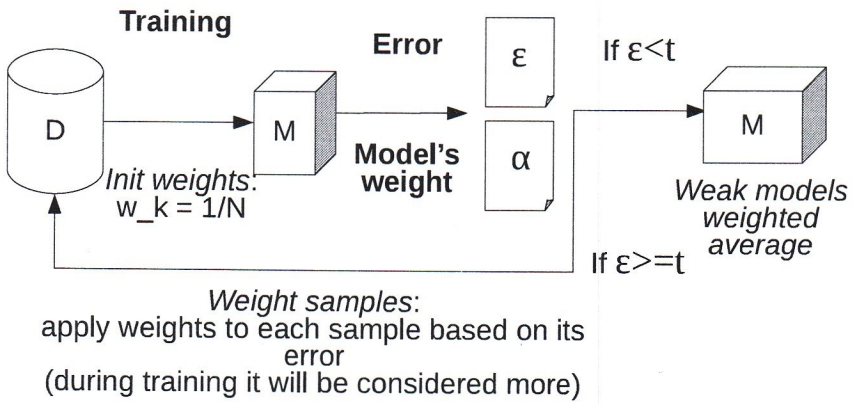
- estimates variable importance
- no overfitting if large number of trees
- high accuracy

OOB sampling: since not all datapoints are sampled, the remaining ones can be used for validation (like K-fold cross validation). Training can be interrupted once OOB error stabilizes.

Boosting

Iteratively build an ensemble of weak learners (*score*>50 %), in an attempt to generate a strong overall model.

Adaboost



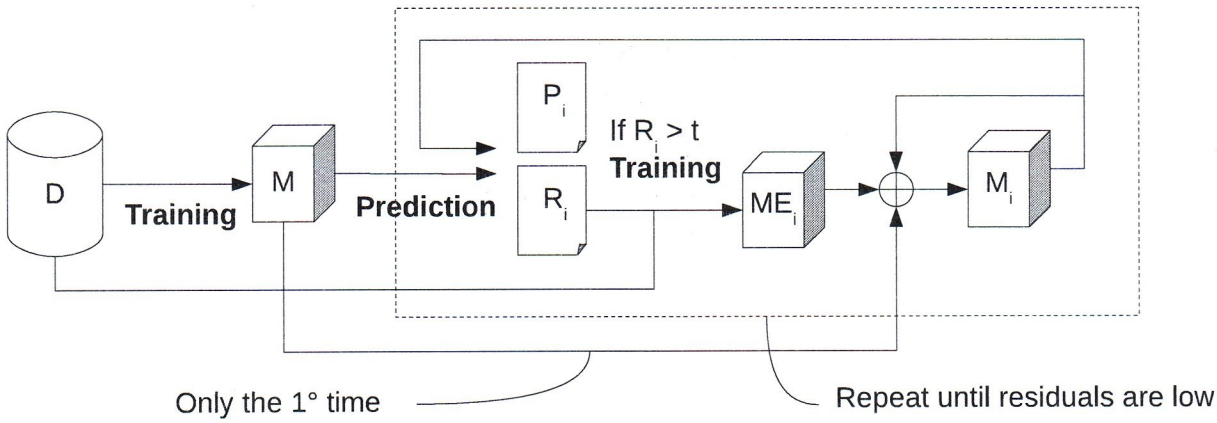
Pros:

- Weak models can be stump or shallow trees

Variants:

- *boosting by sampling:* weights are used to sample data for training
- *boosting by weighting:* weights are used by the learning algorithm

Gradient boosting



- sum of predictions is increasingly accurate
- predictive function is increasingly complex

eXtreme Gradient Boosting

Efficient and scalable implementation of gradient boosting applied to classification and regression trees.

Pros:

- handles sparse data
- fast
- avoid overfitting

Learning ensembles and Stacking

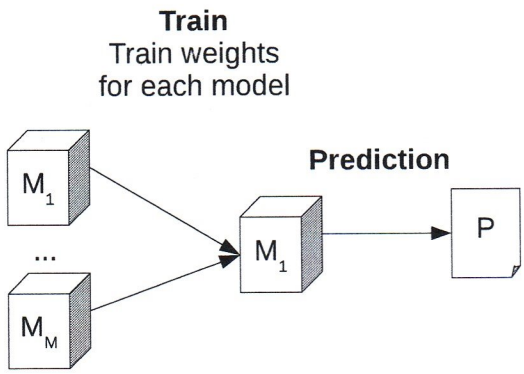
Apply optimization over multiple models:

$$D=\{T_1(X),\ldots,T_M(X)\}$$

$$F(X)=\sum_{m\in D}\alpha_m\cdot T_m(X)$$

Learning ensembles: applied to weak random forests and boosting

Stacking: generalization of learning ensemble



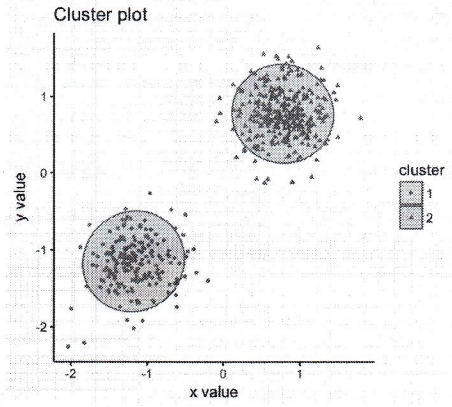
Clustering

Cluster: collection of data objects which have

- high intra-cluster similarity
- low inter-cluster similarity

Requirements for clustering algorithms:

- scalability
- handle dynamic data
- arbitrary data shapes
- deal with noise and outliers
- high dimensionality (**Curse of dimensionality**: with high dimensions all pairs of points equally distant with each other)
- custom constraints



Cluster analysis: given dataset find

- structure
- similarities
- group similar objects
- all hidden patterns

Cluster evaluation:

- intra/inter cluster similarity metrics
- manual inspection
- benchmark on existing labels

Clustering methods

- **Hierarchical vs point assignment**: point assignments maintains a set of clusters which can be split or combined as the algorithm proceeds
- **Numeric vs symbolic data**: whether the clustering supports symbolic data or plain numbers
- **Deterministic vs probabilistic**: deterministic methods each time they are ran, given the same parameters, produce the same clusters
- **Exclusive vs overlapping**: exclusive clustering assigns to each data point only one cluster
- **Hierarchical vs flat**: hierarchical clustering creates a hierarchy of clusters thus creating relationships between clusters whereas flat clustering has none

Distance Measures

Distance measure: $d(x,y)$ maps 2 points in space to a real number which satisfies:

- $d(x,y) \geq 0$
- $d(x,y) = 0 \Leftrightarrow x = y$
- $d(x,y) = d(y,x)$
- $d(x,y) \leq d(x,z) + d(z,y)$

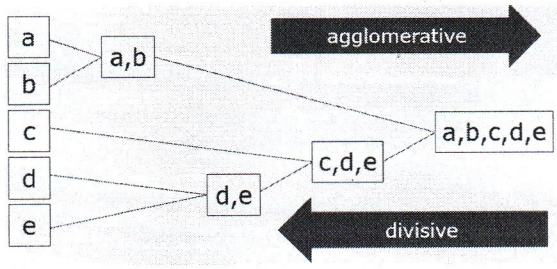
Type of distances measures:

- Euclidean, L-Norm, Manhattan, Jaccard, Cosine, Hamming
- **Edit distance**: $d(x,y)=|x|+|y|-2|LCS|$ longest common subsequence (LCS) of x and y (string similarity)

Hierarchical Clustering

Cluster analysis which builds a hierarchy of clusters. 2 types:

- **Agglomerative (bottom-up)**: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy
  1. compute proximity matrix between all points
  2. merge 2 closest clusters
  3. goto 1 until only one cluster remains
- **Divisive (top-down)**: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy



- no need to specify number of clusters in advance
- may correspond to useful taxonomies
- number of clusters desired by cutting the dendrogram

Update proximity matrix

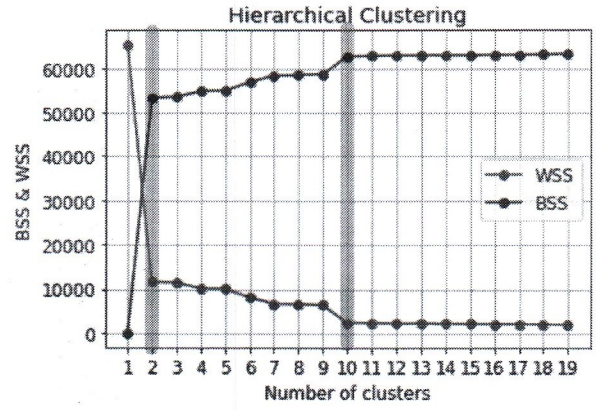
- $d(C_i,C_j)=\mathbf{min}(t_{i,p},t_{j,q})$
- $d(C_i,C_j)=\mathbf{max}(t_{i,p},t_{j,q})$
- $d(C_i,C_j)=\mathbf{avg}(d(t_{i,p},t_{j,q}))$
- $d(C_i,C_j)=d(\mu_i,\mu_j)$ **centroid**

Number of Clusters

Knee/Elbow analysis: given the WSS and BSS plots, determine the appropriate number of clusters

$$\underbrace{WSS(C)}_{\text{Within-cluster Sum of Squares}} = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

$$\underbrace{BSS(C)}_{\text{Between-cluster Sum of Squares}} = \sum_{i=1}^{k} |C_i| \cdot \|\mu - \mu_i\|^2$$



Euclidean vs non-euclidean spaces

- **euclidean space**: cluster identified by its centroid
- **non-euclidean space**: need to define a distance (jaccard, cosine, edit, …). introduce the "clustroid", a data point which represent the cluster and is computed as the centroid with the new defined distance metric.

Representative-based Clustering

Centroid: usual representation of a cluster given by a point that summarizes it $\mu_i = \frac{1}{n_i} \cdot \sum_{x_j \in C_i} x_j$

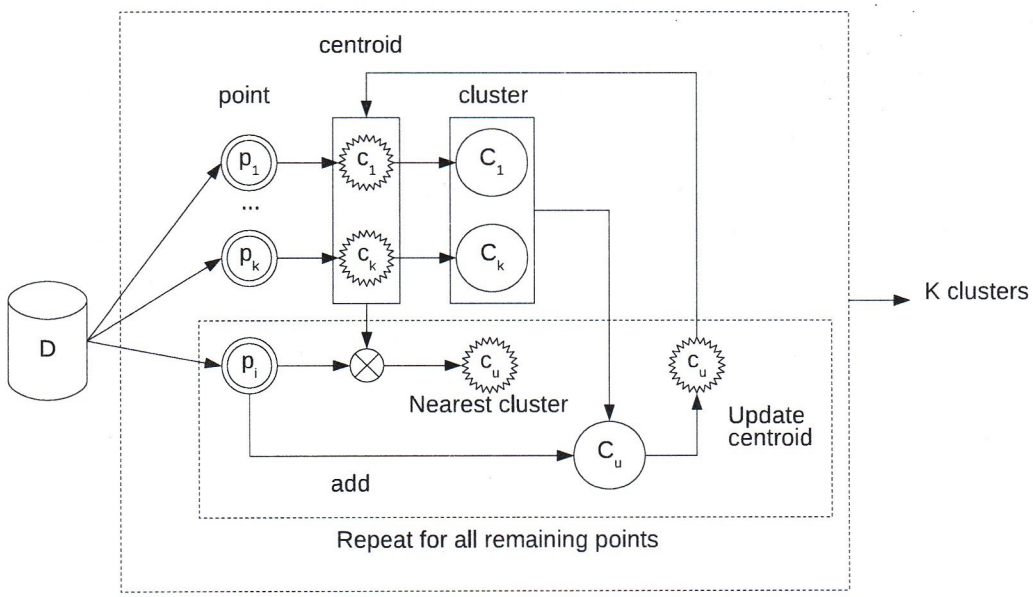
Cluster partition scoring

Clustering goal: select best partition according to scoring function

$$SSE(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2 \qquad C^* = \arg\min_{C} SSE(C)$$

K-Means

Greedy iterative approach that min SSE. Converges to a local optimal clustering (not global).



Strategies to choose centroids:

- pick k random points
- pick random point and add k-1 points most far away from each other
- cluster sample of data getting k clusters. Pick one point from each cluster (the closest to the centroid)

Cons:

- **init centroid problem**: difficult to get correct centroids at beginning $P=\frac{K!}{K^K}$
  - multiple runs
  - hierarchical clustering to choose centroids
  - select more than k initial centroids and select among those
  - bisecting k-means
- **Problems**: cluster with different
  - sizes
  - densities
  - non-globular shapes
  - outliers

Pros:

- efficient
- terminates with local optimum
- simple
- k-mode variant handles categorical data with specific dissimilarity measures

Pre-processing:

- normalize data

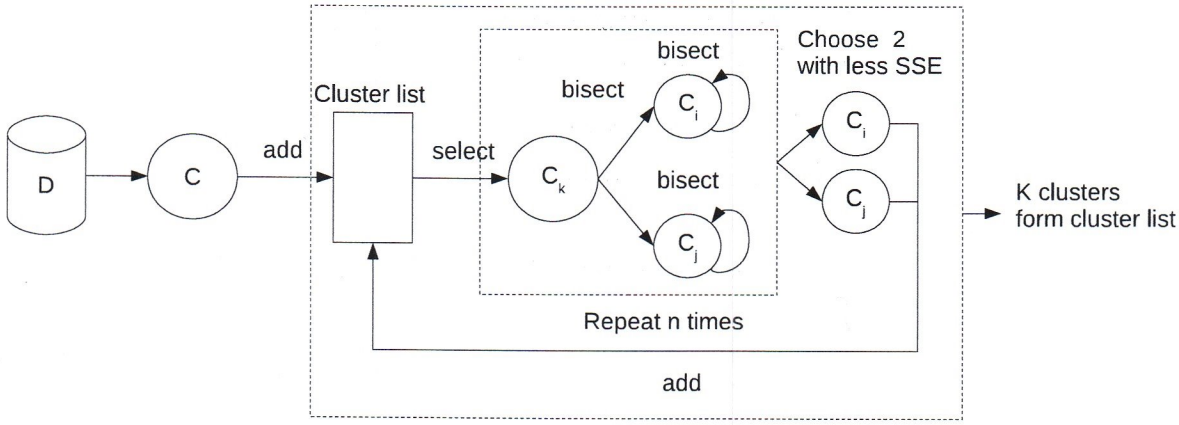
Post-processing:

- remove small clusters

- remove outliers
- split "loose" clusters with high SSE
- merge "close" and with low SSE clusters

Bisecting K-Means

K-means variant **less sensible to init centroids problem**.



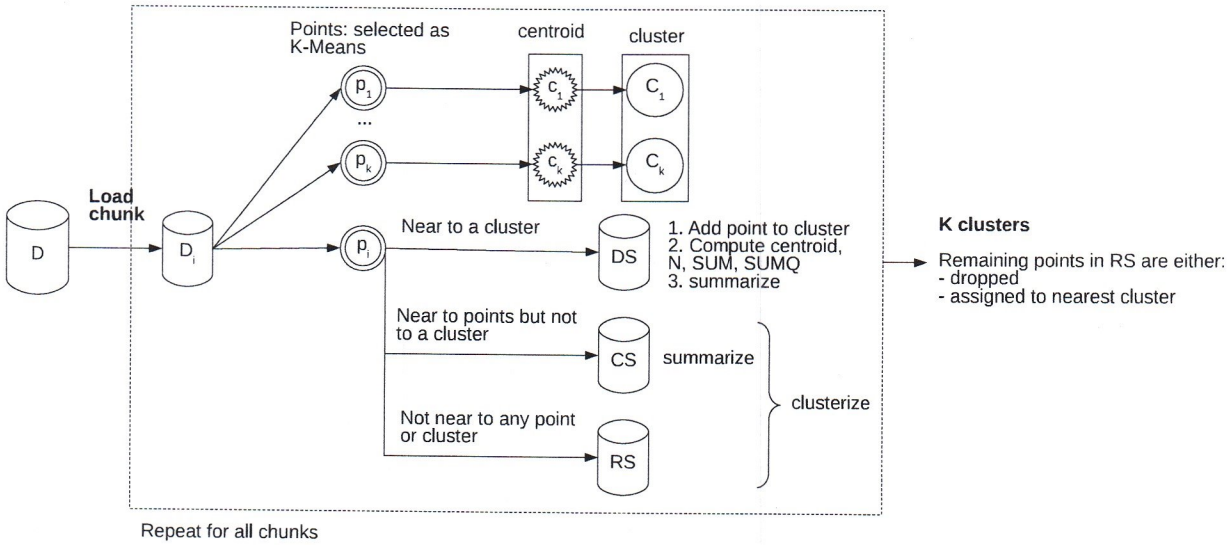
Repeat until k clusters are in the cluster list

BFR

K-means variant that handles very large data quantity. **Summarizes clusters**.

Assumes clusters are normally distributed and axis-aligned eclipses.

Points are loaded in chunks from disk one at a time.



Repeat for all chunks

3 sets are available for BFR:

- **Discard Set (DS)**: points close to centroid → to the centroid's cluster
  - *vector SUM*: each component is the sum of the coords of the points in dimension i
  - *vector SUMQ*: each component is the sum of the squares of coords in dimension i

- **Compression Set (CS)**: points close together but not close to any existing centroid → summarize
- **Retained Set (RS)**: isolated points → wait to be assigned to CS

Various criterion to decide whether a point is near a cluster:

- **near to cluster's centroid**
- **probability that point as part of the cluster** would be found in that cluster (use normal point distribution)
- **Mahalanobis distance**: distance from cluster normalized by the standard deviation (compute distance with SUM and SUMSQ vectors)

Expectation Maximization

Consider point's membership to a cluster as a probability → each cluster charcterized by multivariate normal distribution:

- mean vector
- covariance matrix
- parameter vector (chosen with MLE)

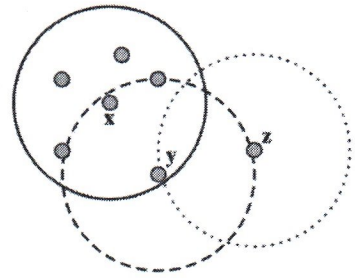
Density Based Clustering

Clustering based on density: at least "minpts" within a specified radius ε.

Core point (x): >="minpts" within a specified radius ε (**density**)

Border point (y): <"minpts" within a specified radius ε, but inside a neighborhood

Noise point (z): not part of neighborhood



Pros:

- handles noise
- arbitrary cluster shapes
- one scan
- density parameters conditions for termination

Density properties (given x, y objects):

- *Directly density reachable*: if x and y belong to same neighborhood and one is core.
- *Density reachable*: if x and y can reach each other with a chain of intersecting neighborhoods.
- *Density connected*: if x and y belong to same neighborhood of another point o.
- *Density-based cluster*: maximal set of density connected points.

Data Preparation

Data extraction, cleaning and transformation are the majority of the work of building a data warehouse.

Data cleaning

- **missing values**: information is missing
- **duplicate data**: information is duplicated (feature or sample level), usually with merging data from different sources
- **outliers**: samples with characteristics considerably different from other sample data

Sampling

Data selection to reduce computation time and needs.

Sampling is good if retains the same properties of the original dataset wrt the problem we are trying to solve.

- **Simple Random Sampling**: equal prob to pick a sample wrt other samples
- **Sampling Without replacement**: picked sample cannot be picked again (removed from sample pool)
- **Sampling With replacement**: picked sample ca be picked again
- **Stratifies sampling**: samples picked from data split in homogenous partitions

Aggregation

Combine 2/more attribute in a single one.

- data reduction
- change of scale
- data with less variability

Feature creation

Create new attributes that capture important information.

- Feature extraction
- mapping data to a new space
- feature construction (combine features)

Discretization

Divide the range of continuous attributes into intervals to reduce data size.

- **Supervised**: discretize attributes using class information (min information loss)
- **Unsupervised**: discretize solely on values

Feature selection

Pros:

- avoid curse dimensionality
- reduce time/memory computation
- reduce noise

Types:

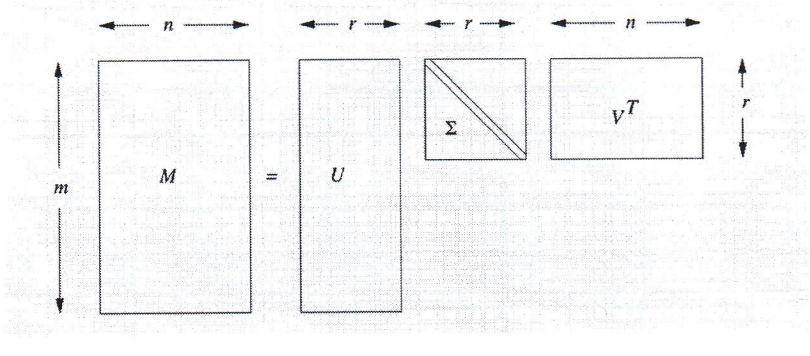
- **Bruteforce**: try all possible feature subsets as input to data mining algorithm
- **Embedded**: data mining algorithm itself makes feature selection
- **Filter approach**: select features independently from data mining technique (correlation matrix, PCA, SVD)
- **Wrapper approach**: use blackbox algorithms to perform feature selection (genetic alg that finds best parameters).
  - *Backward Feature elimination*: start train on n input features, retrain several times removing each time a feature. Stop when error over a fixed threshold. Feature that determines less error variation are removed.
  - *Forward Feature construction*: reverse of backward feature elimination. Start from 1 feature and add1 feature at each iteration.
  - *Recursive Feature elimination*: iteratively train model and keep aside best or worst features. Next iteration train with left out features. At the end rank features by usefulness.

Singular Value Decomposition (SVD)

SVD is a generalization of PCA.

Idea: always possible to decompose $M = U \cdot \Sigma \cdot V^T$

- M original data
- U, V orthonormal columns ($X^T \cdot X = 1$). Represents n examples using r attributes
- $\Sigma_{i,j}$ singular, decreasing order sorted, positive. Represents attribute's strength.



Reduction operates on removing values from $\Sigma_{i,j}$ while trying to retain most energy possible.

$$Energy = \sum \Sigma_{i,j}^2$$

in the example if we remove the value 1.3, we retain the 99% of the energy.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = M'$$

$$\begin{bmatrix} .13 & .02 & -.01 \\ .41 & .07 & -.03 \\ .55 & .09 & -.04 \\ .68 & .11 & -.05 \\ .15 & -.59 & .65 \\ .07 & -.73 & -.67 \\ .07 & -.29 & .32 \end{bmatrix} \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \\ .40 & -.80 & .40 & .09 & .09 \end{bmatrix}$$

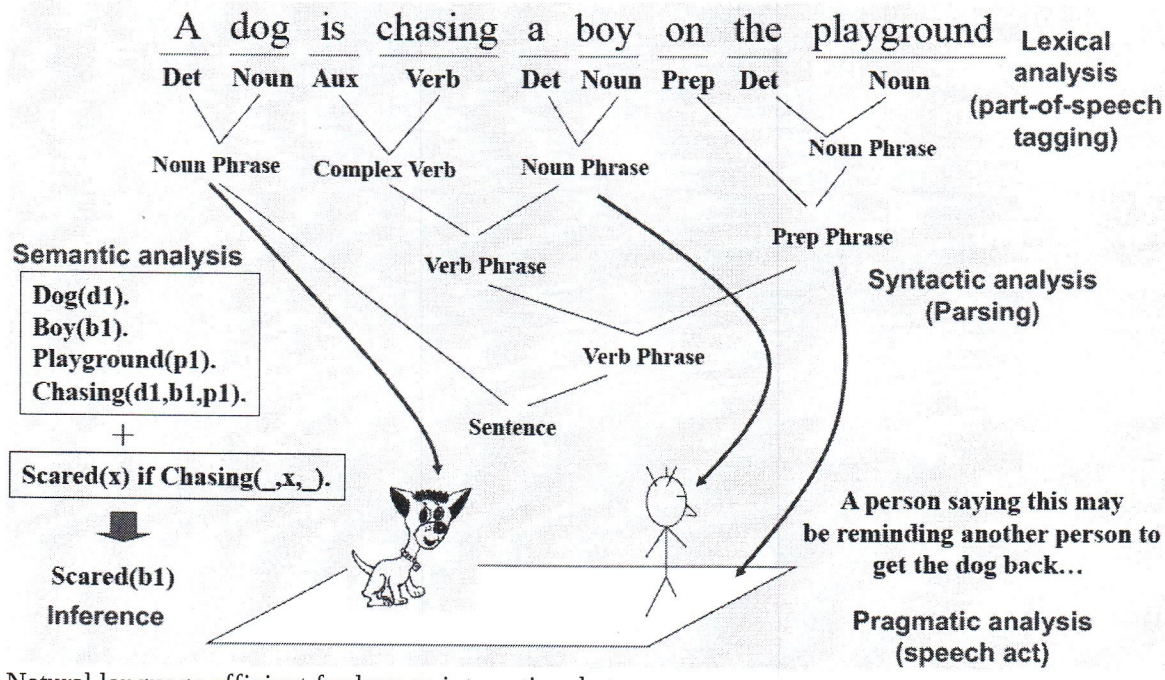
$U \qquad \Sigma \qquad V^T$

Text Mining

Text mining: extract information from different unstructured text documents.

- minimize human effort
- data → high-quality info/actionable data
- knowledge for decision making

Natural Language Processing



Natural language efficient for human interaction, but:

- **Ambiguity**: assume other party knows how to resolve ambiguities
- **Common Sense Reasoning**: assume other party has common sense knowledge

NLP expensive, so:

1. locate promising text fragments with fast methods
2. apply NLP to found text

Information Retrieval

Information retrieval: locate relevant documents wrt user input

Problems:

- unstructured docs
- approximate search
- relevance

2 ways to access text:

- **Pull mode (search engines)**: user takes initiative
- **Push mode (recommender systems)**: system takes initiative

Text retrieval methods:

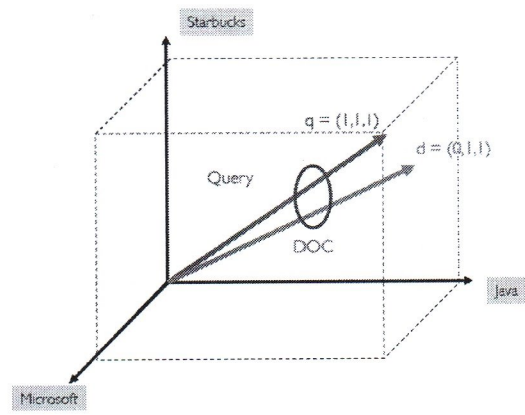
- **Document selection (keyword based)**: queries defines a set of requisites for the documents that has to be returned
- **Document ranking (similarity based)**: documents are relevance ranked wrt user query (Vector Space Model) with a function $f(\underbrace{\vec{q}}_{\text{query keywords}}, \underbrace{\vec{d}}_{\text{document keywords}}) = \mathbb{R}$.
  - $f(\vec{q},\vec{d}) = \textbf{\textit{similarity}}(\vec{q},\vec{d})$
  - $f(\vec{q},\vec{d}) = \boldsymbol{P}(\vec{d} \rightarrow \vec{q})$
  - $f(\vec{q},\vec{d}) = \boldsymbol{P}(R=1|\vec{q},\vec{d})$
  - $f(\vec{q},\vec{d}) = \textbf{\textit{axiom}}$

[precision, recall, f-score]

Vector Space Model

Document and query represented as vectors in high dimensional space corresponding to the vocabulary of the document.

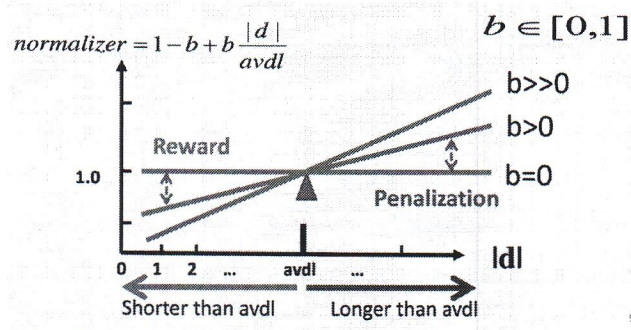
$$\textbf{\textit{similarity}}(\vec{q},\vec{d}) = \vec{q}\cdot\vec{d} = \sum_{i=1}^{N} q_i \cdot d_i$$



- **Basic VSM**: $\begin{cases} q_i & 1 \text{ if keyword } w_i \text{ is present in query, 0 otherwise} \\ d_i & 1 \text{ if keyword } w_i \text{ is present in document, 0 otherwise} \end{cases}$

- **Term Frequency Weighting (TFW):** $\begin{cases} q_i = c(w_i, q) \\ d_i = c(w_i, d) \end{cases}$
- **TFW + Inverse Document Frequency (IDF):** $\begin{cases} q_i = c(w_i, q) \\ d_i = c(w_i, d) \cdot IDF(w_i) \end{cases}$
- **BM25:** $\begin{cases} q_i = c(w_i, q) \\ d_i = \dfrac{(k+1) \cdot c(w_i, d)}{k + c(w_i, d)} \cdot IDF(w_i) \end{cases}$ with $IDF(w) = \log\left(\dfrac{\underbrace{M}_{\#\text{ docs}} + 1}{\underbrace{k}_{\#\text{ docs containing } w_i}} \right)$

Document length normalization: long documents must have more words, but also contain more content so they must be penalized, but not too much.



From text to numerical vectors

- **Keyword selection**: text is tokenized
  - stop words (useless for information: "the", "a", ...) elimination
  - word stemming ("computer", "computing" → "comput")
- **Dimensionality reduction**: having high vector keywords dimensionality is expensive, so apply dimensionality reduction techniques (LSI, LPI, PLSI)
  - LSI: apply SVD $\underbrace{X}_{\text{set of documents}} = U \cdot \Sigma \cdot V^T$ approximate $X \rightarrow \underbrace{X_k}_{\text{first k vectors of U}}$

Text Classification

Text classification: automatically label text documents wrt topic, style, purpose by learning from manually labeled documents.

Similarity based classifiers: based on information retrieval + KNN:

1. given a new document to classify, get k most similar documents
2. document is classified based on the class distribution among k documents (majority/weighted vote)

Naive Bayes

Given: $C = \{C_1, \ldots, C_n\}$ document categories (classes), D document (input to classify)

1. $P(D|C)=\prod_{i=1}^{n} P(\underbrace{w_{ji}}_{\text{word i in document j}}|C)$ by assuming: keywords distributions are inter-independent and order-independent

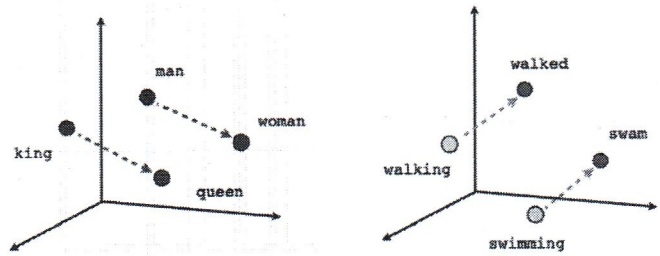
$$P(w_k|C)=\frac{\underbrace{N_{c,k}}_{\text{count of } w_k \text{ word in documents of class C}}+1}{\underbrace{N_c}_{\text{count of words in documents of class C}}\cdot|Vocabulary|} \qquad P(C)=\frac{|\text{docs of class C}|}{|\text{number of docs}|}$$

2. $C^*=\arg\max_C P(C|D)=\arg\max_C P(D|C)\cdot P(C)$

Word Embedding

Representation for text where words with similar meanings have a similar representation.

Each word is a real-valued vector in the space of words.



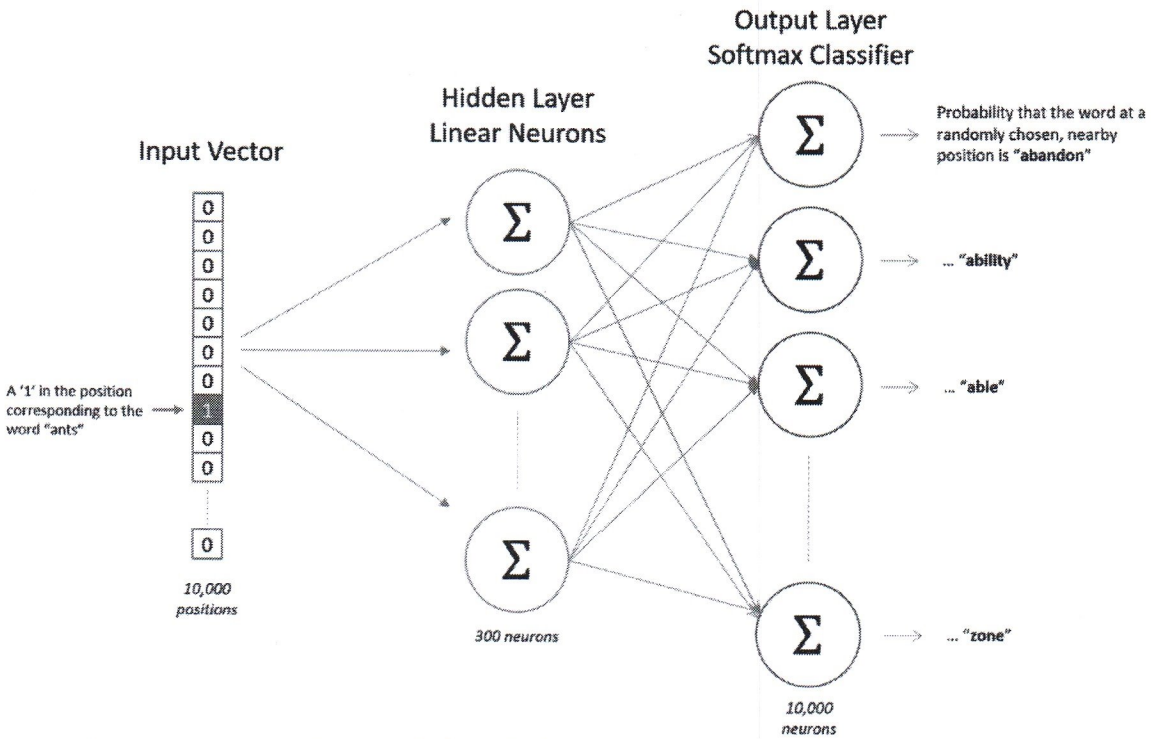
Male-Female Verbe Tense

Word representations:

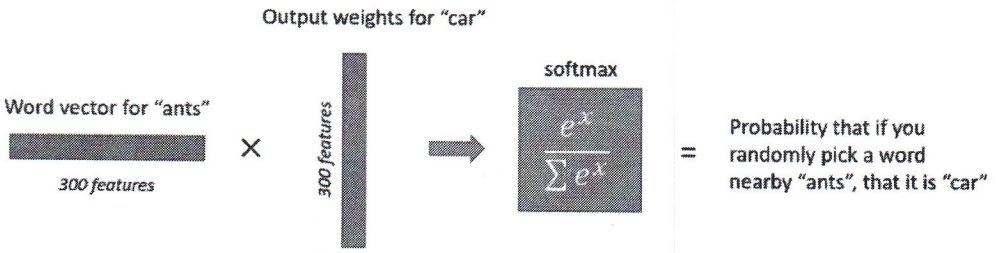
	Word Embedding	*Bag of Words (traditional)*
Word representation	Fixed sized vector	1 bit in one-hot-encoded vector
Context	Used	Not used

Word2Vec

Skip Gram Neural Network architecture.



- **Input vector**: one-hot encoded vocabulary (input keyword has the corresponding cell set at 1)
- **hidden layer**: represents the single word with a vector of 300 features (word embedding)
- **output vector**: size of the vocabulary which outputs the probability of having a determined word near the keyword given as input.



Text Clustering

1. tokenizing, stemming
2. transform corpus into vector space TF-IDF
3. compute cosine distance between each document (similarity measure)
4. apply clustering over similarity measure