Data Mining - Appunti

Francesco Lorenzoni

Febrero 2025

Contents

Part I

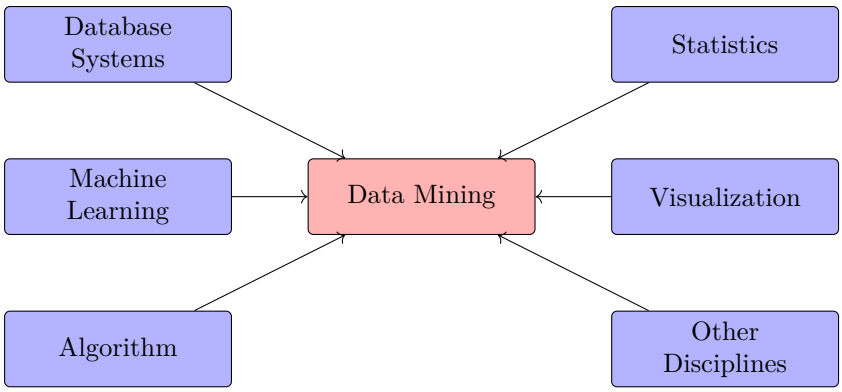
Introduction to Data Mining

Chapter 1

Introduction

Definition 1.1 (Data Mining) *is the use of efficient techniques for the analysis of very large collections of data and the extraction of useful and possibly unexpected patterns in data (hidden knowledge). The goal is to extract (human-readable) knowledge and insight from raw data.*

- ◇ Knowledge implies we are often not just trying to solve a task
- ◇ Insight implies that we should infer non-obvious knowledge
- ◇ Human-readable implies that knowledge should be (when possible) understood by humans: focus on interpretability!
- ◇ Raw data implies we'll need to clean it



1.1 Definitions

1.1.1 Knowledge Discovery Loop

Large collections tend to be heterogeneous in source, domain, language and refinement. The first step is to store the data, which however does not assess its heterogeneity. Data cleaning and integration tackle this problem, so that we get integrated sources, homogenous language, and data cleared of noise and outliers.

To look for insight on the data we have to answer questions on the data as a stakeholder. We may see patterns and ask ourselves their nature. Pattern extraction and validation lead to possible insight. Insight may lead to noticing that some data missing may be useful, and we may want to collect it, going back at previous steps.

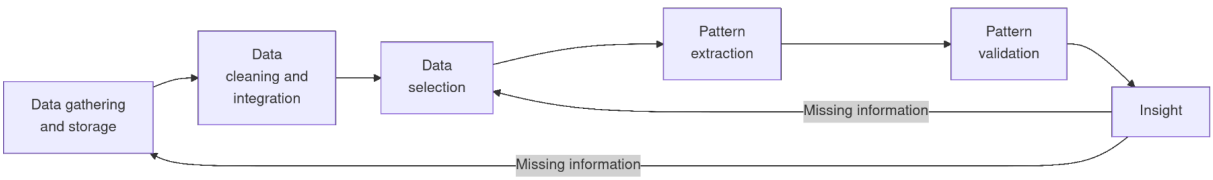


Fig. 1.1: Knowledge Discovery Loop
This essentially summarizes the KDD process.

1.1.2 KDD Process

The KDD process consists of the following steps:

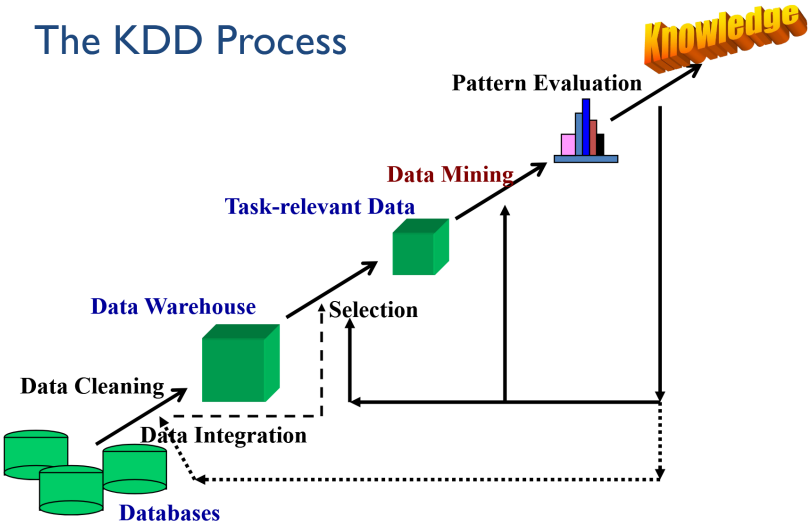
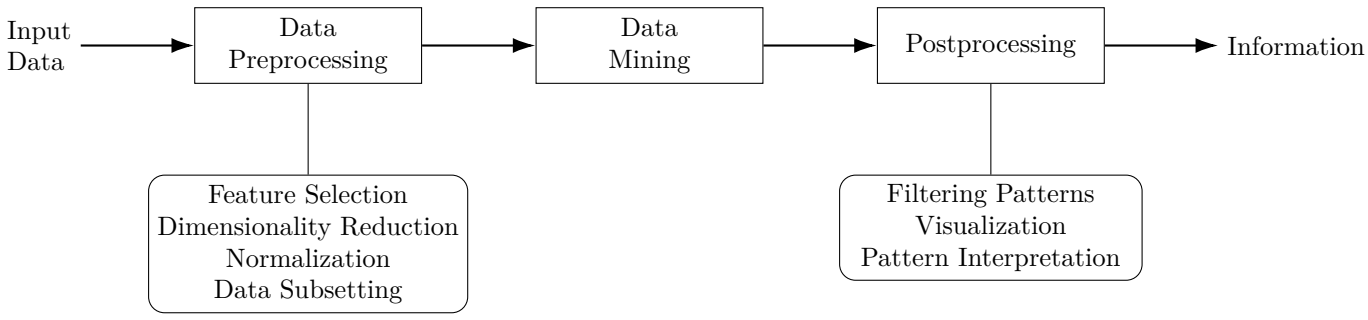


Fig. 1.2: KDD Process

1. **Data Cleaning:** Remove noise and inconsistent data.
2. **Data Integration:** Combine multiple data sources.
   Involves the process of data understanding, data cleaning, merging data coming from multiple sources and transfoming them to load them into a **Data Warehouse**.
   Data Warehouse is a database targeted to answer specific business questions
3. **Data Selection:** Select relevant data for analysis.
4. **Data Transformation:** Transform data into suitable formats for mining (summary, aggregation, etc.).
5. **Data Mining:** Apply algorithms to extract patterns.
   - ◇ *Prediction Methods*
     Use some variables to predict unknown or future values of other variables.
   - ◇ *Description Methods*
     Find human-interpretable patterns that describe the data.
6. **Pattern Evaluation:** Identify truly interesting patterns.
7. **Knowledge Presentation:** Present the mined knowledge in an understandable way.

1.1.3 Data Mining Process



Definition 1.2 (Primary Data) *Original data that has been collected for a specific purpose.*
Primary data is not altered by humans

Definition 1.3 (Secondary Data) *Data that has been already collected and made available for other purposes.*
Secondary data may be obtained from many sources

Definition 1.4 (Association rule discovery) *Given a set of records each of which contain some number of items from a given collection.*
Produce dependency rules which will predict occurrence of an item based on occurrences of other items.

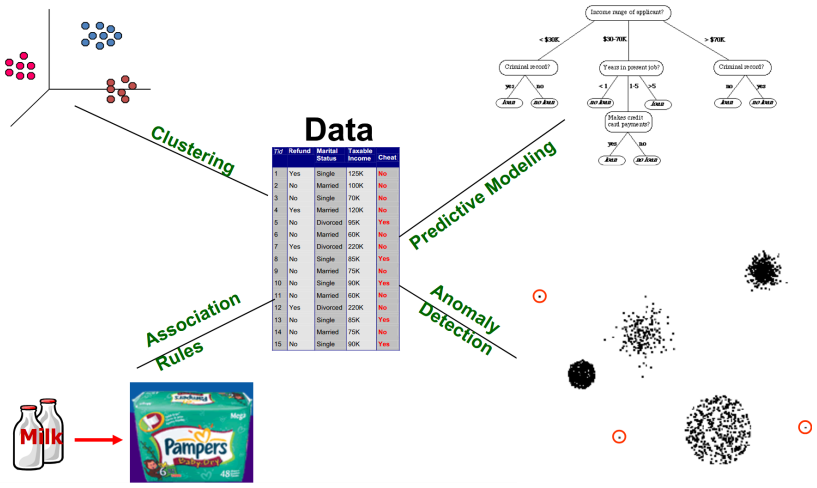


Fig. 1.3: Data Mining methods

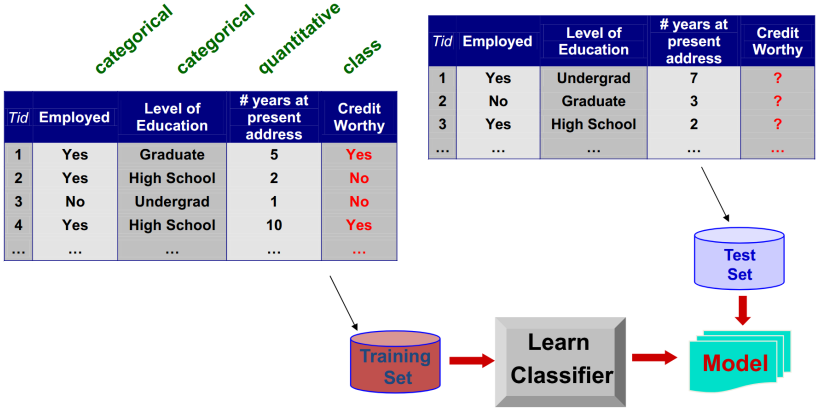


Fig. 1.4: Classification Process

Association Use Cases

- **Market-basket analysis**
  Rules are used for sales promotion, shelf management, and inventory management
- **Telecommunication alarm diagnosis**
  Rules are used to find combination of alarms that occur together frequently in the same time period
- **Medical Informatics**
  Rules are used to find combination of patient symptoms and test results associated with certain diseases

1.2 Data Understanding

Definition 1.5 (Data) *Data is a collection of data objects and their attributes.*
An attribute is a property or characteristic of an object. A collection of attributes describe an object (record).

If data objects have the same fixed set of numeric attributes, then the data objects can be thought of as points in a multi-dimensional space, where each dimension represents a distinct attribute.
Such data set can be represented by an $m \times n$ matrix, where there are m rows, one for each object, and n columns, one for each attribute.

Data Types:
- Document data
- Transaction data
- Graph data
- Ordered data
  - Spatial data
  - Temporal data

The type of the attribute depends on the following properties:
- Distinctness: $=\neq$
- Order: $<>$
- Differences are meaningful: $+-$
- Ratios are meaningful: $*/$

Attribute types:
- **Nominal/Categorical**: attribute values in a finite domain (*distinctness*)
- **Binary**: special case of nominal with two values
- **Ordinal**: attribute values have a total ordering (*distinctness* and *order*)
- **Numeric**: quantity (integer or real-valued) (*distinctness*, *order*, *differences*)
- **Ratio-Scaled**: we can speak of values as being an order of magnitude larger than the unit of measurement (*all 4 properties*)
  length, counts, elapsed time (A baseball game lasting 3 hours is 50% longer than a game lasting 2 hours)
- Discrete/Continuous: attribute values are discrete (finite or countably infinite) or continuous (real-valued).

Attribute Type	Description	Examples	Operations
Nominal	Nominal attribute values only distinguish. $(=, \neq)$	zip codes, employee ID numbers, eye color, sex: {male, female}	mode, entropy, contingency correlation, χ^2 test
Ordinal	Ordinal attribute values also order objects. $(<, >)$	hardness of minerals, {good, better, best}, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Interval	For interval attributes, differences between values are meaningful. $(+, -)$	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, t and F tests
Ratio	For ratio variables, both differences and ratios are meaningful. $(*, /)$	temperature in Kelvin, monetary quantities, counts, age, mass, length, current	geometric mean, harmonic mean, percent variation

Table 1.1: Attribute types, examples and operations

1.2.1 Data Quality

Examples of data quality problems:

- ◇ Wrong data
- ◇ Duplicate data
- ◇ Noise and outliers
- ◇ Missing values

In order to know our data and discovery quality issues we need use descriptive statistics for getting a global picture and summarize properties of data and compare such statistics with the expected behaviour. All around we can exploit visualization techniques that can help in detecting general or unusual patterns and trends, as well as outliers.

1.2.1.1 Histograms

A histogram shows the frequency distribution for a numerical attribute. The range of the numerical attribute is discretized into a fixed number of intervals (**bins**).

The number of bins according to Sturges' rule is:

$$k = \lceil \log_2 n + 1 \rceil$$

where n is the number of records in the data set. Sturges' rule is suitable for data from normal distributions and from data sets of moderate size.

1.2.1.2 Statistics notions

Notoriouses Mean/Median/Mode…The degree in which data tend to spread is called the *dispersion*, or **variance** of the data.

The most common measures for data dispersion are **range** (The distance between the largest and the smallest values), **standard deviation**, the **five-number summary** (based on *quartiles*), and the **inter-quartile range**.

$$\text{variance}(x) = \sigma^2 = s_x^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \bar{x})^2$$

Standard deviation σ is the square root of variance σ^2.

Because of outliers, other measures are often used:

- ◇ absolute average deviation (AAD)

$$\text{AAD}(x) = \frac{1}{m}\sum_{i=1}^{m}|x_i - \bar{x}|$$

- ◇ median average deviation (MAD)

$$\text{MAD}(x) = \text{median}\left(\{|x_1 - \bar{x}|, \ldots, |x_m - \bar{x}|\}\right)$$

1.2.1.3 Box-Plot

1.3 Data Understanding - Lab

Data comes from diverse sources, and generally is not tailor-made for some downstream task. We need to start from basics:

- ◇ What features are available?
- ◇ What are they measuring, exactly?
- ◇ What properties do they have?
- ◇ What are their relations?
- ◇ Are there outliers?
- ◇ …

Data can be of different nature which may co-occur:

- ◇ **Temporal**: the data describes events over time
- ◇ **Sequential**: the data spans some ordering
- ◇ **Relational**: the data describes event in between instances
- ◇ **Spatial**: the data describes space
- ◇ **Independent**: instances in data are independent observations

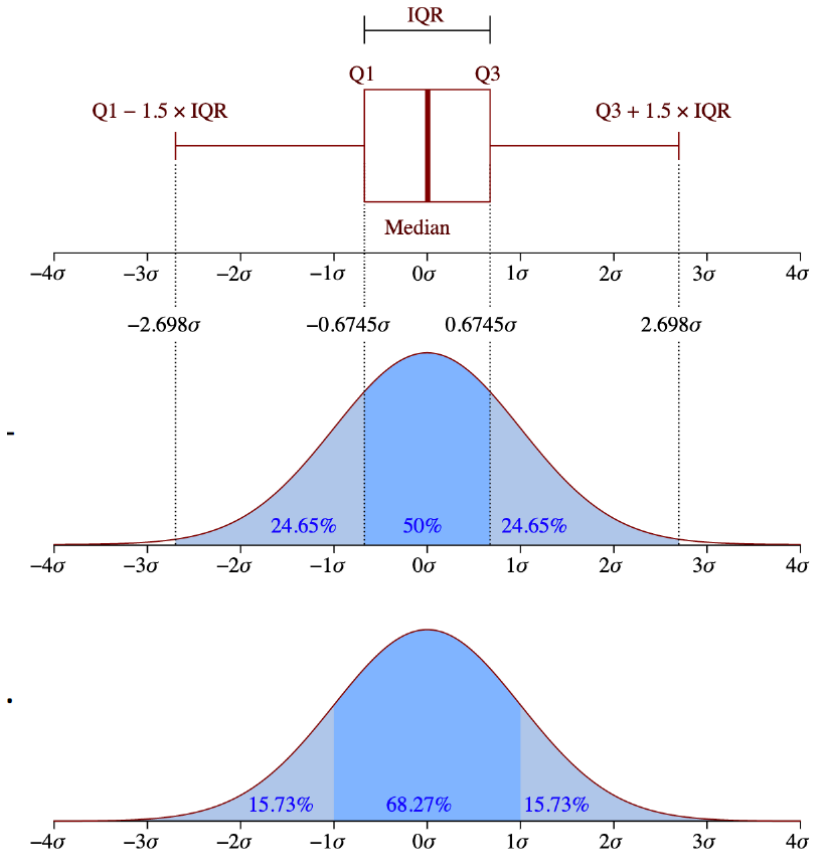


Fig. 1.5: Box-Plot

1.3.1 Data Collections

We refer to single instances in the collections as objects/records/instances, which are described by attributes.

Id	Age	Income	Marital	Loan
0	30	2.5k	Married	Yes
1	24	1.4k	Single	No
...	...	...	...	...

Table 1.2: Grant Data

- ◇ Attributes: `Id , Age , Income , Marital,Loan grant`
- ◇ Records: `0, 30, 2.5k, Married, Yes , 1, 24,1.4k, Single, No`

1.3.2 Data Types

1.3.2.1 Tabular

When records are independent, and described by the same finite set of features, they are often represented in a tabular form: the data matrix. Each row is a record, each dimension is an attribute.

Records on the rows, attributes on the columns.

Id	Age	Bike used	Length	Duration	Date	Cyclist
0	28	Colnago VRS4	152.4	3:43:12	15-5-2025	Alessandro Covi
1	40	Cervelo RS5	72.4	2:55:01	4-3-2024	Gianni Affino

Table 1.3: Cyclist Data

1.3.2.2 Transaction

A feature contains a (multi)set of items.

PurchaseId	Cart	Bought on
0	Bread, Milk	17:12-15-5-2025
1	Notebook, Pens, Bread, Basil	8:04-4-3-2024

Table 1.4: Transaction Data

Records on the rows, attributes on the columns.

1.3.2.3 Graph

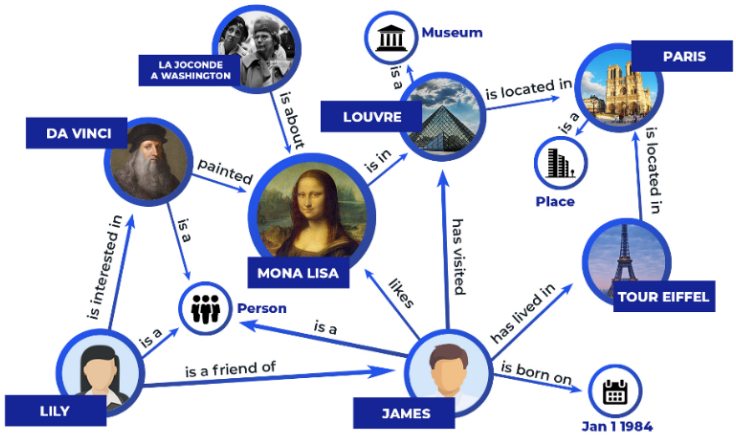


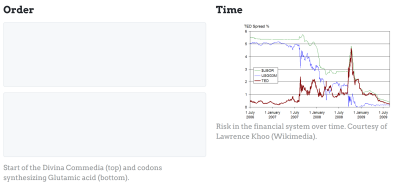
Fig. 1.6: Graph Data

Data is linked, either on records or features. Records are nodes in a graph, attributes can vary wildly across records.

1.3.2.4 Sequential

Records are sequences (of variable length): attributes are indexed (order or time).

Image on the right lacks two images ☺



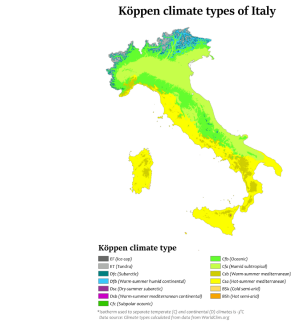
Start of the Divina Commedia (top) and codons synthesizing Glutamic acid (bottom).

Risk in the financial system over time. Courtesy of Lawrence Khoo (Wikimedia).

Fig. 1.7: Sequential Data

1.3.2.5 Spatial

Records are associated with locations in space: attributes can include coordinates, regions, etc.



Climate types of Italy. Courtesy of Adam Peterson (Wikidata). CC BY-SA 4.0.

Fig. 1.8: Spatial Data

1.3.2.6 Attribute types

Type	Description	Example
Numerical	Values have a total ordering, and represent some numerical quantity	Age, dates
Ordinal	Values have a total ordering, and represent some quantity	Dress size, Cup size
Binary	Values are one of two categories: no ordering	Boolean values
Categorical	Values of one of multiple categories: no ordering	Country, Job

Table 1.5: Types of Data

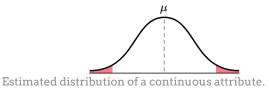
1.3.2.7 Values types

Values can be either:

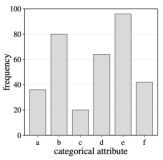
- ◇ **Discrete**
  Defined in a finite or countably finite domain, e.g., country, job, cup size. Note: ordinal values may be discrete too!
- ◇ **Continuous**
  Defined in a continuous and infinite domain, e.g., distance.

1.3.3 Data Syntax and Semantics

Given the categorization of the records and attributes of your data, we can study its general behavior. We leverage some basic statistical tools, first of all by drawing the empirical distribution of the attributes.



Estimated distribution of a continuous attribute.



Distribution of a categorical attribute in a bar chart.

Fig. 1.9: Data Syntax and Semantics

Useful statistics for data semantics

- **Expected value**

$$\mathbb{E}[X] = \sum_{x \in \text{dom}(X)} \Pr(X = x)\, x$$

  A statistic representative of the value of an attribute, weighing values and their probability
- **Variance**

$$\sigma^2(X) = \mathbb{E}\left[\sum_{x \in \text{dom}(X)} \big(x - \mathbb{E}[X]\big)^2\right]$$

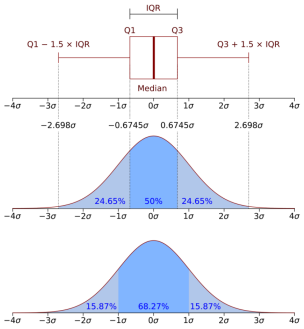
  Distance from the expected value of all records: the data spread
- **Quantiles**

$$q^p = x \;\; s.t. \;\; \Pr(X \leq x) = q^p$$

  Inflection points defining values for a threshold, e.g., if the 99-th percentile is , then we
- **Interquantile range**

$$q^{75} - q^{25}$$

  Distance between quantiles: how spread are inflection points?



A Normal distribution, its q^{25} and q^{75} quantiles, and interquartile range between them. On top, a box plot visualization of the distribution.

Fig. 1.7: Statistics Graph

Statistical summary of the distribution are typically accompanied by visual and semantic one.
Erroneous or weird values to be cleaned later may already pop up in these basic steps. Outlier values typically skew statistics. Variance is often replaced by absolute/median average deviation

1.4 Data Cleaning

There are some concepts to be aware of when dealing with data quality, hence data cleaning.

Data accuracy is the degree to which data correctly describes the "real world" object or event being described.

- Syntactic: values outside domain, e.g., Eataly in Country
- Semantic: values in domain, but semantically wrong, e.g., age is 3, and weight is 82kg

Completeness is the degree to which all required data is known.
Some attributes are not collected, or are collected partially, e.g., temperature was not recorded by the sensor.

Biased gathering is the degree to which data may be over/under-representative, e.g., the bank may only provide data about successful loan applicants.

Timeliness is the degree to which data is up to date.

Remember: *garbage in, garbage out!*[1] In a task-agnostic view, we are interested in addressing the above by tackling:

- **Duplicates**: skews the data distribution
- **Missing values**: give false/partial information

[1] i.e. if you have garbage data, you'll get garbage results

- ◇ **Noise**: uninformative of the data
- ◇ **Poor accuracy**: gives wrong data
- ◇ **Outliers**: skews the data distribution and models of the data

1.4.1 Handling Duplicates

Remove them... when appropriate! Not all duplicates are garbage, it depends on what insight you can gather from it.

Case A
You have data on registration to your website, with several duplicate e-mails. Insights:

- ◇ The "Sign in" button is hard to find
- ◇ The "Sign in" button is less visible than the "Sign up" button
- ◇ Your site is so anonymous people forget they signed up already

Case B
You have data on credit account opening from Poste (Italian postal service) with several duplicate e-mails. Insights:

- ◇ The client hacked the database and added themselves to ask more credit (unlikely)
- ◇ Poste's tech staff is underwhelming (very likely)

1.4.1.1 Duplicate Features

Duplicate features may be more tricky. Features convey similar, although not equal, information to others.
Examples:

- ◇ Resting heart rate and heart rate under continuous high effort
- ◇ Education level and reading skills
- ◇ Rent and available bank deposit

These pairs of features are not per se one duplicate of the other, but are strongly related: when one grows, so does the other, and when one goes down, so does the other.

Linear (and rank) relationships between two features X, Y can be quantified with their correlation. Correlation ranges in $[-1, 1]$, from perfectly negative to perfectly positive correlation.
Given two lists of values x^{i^n}, y^{i^n} we can compute two main correlation types.

- ◇ **Pearson correlation**
$$\rho_P^{X,Y} = \frac{\mathbb{E}\left[(x^i - \mathbb{E}[X])(y^i - \mathbb{E}[Y])\right]}{\sigma_X \sigma_Y}$$
Measures linear correlation between two numerical features and their values.
- ◇ **Spearman correlation**
$$\rho_S = \rho_P^{\text{rank}(X),\text{rank}(Y)}$$
Measures monotonic correlation between two ordinal or numerical features.
rank is the position of the value in the sorted list of values.

1.4.2 Handling Missing Values

Data may be missing for any number of reasons (at random or not at random).

- ◇ A record has a large and/or significant set of missing attributes
- ◇ An attribute has a large percentage of missing values

We have two choices: **dropping** or **imputing**.

Dropping
If a record has a large and/or significant set of missing attributes, or an attribute has a large percentage of missing values, we can drop the record/attribute.

- ◇ High percentage of missing values
- ◇ Missing values in critical attributes, e.g., a patient in cardiology has no heart rate data

Imputing
Imputing means replacing the missing value with a "best guess" value.
If a record has a small set of missing attributes, or an attribute has a small percentage of missing values, we can impute the missing values. We have to create a model to predict the missing value.

- ◇ Low percentage of missing values
- ◇ Reasonably good understanding of the attribute semantics/distribution

◇ Presence of related attributes

1.4.3 Outliers

Quantiles and distributions inform us on what values may be outliers. They are typically dropped, and unlike missing values, almost never imputed. We'll tackle algorithms later in the course.

1.4.3.1 Flower Example

There is a dataset with 5 attributes: sepal length, sepal width, petal length, petal width, and species (type).

Sepal L.	Sepal W.	Petal L.	Petal W.	Type
5.1	3.5	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
...	...	...	...	...

Table 1.6: Flower Dataset Example

Fig. 1.8: Flower Data plotted

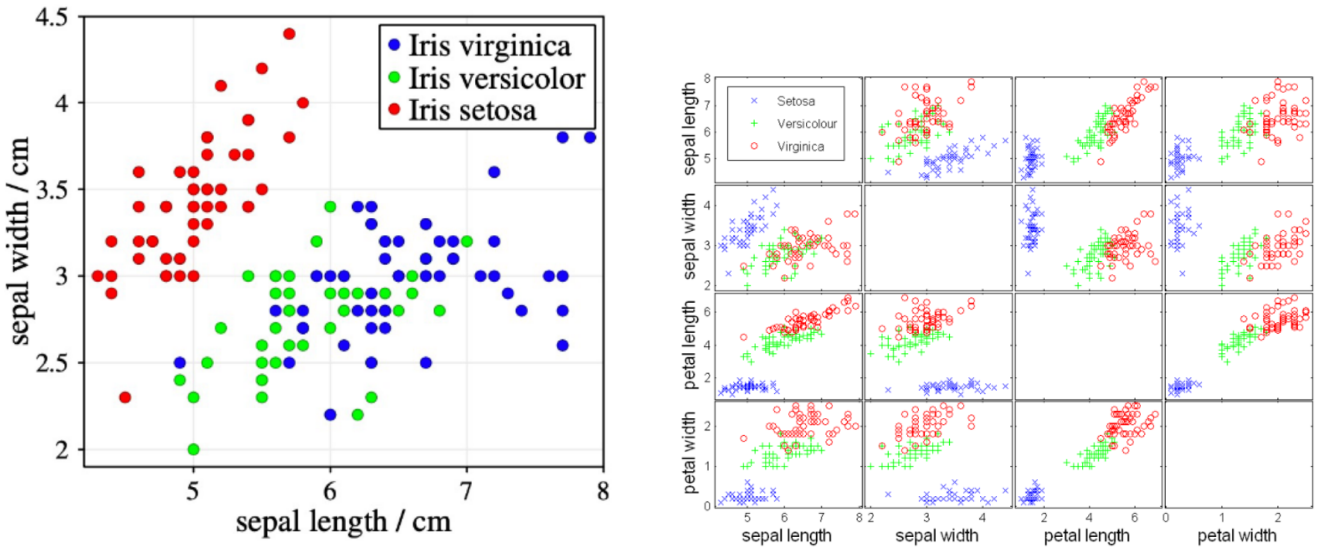


Fig. 1.9: Scatter plot of sepal length and width, and scatter matrix: scatter plots of all pairs of attributes in the Iris dataset.

Plot bivariate (or trivariate) data, eyeing data correlation and outliers.

1.5 Data Preparation

We will delve into the following techniques of data preparation:

◇ Aggregation
◇ Data Reduction: Sampling
◇ Dimensionality Reduction

- ◇ Feature subset selection
- ◇ Feature creation
- ◇ Discretization and Binarization
- ◇ Attribute Transformation

1.5.1 Aggregation

Aggregation is the process of combining two or more attributes (or objects) into a single attribute (or object).

Purpose

- ◇ Data reduction
  - Reduce the number of attributes or objects
- ◇ Change of scale
  - Cities aggregated into regions, states, countries, etc.
  - Days aggregated into weeks, months, or years
- ◇ More "stable" data
  - Aggregated data tends to have less variability

1.5.2 Reduction

Reduction is simply reducing the amount of data We may reduce the number of **records** by sampling or clustering, or the number of **attributes** (*columns*) by selecting a subset of them, or by creating a new —smaller— set of attributes from the old one.

1.5.2.1 Sampling

Sampling is the main technique employed for data reduction.
It is often used for both the preliminary investigation of the data and the final data analysis.

Sampling is typically used in data mining because processing the entire set of data of interest is too expensive or time consuming.

The key principle for effective sampling is the following:

- ◇ Using a sample will work almost as well as using the entire data set, if the sample is representative
- ◇ A sample is representative if it has approximately the same properties (of interest) as the original set of data

- ◇ **Simple Random Sampling**
  - There is an *equal probability* of selecting any particular item
  - Sampling **without replacement**
    - ○ As each item is selected, it is removed from the population
  - Sampling **with replacement**
    - ○ Objects are not removed from the population as they are selected for the sample.
    - ○ In sampling with replacement, the same object can be picked up more than once
  - **Stratified sampling**
  - Split the data into several partitions; then draw random samples from each partition
  - Approximation of the percentage of each class
  - Suitable for distribution with peaks: each peak is a **layer**

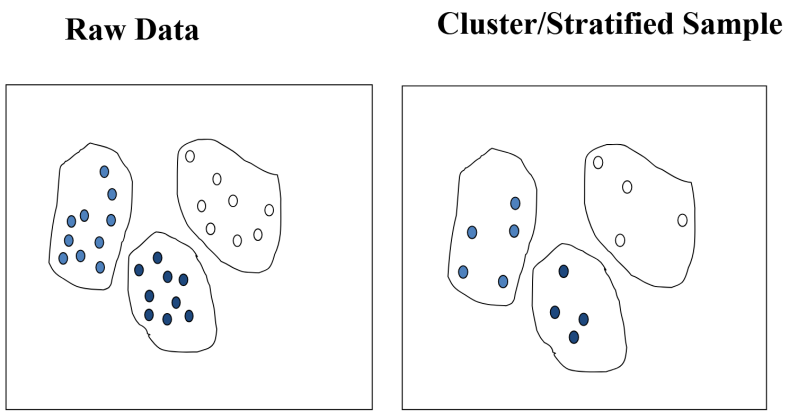


Fig. 1.10: Stratified Sampling

1.5.2.2 Dimensionality Reduction

This consists in reducing the number of attributes (or features) in the data. We want a selection of a subset of attributes that is as small as possible and sufficient for the data analysis.

- ◇ removing (more or less) irrelevant features
  - – Contain no information that is useful for the data mining task at hand
  - – Example: students' ID is often irrelevant to the task of predicting students' GPA
- ◇ removing redundant features
  - – Duplicate much or all of the information contained in one or more other attributes
  - – Example: purchase price of a product and the amount of sales tax paid

Curse of Dimensionality

When dimensionality increases, data becomes **increasingly sparse** in the space that it occupies.

Definitions of density and distance between points, which are critical for clustering and outlier detection, become less meaningful.

This phenomenon is known as the **curse of dimensionality**.

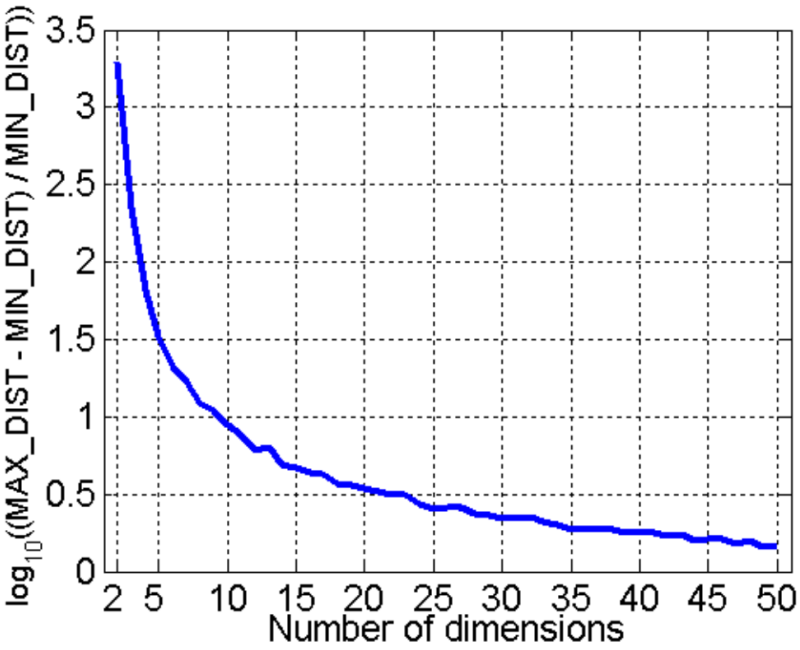


Fig. 1.11: $log_{(MAXDIST-MINDIST)/MINDIST}$ decreases as the dimensionality increases, meaning that the difference between the farthest and nearest neighbor distances becomes less significant

Purposes of dimensionality reduction include:
- ◇ Avoid curse of dimensionality
- ◇ Reduce amount of time and memory required by data mining algorithms
- ◇ Allow data to be more easily visualized
- ◇ May help to eliminate irrelevant features or reduce noise

Techniques to do so include:
- ◇ Principal Components Analysis (PCA)
- ◇ Singular Value Decomposition
- ◇ Others: supervised and non-linear techniques

1.5.2.3 Feature Subset Selection

Feature subset selection consists in selecting a subset of the original features. The goal is to find a minimal subset of features that is as good as the entire set of features for the data analysis task at hand.

For removing irrelevant features, it is needed a **performance measure** indicating how well a feature or subset of features performs w.r.t. the considered data analysis task.

For removing **redundant features**, either a *performance measure* for subsets of features or a *correlation measure* is needed.

Filter Methods

- ◇ Selection after analyzing the **significance** and **correlation** with other attributes
- ◇ Selection is independent of any data mining task
- ◇ The operation is a pre-processing

Wrapper Methods

- ◇ Selecting the top-ranked features using as reference a DM task
- ◇ Incremental Selection of the "best" attributes
  "Best" = with respect to a specific measure of statistical significance (e.g.: information gain)

Embedded Methods

- ◇ Selection as part of the data mining algorithm
- ◇ During the operation of the DM algorithm, the algorithm itself decides which attributes to use and which to ignore (e.g. Decision tree)

Feature Selection Techniques

- ◇ **Selecting the top-ranked features**: Choose the features with the best evaluation when single features are evaluated.
- ◇ **Selecting the top-ranked subset**: Choose the subset of features with the best performance. This requires exhaustive search and is impossible for larger numbers of features. (For 20 features there are already more than one million possible subsets.)
- ◇ **Forward selection**: Start with the empty set of features and add features one by one. In each step, add the feature that yields the best improvement of the performance.
- ◇ **Backward elimination**: Start with the full set of features and remove features one by one. In each step, remove the feature that yields to the least decrease in performance.

Chapter 2

Data Representation

- By **correlation**
  I want to represent data according to the correlation of the dataset
  Algorithm: `PCA`
- By **neighborhood**
  I want to represent the data so that similar instances are similar
  Algorithm: `t-SNE`
- By **manifold**
  I want to represent the data so that its manifold is preserved
  Algorithm: `UMAP`

2.1 Principal Component Analysis (PCA)

PCA is a linear dimensionality reduction technique that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Essentially, PCA exploits spectral decomposition of the whole dataset to find a new basis for the data.

Data can often be correlated, and linear dependencies can exist among variables, e.g.,

- Rent is linearly dependent on salary and food expenses
- Bank deposit is linearly dependent on salary and work
- Cardio is linearly dependent on VO_2max

Vectors are m-dimensional elements in a field, and enjoy both addition and multiplication by scalar.
Composing these two, we can generate an infinite number of vectors: this is a **vector space**, and is defined by the basis vectors involved in the composition.

A matrix A defines a space…and thus a linear transformation! Av linearly combines the columns of with coefficients given by v.

Eigenpairs (λ, v) of a square matrix A are defined by the equation $Av = \lambda v$.

The eigenvectors $v_1, \dots, v_m$ of a matrix A define the stretching of the space, and their eigenvalues $\lambda_1 > \dots > \lambda_m$ define the stretching factor.

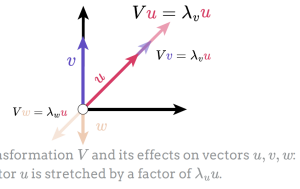
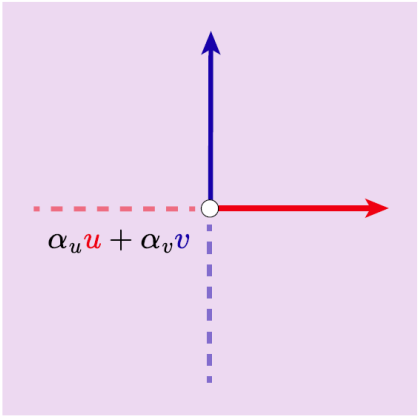


Fig. 2.2: Eigenvectors of a matrix

PCA projects some data X to $\hat{X}$ through a linear transformation A: $AX = \hat{X}$.

Fun fact #1: for a mean-centered $\bar{X}$, the slope is directly proportional to the covariance!



Two vectors u, v (in red and blue), and the plane spanned by all their linear combinations $\alpha_u u + \alpha_v v$ (in purple).

Fig. 2.1: Vector space spanned by two vectors

$$\bar{\Sigma} = \begin{bmatrix} \sigma^2_{\bar{X}^1} & \cdots & \text{cov}(\bar{X}^1, \bar{X}^n) \\ \cdots & \cdots & \cdots \\ & & \sigma^2_{\bar{X}^n} \end{bmatrix}$$

PCA in a nutshell

There is some pretty complicated linear algebra behind PCA, but the main steps are the following:

1. Mean-center your data X to get $\bar{X}$
   This means subtracting the mean of each column from the respective column entries, obtaining relative values with respect to the mean.
2. Compute its eigenvectors matrix $\bar{V}$
   We build the covariance matrix $\bar{\Sigma}$ of $\bar{X}$, and compute its eigenvectors and eigenvalues. The eigenvectors represent the directions of maximum variance (the principal components), and the eigenvalues indicate the magnitude of variance in those directions. Eigenvectors are sorted in decreasing order of their eigenvalues and stored in the columns of matrix V.
   A *correlation matrix* is the same as a covariance matrix, but with normalized values between -1 and 1, obtained by dividing each covariance by the product of the standard deviations of the respective variables.
3. Transpose V to obtain the transformation V^T
   We transpose V to prepare for the projection step, as we want to project our data onto the new basis defined by the principal components, allowing for easy computation of dot products.
4. Project the data: $\bar{X}$ through $V^T\bar{X}$, obtaining the PCA-transformed data $\hat{X}$
   Here we perform the actual projection of our mean-centered data onto the new basis defined by the principal components, essentially rotating our data to align with the directions of maximum variance. Each axis in $\hat{X}$ corresponds to a principal component (PC1,PC2,...), ordered by the amount of variance they capture from the original data.

In simple terms: PCA looks at your data and finds the "most important directions" - imagine you have a cloud of points and you want to find the best line that captures the main trend. PCA finds not just one line, but multiple directions ordered by importance. It then rotates your data so that the first dimension captures the most variation, the second dimension captures the second most variation. This allows you to keep only the first two dimensions while retaining most of the information, effectively reducing the complexity of your data while preserving its essential structure.

2.1.1 Observations

- ◇ PCA redefines data by removing collinearity: if your data has low covariance, the transformation will have minimal effect.
- ◇ PCA performs a linear transformation to tackle linear relationships between variables. Nonlinear relationships are not influenced.
- ◇ PCA does not work very well for high complexity data.

Uses

- ◇ **Feature selection**: high covariance of a feature may indicate disposability.
- ◇ **Dimensionality reduction**: trimming columns of lets us reduce the dimension of the resulting data.
- ◇ **Clustering preprocessing**: correlated features inflate object similarity

2.2 t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique particularly well suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. It works by modeling the data as a distribution of points in a high-dimensional space and then finding a lower-dimensional representation that preserves the pairwise similarities between points.

`t-SNE` focuses on data clusters rather than subspace representation, and again maps the original data X to a representation $\hat{X}$.

`t-SNE` tackles this problem in two phases:

1. **Similarity phase** In the original space $\mathcal{X}$, how similar is x_i to x_j?
   How similar is x_i to x_j? Even better, what is the probability that x_i is a neighbor of x_j?
2. **Embedding phase** In the mapped space $\hat{\mathcal{X}}$, how similar is $\hat{x}_i$ to $\hat{x}_j$?

2.2.1 Similarity phase

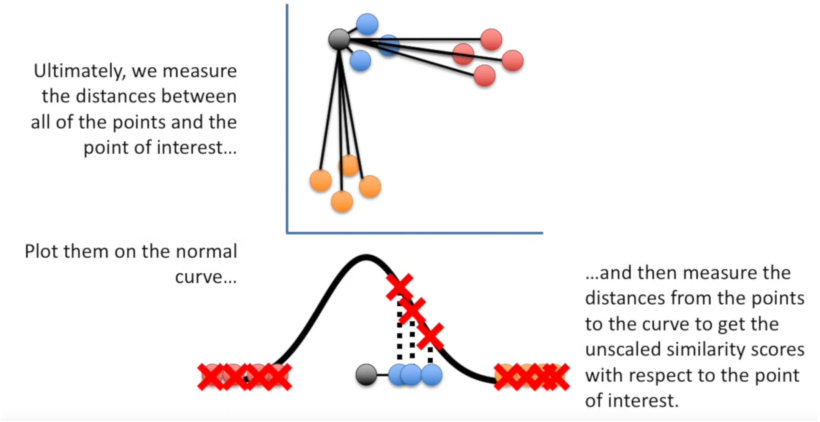


Fig. 2.2: t-SNE plotting distance on an X axis and then projecting it on a normal distribution curve, to get the probability of being a neighbor

The similarity phase computes the similarity between points in the original high-dimensional space. This is typically done by converting the Euclidean distances between points into conditional probabilities that represent similarities. The probability that point x_j is a neighbor of point x_i is given by a Gaussian distribution centered at x_i. The variance of this Gaussian is controlled by a parameter called **perplexity**, which can be thought of as a smooth measure of the effective number of neighbors.

This yields a neighboring matrix P where each entry p_{ij} represents the probability that point x_j is a neighbor of point x_i in the original high-dimensional space.

The conditional probability is computed as:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

where σ_i is the variance of the Gaussian centered at point x_i. The perplexity parameter determines σ_i through a binary search to match the desired effective number of neighbors.

2.2.2 Embedding phase

In the embedding phase, t-SNE defines a similar probability distribution over the points in the low-dimensional map. However, instead of using a Gaussian distribution, it uses a Student's t-distribution with one degree of freedom (heavy-tailed distribution) to avoid the "crowding problem" where moderate distant points are forced to be too far apart in the low-dimensional representation.

The probability in the low-dimensional space is:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

where y_i and y_j are the low-dimensional counterparts of x_i and x_j.

2.2.3 Optimization

t-SNE minimizes the Kullback-Leibler divergence between the probability distributions P (high-dimensional) and Q (low-dimensional):

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

The algorithm uses gradient descent to find the low-dimensional representation Y that minimizes this cost function, effectively preserving the neighborhood structure of the original high-dimensional data.

Key advantages:

- ◇ Excellent for visualization of high-dimensional data
- ◇ Preserves local neighborhood structure
- ◇ Can reveal clusters and patterns not visible in linear methods

Key limitations:

- ◇ Computationally expensive (quadratic in the number of points)
- ◇ Non-deterministic (different runs can give different results)
- ◇ Sensitive to hyperparameters, especially perplexity
- ◇ Not suitable for embedding new data points (no explicit mapping function)

2.2.4 t-SNE — Practical summary

t-SNE (t-distributed Stochastic Neighbor Embedding) is a nonlinear method to embed high-dimensional data into 2–3 dimensions for visualization. It prioritizes preserving local neighborhoods rather than global linear structure.

2.2.4.1 Workflow (practical)

- ◇ **Similarity in high-dim:** for each point x_i compute conditional probabilities

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

  where σ_i is chosen (by binary search) to match a user-specified *perplexity* (effective neighbor count).
- ◇ **Symmetrize:** form

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

  producing a joint probability matrix P.
- ◇ **Low-d similarity:** place points y_i in low-d and define heavy-tailed affinities

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}.$$

- ◇ **Cost and optimization:** minimize KL divergence

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

via gradient descent with momentum; common practical boosts include *early exaggeration* (multiply P by a factor early on), learning-rate scheduling, and momentum.
- ◇ **Scaling tricks:** for large n use approximations (Barnes–Hut or FFT-based) to reduce complexity from $O(n^2)$.

2.2.4.2 Practical notes

- ◇ Good for revealing clusters and local structure, but distances between far clusters are not reliable.
- ◇ Sensitive to hyperparameters (perplexity, learning rate, iterations, early-exaggeration).
- ◇ No explicit mapping function: embedding new points requires recomputing or using approximations.

2.2.4.3 Why t-SNE is non-deterministic

Non-determinism arises from several sources commonly used in implementations:

- ◇ random initialization of low-dimensional points Y;
- ◇ stochasticity in the optimization (random tie-breaks, minibatching or noisy gradients in some variants);
- ◇ randomized approximations (e.g. tree construction in Barnes–Hut, random projection steps);
- ◇ sensitivity to optimizer hyperparameters and early-exaggeration phase which amplify small random differences.

These factors make different runs (or different implementations/settings) produce different embeddings even on the same data.

2.3 UMAP

UMAP (Uniform Manifold Approximation and Projection) is a nonlinear dimensionality reduction technique that is particularly effective for visualizing high-dimensional data in a low-dimensional space. It is based on manifold learning and topological data analysis, aiming to preserve both local and global structure of the data.

The computed distances induce a connectivity graph, and thus an adjacency matrix A, its edges measuring distances among instances. After turning distances into probabilities, UMAP optimizes a distance on A, to make it so that all and only the edges on the original manifold also appear in the transformed manifold with the same magnitude.

For the set of edges E, UMAP minimizes

$$-\sum_{e\in E}\left(\underbrace{\Pr(e;X)\log(\Pr(e;Z))}_{\text{existing edges}}+\underbrace{(1-\Pr(e;Z))\log(1-\Pr(e;X))}_{\text{non-existing edges}}\right),$$

where $\Pr(e;X), \Pr(e;Z)$ indicate the probability of edge e in the original and transformed representation, respectively.

Note that since the probability is $\in [0,1]$, its logarithm is negative and spans $(-\infty, 0]$, that's the reason behind the minus "$-$" before the $\sum$ and the overall functioning of the formula.
In this way, when two points in the original space are close ($\Pr(e;X) \approx 1$), the first term dominates and the cost is minimized by making them close in the transformed space as well ($\Pr(e;Z) \approx 1 \Rightarrow \log(\Pr(e;Z)) \approx 0$), because if instead they are far apart in the transformed space Z ($\Pr(e;Z) \approx 0$), then $\log(\Pr(e;Z)) \approx -\infty$ and the cost becomes very large.
Conversely, when two points in the original space are far apart ($\Pr(e;X) \approx 0$), the second term dominates and the cost is minimized by making them far apart in the transformed space as well ($\Pr(e;Z) \approx 0 \Rightarrow \log(1-\Pr(e;Z)) \approx 0$), because if instead they are close in the transformed space ($\Pr(e;Z) \approx 1$), then $\log(1-\Pr(e;Z)) \approx -\infty$ and the cost becomes very large.

2.4 Brief conclusions

In PCA the process is deterministic and the transformations are all linear, making the results interpretable.
UMAP and t-SNE instead are non interpretable methods: the transformations are nonlinear and non-deterministic, since they rely on random initialization and stochastic (probability-related) optimization. Thus, the results may vary between different runs even on the same data, and the axes in the transformed space do not have a clear meaning.

Chapter 3

Data Cleaning

Key Points

- How to handle **anomalous values**
- How to handle **outliers**
- **Data Transformations**

3.1 Anomalous Values

- **Missing** values - `NULL`, `?`, `<NA>`, `""`
- **Unknown** Values - Values without a real meaning (e.g. `gH43cz` as "age")
- **Not Valid** Values - Values not significant (e.g. `-24` as "age")

We can handle anomalous values with various techniques. First of all we can **eliminate** the records with anomalous values.
We could also **substitute** anomalous values, knowing, however, that it could influence the original distribution of numerical values.
To mitigate this effect, we can use **mean/median/mode** to substitute missing values, or estimate missing values using the **probability distribution** of existing values.
We could also **segment data** and apply the above for every segment where there happen to be missing values, hence using mean/mode/median or the probability distribution of the segment.
Finally, we could use *predictive models* (**classification/regression**) to estimate missing values, using the other attributes as predictors.

3.1.1 Discretization

*Discretization is the process of converting a **continuous** attribute into an **ordinal** attribute.*

- A potentially infinite number of values are mapped into a small number of categories
- Discretization is commonly used in classification
- Many classification algorithms work best if both the independent and dependent variables have only a few values

Unsupervised Discretization

- No label for instances
- The number of classes is unknown

Techniques of binning:

- **Natural** binning - Intervals with the same width
- **Equal Frequency** binning - Intervals with the same frequency
- **Statistical** binning - Use statistical information (Mean, variance, Quartile)

3.1.1.1 Natural Binning

This is a fairly simple approach: we sort the values, subdividing the range of values into k parts with the same size.

$$\sigma = \frac{x_{max} - x_{min}}{k}$$

Then, element x_j is assigned to bin —i.e. belong to the class— i if:

$$x_j \in [x_{min} + i\sigma, x_{min} + (i+1)\sigma)$$

Note however that this approach is sensitive to outliers, which can skew the range of values, generating very unbalanced distributions.

3.1.1.2 Equal Frequency Binning

Sort and count the elements, definition of k intervals of f, where, having N elements:

$$f = \frac{N}{k}$$

The element x_j is assigned to bin i if:

$$i \times f \leq j < (i+1) \times f$$

This is not always suitable for highlighting interesting correlations.

3.1.1.3 How many bins?

In both cases, the number of bins k is a parameter to be set. A rule of thumb is to set k accorgin to Sturges' optimal number of classes C for N elements:

$$C = 1 + \frac{10}{3}\log_{10}(N)$$

where N is the number of elements. The optimal width of the classes depends on the variance and the number of data points:

$$h = \frac{3.5 \cdot s}{\sqrt{N}}$$

3.2 Supervised discretization

The discretization has a quantifiable goal and the number of classes is known.

There are two main approaches: **entropy** and **percentiles** based discretization.

3.2.1 Entropy-based Discretization

Minimizes the entropy wrt[1] a label, with the goal of maximizing the purity of intervals (information gain).

We start with each attribute value as a separate interval and then create larger intervals by merging adjacent intervals that are similar according to a statistical test.

Starts by bisecting the initial values so that the resulting two intervals give minimum entropy.
The splitting process goes on with another interval, typically choosing the interval with the worst (highest) entropy.
Stop when a user-specified number of intervals is reached, or a stopping criterion is satisfied.

3.3 Binarization

Binarization is the process of converting a continuous attribute into a binary attribute.
This can be useful in various scenarios, such as when we want to simplify the model or when we need to handle categorical variables. It is common to apply this for **association** analysis.

There are several techniques for binarization of continuous attributes:

- ◇ **Thresholding** - Assigning values above a certain threshold to one class and values below to another.
- ◇ **One-Hot Encoding** - Creating binary columns for each category in a categorical variable.
- ◇ **Binarization with Decision Trees** - Using decision tree algorithms to find optimal splits for binarization.

[1] "with respect to"

There are also techniques for binarization of categorical attributes:

Table 3.1: Conversion of a categorical attribute to three binary attributes.

Categorical Value	**Integer Value**	x_1	x_2	x_3
awful	0	0	0	0
poor	1	0	0	1
OK	2	0	1	0
good	3	0	1	1
great	4	1	0	0

This however leads to highlighting associations that are not really there, for example, in the table above, x_2 and x_3 may look correlated, but, in fact, they are not.

Table 3.2: Conversion of a categorical attribute to five asymmetric binary attributes.

Categorical Value	**Integer Value**	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
OK	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

3.3.1 Attribute Transformation

An attribute transform is a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values.

- ◇ Simple functions: x^k, $log(x)$, e^x, $|x|$
- ◇ **Normalization** Refers to various techniques to adjust to differences among attributes in terms of frequency of occurrence, mean, variance, range
- ◇ Take out unwanted, common signal, e.g., seasonality
- ◇ In statistics, **standardization** refers to subtracting off the mean and dividing by the standard deviation

The transformation $Y = T(X)$ should:
- ◇ preserve the relevant information of X
- ◇ eliminates at least one of the problems of X
- ◇ is more useful of X

3.3.1.1 Normalization

- ◇ Min-Max Normalization

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

- ◇ z-score Normalization

$$v' = \frac{v - \mu}{\sigma} = \frac{v - mean_A}{stddev_A}$$

- ◇ Decimal Scaling

$$v' = \frac{v}{10^j} \qquad \textit{where } j \textit{ is the smallest integer such that } max(|v'|) < 1$$

3.3.1.2 Transformation functions

- ◇ Exponential Transformation

$$T_p(x) = \begin{cases} ax^p + b & (p \neq 0) \\ c \log x + d & (p = 0) \end{cases}$$

where $a, b, c, d, p \in \mathbb{R}$
  - – Preserve the order
  - – Preserve some basic statistics
  - – They are continuous functions
  - – They are derivable
  - – They are specified by simple functions

- ◇ Logarithmic Transformation (- Stabilizing Variance)

$$T(x) = c \log x + d$$

  - Applicable to positive values
  - Makes homogenous the variance in log-normal distributions
    E.g.: normalize seasonal peaks
- ◇ Square Root and Reciprocal (- Stabilizing Variance)

$$T(x) = ax^p + b$$

  - Square-root Transformation
    - p = 1/c, c integer number
    - To make homogenous the variance of particular distributions e.g., Poisson Distribution
  - Reciprocal Transformation
    - p < 0
    - Suitable for analyzing time series, when the variance increases too much wrt the mean

To detect outliers on one dimension we can use also the **IQR** (Interquartile Range) method or the **Z-Score** method.

- ◇ **IQR** method:
  - Calculate Q1 (25th percentile) and Q3 (75th percentile)
  - Compute IQR = Q3 - Q1
  - Define lower bound = Q1 - 1.5 * IQR
  - Define upper bound = Q3 + 1.5 * IQR
  - Any data point outside these bounds is considered an outlier
- ◇ **Z-Score** method:
  - Calculate the mean (μ) and standard deviation (σ) of the dataset
  - For each data point x, compute the Z-score: $Z = \frac{(x-\mu)}{\sigma}$
  - A common threshold is $|Z| > 3.29$; data points with Z-scores beyond this threshold are considered outliers

Part II

Clustering

Chapter 4

Cluster analysis

Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups.

The aim of clustering is to ease data **understanding** and **summarization**, i.e. reduce the size of data sets.

The following are *NOT* cluster analysis:

- Simple segmentation
  - Dividing students into different registration groups alphabetically, by last name
- Results of a query
  - Groupings are a result of an external specification
  - Clustering is a grouping of objects based on the data
- Supervised classification
  - Have class label information
- Association Analysis
  - Local vs. global connections

These ain't clustering essentially because there is no **similarity** measure involved, which instead is the core of clustering.

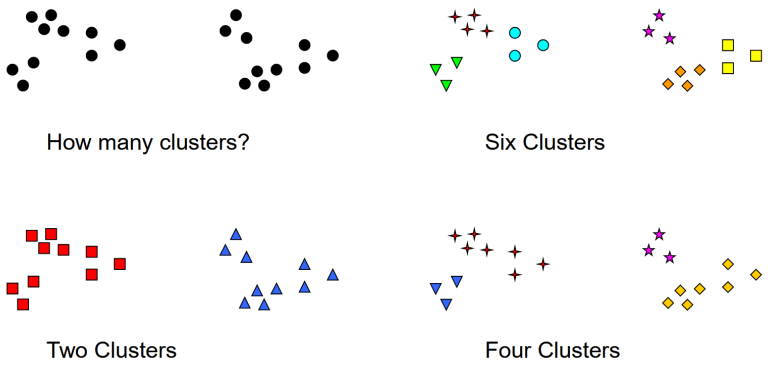


Fig. 4.1: "Cluster" may be an ambiguous term. How to define a cluster? How big is a cluster? How many clusters are there?

4.1 Definitions

A **clustering** is a set of clusters. There is an important distinction between hierarchical and partitional sets of clusters.

- *Partitional* Clustering
  A division of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset.
- *Hierarchical* Clustering
  A set of nested clusters organized as a hierarchical tree.

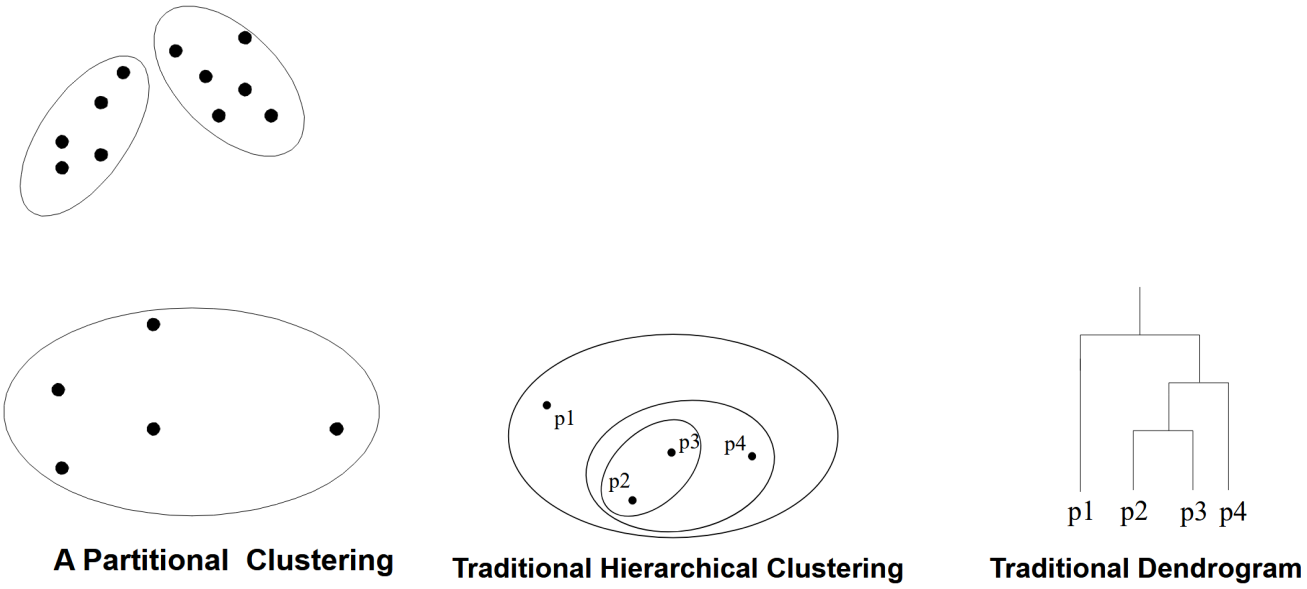


Fig. 4.2: Partitional vs Hierarchical clustering

There are other distinctions among clusterings:

- ◇ Exclusive versus non-exclusive
  - In non-exclusive clusterings, points may belong to multiple clusters.
  - Can represent multiple classes or 'border' points
- ◇ Fuzzy versus non-fuzzy
  - In fuzzy clustering, a point belongs to every cluster with some weight between 0 and 1
  - Weights must sum to 1
  - Probabilistic clustering has similar characteristics
- ◇ Partial versus complete
  - In some cases, we only want to cluster some of the data
- ◇ Heterogeneous versus homogeneous
  - Clusters of widely different sizes, shapes, and densities

4.2 Types of Clustering

- ◇ **Well-separated** clusters
  A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster
- ◇ **Center-based** clusters

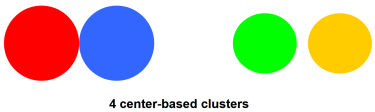


Fig. 4.3: Center-based clusters

  - A cluster is a set of objects such that an object in a cluster is closer (more similar) to the "center" of a cluster, than to the center of any other cluster
  - The center of a cluster is often a centroid, the average of all the points in the cluster, or a medoid, the most "representative" point of a cluster

- ◇ **Contiguous** clusters (Nearest neighbor or Transitive)
  - Each point is closer to at least one point in its cluster than to any point in another cluster.
  - Graph based clustering
  - This approach can have trouble when noise is present since a small bridge of points can merge two distinct clusters

Fig. 4.3: Contiguity-based clusters

- ◇ **Density-based** clusters

  - A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
  - Used when the clusters are irregular or intertwined, and when noise and outliers are present.
  - Points which are not classfied as part of any cluster are considered noise. In the figure they may be the gray blackground points.

Fig. 4.4: Density-based clusters

◇ **Property or Conceptual** clusters

◇ **Described by an Objective Function** clusters

  - Finds clusters that minimize or maximize an objective function.
  - Enumerate all possible ways of dividing the points into clusters and evaluate the 'goodness' of each potential set of clusters by using the given objective function.
  - NP Hard)
  - Can have global or local objectives.
    - Hierarchical clustering algorithms typically have local objectives
    - Partitional algorithms typically have global objectives

4.3 Similarity

- ◇ Similarity
  - – Numerical measure of how alike two data objects are.
  - – Is higher when objects are more alike.
  - – Often falls in the range [0,1]
- ◇ Dissimilarity
  - – Numerical measure of how different are two data objects
  - – Lower when objects are more alike
  - – Minimum dissimilarity is often 0
  - – Upper limit varies
- ◇ Proximity refers to a similarity or dissimilarity

4.3.1 Similarity and Dissimilarity for Different Attribute Types

Table 4.1: Similarity and dissimilarity for simple attributes

Attribute Type	**Dissimilarity**	**Similarity**
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal	$d = \frac{\lvert p-q \rvert}{n-1}$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - \frac{\lvert p-q \rvert}{n-1}$
Interval or Ratio	$d = \lvert p - q \rvert$	$s = -d$, $s = \frac{1}{1+d}$ or $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

p and q are the attribute values for two data objects.

4.3.2 Euclidean Distance

The most common way to measure distance between two data objects is the Euclidean distance:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

where n is the number of dimensions (attributes) and x_k and y_k are, respectively, the k^{th} attributes (components) or data objects $\mathbf{x}$ and $\mathbf{y}$. Standardization is necessary, if scales differ.

- ◇ Standardization is necessary, if scales differ.

4.3.2.1 Minkowski Distance

Minkowski Distance is a generalization of Euclidean Distance. r is a parameter that defines the type of distance, n is the number of dimensions (attributes) and x_k and y_k are, respectively, the k^{th} attributes (components) or data objects $\mathbf{x}$ and $\mathbf{y}$.

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{\frac{1}{r}}$$

- ◇ For $r = 1$, it is the Manhattan distance (or city block/Hamming distance)
- ◇ For $r = 2$, it is the Euclidean distance
- ◇ For $r \to \infty$, it is the supremum distance (or Chebyshev distance)

Metrics and Similiraties

1. $d(x, y) \geq 0$ for all x and y and $d(x, y) = 0$ only if $x = y$. (Positive definiteness)
2. $d(x, y) = d(y, x)$ for all x and y. (Symmetry)
3. $d(x, z) \leq d(x, y) + d(y, z)$ for all points x, y, and z (Triangle Inequality).

A *distance* d is a *metric* if it satisfies the three conditions above.

1. $s(x, y) = 1$ (or maximum similarity) only if $x = y$.
2. $s(x, y) = s(y, x)$ for all x and y. (Symmetry)

These instead are properties of a *similarity* measure.

4.3.3 Binary Similarity

Computing similarity among objects described by binary attributes is slightly different from numerical attributes.

◇ Simple Matching Coefficient (SMC)

$$SMC = \frac{\texttt{\#matches}}{\texttt{\#attributes}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

◇ Jaccard Coefficient

$$J = \frac{\texttt{\#11matches}}{\texttt{\#non-zero attribute values}} \frac{f_{11}}{f_{11} + f_{10} + f_{01}}$$

$$\begin{aligned}
p &= 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
q &= 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1 \\
f_{01} &= 2 \text{ (\#attributes where } p \text{ was 0 and } q \text{ was 1)} \\
f_{10} &= 1 \text{ (\#attributes where } p \text{ was 1 and } q \text{ was 0)} \\
f_{00} &= 7 \text{ (\#attributes where } p \text{ was 0 and } q \text{ was 0)} \\
f_{11} &= 0 \text{ (\#attributes where } p \text{ was 1 and } q \text{ was 1)} \\
SMC &= \frac{0+7}{10} = 0.7 \\
J &= \frac{0}{0+1+2} = 0
\end{aligned}$$

4.3.4 Cosine Similarity

Used often in text mining, where each dimension corresponds to a term (word) and the value of the dimension is the frequency of the term in the document.

If d_1 and d_2 are two document vectors, then

$$\cos(d_1, d_2) = (d_1 \cdot d_2)/||d_1||||d_2||$$

where $\cdot$ indicates vector dot product and $||d||$ is the length of vector d.

A "document vector" is a representation of a document as a vector in a multi-dimensional space, where each dimension corresponds to a unique term (word) from the document corpus. The value of each dimension is typically the frequency of the term in the document, or it can be weighted using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).

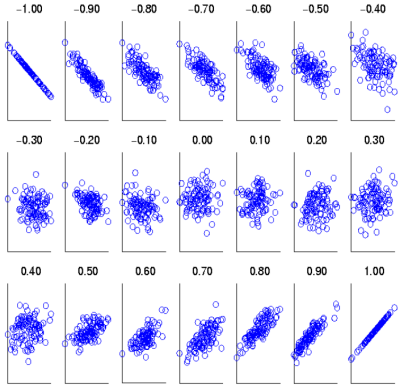
In BERT embeddings, cosine similarity is often used to measure the similarity between two text embeddings, which are high-dimensional vectors representing the semantic content of the texts.

It's called cosine similarity because it measures the cosine of the angle between two vectors in a multi-dimensional space. If the angle is 0 degrees (i.e., the vectors point in the same direction), the cosine similarity is 1, indicating maximum similarity. If the angle is 90 degrees (i.e., the vectors are orthogonal), the cosine similarity is 0, indicating no similarity. If the angle is 180 degrees (i.e., the vectors point in opposite directions), the cosine similarity is -1, indicating maximum dissimilarity.

Example:

$$
\begin{aligned}
d_1 &= 3\ 2\ 0\ 5\ 0\ 0\ 0\ 2\ 0\ 0 \\
d_2 &= 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 2 \\
d_1 \cdot d_2 &= 3*1+2*0+0*0+5*0+0*0+0*0+0*0+2*1+0*0+0*2=5 \\
||d_1|| &= \sqrt{3^2+2^2+0^2+5^2+0^2+0^2+0^2+2^2+0^2+0^2} = \sqrt{42} \\
||d_2|| &= \sqrt{1^2+0^2+0^2+0^2+0^2+0^2+0^2+1^2+0^2+2^2} = \sqrt{6} \\
\cos(d_1,d_2) &= \frac{5}{\sqrt{42}*\sqrt{6}} = \frac{5}{\sqrt{252}} \approx 0.315
\end{aligned}
$$

4.3.5 Correlation



Correlation measures the linear relationship between objects (binary or continuous). To compute correlation, we standardize data objects, p and q, and then take their dot product (covariance/standard deviation).

Fig. 4.4: Scatter plots showing the similarity from -1 to 1

$$
corr(\mathbf{x},\mathbf{y}) = \frac{covariance(\mathbf{x},\mathbf{y})}{stddev(\mathbf{x}) * stddev(\mathbf{y})} = \frac{\sum_{k=1}^{n}(x_k-\bar{x})(y_k-\bar{y})}{\sqrt{\sum_{k=1}^{n}(x_k-\bar{x})^2}\sqrt{\sum_{k=1}^{n}(y_k-\bar{y})^2}}
$$

4.3.6 Entropy

Information relates to possible outcomes of an event such as transmission of a message, flip of a coin, or measurement of a piece of data. The amount of information is inversely related to the probability of an event. Entropy is the commonly used measure of information content. For a discrete random variable X with n possible values $x_1, x_2, ..., x_n$ each having probability $p_1, p_2, \ldots, p_n$, the entropy $H(X)$ is defined as:

$$
H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)
$$

Entropy is measured in bits and is $0 \leq H(X) \leq \log_2(n)$. Thus, entropy is a measure of how many bits it takes to represent an observation of X on average.

The information one variable provides about another is called mutual information. The mutual information $I(X;Y)$ of two discrete random variables X and Y is defined as:

$$
I(X,Y) = H(X) + H(Y) - H(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x,y)
$$

where $H(X,Y)$ is the joint entropy of X and Y, and $p(x,y)$ is the probability that x and y occur together.

4.4 K-Means

- ◇ *Partitional* clustering approach
- ◇ The number of clusters K, must be specified
- ◇ Each cluster is associated with a **centroid** (center point), typically being the mean of the points in the cluster
- ◇ Each point is assigned to the cluster with the closest centroid. This may measured as Euclidean distance, cosine similarity, correlation, etc.
  Using these measures makes K-Means to converge in the first few iterations.

- ◇ Complexity is $O(nKId)$, where n is the number of data points, K is the number of clusters, I is the number of iterations, and d is the number of dimensions (attributes)
- ◇ The basic algorithm is very simple

Algorithm 1 K-Means Clustering Algorithm

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

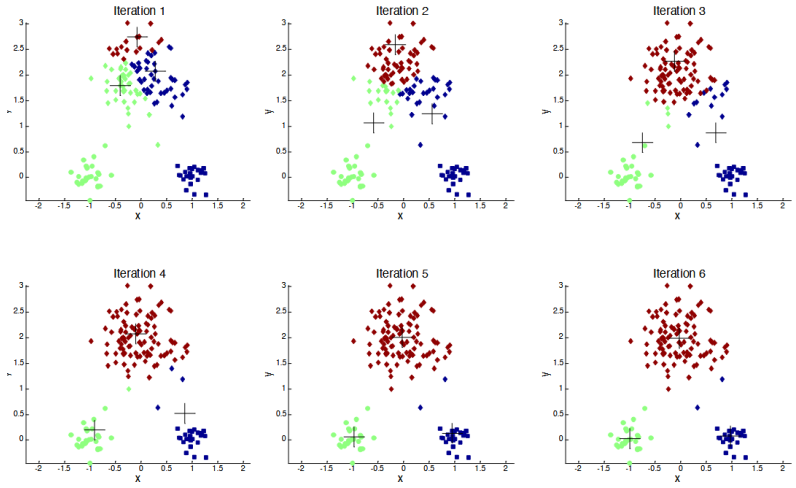


Fig. 4.5: K-Means Clustering Iterations

4.4.1 Evaluating K-Means clusters

The most common measure to evaluate the quality of K-Means clusters is the **Sum of Squared Errors** (SSE), also known as **Inertia**. It quantifies how tightly the data points in a cluster are grouped around their centroid. The SSE is calculated as follows:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

- ◇ K is the number of clusters,
- ◇ C_i is the set of points in cluster i,
- ◇ x is a data point in cluster C_i,
- ◇ μ_i is the centroid of cluster C_i,
- ◇ $||x - \mu_i||^2$ is the squared Euclidean distance between point x and centroid μ_i.

4.4.2 Limitations of K-Means

K-means has problems when clusters are of differing

- ◇ Clusters have different sizes
- ◇ Clusters have different densities
- ◇ Clusters have Non-globular shapes
- ◇ The data contains **outliers**.

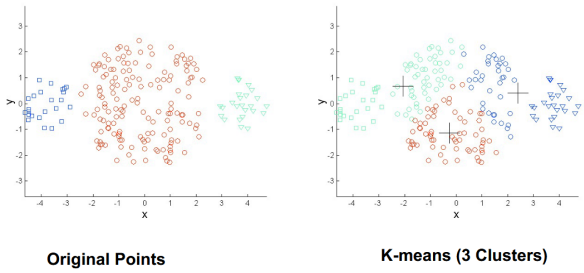


Fig. 4.6: K-Means with different cluster sizes
A solution may be to use more clusters, hence increasing K, and then merging them.

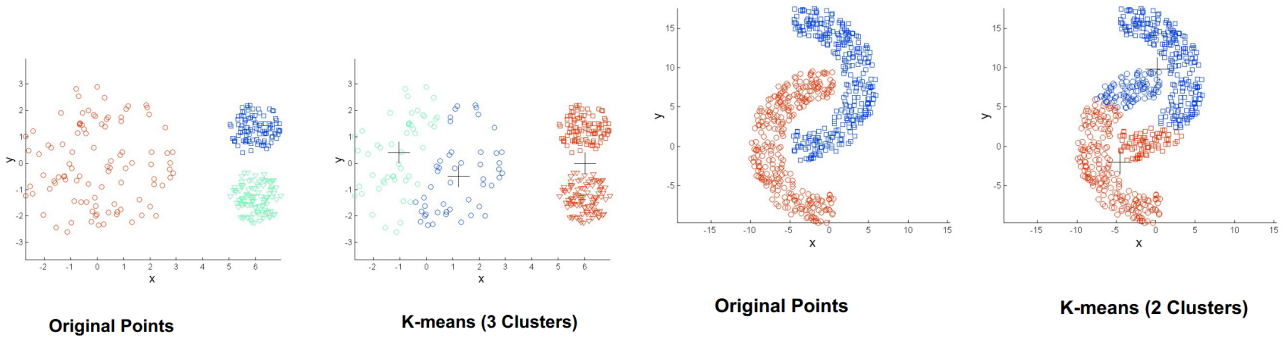


Fig. 4.6: K-Means with different cluster densities and non-globular shapes

4.4.2.1 Empty Clusters

K-Means sometimes produces empty clusters. This happens when no points are assigned to a cluster during the assignment step. This can occur if the initial centroids are poorly chosen or if the data distribution is such that certain centroids end up being too far from any data points.

Solutions include the following strategies, which may be iterated until no empty clusters remain:

- ◇ Choose a point and assign it to the cluster
  - – Choose the point that contributes most to SSE
  - – Choose a point from the cluster with the highest SSE

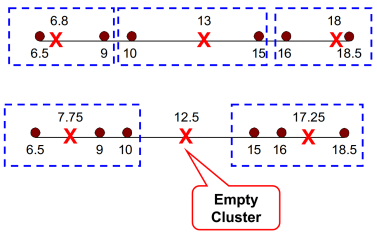


Fig. 4.7: K-Means with Empty Clusters

4.4.3 Workflow

1. Data Preprocessing
   - ◇ Handle missing values
   - ◇ Normalize/standardize data
2. Choose the number of clusters K
3. Post processing and validation
   - ◇ Eliminate small clusters that may represent outliers
   - ◇ Split 'loose' clusters, i.e., clusters with relatively high SSE
   - ◇ Merge clusters that are 'close' and that have relatively low SSE
   - ◇ Can use these steps during the clustering process
     `ISODATA`

4.4.4 Choosing the number of clusters K

If there are K "real" clusters then the chance of selecting one centroid from each cluster is small, especially if K is large.

A solution is to run K-Means multiple times with different initial centroids and choose the best result, but probability of getting a good result is still small.

A different approach is to use **hierarchical clustering** to determine initial centroids for K-Means.

Another option is to **incrementally** update centroids as points are assigned to clusters, instead of waiting until all points are assigned. This is more expensive and introduces order dependence.

4.4.5 Convergence of K-Means

We can define a goodness measure of a cluster c (hence of the set of Clusters C) as the SSE from the cluster centroid:

$$SSE(c, s_c) = \sum_{i=1}^{n} (d_i, s_c)^2 \tag{4.1}$$

$$G(C, s) = \sum_{c \in C} SSE(c, s_c) \tag{4.2}$$

where s is the set of centroids, s_c is the centroid of cluster c, d_i is a point in cluster c, and C is the set of clusters. The objective of K-Means is to minimize $G(C, s)$, so actually it is more of a *cost* rather than a *goodness* measure.

Re-assignment of points to clusters and re-computation of centroids is guaranteed to *not increase* (or *monotonically decreases*) $G(C, s)$, hence K-Means converges to a local minimum.

At each iteration, given $G(C, s)$, K-Means performs the following two steps:
1. Fix s, optimize $C \rightarrow$ **Assignment step**: assign d_i to the closest centroid $\Rightarrow G(C', s) \leq G(C, s)$
2. Fix C', optimize $s \rightarrow$ **Update step**: recompute centroids $\Rightarrow G(C', s') \leq G(C', s) \leq G(C, s)$

In this way the new cost results smaller than the original one, leading to convergence to a local minimum.

Recall that K-Means stops when relatively few points change clusters (or after a fixed number of iterations), i.e. when the first (assignment) step involves only few points, because typically the major decrease (convergence) happens in the first few iterations.

4.5 Hierarchical Clustering

Hierarchical clustering produces a set of nested clusters organized as a hierarchical tree can be visualized as a dendrogram, which is a tree-like diagram that records the sequences of merges or splits.

We do not have to assume any particular number of clusters, any desired number of clusters can be obtained by "cutting" the dendrogram at the proper level, each possibly corresponding to meaningful taxonomies.

4.5.1 Agglomerative vs Divisive

There are two types of hierarchical clustering, both exploiting a similarity or distance matrix:

- ◇ **Agglomerative** (bottom-up)
  Start with the points as individual clusters, and, at each step, merge the closest pair of clusters until only one cluster (or k clusters) remain.
- ◇ **Divisive** (top-down)
  Start with all points in one cluster and, at each step, split the cluster into smaller clusters until each point is its own cluster (or k clusters are formed).

Algorithm 2 Agglomerative Hierarchical Clustering Algorithm

```
1: Compute the proximity matrix
2: Let each data point be a cluster
3: repeat
4:     Merge the two closest clusters
5:     Update the proximity matrix
6: until only a single cluster remains
```

Proximity matrix is a square matrix that contains the distances (or similarities) between each pair of data points. "Proximity" is just a general term that can refer to either distance or similarity, depending on the context.

4.5.1.1 Updating Proximity Matrix

When two clusters c_i and c_j are merged into a new cluster c_k, we need to update the proximity matrix to reflect the distances between the new cluster c_k and all other existing clusters. There are several methods to do this, each defining the distance between clusters differently:

- **Single Linkage** (or Nearest Neighbor) The distance between two clusters is defined as the minimum distance between any single point in the first cluster and any single point in the second cluster.
$$d(c_i, c_j) = \min_{x \in c_i, y \in c_j} d(x, y)$$
- **Complete Linkage** (or Farthest Neighbor) The distance between two clusters is defined as the maximum distance between any single point in the first cluster and any single point in the second cluster.
$$d(c_i, c_j) = \max_{x \in c_i, y \in c_j} d(x, y)$$
- **Average Linkage** The distance between two clusters is defined as the average distance between all pairs of points, one from each cluster.
$$d(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{x \in c_i} \sum_{y \in c_j} d(x, y)$$
Note that **Single**, **Complete** and **Average** Linkage have all the same complexity! $O(n^3)$, where n is the number of data points.
However, we could use partial updates and intermediate results to significantly reduce the time complexity, by storing for each cluster the average, and minimum/maximum distances to other clusters. Especially for the first iterations when clusters are small, this can lead to significant space usage.
- **Centroid Linkage** The distance between two clusters is defined as the distance between their centroids.
$$d(c_i, c_j) = d(\mu_i, \mu_j)$$
where μ_i and μ_j are the centroids of clusters c_i and c_j, respectively.
- **Ward's Method** This method minimizes the total within-cluster variance. When two clusters are merged, the increase in the total within-cluster variance is calculated, and the pair of clusters that results in the smallest increase is merged.
$$d(c_i, c_j) = \frac{|c_i||c_j|}{|c_i| + |c_j|} ||\mu_i - \mu_j||^2$$
where $|c_i|$ and $|c_j|$ are the sizes of clusters c_i and c_j, and μ_i and μ_j are their centroids.
This may be used to initialize K-Means centroids, and it's the K-Means analogue of hierarchical clustering.

4.5.2 Divisive Hierarchical Clustering

First of all, build MST (Minimum Spanning Tree) of the data points, then iteratively remove the longest edge in the MST to split the data into clusters.
Start with a single cluster which consists of any point, and at each step, look for the closest pair of points p, q such that p is in the current tree while q is not. Add q to the tree and the edge (p, q) to the MST. Repeat until all points are in the tree.

Algorithm 3 Divisive Hierarchical Clustering Algorithm

1: Compute a MST for the proximity graph
2: **repeat**
3: Create a new cluster by breaking the link corresponding to the longest edge in the MST (smallest proximity)
4: **until** only singleton clusters remain

4.5.3 Complexity and Limitations

- Time Complexity - $O(n^3)$ for naive implementations of hierarchical clustering
  - $O(n^2)$ to compute the proximity matrix
  - $O(n)$ to find the closest pair of clusters
  - $O(n)$ to update the proximity matrix
  - Repeated n times

  More efficient algorithms can achieve $O(n^2 \log n)$.
- Space Complexity - $O(n^2)$ for storing the proximity matrix

4.5.4 Limitations

Once a decision is made to combine two clusters, it cannot be undone.
No global objective function is directly minimized.
Different schemes have problems with one or more of the following:

- ◇ Sensitivity to noise and outliers
- ◇ Difficulty handling clusters of different sizes and non- globular shapes
- ◇ Breaking large clusters

4.6 Density based - DBSCAN

Definition 4.1 (Density) ***Density*** *is the number of points within a specified radius (Eps or ϵ).*

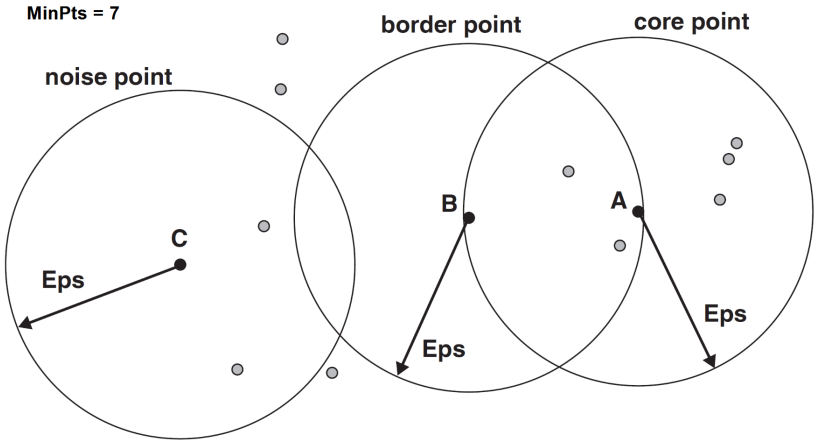


Fig. 4.7: Core, Border, and Noise Points in DBSCAN

A point is a **core point** if it has at least MinPts points within distance Eps (including itself). These are points which are at the interior of a cluster.
A point is a **border point** if it has fewer than MinPts points within distance Eps, but is in the neighborhood of a core point. These are points which are on the edge of a cluster.
A point is a **noise point** (or outlier) if it is neither a core point nor a border point. These are points which are not part of any cluster.

DBSCAN

1. Label points as core, border, or noise based on thresholds R (radius of neighborhood) and `min_pts` (min number of neighbors)
   - ◇ Every point having at least `min_pts` points within distance R becomes a core point
2. Connect core points that are within R of each other, hence are neighbors, putting them in the same cluster
3. Associate border points to the nearest core point, hence to the same cluster as the core point, and remove noise.

Algorithm 4 DBSCAN Algorithm

```
current_cluster_label ← 0
for all core points do
    if the core point has no cluster label then
        current_cluster_label ← current_cluster_label + 1
        Label the current core point with cluster label current_cluster_label
    for all points in the Eps-neighborhood, except i^th the point itself do
        if the point does not have a cluster label then
            Label the point with cluster label current_cluster_label
```

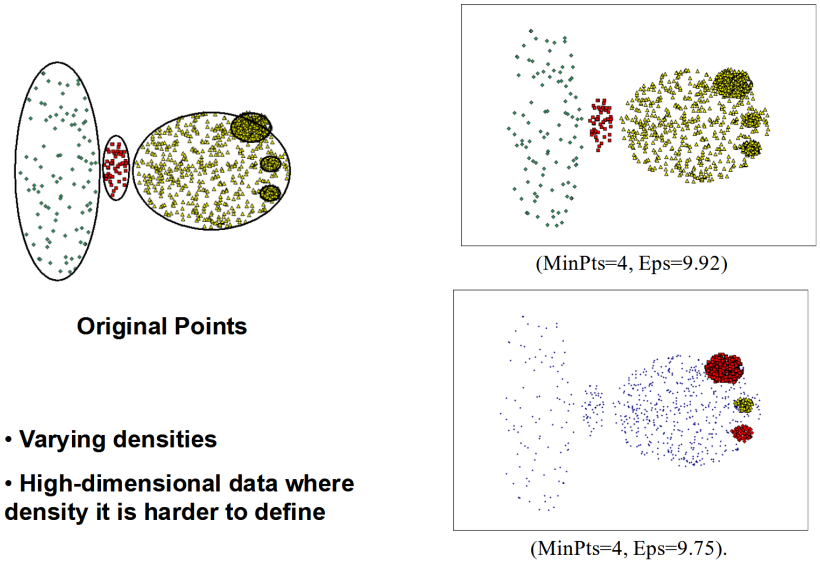


Fig. 4.8: DBSCAN Not behaving well in given situations

Determining Eps and MinPts

To determine Eps and MinPts, we can use the k-distance graph.

The idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance Noise points have the k^{th} nearest neighbor at distance.

So, plot sorted distance of every point to its k^{th} nearest neighbor.

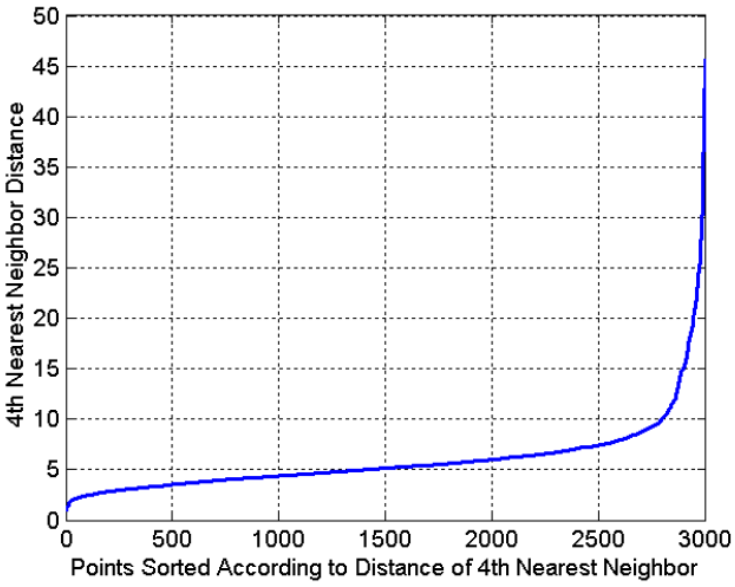


Fig. 4.9: We have 3000 points (instances), we can see that we have a few noise points (~ 300) because their distance from the 4^{th} is way higher. Here $k = 4$ looks optimal but also for $k \neq 4$ we could have almost the exact curve shape, as the 5^{th} or 6^{th} nearest neighbor could be way further than the 4^{th} as well as pretty close to it, we can't know judjing only from this graph.

Chapter 5

Cluster Validity

For supervised classification we have a variety of measures to evaluate how good our model is. such as accuracy, precision, recall, etc…

" For cluster analysis, the analogous question is how to evaluate the "goodness" of the resulting clusters? " —

"Clusters are in the eye of the beholder", but we still want to evaluate them to avoid finding patterns in noise, compare clustering algorithms, or to compare clusters or sets of clusters.

5.1 Towards cluster validation

1. Determining the **clustering tendency** of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
3. Evaluating how well the results of a cluster analysis fit the data without reference to external information.
4. Comparing the results of two different sets of cluster analyses to determine which is better
5. Determining the "correct" number of clusters.

Numerical measures that are applied to judge various aspects of cluster validity, are classified into the following three types:

- ◇ **External** Index: Used to measure the extent to which cluster labels match externally supplied class labels.
  Entropy
- ◇ **Internal** Index: Used to measure the goodness of a clustering structure without respect to external information.
  Sum of Squared Error (SSE)
- ◇ **Relative** Index: Used to compare two different clusterings or clusters.
  Often an external or internal index is used for this function, e.g., SSE or entropy

Sometimes these are referred to as ***criteria*** instead of *indices*; however, sometimes criterion is the general strategy and index is the numerical measure that implements the criterion.

5.1.1 Measuring validity through correlation

Two matrices:

- ◇ Proximity matrix
- ◇ Ideal similarity matrix
  - One row and one column for each data point
  - An entry is 1 if the associated pair of points belong to the same cluster
  - An entry is 0 if the associated pair of points belongs to different clusters

Then we compute the correlation between the two matrices. Note that the correlation will be between -1 and 0. This happens because the proximity matrix contains distances (the smaller the distance, the more similar the points are), while the ideal similarity matrix contains similarities (the larger the similarity, the more similar the points are). Therefore, we expect a negative correlation between the two matrices.
The closer the correlation is to -1, the better the clustering, because, points which are far apart and are in different

clusters have inversely correlated values (high distance, low —0 actually— similarity), while points which are close together and are in the same cluster have also inversely correlated values (low distance, high similarity).

note that since the matrices are symmetric only the correlation between $n(n-1)/2$ entries needs to be calculated.

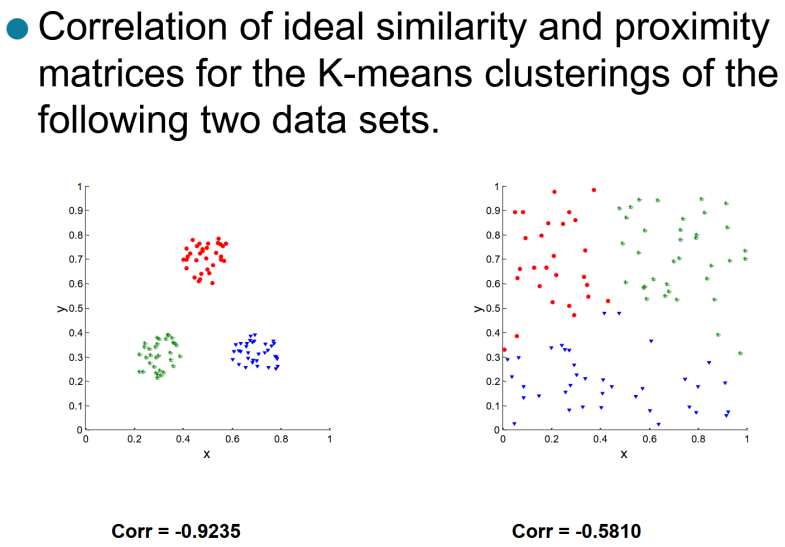


Fig. 5.1: Correlation of two sets. The first set has an almost perfect clustering, and in fact enjoys high "inverse" correlation of -0.9235, while the second, performed on random data, has a correlation of 0.5810, indicating poor clustering, even though probably there ain't any real clusters in the data, so no real improvement is possible.
Actually, in real world data it's very rare to observe a -0.9235 correlation score.

5.1.2 Internal measures

5.1.2.1 SSE - Sum of Squared Error

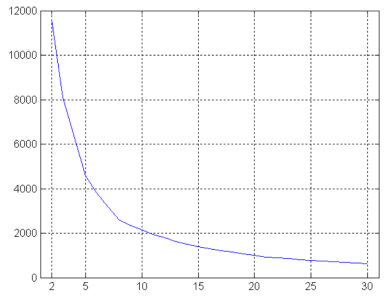


Fig. 5.2: SSE of cluster found using K-means with various values of k

SSE (Sum of Squared Error) is a common measure of cluster cohesion, defined as:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist^2(m_i, x)$$

where m_i is the centroid of cluster C_i.

This is a measure of how tightly the data points in a cluster are grouped together. Lower values of SSE indicate more cohesive clusters. It can be a nice measure which does not need external information, and may also be used to determine the appropriate number of clusters by plotting the SSE for different values of k and looking for an "elbow" in the graph.

5.1.2.2 Cohesion and Separation

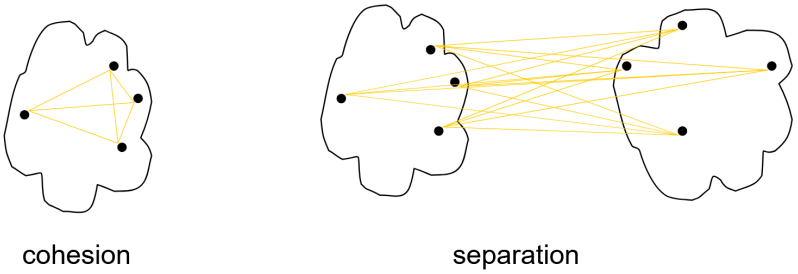


Fig. 5.3: Cohesion and separation

A proximity graph based approach can also be used for cohesion and separation.

- Cluster **cohesion** is the sum of the weight of all links within a cluster, it measures how closely related are objects in a cluster. (SSE is a measure of cohesion)

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

- Cluster **separation** is the sum of the weights between nodes in the cluster and nodes outside the cluster. Measures how distinct or well-separated a cluster is from other clusters. (Squared error between clusters is a measure of separation)

$$BSS = \sum_i |C_i|(m - m_i)^2 \qquad |C_i| \text{ size of cluster } C_i$$

Note that $BSS + WSS = TSS$ where TSS is the total sum of squares, and is constant and independent of the clustering.
Here m_i is the centroid of cluster C_i and m is the overall mean of the data points.

5.1.2.3 Silhouette coefficient

Silhouette coefficient combines ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings.

Consider a point i in cluster A:

- Calculate a = average distance of i to the points in its cluster
- Calculate b = min (average distance of i to points in another cluster)
  - Suppose there are clusters A, B, C, D and i is in cluster A
  1. Calculate average distance of i to points in B, call it d_B, do the same for C and D, obtaining d_C and d_D
  2. Then $b = \min(d_B, d_C, d_D)$
- The silhouette coefficient for a point is then given by

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

- Typically between 0 and 1. The closer to 1 the better.
- Can calculate the average silhouette coefficient for a cluster or a clustering.

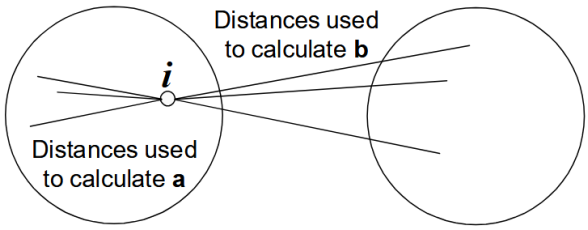


Fig. 5.4: Silhouette coefficient for i

5.1.3 External measures

External measures compare the clustering results to an externally known class labels. Examples may be **entropy** and **purity**.

Table 5.1: K-means Clustering Results for LA Document Data Set

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Entropy	Purity
1	3	5	40	506	96	27	1.2270	0.7474
2	4	7	280	29	39	2	1.1472	0.7756
3	1	1	1	7	4	671	0.1813	0.9796
4	10	162	3	119	73	2	1.7487	0.4390
5	331	22	5	70	13	23	1.3976	0.7134
6	5	358	12	212	48	13	1.5523	0.5525
Total	354	555	341	943	273	738	1.1450	0.7203

5.1.3.1 Entropy

Entropy measures how the various clusters are distributed with respect to the class labels.

For each cluster, the class distribution of the data is calculated first, i.e., for cluster j we compute p_{ij}, the "probability" that a member of cluster j belongs to class i as follows: $p_{ij} = m_{ij}/m_j$, where m_j is the number of values in cluster j and m_{ij} is the number of values of class i in cluster j.

Then using this class distribution, the entropy of *each* cluster j is calculated using the standard formula:

$$e_j = -\sum_{i=1}^{L} p_{ij} \log_2 p_{ij}$$

where L is the number of classes.

The total entropy for a set of clusters is calculated as the sum of the entropies of each cluster weighted by the size of each cluster, i.e.:

$$e = \sum_{i=1}^{K} \frac{m_i}{m} e_j$$

where m_j is the size of cluster j, K is the number of clusters, and m is the total number of data points.

5.1.3.2 Purity

Using the terminology derived for entropy, the purity of cluster j, is given by:

$$purity_j = \max_i p_{ij}$$

and the overall purity of a clustering by:

$$purity = \sum_{j=1}^{K} \frac{m_j}{m} purity_j$$

5.1.4 Statistics to evaluate metrics

" Suppose you get a 0.6 silhouette score for a clustering. Is that good or bad? " — Data miner

To answer this question, we can use **statistical** hypothesis testing. The idea is to compare the observed silhouette score to a distribution of silhouette scores obtained on random data with no inherent cluster structure.
If the observed silhouette score is significantly higher than the scores from random data, we can conclude that the clustering is meaningful.

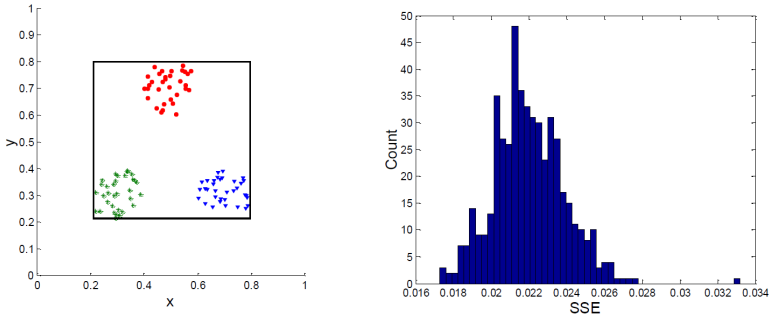


Fig. 5.4: Histogram shows SSE of three clusters in 500 sets of random data points of size 100 distributed over the range $[0.2, 0.8]$ for x and y values. The clustering on the left has SSE of 0.005, which is significantly lower than the random data —in fact it is out of scale—, indicating a good clustering.

Instead if we have two clustering results, one yielding 0.6 silhouette score and another yielding 0.8 silhouette score, we can easily conclude that the second clustering is better, since it has a higher silhouette score. We do not want to assert whether the value is *good* in an "absolute" way, but rather whether it is *better* than another clustering result.

Chapter 6

K-Means

6.1 Bisecting K-Means

Instead of partitioning the data set into K clusters in each iteration, bisecting k-means algorithm splits one cluster into two sub clusters at each bisecting step (by using k-means) until k clusters are obtained.

Algorithm 5 Bisecting K-means algorithm

1: Initialize the list of clusters to contain the cluster consisting of all points.
2: **repeat**
3: Remove a cluster from the list of clusters.
4: {Perform several "trial" bisections of the chosen cluster.}
5: **for** $i = 1$ to $number\_of\_trials$ **do**
6: Bisect the selected cluster using basic K-means.
7: Select the two clusters from the bisection with the lowest total SSE.
8: Add these two clusters to the list of clusters.
9: **until** Until the list of clusters contains K clusters.

The algorithm is exhaustive terminating at singleton clusters (unless K is known)

- ◇ Note that Terminating at singleton clusters
  - – Is time consuming
  - – Singleton clusters are meaningless
  - – Intermediate clusters are more likely to correspond to real classes
  - – No criterion for stopping bisections before singleton clusters are reached

The resulting clusters can be refined by using their centroids as the initial centroids for the basic K- means.

6.2 X-Means

X-Means clustering algorithm is an extended K-Means which tries to automatically determine the number of clusters based on `BIC` scores.

The X-Means goes into action after each run of K-Means, making local decisions about which subset of the current centroids should split in order to better fit the data.

The splitting decision is done by computing the *Bayesian Information Criterion* (`BIC`).

6.2.0.1 Bayesian Information Criterion

- ◇ A strategy to stop the Bisecting algorithm when meaningful clusters are reached to avoid over-splitting
- ◇ Using BIC as splitting criterion of a cluster in order to decide whether a cluster should split or no
- ◇ BIC measures the improvement of the cluster structure between a cluster and its two children clusters.
- ◇ Compute the BIC score of:

  - A cluster
  - Two children clusters
- ◇ BIC approximates the probability that the Mj is describing the real clusters in the data

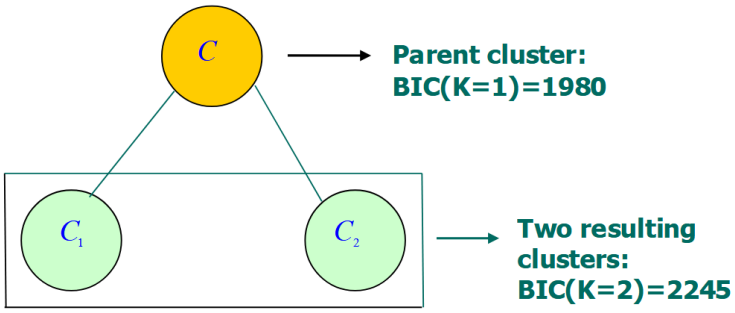


Fig. 6.1: BIC splitting
The BIC score of the parent cluster is less than BIC score of the generated cluster structure ⇒ we accept the bisection

Forward search for the appropriate value of k in a given range $[r_1, r_{max}]$; we recursively split each cluster and use BIC score to decide if we should keep each split.

1. Run K-means with k=r1
2. Improve structure
3. If k > rmax Stop and return the best-scoring model

- ◇ Use local BIC score to decide on keeping a split
- ◇ Use global BIC score to decide which K to output at the end

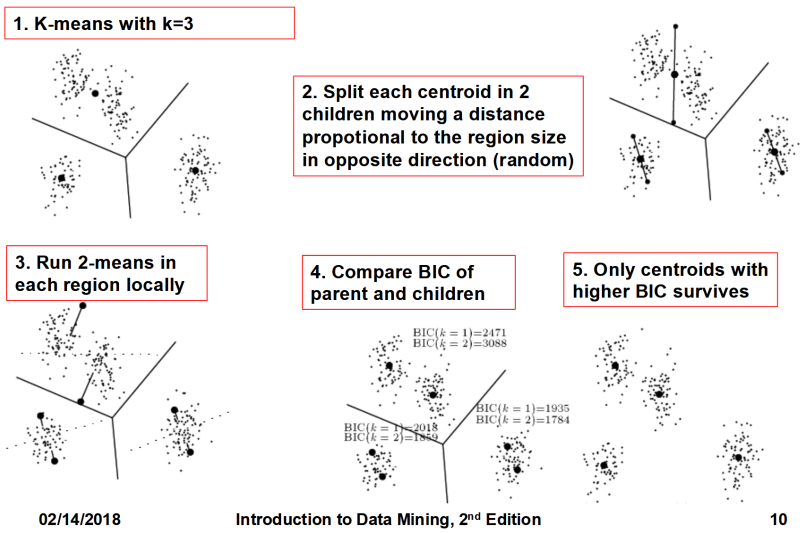


Fig. 6.2: X-Means process

6.3 Mixture Models and the EM Algorithm

A **mixture model** is a probabilistic model for representing the presence of subpopulations within an overall population, without requiring that an observed data set should identify the subpopulation to which an individual observation belongs. The model assumes that the data is generated from a mixture of several distributions, each representing a different subpopulation.

Algorithm 6 EM algorithm

1: Select an initial set of model parameters.
2: (As with K-means, this can be done randomly or in a variety of ways.)
3: **repeat**
4: **Expectation Step** For each object, calculate the probability that each object belongs to each distribution, i.e., calculate $prob(distribution\_j|\mathbf{x}_i, \Theta)$.
5: **Maximization Step** Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
6: **until** The parameters do not change.
7: (Alternatively, stop if the change in the parameters is below a specified threshold.)

Chapter 7

Anomaly Detection

We may refer to anomalies in data also as *outliers.*

7.1 Outliers

- ◇ Inherently **fuzzy** - An instance has a degree of outlierness, which we can threshold to decide whether an instance is an outlier or not.
- ◇ **Data-dependent** - Outlier are exceptions to the data. But outliers themselves define the data...?
- ◇ **Not noise** - Noise is random, outliers are exceptional.
- ◇ **Mono/multi-dimensional** - An outlier can be so on just one dimension, or on multiple.

Outliers are something that is either **unusual** or **extreme**, or both.
Outliers are, by nature, defined in terms of other instances. Whatever approach we use to detect them, we should take into account that they influence it as well.

7.2 Outlier Detection Algorithms

Algorithms used to detect outliers usually involve two key steps:

1. **Grading** - Define a grading function $\tilde{o}$ that assigns to each instance x a degree of outlierness/anomaly $\tilde{o}(x)$.
2. **Thresholding** - Decide on a threshold $\hat{o}$ such that instances with $\tilde{o}(x) > \hat{o}$ are considered outliers.

We can categorize outlier detection algorithms depending on the *axis* on which they operate:

- ◇ **Locality** - is the outlier detection performed in a local neighborhood (local) or considering the entire dataset (global)?
- ◇ **Sensitivity** - is the outlier detection sensitive to the presence of other outliers (sensitive) or not (robust)?
- ◇ **Interpretability** - can we interpret/explain why an instance is considered an outlier (interpretable) or not (black-box)?

Locality	*Global/Local*
Sensitive	*Robust/Sensitive*
Interpretable	*Black-box/Interpretable*

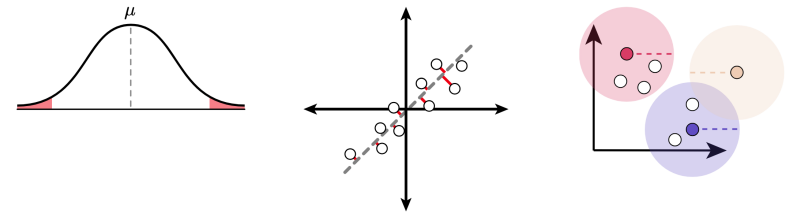


Fig. 7.1: Distribution, manifold, neighborhood

7.2.1 Distributions

In case of a normal distribution $\mathcal{N}(\mu, \sigma)$, we can define outliers as instances that are more than k standard deviations away from the mean μ (see Figure 7.2).
This approach is **global**, **robust** and **interpretable**.

The degree of anomaly $\tilde{o}(x)$ can be defined as:

$$\text{z-score} = \tilde{o}(x) = \frac{|x - \mu|}{\sigma}$$

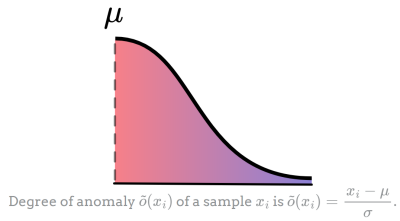


Fig. 7.2: For a normal distribution $\mathcal{N}(\mu, \sigma)$, we can define outliers as instances that are more than k standard deviations away from the mean μ.

7.2.1.1 Grubbs test

$z - scores$ generate sample-dependent outlier degrees $\tilde{o}(x_1), \tilde{o}(x_2), \ldots, \tilde{o}(x_n)$, but does not tackle the **+1 problem**.

The +1 problem is the problem of deciding whether the most extreme instance in a dataset is an outlier or not.
Grubb's test iterates over detected outliers, removing *one layer* of outliers at a time, until no more outliers are found.
The key point is that after removing outliers, the distribution parameters μ, σ are recomputed, thus changing the z-scores of the remaining instances. This is key to detect multiple outliers in a dataset.

Algorithm 7 Grubb's test for outliers

1: Find current outlier set $\hat{X}$
2: If $\hat{X} = \emptyset$, stop
3: $X = X \setminus \hat{X}$ - remove outliers from dataset
4: Go to step 1

Locality	**Global**
Sensitive	Outliers influence the distribution but may be removed by Grubb's test
Interpretable	**Black-box**, no clear explanation for outlierness, simply, there are not many similar instances.

Data may vary *locally*: subsets of the data each follow a different distribution.
Assumption: there exists a partition of the data, each block distributed according to a Normal distribution.
Thus we could use multiple models to model the data, and define outliers as instances that are outliers with respect to their local distribution.

One of k models $M_{\theta_0}, M_{\theta_1}, \ldots, M_{\theta_{k-1}}$ is sampled, each with a sampling probability m_i. Different distributions sample different regions of the density.

Locality	**Local**
Sensitive	Outliers influence the distribution, may be unstable, but may be removed by Grubb's test (?)
Interpretable	**Black-box**, no clear explanation for outlierness, simply, there are not many similar instances.

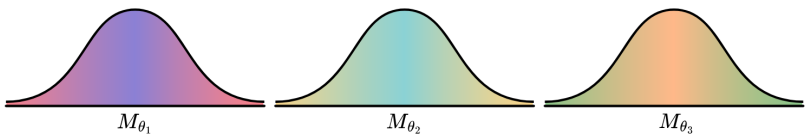


Fig. 7.2: A mixture of Normals $M_{\theta_0}, M_{\theta_1}, M_{\theta_2}$; each sampled with probability m_0, m_1, m_2, respectively.

7.2.2 Thresholding

Grubb's test

Choosing a threshold $\hat{o}$ is arbitrary, but there are algorithms such as Grubb's test which define their own thresholding mechanism. In Grubb's test, the threshold is defined as:

$$\hat{o} = n\frac{\sum(x_i - \bar{X})^4}{(\sum(x_i - \bar{X})^2)^2}$$

z-scores assume a Normal distribution, but often this is not the case. Yet, we can still identify tails of a distribution, and in turn, anomalies, by exploiting *Markov* and *Chebyshev* **inequalities**, that hold for any distribution.

- ◇ *Markov* inequality - for a variable (distribution)X with positive values and threshold β, it holds

$$P(X \geq \beta) \leq \frac{\mathbb{E}(X)}{\beta}$$

  Thus, given an estimate of the variable's expected value, we can retrieve the inverse of an image of its cumulative distribution
- ◇ *Chebyshev* inequality - for a variable (distribution) X with mean μ $(= \mathbb{E}(X))$ and standard deviation σ, and threshold β, it holds

$$P(|X - \mathbb{E}(X)| > \beta) \leq \frac{\sigma^2}{\beta^2}$$

  That is, the probability of deviation from the mean is inversely proportional to the deviation

7.2.3 Manifold

Distributional approaches define the density, but do not describe the data itself. $\tilde{o}$ is defined in terms of the manifold: does the given instance lie in the manifold? Just like the distributional approach, we must assume the manifold family. To preserve the interpretability of our results, we initially stick to linear manifolds.

We can define the anomaly degree $\tilde{o}(x)$ as the distance of x from the manifold.
We can use PCA to find the linear manifold that best fits the data.
PCA finds the directions of maximum variance in the data, and uses them as a new basis for the data.

A matrix A spans a linear space, thus every vector b in its spanned space is defined as a linear combination of $A : b = Ax$. For non fullrank matrices A, such a solution x may not exist. Thus, we need to project on the data manifold, to look for the closest point $\hat{b}$ to b that lies in the space spanned by A.

Least Squares

Least Squares is a method to find the best fitting solution to an overdetermined system of equations $Ax = b$ (more equations than unknowns).
The best fitting solution is the one that minimizes the residual sum of squares (RSS):

$$RSS = ||Ax - b||_2^2$$

Least squares assumes a linear manifold, and squared norm as distance metric.

The instability of least squares is due to the data collinearity. A possible solution: de-correlate the data! PCA does exactly that.

Some mathy examples are displayed in the lecture slides…

Locality	**Global**
Sensitive	Strongly influenced by outliers
Interpretable	Partial: which instances have lower degrees? What even is a "low" degree?

Table 7.1: Least squares

Manifold-based algorithms are as flexible as the defined manifold. Like with mixture models, neighbor-based approaches reintroduce locality: outliers are defined in function of their neighbors:

- ◇ **Connectivity** - An outlier is defined in terms of the connectivity to its neighbors
- ◇ **Concentration** - An outlier is defined in terms of its neighbor concentration

$$\begin{bmatrix} 5 & 11 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 54 & 27 & \cdots & 3 \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

In the above matrix we display connectivity as **postings**, rather than distances.
$A_{i,j}$ is the —row ID?— j^{th} neighbor of instance i, making each row ordered by increasing distance from i.

The underlying assumption is that an in instance is an inlier if it is well connected to its neighbors, and lies in a dense area of the space.
Each instance has a posting list of neighbors, from the closest to the farthest: the lower the aggregated position in other lists, the higher the connectivity degree.

- ◇ Posting position defines connectivity: it is not density
- ◇ Connectivity is asymmetric: I may be your closest instance, you may not be mine

7.2.3.1 Grading neighbors connectivity

Posting matrices are often used as a base on which to measure different indices of connectivity, e.g.,

- ◇ **hub** - instance x_i is at least the t^{th} neighbor of at least k instances.
  Definition used by ODIN: given a posting matrix A,x_i is a *hub* if it appears at least k times in the first t columns of A. Hence,x_i is an outlier if the opposite is true:

$$\tilde{o}(x_i) = \begin{cases} 1 & if \quad |i|i \in A_{\neq i,\leq t}| < k \\ 0 & otherwise \end{cases}$$

  That is, if x_i appears less than k times in the first t columns of A, it is an outlier.
  You must be at most the t^{th} neighbor of at least k instances to be considered a hub, hence, not an outlier.
- ◇ **popularity** - instance x_i is on average the t^{th} neighbor of at least k instances
  Given a posting matrix A, x_i is an outlier if, on average, is not less than the $t-th$ neighbor of other instances:

$$\hat{o}(x_i) = \frac{\sum_{l=0,l\neq i}^{n-1} \sum_{j=0}^{n-1} \mathbb{1}\{a_{l,j} = x_i\} l}{n-1} > t.$$

  where $\mathbb{1}\{a_{l,j} = x_i\}$ is an indicator function that equals 1 if $a_{l,j} = x_i$ and 0 otherwise, representing the position in the posting list.
- ◇ **ostracism** - instance x_i is at worst t^{th} neighbor of other k instances

Connectivity and concentration can be approximated through similar structures: we go from *postings* matrix to distance matrix! To ease notation, we use a row-sorted distance matrix A_γ, so that row i holds increasing distances from instance x_i.

$$A = \begin{bmatrix} 0 & 2.28 & 0.16 & 0.21 \\ 2.21 & 0 & 1.21 & 3.91 \\ 0.16 & 1.21 & 0 & 0.76 \\ 0.21 & 3.91 & 0.76 & 0 \end{bmatrix}$$

$$A_\gamma = \begin{bmatrix} 0.16 & 0.21 & 2.28 \\ 1.21 & 2.28 & 3.91 \\ 0.16 & 0.76 & 1.21 \\ 0.21 & 0.76 & 3.91 \end{bmatrix}$$

A distance matrix A (top), and its row-sorted version A_γ (bottom). First column of 0s trimmed from A_γ.
A_γ is also called the **reach matrix**, which will be discussed below.

7.2.4 Reach

An instance x has reach $\gamma^k(x)$ if the $k-th$ nearest neighbor is at distance γ^k, and average reach $\bar{\gamma}^k(x)$ if the average of $\{\gamma^1, \ldots, \gamma^k\}$ is $\bar{\gamma}^k(x)$.

Our row-sorted distance matrix A_γ is the *reach* matrix of the data! Indeed, A_γ defines both reach and average reach.

The reach of instance x_i is encoded in the row-sorted distance matrix A_γ as follows:

$$A_\gamma = \begin{bmatrix} \gamma^1(x_1) & \gamma^2(x_1) & \gamma^3(x_1) \\ \gamma^1(x_2) & \gamma^2(x_2) & \gamma^3(x_2) \\ \gamma^1(x_3) & \gamma^2(x_3) & \gamma^3(x_3) \end{bmatrix}$$

where $\gamma^j(x_i)$ denotes the j-th nearest neighbor of instance x_i.

For example, consider the following transformation:

$$A_\gamma \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \bar{\gamma}^1(x_1) & \bar{\gamma}^2(x_1) & \bar{\gamma}^3(x_1) \\ \bar{\gamma}^1(x_2) & \bar{\gamma}^2(x_2) & \bar{\gamma}^3(x_2) \\ \bar{\gamma}^1(x_3) & \bar{\gamma}^2(x_3) & \bar{\gamma}^3(x_3) \end{bmatrix}$$

A_γ explicitly encodes reach (A_γ itself) and average reach. Note that the left part of the equation above is a matrix multiplication! The matrix with the fractions is a **transformation matrix** that transforms reach into average reach, by assigning weights to each neighbor distance.

Average Reach as concentration

The key point is that the average reach defines an empirical approximate **concentration**, since it measures what's the average distance of the k nearest neighbors.

7.2.4.1 Reach ratio factor

Assumption: Inliers have lower reach than their neighbors. We formalize this in a reach (ratio factor:

$$\tilde{o}^k_{i,j} = \frac{\bar{\gamma}^k(x_i)}{\bar{\gamma}^k(x_j)}$$

which 1 is for pairs x_i, x_j with equal k-neighbors concentration, and > 1 for instances with different concentrations, x_i laying in a sparser area of the space. It can be also < 1 if x_i is in a denser area than x_j.

Local outlier factor generalizes outlier (*reach ratio*) factor by averaging the outlier factor over the neighbors of an instance:

$$\tilde{o}(x_i) = \sum_{x_j \in neigh(x_i)} \tilde{o}^k_{i,j}$$

" But...how many neighbors should we consider? " —

It matches the number of columns in the reach matrix A_γ, which is a parameter to be set by the user, which should be k (if I'm not mistaken).

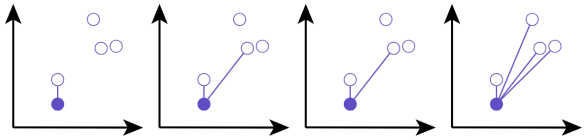


Fig. 7.3: Neighbors at different k values. Local outlier factor respects the postings matrix, as it creates *clusters* of neighbors.

Connectivity outlier factor (COF) is based on the idea that outliers are poorly connected to their neighbors. It is defined as:

$$\tilde{o}(x_i) = \sum_{x_j \in connect\_neigh(x_i)} \tilde{o}^k_{i,j}$$

The connected neighbors of an instance x_i is recursively defined as the 1-nearest neighbor to the last element in the chain.

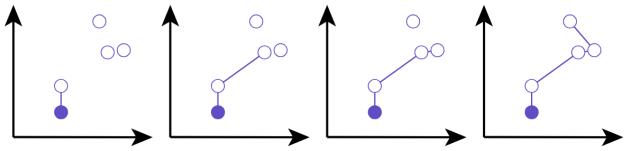


Fig. 7.4: Neighbors with different k values. Connectivity Outlier Factor does not respects the posting matrix. Rather, it creates *chains* of neighbors.

k-NN outlier factor (KOF) replaces the average reach at k (denoted with $\bar{\gamma}^k$) with the maximum reach at k (denoted with $\hat{\gamma}^k$):

$$\tilde{o}(x_i) = \gamma^k(x_i)$$

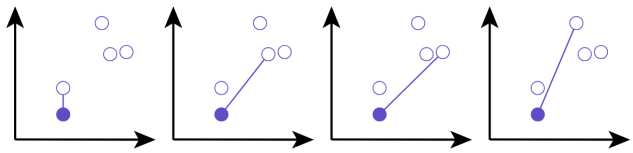


Fig. 7.5: Neighbors at different k values

7.2.5 Concentration

Reach degrees approximate space concentration with (inverse) reach. Rather than picking a k, we can swap in a more natural definition of concentration: instances found per unit of space. Even better, instances found within an hypersphere $B(\cdot, \varepsilon)$ of radius ε centered in the instance.

Clearly, the assumption is, again, that outliers are found in low-concentration areas of the space.

We compute concentration on a two-radii approach:

- ◇ **concentration radius** ε: determines the hyperspheres $B(x_i, \varepsilon)$ estimating concentration $c^\varepsilon(x_i)$ of x_i within a radius ε
- ◇ **neighborhood radius** δ: proportional to ε, determines the neighborhood B_i of x_i as the instances laying within $B(x_i, \delta)$

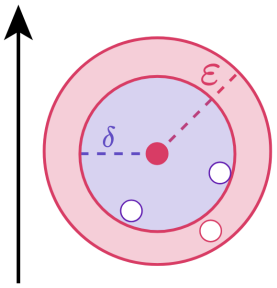


Fig. 7.6: The two radii ε, δ: the former is used to estimate *concentration*, the latter to choose which neighbors to compare concentration against. **Note:** δ may also be larger than ε!

Like reach-based concentration, degrees are defined on a basis of comparisons between some degree of an instance, and its neighbors:

$$\tilde{o}(x_i) = \bar{c}^\varepsilon(B_i) - c^\varepsilon(x_i), \quad \text{with} \quad \bar{c}^\varepsilon(B_i) = \frac{\sum_{x_j \in B_i} c^\varepsilon(x_j)}{|B_i|}$$

Typically we define concentration as a ratio with n the number of instances in the dataset but I guess we could also use $\max_i(|B_i|)$

$$c^\varepsilon(x_i) = \frac{x_j \in \mathcal{X} : d(x_i, x_j) \leq \varepsilon - 1}{n - 1}$$

Note that this is different from picking $x_j \in B_i$, as B_i considers δ rather than ε: $x_j \in \mathcal{X} : d(x_i, x_j) \leq \delta$.

that is, two-radii concentration compares the concentration of an instance, with the concentration of its neighbors. For

- $\tilde{o}(x_i) > 0$ neighbors have a higher concentration. x_i is in a sparse area, it could be an outlier!
- $\tilde{o}(x_i) \approx 0$ neighbors have a the same concentration
- $\tilde{o}(x_i) < 0$ neighbors have a lower concentration. x_i is in a dense area, likely not an outlier!

Locality	**Local**
Sensitive	Choice of neighborhood, connectivity parameter
Interpretable	Partial: can inspect what instances lead to different reaches

Table 7.2: Grading connectivity factors

7.2.6 Neighborhoods

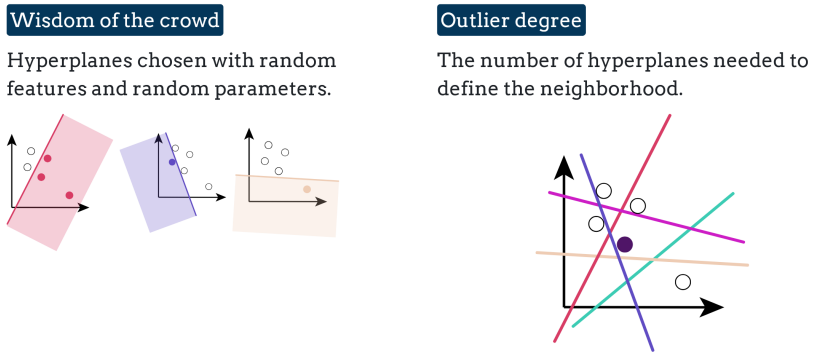


Fig. 7.3: Faster and easier to compute hyperplane-based neighborhoods.

An **isolation tree** t is a random tree which randomly partitions the space into a set of blocks (attributes, typically two at a time to get a binary tree). In the typical isolation trees the splits are univariate, that is, they split the space along one attribute at a time, so it follows a hyperplane orthogonal to the axis of the chosen attribute (see Figure 7.3). However in EIF (Extended Isolation Forests) splits can be multivariate, that is, they can split the space along hyperplanes not orthogonal to any axis, as they are displayed here

- Splits are sampled randomly
- Tree grows up to a predefined height, or until all leaves contain one instance

Outlier degree

$$\tilde{o}^t(x_i) = \frac{path(x_i, t)}{c}$$

where c defines the average path length in the tree, which in case of a binary tree, for instance, is $\log_2(n)$, with n number of instances. $path(x_i, t)$ is the length of the path from the root to the leaf containing x_i.
That is, the instance is an outlier it the distance from the root is **lower** than average.

Note that, differently from other approaches, here *lower* outlier degree means *higher* outlierness.

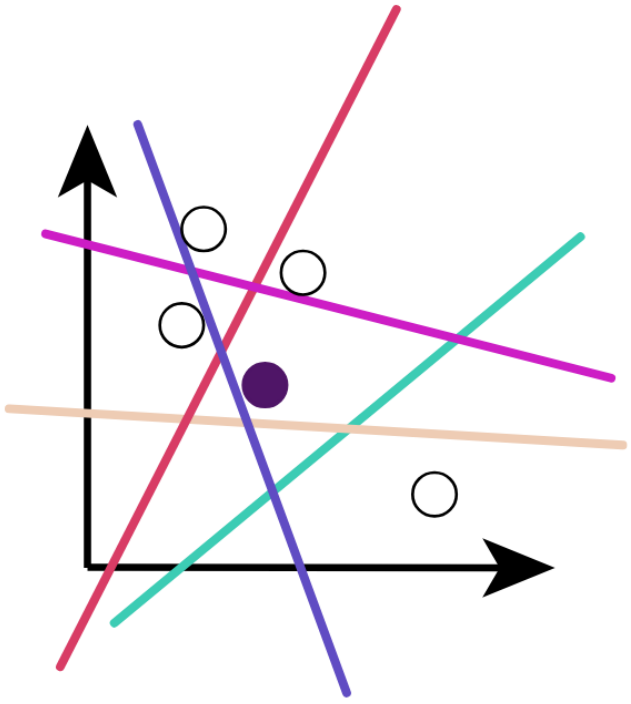


Fig. 7.4: Isolation tree

Consider Figure 7.4: every line (split) is a node in the isolation tree, which in fact splits the previous node in two. That is each split partitions the space in two half-spaces, and the tree grows downwards until each leaf contains exactly one instance —probably but not necessarily an outlier, depends on the length of the path—, or until a predefined height is reached. The path of the colored instance in Figure 7.4 is 5, as it is isolated by 5 splits (actually 2 well placed splits would be enough, but they are chosen randomly, so…); this means that in the tree, the path from the root to the leaf

containing the instance is 5 edges long.

Suppose there are 1000 instances in a very small portion of the plane, representing the main distribution, and one instance very far away from them. The far away instance will likely be isolated in just a few splits, as the random splits will likely isolate it quickly.
Isolating one of the 1000 instances will likely take *a lot* of splits, as the splits are random, and the instance is surrounded by many other instances, making it hard to isolate it quickly. Dense data would end up in a super deep leaf, potentially making less dense but still normal instances to be isolated *"relatively"* quickly, and thus misclassified as outliers. That's the reason why we set a maximum height for the tree, to have a cutoff point after which we stop splitting the space and we consider the instances in the leaf as normal.

Please note again that the outlier degree defined above is "inversed": the lower it is, the more likely the instance is an outlier.
The higher the path length, the higher the outlier degree, as the instance is likely not an outlier.

An isolation forest T is comprised of several isolation trees, further sampling the hyperplane space.
The key point is that we can combine multiple trees to get a more comprehensive outlier degree:

Outlier degree

$$\tilde{o}(x_i) = 2 - \frac{\sum_{t \in T} path(x_i, t)}{|T|c}$$

where $|T|$ is the number of trees in the forest.

Locality	**Global** and **local**
Sensitive	Dataset noise can be interpreted as outlier
Interpretable	One of the most interpretable. Splits are induced by the tree, if the tree is univariate.

Table 7.3: Grading Isolation Forests

Part III

Association Rules and Sequential Patterns

Chapter 8

Association Analysis

Association Rule Mining refers to, given a set of transactions, finding rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

8.1 Basic Concepts

8.1.1 Frequent Itemset

An **itemset** is a collection of one or more items. An itemset with k items is called a k-itemset.

TODO does this apply only to transactional data?

Definition 8.1 (Frequent Itemset) *An itemset is **frequent** if its **support** (the fraction of transactions that contain the itemset) is greater than or equal to a user-specified minimum support threshold minsup.*

Transaction ID	Items Purchased
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Coke, Beer, Diaper
4	Bread, Milk, Beer, Diaper
5	Bread, Milk, Diaper, Coke

Table 8.1: Example of transactions

- Itemset - $Milk, Bread, Diaper$
- Suppport count (σ) - Frequency of occurrence of an itemset - $\sigma(Milk, Bread, Diaper) = 2$
- Support $\in [0, 1]$ - Fraction of transactions that contain an itemset - $s(Milk, Bread, Diaper) = \frac{2}{5}$
- An itemset whose support is greater than or equal to a *minsup* threshold is called a frequent itemset.
- Association Rule - An implication expression of the form $X \Rightarrow Y$, where X and Y are itemsets - $Milk, Diaper \Rightarrow Beer$
- Rule Evaluation Metrics -
  - Support ($s \in [0, 1]$)- The proportion of transactions that contain the itemset .

$$s = \frac{\sigma(X \cup Y)}{|T|}$$

  - Confidence ($c \in [0, 1]$)- The proportion of the transactions that contain X which also contain Y.

$$c = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Given a set of transactions T, the goal of association rule mining is to find all rules having

- support $\geq$ *minsup* threshold
- confidence $\geq$ *minconf* threshold

Brute-force approach:

- ◇ List all possible association rules
- ◇ Compute the support and confidence for each rule
- ◇ Prune rules that fail the minsup and minconf thresholds

Computationally prohibitive!

Two-step approach:

- ◇ Frequent Itemset Generation - Generate all itemsets whose support $\geq$ *minsup*
  This is still computationally expensive
- ◇ Rule Generation - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

The problem is finding the frequent itemsets efficiently.

Definition 8.2 (Apriori principle) *A subset of a frequent itemset is also a frequent itemset. Conversely, if an itemset is not frequent, none of its supersets can be frequent. We can exploit this property to prune the search space.*

The support of an itemset never exceeds the support of its subsets. This is also known as the anti-monotone property of support.

8.2 Apriori Algorithm

The purpose of the Apriori algorithm is to find all frequent itemsets in a transaction database, which means identifying itemsets that satisfy the support constraint (threshold), i.e. finding all itemsets whose support is greater than or equal to a user-specified minimum support threshold.

The Apriori principle holds due to the following property of support of the support measure:

Definition 8.3 (Anti-monotone Property) *Anti-monotone property of support is formulated as:*

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

This states that the support of an itemset never exceeds the support of its subsets.

Algorithm 8 Apriori Algorithm

1: $k \leftarrow 1$
2: $F_1 \leftarrow$ {frequent 1-itemsets} ▷ Scan database and count support of each item
3: **repeat**
4: $L_{k+1} \leftarrow$ **Candidate Generation** from F_k ▷ Generate candidate $(k+1)$-itemsets
5: $L_{k+1} \leftarrow$ **Candidate Pruning** of L_{k+1} ▷ Prune candidates with infrequent k-subsets
6: Scan transaction database to count support of each candidate in L_{k+1}
7: $F_{k+1} \leftarrow$ candidates in L_{k+1} with support $\geq$ minsup
8: $k \leftarrow k+1$
9: **until** $F_k = \emptyset$
10: **return** $\bigcup_k F_k$ ▷ Return all frequent itemsets

8.2.1 Candidate Generation

We can generate candidates using the $F_{k-1} \times F_{k-1}$ prefix method: merge two frequent $(k-1)$-itemsets if their first $k-2$ items are the same. For example, $\{A, B, C\}$ and $\{A, B, D\}$ can be merged to form the candidate $\{A, B, C, D\}$. Instead, $\{A, B, C\}$ and $\{A, C, D\}$ cannot be merged because their first $k-2$ items are not the same.

There is also an alternative $F_{k-1} \times F_1$ method: Merge two frequent $(k-1)$-itemsets if the last $(k-2)$ items of the first one is identical to the first $(k-2)$ items of the second. For instance, $Merge(ABC, BCD) = ABCD$, but $Merge(ABC, ABE)$ is not possible.

8.2.2 Candidate Pruning

Suppose we have have this set of *frequent* 3-itemset and the generated L_4 candidates with alternative $F_{k-1} \times F_1$ method:

$$F_3 = ABC, ABD, ABE, ACD, BCD, BDE, CDE$$
$$L_4 = ABCD, ACDE, ABDE, BCDE$$

To prune the candidates in L_4 we must check if all their possible 3-item subsets are in F_3:
- ◇ $ABCD$ - all subsets are frequent - keep
- ◇ $ACDE$ - ADE and ACE are not frequent - prune
- ◇ $ABDE$ - ADE is not frequent - prune
- ◇ $BCDE$ - BCE is not frequent - prune

At this point, the only candidate left is $ABCD$ which *might* be frequent, while the others could not possibly be, thus they were pruned. We need to scan the database to count its support.

This can be done with the aid of a hash tree structure to store the candidates and facilitate efficient support counting, to avoid checking each candidate against every transaction (computationally expensive).

8.2.3 Rule Generation

Once we have found all frequent itemsets, we can generate high-confidence association rules from each frequent itemset. Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow (L - f)$ has confidence $\geq$ *minconf*. If $|L| = k$, there are $2^k - 2$ possible rules that can be generated from L (excluding the empty set $L \rightarrow \emptyset$ and the set itself $\emptyset \rightarrow L$).

In general, confidence does not have an antimonotone property, so $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$. However, confidence of rules generated from the same itemset has an anti-monotone property. E.g., Suppose A, B, C, D is a frequent 4-itemset:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

Confidence is anti-monotone w.r.t. number of items on the right-hand-side (RHS) of the rule. This happens because, recalling the confidence formula

$$c(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$$

the numerator $s(X \cup Y)$ remains the same while the denominator $s(X)$ decreases as we move items from the RHS to the left-hand-side (LHS) of the rule, because $\sigma(A) \geq \sigma(AB) \geq \sigma(ABC)$ according to the Apriori principle, yielding an increase in confidence.

This allows to prune rules during generation during the process, as in the figure below.

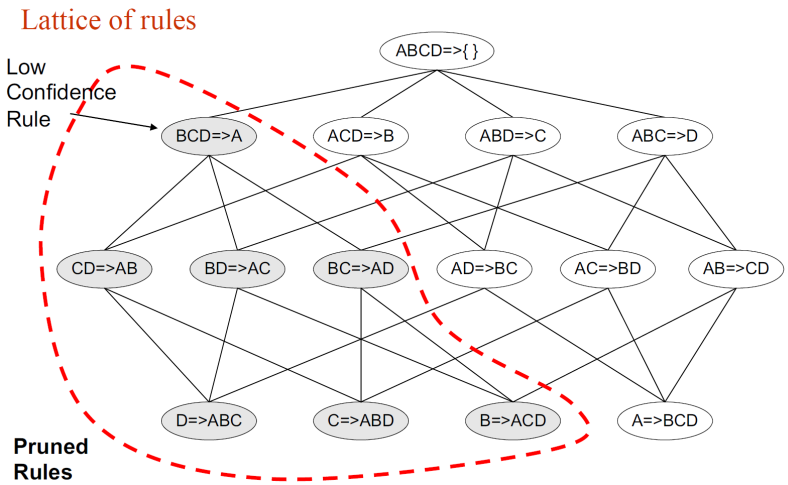


Fig. 8.1: Confidence Lattice

Definition 8.4 (Closed Itemset) *An itemset X is* ***closed*** *if none of its immediate supersets has the same support as the itemset X.*
X is not closed if at least one of its immediate supersets has support count as X.

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

Table 8.2: Example Transaction Database

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

Table 8.2: Frequent Itemsets and Their Support

8.2.4 Closed Itemsets

From the example above, the **closed itemsets** are:

- ◇ {B} with support 5 - closed because its immediate superset {A,B} has support $4 \neq 5$
- ◇ {A,B} with support 4 - closed because its immediate supersets {A,B,C} and {A,B,D} have support 2 and 3 respectively, both $\neq 4$
- ◇ {B,D} with support 4 - closed because its immediate superset {A,B,D} has support $3 \neq 4$ and {B,C,D} has support $2 \neq 4$
- ◇ {A,B,D} with support 3 - closed because its immediate superset {A,B,C,D} has support $2 \neq 3$
- ◇ {C,D} with support 3 - closed because its immediate supersets have different support
- ◇ {A,B,C,D} with support 2 - closed because it has no supersets

Examples of **non-closed itemsets**:

- ◇ {A} with support 4 is **not closed** because its immediate superset {A,B} has the same support 4
- ◇ {A,C} with support 2 is **not closed** because its immediate superset {A,B,C} has the same support 2
- ◇ {A,B,C} with support 2 is **not closed** because its immediate superset {A,B,C,D} has the same support 2

Closed itemsets are important because they provide a compact representation of all frequent itemsets while preserving complete support information.

8.2.5 Maximal Itemsets

Definition 8.5 (Maximal Itemset) *An itemset X is **maximal** if none of its immediate*[1] *supersets is frequent. X is not maximal if at least one of its immediate supersets is frequent.*

Assuming a minimum support threshold of 2, the **maximal frequent itemsets** from the example are:

- ◇ {B} with support 5 - maximal if we consider only singleton itemsets, but **not maximal** overall because its supersets {A,B}, {B,C}, {B,D} are frequent
- ◇ {A,B,D} with support 3 - maximal because its only superset {A,B,C,D} has support 2, which is still frequent, so **not maximal**
- ◇ {B,D} with support 4 - **not maximal** because its superset {A,B,D} is frequent
- ◇ {C,D} with support 3 - **not maximal** because its supersets {A,C,D}, {B,C,D}, and {A,B,C,D} are frequent (support ≥ 2)
- ◇ {A,B,C,D} with support 2 - **maximal** because it has no supersets and it is frequent

Therefore, with minsup = 2, the only **maximal frequent itemset** is:

- ◇ {A,B,C,D} with support 2

[1] immediate means? …TODO

Every maximal frequent itemset is also closed, but not every closed itemset is maximal. Maximal itemsets provide an even more compact representation than closed itemsets, but they only preserve information about which itemsets are frequent, not their exact support counts.

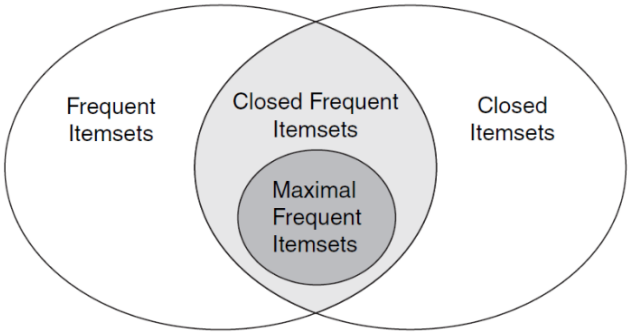


Fig. 8.2: Relationships between frequent, closed, closed frequent and maximal frequent itemsets

8.3 Confidence

The **confidence** of an association rule $X \Rightarrow Y$ is a measure of the reliability of the rule. It is defined as the conditional probability that a transaction contains the itemset Y given that it contains the itemset X. Mathematically, confidence is expressed as:

$$c(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$$

where:

- $s(X \cup Y)$ is the support of the itemset that contains both X and Y.
- $s(X)$ is the support of the itemset X.

The confidence value ranges from 0 to 1, where a higher confidence indicates a stronger association between the itemsets X and Y. For example, a confidence of 0.8 means that 80% of the transactions that contain X also contain Y.

8.3.1 Contingency tables

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

Table 8.3: Contingency table for itemsets X and Y

Given a dataset with N transactions, we can represent the occurrences of itemsets X and Y using a contingency table, where the entries are the support counts of the combinations of presence and absence of X and Y:

Customers	**Tea**	**Coffee**	$\cdots$
C1	0	1	$\cdots$
C2	1	0	$\cdots$
C3	1	1	$\cdots$
C4	1	0	$\cdots$
$\cdots$			

Table 8.4: Transaction data for Tea and Coffee

	Coffee	$\overline{Coffee}$	
Tea	150	50	200
$\overline{Tea}$	650	150	800
	800	200	1000

	Honey	$\overline{Honey}$	
Tea	100	100	200
$\overline{Tea}$	20	780	800
	120	880	1000

Table 8.4: Contingency table for Tea and Coffee, and another one for Tea and Honey

$Confidence \approx P(Coffee|Tea) = \frac{150}{200} = 0.75$, suggesting that people who drink tea are more likely to drink coffee than not drink coffee, given that the confidence is greater than 0.5.
But, in fact, $P(Coffee) = \frac{800}{1000} = 0.8$, meaning that people who drink tea are actually less likely to drink coffee than the general population.

$Confidence \approx P(Honey|Tea) = \frac{100}{200} = 0.5$, suggesting that people who drink tea are equally likely to drink honey or not drink honey.
But, actually, $P(Honey) = \frac{120}{1000} = 0.12$, meaning that people who drink tea are much more likely to drink honey than the general population.

8.3.2 Drawbacks

Confidence has some drawbacks:

- ◇ It does not consider the overall frequency of the consequent itemset Y in the dataset.
- ◇ It can be misleading in cases where the consequent itemset Y is very common in the dataset, leading to high confidence values even when there is no real association between X and Y.

To address these issues, other measures such as **lift** and **conviction** are often used in conjunction with confidence to provide a more comprehensive evaluation of association rules.

8.3.2.1 What rules do we want

- ◇ Confidence($X \Rightarrow Y$) should be sufficiently high
  - To ensure that people who buy X will more likely buy Y than not buy Y
- ◇ Confidence($X \Rightarrow Y$) > $support(Y)$
  - Otherwise, rule will be misleading because having item X actually reduces the chance of having item Y in the same transaction
- ◇ Is there any measure that capture this constraint?
  - Answer: *Yes.* There are many of them.

8.4 Other criteria

$confidence(X \Rightarrow Y) = support(Y)$ is equivalent to:

$$P(Y|X) = P(Y)$$
$$P(X,Y) = P(X)P(Y) \quad \text{(independence)}$$

$$P(X,Y) < P(X)P(Y) \quad \text{(negative correlation)}$$
$$P(X,Y) > P(X)P(Y) \quad \text{(positive correlation)}$$

8.4.1 Lift

The **lift** of an association rule $X \Rightarrow Y$ is a measure of how much more likely the occurrence of itemset Y is when itemset X is present, compared to when Y occurs independently of X. It is defined as:

$$\text{lift}(X \Rightarrow Y) = \frac{c(X \Rightarrow Y)}{s(Y)} = \frac{s(X \cup Y)}{s(X) \times s(Y)}$$

where:

- ◇ $c(X \Rightarrow Y)$ is the confidence of the rule.
- ◇ $s(Y)$ is the support of the itemset Y.

In the slides it is defined as:

$$\text{lift}(X \Rightarrow Y) = \frac{P(X,Y)}{P(X) \times P(Y)}$$

A lift value greater than 1 indicates a positive association between X and Y, meaning that the presence of X increases the likelihood of Y. A lift value less than 1 indicates a negative association, while a lift value equal to 1 suggests that X and Y are independent.

Lift is particularly useful for identifying interesting rules that may not be apparent from confidence alone, as it accounts for the overall frequency of the consequent itemset Y in the dataset.

Lift would fix the coffee problem, because, $0.75 \,/\, 0.8 = 0.9375 < 1$, indicating a negative correlation between tea and coffee. Also the honey problem would be fixed, because $0.5 \,/\, 0.12 = 4.1667 > 1$, indicating a positive correlation between tea and honey.

8.4.2 Interest

The **interest** of an itemset X is a measure of how much the actual support of X deviates from what would be expected if the items in X were independent. It is defined as:

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

A positive interest value indicates that the items in X co-occur more frequently than would be expected under independence, suggesting a positive association. A negative interest value indicates that the items co-occur less frequently than expected, suggesting a negative association. An interest value of zero suggests that the items are independent.

Alternative copilot definition

$$\text{interest}(X) = s(X) - \prod_{i \in X} s(\{i\})$$

where:

- $s(X)$ is the support of the itemset X.
- $\prod_{i \in X} s(\{i\})$ is the product of the supports of the individual items in X.
  - Written like this it should always be negative…that's weird

8.4.3 Other measures

In the slide two other measures are defined:

$$PS = P(X,Y) - P(X)P(Y)$$

$$\sigma\text{-}coefficient = \frac{P(X,Y) - P(X)P(Y)}{\sqrt{P(X)P(Y)(1-P(X))(1-P(Y))}}$$

These measures also aim to capture the degree of association between itemsets, taking into account their individual supports and the expected co-occurrence under independence.

8.5 Non-binary Attributes

Gender	⋯	Age	Annual Income	No of hours spent online per week	No of email accounts	Privacy Concern
Female	⋯	26	90K	20	4	Yes
Male	⋯	51	135K	10	2	No
Male	⋯	29	80K	10	3	Yes
Female	⋯	45	120K	15	3	Yes
Female	⋯	31	95K	20	5	Yes
Male	⋯	25	55K	25	5	Yes
Male	⋯	37	100K	10	1	No
Male	⋯	41	65K	8	2	No
Female	⋯	26	85K	12	1	No
⋯	⋯	⋯	⋯	⋯	⋯	⋯

Male	Female	⋯	Age < 13	Age $\in [13,21)$	Age $\in [21,30)$	⋯	Privacy = Yes	Privacy = No
0	1	⋯	0	0	1	⋯	1	0
1	0	⋯	0	0	0	⋯	0	1
1	0	⋯	0	0	1	⋯	1	0
0	1	⋯	0	0	0	⋯	1	0
0	1	⋯	0	0	0	⋯	1	0
1	0	⋯	0	0	1	⋯	1	0
1	0	⋯	0	0	0	⋯	0	1
1	0	⋯	0	0	0	⋯	0	1
0	1	⋯	0	0	1	⋯	0	1
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯

Fig. 8.3: Handling non-binary attributes by changing the actual columns

8.5.1 Categorical attributes

Categorical attributes are variables that can take on a limited, fixed number of possible values, representing distinct categories or groups. These attributes are often used in data mining and machine learning tasks to classify or group data points based on their characteristics.

We have to apply association analysis to non-asymmetric binary attributes, so we have to formulate rules like:

$$Gender = Male, Age \in [21, 30) \Rightarrow No\ of\ hours\ online \geq 10$$

Some attributes can have many possible values, with many of these values having very low support. To deal with this, we can use **attribute generalization**, which involves replacing specific attribute values with more general categories based on a predefined **hierarchy** or **taxonomy**. For example, instead of using specific ages, we can group them into age ranges like [0-10), [10-20), [20-30), etc. This helps to reduce the number of distinct values and increases the support for each category, making it easier to identify meaningful patterns in the data.

In some other cases we also may have a hierarchy of values for an attribute. For example, for the attribute "Location", we may have a hierarchy like City → State → Country. In such cases, we can use the hierarchy to generalize the attribute values and find association rules at different levels of granularity.

8.5.2 Continuous attributes

Continuous attributes are variables that can take on an infinite number of values within a given range. These attributes are often used in data mining and machine learning tasks to represent measurements or quantities that can vary continuously.

To handle such values we have various methods:

- ◇ Discretization-based
- ◇ Statistics-based
- ◇ Non-discretization (such as MINAPRIORI)

// TODO skipped support counting with hash tree

8.6 Rule Extraction

8.6.1 Association Rule Mining

Association Rule Mining refers to (everything in the previous sections ☺), given a set of transactions, finding rules that will predict the occurrence of an item based on the occurrences of other items in the transaction. Formally, an association rule is an implication of the form $X \Rightarrow Y$, where X and Y are disjoint itemsets. The goal is to discover interesting relationships between items in large datasets.

Rules obtained in this way are unordered, so no priority is given to any of them. When applying the rules to a new instance, we may have multiple rules that apply, and we need a way to resolve conflicts.

They induce a form of local optimization (?) since each rule is evaluated independently of the others.

8.6.2 Rule lists

A rule list is a sorted set of rules: starting from rule 1, if rule i does not support the given instance, move to rule $i + 1$. Here, we do have priority: the first rule that supports the instance is the one predicting.

```
(1) if age in (23, 26) and priors in (2, 3) then recidivous
(2) if age in (18, 20) then recidivous
(3) if sex is male and age in (21, 22) then recidivous
(4) if priors > 3 then recidivous
(5) else not_recidivous
```

Support, confidence, and other measures still hold, but we need to reconsider support: in a rule list, the first rule to support an instance is the one predicting. To adjust, we treat support as an indicator variable, evaluating to 1 for the *first* rule of a given list to apply, and otherwise: 0.

$$supp_A(r,x) = \begin{cases} 1 & \textit{if } r \in A \textit{ is the first rule to satisfy instance } x \\ 0 & \textit{otherwise} \end{cases}$$

age	priors	sex	r_1	r_2	r_3	r_4	r_5
24	4	m	0	0	0	1	0
20	1	f	0	1	0	0	0
20	5	m	0	1	0	0	0

Table 8.5: Support indicator variable for a rule list

8.6.3 Branch and bound algorithms

Family of optimization algorithms that defines a space of solutions to search, according to a given objective function. Exploring the whole space is unfeasible, hence branch and bound algorithms define:

- ◇ A set of feasible solutions to explore from a starting set: the branches of the solution tree to explore
- ◇ A bound for the objective: it allows to prune branches, thus reducing the search space

A search tree has an exponentially large number of states! For a tree of order k and depth d, $\mathcal{O}(k^d)$ states! So we do not need to go very deep in the tree, just enough to find a good solution.

Exploration populates a queue of states to consider: the larger the queue, the larger the computational cost. A search algorithm simply

- ◇ Inserts states in the queue
- ◇ Pops states from the queue

8.6.4 CORELS - Certifiably Optimal Rule Lists

CORELS is a branch-and-bound algorithm to learn optimal rule lists for categorical data. It uses a variety of techniques to prune the search space and efficiently find the optimal rule list.

We model our rule list using a tuple $R = \langle P_R, Y_R, y_r \rangle$, where:

- ◇ P_R is the ordered list of antecedents (if-clauses)
- ◇ Y_R is the list of consequent predictions (then-clauses)
- ◇ y_r^0 is the default prediction (else-clause)
- ◦ k^R is the number of rules in the list (length of P_R and Y_R)

Rules are ordered, and a Rule list R may be prefix of another Rule list R', having $R \preceq R^+$. In the search tree, child nodes extend parent nodes by adding a new rule at the end of the list.

(1) **if** age $\in (23, 26)$ **and** priors $\in (2, 3)$ **then recidivous**
(2) **if** age $\in (18, 20)$ **then recidivous**
(3) **else not_recidivous**

- ◇ $k^R = 2$
- ◇ $P_R : (p^1, \ldots, p^{k^R}) = (\text{age} \in (23, 26) \text{ and priors} \in (2, 3),\ \text{age} \in (18, 20))$
- ◇ $Y_R : (y_R^1, \ldots, y_R^{k^R}) = (\text{recidivous},\ \text{not\_recidivous})$
- ◇ $y_R^0 = \text{not\_recidivous}$

8.6.4.1 CORELS empirical Error

$$l(R, X, Y) = l_R(R, X, Y) + l_0(R, X, Y),$$

dove

$$l_R(R, X, Y) = \frac{1}{n} \sum_{(x,y) \in X \times Y} \sum_{j=1}^{k^R} \text{supp}_{P_R}(p_R^j, x)\ \mathbf{1}[y \neq y_R^j],$$

$$l_0(R, X, Y) = \frac{1}{n} \sum_{(x,y) \in X \times Y} \Big(1 - \text{supp}(P_R, x)\Big) \, \mathbf{1}[y \neq y_R^0]$$

To read the formula, the fract and the sum over $X \times Y$ is just the average over all instances, which will yield a ratio. Recall that $\text{supp}_{P_R}(p_R^j, x)$ is 1 if rule j is the first rule to satisfy instance x, and 0 otherwise, and that *exactly one* rule will satisfy x. The rightmost indicator function is 1 if the prediction of rule j (y_R^j) does not match the actual label y of instance x, and 0 otherwise.
In other words, $l_R(R, X, Y)$ counts the fraction of misclassified instances by the rules in the list. The second term, $l_0(R, X, Y)$, handles the default rule (else-clause). It counts the fraction of misclassified instances that are not covered by any of the rules in the list.

We may compute bounds on **length** and **accuracy**. The explanation related to length is rather confused, I couldn't get it all, the idea is that we have an an upper bound on the maximum number of rules (k^R) we want in the list, and if we reach that number, we can prune that branch of the search tree.
For the accuracy, the idea is that the number of instances correctly classified by a rule p to be added to the rule list should be higher than some threshold, in order to allow the addition of that rule to the list. If not, we can prune that branch of the search tree.

8.6.4.2 Trimming the search space

Trimming is related to **Loss** L_R and **Lower bound** $b(R)$. Here we see appearing a **regularization** term λk^R, where λ is a hyperparameter that controls the trade-off between *accuracy* and *complexity* of the rule list, and k^R is the number of rules in the list.
In general less rules mean more generalization and interpretability, so we want to penalize longer rule lists, trading with the potential increase in accuracy they might yield.

$$L(R, X, Y) = l(R, X, Y) + \lambda k^R$$
$$b(R, X, Y) = l_R(R, X, Y) + \lambda k^R \leq L(R, X, Y)$$

In formulas this is a mess, and I couldn't get it.

The overall idea, if I understood correctly is that extensions R^+ of a rule list R will always have a loss $L_R(R^+)$ greater than or equal to the loss of the parent rule list $L_R(R)$, because adding more rules can only increase the number of misclassifications (or leave it unchanged, because the rules in R have priority, and if they were picked before, they will still be picked even if there are more rules "at the bottom"). Thus, if we have already found a rule list R_{best} with a certain loss lower than R's, we can prune any extensions of R that cannot improve upon this loss.

8.7 FP Tree Growth

The FP-tree contains a compressed representation of the transaction database.

A trie (prefix-tree) data structure is used

Each transaction is a path in the tree (paths may overlap). Once the FP-tree is constructed the recursive, divide-and-conquer FP-Growth algorithm is used to enumerate all frequent itemsets.

Since transactions are sets of items, we need to transform them into ordered sequences so that we can have prefixes We need to impose an order to the items.Initially, we assume a lexicographic order.

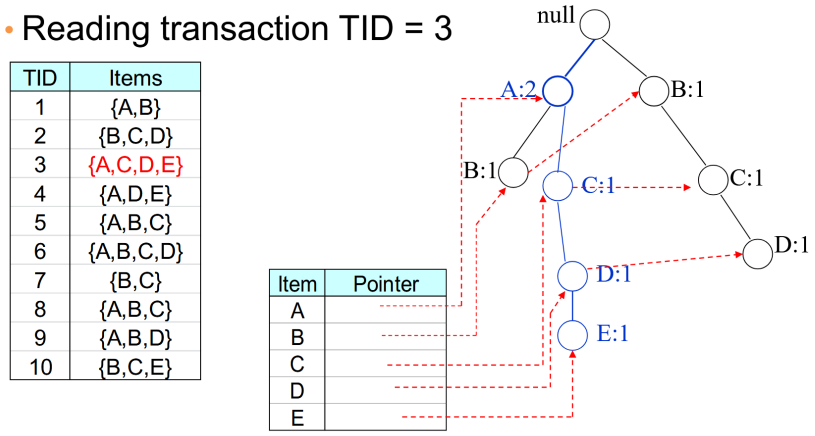


Fig. 8.4: Each transaction is a path in the trie. There is a header table to point to the first occurrence of each item in the tree.

The size of the tree depends on the compressibility of the data

- ◇ *Extreme case*: All transactions are the same, the FP-tree is a single branch
- ◇ *Extreme case*: All transactions are different the size of the tree is the same as that of the database (bigger actually since we need additional pointers)

The size of the tree also depends on the ordering of the items. We could order the items according to their frequency from larger to smaller, but we need to do an extra pass over the dataset to count frequencies.

Algorithm 9 FP-Growth Algorithm

1: **for** each suffix X **do**
2: **Phase 1:** Construct the prefix tree for X and compute the support using the header table and the pointers
3: **if** X is frequent **then**
4: **Phase 2:** Construct the *conditional* FP-tree for X:
5: 1. Recompute support
6: 2. Prune infrequent items
7: 3. Prune leaves and recurse

Chapter 9

Sequential Pattern Mining

Sequential pattern mining is the task of discovering statistically relevant patterns between data examples where the values are delivered in a sequence. A sequence is an ordered list of elements (transactions). Each element is a set of events (items) that occur together. A sequential pattern is a sequence that occurs frequently in a sequence database.

Consider the following example of a sequence of different transactions by a customer at an online store:
`< {Digital Camera,iPad} {memory card} {headphone,iPad cover} >`
This sequence indicates that the customer first bought a digital camera and an iPad, then later bought a memory card, and finally bought a headphone and an iPad cover together.

The knowledge we can extract here is that probably the customer didn't realized he needed the memory card when he bought the digital camera, so he bought it later. Also, after buying the iPad, he probably realized he needed accessories for it, so he bought them together.

Databases of transactions usually have a temporal information which *Sequential patterns* can exploit.

9.1 Definitions

A sequence is an ordered list of elements (transactions/itemsets). An element is a set of items that occur at the same time, and it is also associated with a timestamp.

$$s = \langle e_1 e_2 e_3 \rangle$$

$$e_i = \{i_1, i_2, \ldots, i_k\}$$

Length of a sequence, $|s|$, is given by the number of elements of the sequence.
A k-sequence is a sequence that contains k events (items).

A sequence $s_a = \langle a_1 a_2 \ldots a_n \rangle$ is a subsequence of another sequence $s_b = \langle b_1 b_2 \ldots b_m \rangle$ (denoted as $s_a \subseteq s_b$) if there exist integers $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$.

Data sequence	Subsequence	T/F
$\langle\{2,4\}\{3,5,6\}\{8\}\rangle$	$\langle\{2\}\{3,5\}\rangle$	$\mathbb{T}$
$\langle\{1,2\}\{3,4\}\rangle$	$\langle\{1\}\{2\}\rangle$	$\mathbb{F}$
$\langle\{2,4\}\{2,4\}\{2,5\}\rangle$	$\langle\{2\}\{4\}\rangle$	$\mathbb{T}$

Table 9.1: Examples of subsequence containment

It may occur that there are multiple ways in which a subsequence can be contained in a data sequence. Each such way is called an **instance** or **occurrence** of the subsequence in the data sequence. In sequential pattern mining, it's sufficient to find only one instance.

Definition 9.1 (Subsequence support) *The **support** of a subsequence w is the fraction of data sequences in the sequence database S that contain w.*

9.1.1 Exercises

9.1.1.1 Exercise 1

Find instances/occurences of the following subsequences in the sequence database

$$\langle\{C\}\{H\}\{C\}\rangle$$
$$\langle\{A\}\{F\}\rangle$$
$$\langle\{A\}\{A\}\{D\}\rangle$$
$$\langle\{A\}\{A,B\}\{F\}\rangle$$

$$\langle\{A,C\}\{C,D\}\{F,H\}\{A,B\}\{B,C,D\}\{E\}\{A,B,D\}\{F\}\rangle$$
$$t=0\ t=1\ t=2\ t=3\ t=4\ t=5\ t=6\ t=7$$

9.1.1.2 Exercise 2

Find instances/occurences of the following subsequences in the sequence database

$$\langle\{C\}\{H\}\{C\}\rangle$$
$$\langle\{A\}\{B\}\rangle$$
$$\langle\{C\}\{C\}\{E\}\rangle$$
$$\langle\{A\}\{E\}\rangle$$

$$\langle\{A,C\}\{C,D,E\}\{F\}\{A,H\}\{B,C,D\}\{E\}\{A,B,D\}\{F\}\rangle$$
$$t=0\ t=1\ t=2\ t=3\ t=4\ t=5\ t=6\ t=7$$

9.2 Towards an Algorithm

Definition 9.2 (Sequential Pattern Mining) *Given a sequence database S and a minimum support threshold min_sup, the **sequential pattern mining** task is to find all subsequences w such that $support(w) \geq min\_sup$.*

The most trivial, yet inefficient, way to find all sequential patterns is to generate all possible $k-subsequences$ for $k = 1, 2, \ldots$ and count their support in the sequence database. This approach is computationally expensive due to the exponential number of possible subsequences.

Note that a k-sequence contains k **events**, not necessarily k **elements**!

9.2.1 GSP - Generalized Sequential Pattern

Follows the same structure of the Apriori algorithm, i.e. starting from short patterns and finding longer ones at each iteration.
It is based on "Apriori principle" (or "anti-monotonicity of support") which states that if a sequence is not frequent, none of its super-sequences can be frequent.

$$S_1 \subseteq S_2 \implies support(S_1) \geq support(S_2)$$

Proof: Any input sequence that contains S2 will also contain S1

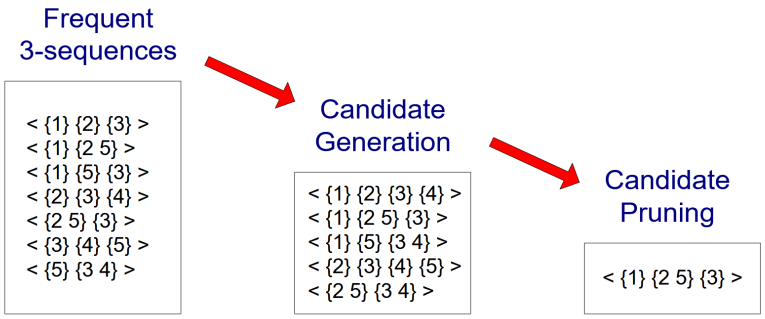


Fig. 9.1: GSP example

Algorithm 10 GSP - Generalized Sequential Pattern

Step 1: Make the first pass over the sequence database D to yield all the 1-element frequent sequences

repeat

Candidate Generation:

Merge pairs of frequent subsequences found in the $(k-1)$-th pass to generate candidate sequences that contain k items

Candidate Pruning:

Prune candidate k-sequences that contain infrequent $(k-1)$-subsequences

Support Counting:

Make a new pass over the sequence database D to find the support for these candidate sequences

Candidate Elimination:

Eliminate candidate k-sequences whose actual support is less than min_sup

until no new frequent sequences are found

9.2.1.1 Candidate Generation in GSP

Given n events: $i_1, i_2, i_3, \ldots, i_n$

- Candidate 1-subsequences:

$$\langle\{i_1\}\rangle, \langle\{i_2\}\rangle, \langle\{i_3\}\rangle, \ldots, \langle\{i_n\}\rangle$$

- Candidate 2-subsequences:

$$\langle\{i_1, i_2\}\rangle, \langle\{i_1, i_3\}\rangle, \ldots, \langle\{i_1\}\{i_1\}\rangle, \langle\{i_1\}\{i_2\}\rangle, \ldots, \langle\{i_{n-1}\}\{i_n\}\rangle$$

- Candidate 3-subsequences:

$$\langle\{i_1, i_2, i_3\}\rangle, \langle\{i_1, i_2, i_4\}\rangle, \ldots, \langle\{i_1, i_2\}\{i_1\}\rangle, \langle\{i_1, i_2\}\{i_2\}\rangle, \ldots,$$
$$\langle\{i_1\}\{i_1, i_2\}\rangle, \langle\{i_1\}\{i_1, i_3\}\rangle, \ldots, \langle\{i_1\}\{i_1\}\{i_1\}\rangle, \langle\{i_1\}\{i_1\}\{i_2\}\rangle, \ldots$$

Remark: events within an element are ordered

- YES: $\langle\{i_1, i_2, i_3\}\rangle$ NO: $\langle\{i_3, i_1, i_2\}\rangle$

9.2.1.2 Candidate Generation - Merging Procedure

- **Base case ($k = 2$):**
  - Merging two frequent 1-sequences $\langle\{i_1\}\rangle$ and $\langle\{i_j\}\rangle$ will produce two candidate 2-sequences: $\langle\{i_1\}\{i_j\}\rangle$ and $\langle\{i_1, i_j\}\rangle$
  - Special case: i_1 can be merged with itself: $\langle\{i_1\}\{i_1\}\rangle$
- **General case ($k > 2$):**
  - A frequent $(k-1)$-sequence w_1 is merged with another frequent $(k-1)$-sequence w_2 to produce a candidate k-sequence if the subsequence obtained by removing the **first event** in w_1 is the same as the one obtained by removing the **last event** in w_2
    - The resulting candidate after merging is given by the sequence w_1 extended with the last event of w_2.

- If last two events in w_2 belong to the same element $\Rightarrow$ last event in w_2 becomes part of the last element in w_1:
$$\langle\{d\}\{a\}\{b\}\rangle + \langle\{a\}\{b,c\}\rangle = \langle\{d\}\{a\}\{b,c\}\rangle$$
This is allowed because if we trim the first event from w_1 and the last event from w_2 we get the same sequence: $\langle\{a\}\{b\}\rangle$
However, the sequence must be exactly the same in terms of *items*, not only *events*! The following case would not be allowed:
$$\langle\{d\}\{a,b\}\rangle + \langle\{a\}\{b,d\}\rangle$$
Because on the left we have $\langle\{a,b\}\rangle$ and on the right $\langle\{a\}\{b\}\rangle$ after trimming.
- Otherwise, the last event in w_2 becomes a separate element appended to the end of w_1:
$$\langle\{a,d\}\{b\}\rangle + \langle\{d\}\{b\}\{c\}\rangle = \langle\{a,d\}\{b\}\{c\}\rangle$$

- Special case: check if w_1 can be merged with itself
  - Works when it contains only one event type: $\langle\{a\}\{a\}\rangle + \langle\{a\}\{a\}\rangle = \langle\{a\}\{a\}\{a\}\rangle$
  - This woudl not work
$$\langle\{a,b\}\{a,b\}\rangle + \langle\{a,b\}\{a,b\}\rangle = !!!$$

9.2.1.3 Merging Examples

- Merging the sequences $w_1 = \langle\{1\}\{2\ 3\}\{4\}\rangle$ and $w_2 = \langle\{2\ 3\}\{4\ 5\}\rangle$
  - will produce the candidate sequence $\langle\{1\}\{2\ 3\}\{4\ 5\}\rangle$ because the last two events in w_2 (4 and 5) belong to the same element
- Merging the sequences $w_1 = \langle\{1\}\{2\ 3\}\{4\}\rangle$ and $w_2 = \langle\{2\ 3\}\{4\}\{5\}\rangle$
  - will produce the candidate sequence $\langle\{1\}\{2\ 3\}\{4\}\{5\}\rangle$ because the last two events in w_2 (4 and 5) do not belong to the same element
- We **do not have to** merge the sequences $w_1 = \langle\{1\}\{2\ 6\}\{4\}\rangle$ and $w_2 = \langle\{1\}\{2\}\{4\ 5\}\rangle$ to produce the candidate $\langle\{1\}\{2\ 6\}\{4\ 5\}\rangle$
  - Notice that if the latter is a viable candidate, it will be obtained by merging w_1 with $\langle\{2\ 6\}\{4\ 5\}\rangle$

9.2.1.4 Candidate Pruning

Candidate pruning follows —again— the Apriori principle: If a k-sequence W contains a $(k-1)$-subsequence that is not frequent, then W is not frequent and can be pruned.

Method:

- Enumerate all $(k-1)$-subsequences:
  - $\{a,b\}\{c\}\{d\} \rightarrow \{b\}\{c\}\{d\}, \{a\}\{c\}\{d\}, \{a,b\}\{d\}, \{a,b\}\{c\}$
- Each subsequence generated by cancelling 1 event in W
  - Number of $(k-1)$-subsequences $= k$
- Remark: candidates are generated by merging two "mother" $(k-1)$-subsequences that we know to be frequent
  - Correspond to remove the first event or the last one
  - Number of significant $(k-1)$-subsequences to test $= k-2$
  - Special cases: at step $k = 2$ the pruning has no utility, since the only $(k-1)$-subsequences are the "mother" ones

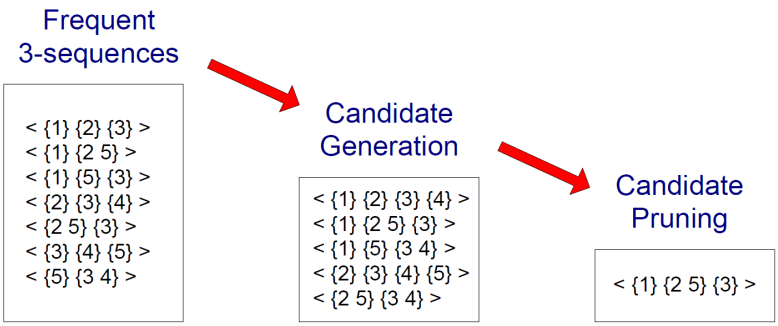


Fig. 9.2: Example of candidate Generation and Pruning

To understand how this works, consider the first generated candidate in Figure 9.2: $\langle\{1\}\{2\}\{3\}\{4\}\rangle$

- We generate the following $(k-1)$-subsequences:

$$\langle\{2\}\{3\}\{4\}\rangle, \langle\{1\}\{3\}\{4\}\rangle, \langle\{1\}\{2\}\{4\}\rangle, \langle\{1\}\{2\}\{3\}\rangle$$

- The first and the last subsequences are the "mother" ones, so we only need to check the other two
- They do not appear in the list of frequent 3-sequences, so we can prune this candidate

The candidate $\langle\{1\}\{25\}\{3\}\rangle$ instead, generates the following $(k-1)$-subsequences:

$$\langle\{25\}\{3\}\rangle, \langle\{1\}\{2\}\{3\}\rangle, \langle\{1\}\{25\}\rangle, \langle\{1\}\{5\}\{3\}\rangle$$

All of them are frequent, so we keep this candidate for the next step.

9.3 Timing Constraints

In some applications, it is useful to impose timing constraints on the sequential patterns to be mined. Typical timing constraints include:

- x_g - **max gap**: Each element of the pattern instance must be at most x_g time after the previous one
- n_g - **min gap**: Each element of the pattern instance must be at least n_g time after the previous one
- m_s - **max span**: The overall duration of the pattern instance must be at most m_s time

An approach to enforce timing constraints is to mine sequential patterns without constraints and then filter the resulting patterns by checking their instances against the timing constraints. This however could be inefficient if many patterns are generated that do not satisfy the timing constraints.

It would be better to modify GSP to take timing constraints into account during candidate generation and support counting. This would reduce the number of candidates generated and the number of instances counted that do not satisfy the timing constraints.
However we must ensure that the Apriori principle still holds when timing constraints are applied.

- SPAN
  If we remove instances in the middle, the span remains the same. If we remove from the edges, the span can only decrease.
- MIN-GAP
  If we remove instances, the gaps between remaining instances can only increase.
- MAX-GAP
  If we remove instances inbetween, the gaps between remaining instances may increase, potentially violating the max-gap constraint. Therefore, the Apriori principle does not hold for max-gap constraints.
  We need to use **contiguous subsequences** to ensure that the max-gap constraint is preserved when generating candidates and counting support.

9.3.1 Contiguous Subsequences

Definition 9.3 (Contiguous Subsequence) *s is a **contiguous subsequence** of $w = \langle e_1\rangle\langle e_2\rangle \dots \langle e_k\rangle$ if any of the following conditions hold:*

1. *s is obtained from w by deleting an item from either e_1 or e_k* *(avoids internal "jumps")*

2. *s is obtained from w by deleting an item from any element e_i that contains* ***at least 2 items*** *(not interesting for our usage)*
   This is a key point!
3. *s is a contiguous subsequence of s' and s' is a contiguous subsequence of w (recursive definition)*

Examples: Consider $s = \langle\{1\}\{2\}\rangle$

- s **is** a contiguous subsequence of:
$$\langle\{1\}\{2,3\}\rangle$$
- s is **not** a contiguous subsequence of:
$$\langle\{1\}\{3\}\{2\}\rangle \quad \text{and} \quad \langle\{2\}\{1\}\{3\}\{2\}\rangle$$

So, when generating candidates with max-gap constraints, we only merge sequences that produce candidates where the "mother" sequences are contiguous subsequences of the candidate.

In some domains, we may have only one very long time series, for example:

- monitoring network traffic events for attacks
- monitoring telecommunication alarm signals

Goal is to find frequent sequences of events in the time series, so we have to count "instances", but which ones? This problem is also known as *frequent episode mining*.

Chapter 10

Supervised Machine Learning

A machine is said to learn if, when tackling a task, it is able to improve its own performance through experience.

- ◇ Task T: the problem we are trying to solve, e.g., predict the cancer risk of a patient
- ◇ Experience E: the experience on the task provided to the model, e.g., some dataset
- ◇ Performance P: a measure of success, e.g., the success rate in predicting cancer
- ◇ Model F: a function f solving the task with a learning algorithm A

Task	**Predict**	**Example**
Binary classification	one of two discrete labels	Is the patient at high risk of developing cancer, or not?
Multilabel classification	any of several discrete labels	Of all the possible syndromes, which is the patient going to develop?
Multiclass classification	one of several discrete labels	The student is going to major in...?
Regression	a continuous label	The student's grade is going to be...?

Table 10.1: Types of supervised learning tasks

10.1 Experience or not

Models are not developed to aid on known experiences, rather on **unknown** ones. The performance of a model can't be uniquely measured on its performance on the given experience, but rather on novel experiences which the model was not preview to. We want to achieve a low generalization error.

- ◇ **Optimization**
  Maximizes performance on the given experience E: optimization performance
- ◇ **Machine Learning**
  Maximizes performance on the given, and expected non-given, experience E and $\bar{E}$: generalization performance

If we do not know the non-given experience $\bar{E}$ how can we ever expect to be effective on it?
Equal distribution assumption. It is assumed that the given experience E, and the non- given experience $\bar{E}$ are sampled from the same distribution Pr^E.
Given a learning algorithm A, can I always expect the performance to transfer?
No.

Sampling from the data distribution is a random process, and the resulting models learned end up erring in terms of:

- ◇ Bias: the expected performance decrease w.r.t. the best model
- ◇ Variance: the variance with respect to different samples

Ideally, we want to have learning algorithms with low enough bias and variance.

10.2 Improper models

There are two categories of improper models:

- Underfit. High bias, high variance. Models which have poor performance, regardless of samples. They have not learnt enough!
  High variance, low bias. Models which ought to improve their performance on the given experience E
- Overfit. Low bias, high variance. Models which have overfit on the given experience, and ought to improve their generalization performance. They have learnt too much!
  Low variance, high bias. Models which ought to improve their generalization performance on the unknown experience $\bar{E}$

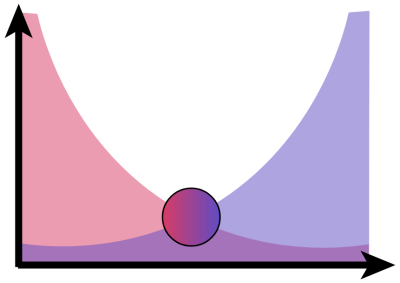


Fig. 10.1: Bias (red) and variance (blue) as a decomposition of model performance

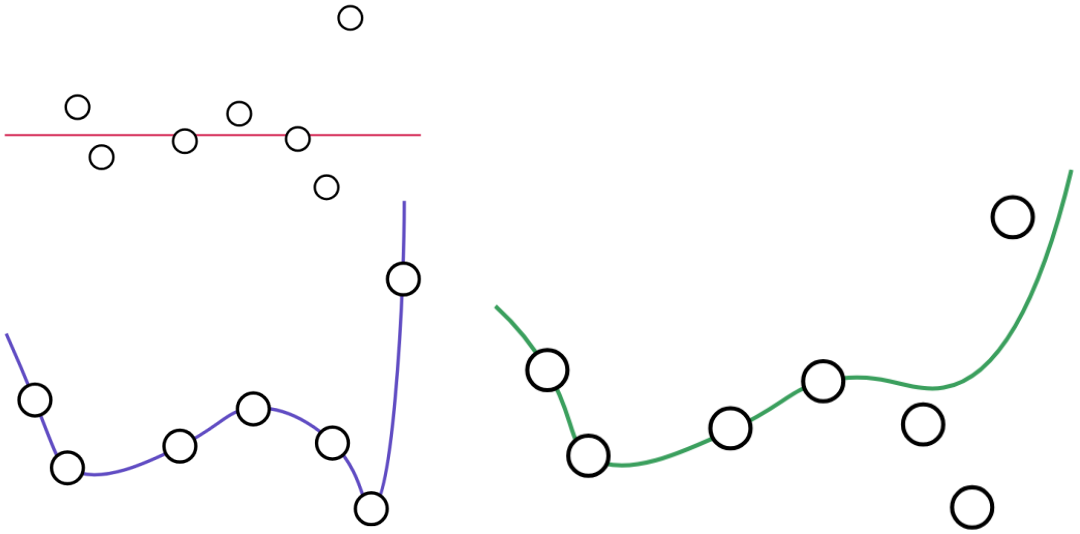


Fig. 10.1: Two models approximating a dataset: one model has too low a capacity and underfits the data (in red), while another has too high a capacity and overfits the data (in blue). These are opposed to a well-fitted model (in green) which generalizes well, without being too strict and leaving some space for variance in data.

10.3 Searching for models

- *Tackling the generalization gap.* Data-based strategies to maximize generalization performance
- *Performance evaluation.* How to measure model performance/error
- *Parameterization.* Exploring the space of models

We design two phases in the model search:

- **Model selection**
  A learning phase wherein, among all possible models in a model space $\mathcal{F}$, we select a model f
- **Model Validation**
  A learning phase wherein we estimate the generalization performance (error) of the selected model f

Model validation cannot affect model selection: it only goes one way, from selection to validation. Thus, we need to incorporate in model selection some strategy to avoid under/overfitting

10.3.1 Data Partitioning

The standard approach is model agnostic: we can apply this to any learning algorithm or family of models we want. We operate a tripartite partitioning of (X, Y):

- ◇ **Training dataset** (X^{tr}, Y^{tr}): search through the models' space $\mathcal{F}$
- ◇ **Validation dataset** (X^{vl}, Y^{vl}): guesstimate the generalization performance of candidate models $f_1, \ldots, f_k$
- ◇ **Test dataset** (X^{ts}, Y^{ts}): estimate the generalization performance of the selected model f_i

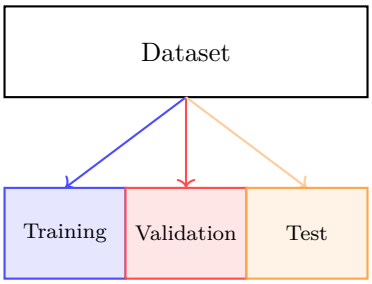


Fig. 10.2: A partition of the dataset (in black) in training (blue), validation (red), and test (beige).

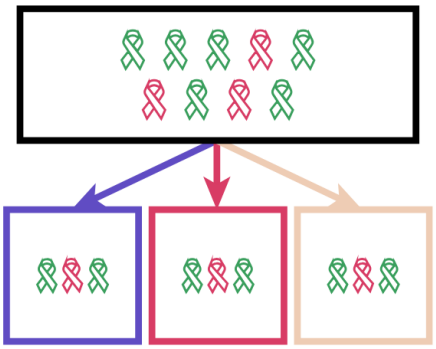
Task	Training	Validation	Test
Disease diagnosis	Biology lectures	Homework	Exam
Learning a language	Duolingo	Exchange student	Living abroad
Pandemic diffusion	Black plague	Ebola	Covid

Table 10.2: Analogy between data partitioning and real-world learning

10.3.1.1 Partitioning the dataset

How to partition the dataset properly?

- ◇ **Size** - Test and validation set of similar size, training set of much larger size, e.g., a ratio of 4:1. Some learning algorithms are more data- hungry, so this is a starting baseline.
- ◇ **Distribution** - Ideally, same distribution for all three datasets. Random stratified sampling is used



A partition of the dataset (in black) in training (blue), validation (red), and test (beige). Patients with (red cross) and without (green cross) cancer are split evenly among the blocks.

Fig. 10.2: Partitioned data

10.3.2 Model selection

There are two key model selection strategies: **hold-out** and **cross-validation**.

Hold-out leverages the three-blocks partition train-validation test.

- ◇ Learn candidate models $f_1, \ldots f_k$ on the training set
- ◇ Evaluate them on the validation set
- ◇ Estimate the generalization error on the test set

K-fold Cross-validation aims at run multiple times the hold-out strategy, to increase the size of the validation set. We partition a given set of data, e.g., the training dataset, in k blocks (folds). In each fold, one block acts as validation set, while the other $k-1$ blocks act as training set. We repeat this k times, each time changing the validation block.

Stretching hold-out, we aim to further increase the size of the validation set. We partition a given set of data, e.g., the training dataset, in two folds:

- ◇ in set 1, block 1 is a training dataset, block 2 is the validation dataset
- ◇ in set 2, block 1 is a training dataset, block 2 is the validation dataset

Now I can learn and guesstimate a model f_i on both folds!

This is called **2-fold cross-validation**, but we can generalize it to k folds.

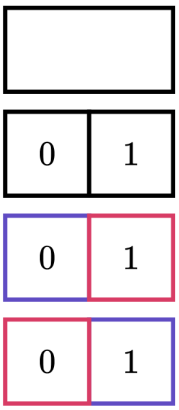
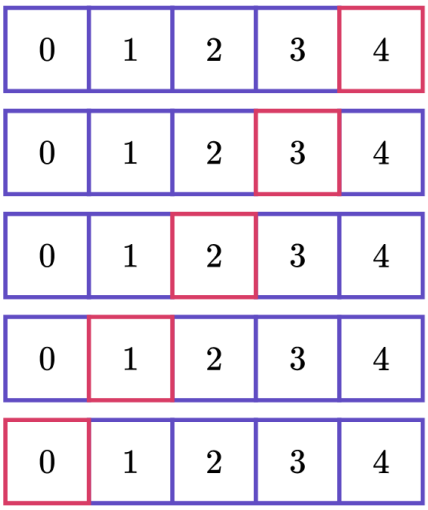


Fig. 10.3: A set partitioned in two blocks, and the resulting folds. In each fold, a block acts as validation set (in red), and the other as training set (in blue).



A set partitioned in $k = 5$ blocks, and the resulting *folds*. In each fold, a block acts as validation set (red), and the other as training set (blue).

Fig. 10.3: A set partitioned in k blocks, and the resulting folds. In each fold, a block acts as validation set (in red), and the other as training set (in blue).

10.4 Performance evaluation

With proper partitioning, we are now able to feed experiences (data) to the candidate models $f_1, \ldots, f_k$ we wish to select. How do we evaluate them?

Classification - Classification tasks generally aim to measure a Hamming distance ($\ominus$) between the gold labels, and the labels given by the model. Either measured as error or performance.

Reminder: we indicate the vector of n gold labels with $Y \in y$, the model of interest f with f, its prediction on an instance x^i with $f(x^i)$, and the indicator function with $\mathbb{1}$.

Task	Measure	Formulation
Binary, Multiclass	Accuracy	$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(Y_i = f(x^i))$ or $1 - \textit{Error rate}$
Binary, Multiclass	Error rate	$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(Y_i \neq f(x^i))$ or $1 - \textit{Accuracy}$
Multilabel	Jaccard similarity	$\frac{1}{n}\sum_{i=1}^{n} \frac{Y_i \cap f(x^i)}{Y_i \cup f(x^i)}$
Multilabel	Hamming error	$\frac{1}{n}\sum_{i=1}^{n} 1 - (Y_i \ominus f(x^i))$

Table 10.3: Performance measures for classification tasks

10.4.1 Confusion Matrix

In some binary cases, one label y is for us of interest (positive label), e.g., patients we predict will have cancer, while the other is not (negative label). We can construct a *confusion matrix* out of the predictions $f(x^i)$ of the model, that we can then leverage to define more performance measures.

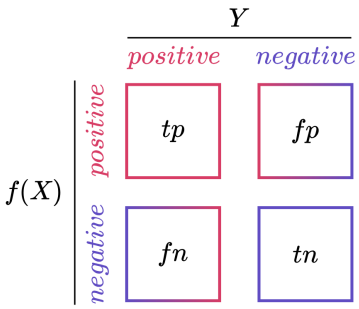


Fig. 10.4: A confusion matrix: columns defined by the gold labels Y, and rows defined by the predicted labels $f(X)$.

Task	Measure	Formulation	Description
Binary	Precision	$\frac{tp}{tp+fp}$	Of all the positive predictions, how many, in proportion, are correct?
Binary	Recall	$\frac{tp}{tp+fn}$	Of all the positive instances, how many, in proportion, have been correctly predicted?
Binary	$f1$-score	$h(\text{precision}, \text{recall})$	Harmonic mean of precision and recall
Binary	Accuracy	$\frac{tp+tn}{tp+tn+fp+fn}$	Accuracy

Table 10.4: Binary classification performance measures from confusion matrix

Confusion matrices are limited in their weighting, as any entry, e.g., true positives (tp), has a unitary weight in all performance measures. Yet, in some cases, some false weigh heavier than others, e.g., diagnosing a false positive cancer is far worse than diagnosing a false negative. Thus, we introduce the cost matrix, holding one weight per each entry in the confusion matrix, weighing it in performance measures.

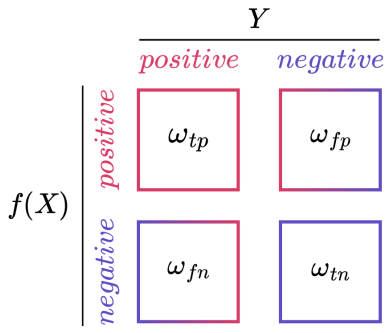


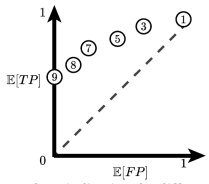
Fig. 10.5: Confusion Matrix having weights for each entry, forming a cost matrix.

For non-binary classification tasks, we can set one label as positive, and all others as negative, and build multiple confusion matrices, one per label, and then aggregate the results.

10.4.2 ROC Curve

Among all possible model solving binary classification tasks, some do not compute a binary label per se, rather a score

or probability α of a label. We need to to try multiple thresholds τ, each yielding different labellings. We may plot these results in a **ROC curve**.



A ROC curve, numbers indicating the different thresholds that have generated them. $\mathbb{E}[TP]$, $\mathbb{E}[FP]$, indicate the *rates* of true positives and false positives.

Fig. 10.6: Points are just threshold IDs

There are some points of interest in the ROC curve:

- $(0, 1)$ - **Perfect classifier** - no false positives, all true positives
- $(1, 0)$ - **Worst classifier** - all false instances are classified as positive, none of the positive instances is classified as positive
- $(1, 1)$ - **All positive classifier** - classifies all instances as positive, yielding all true positives, but also all false positives
- (x, x) - **Random classifier** - the diagonal line from $(0, 0)$ to $(1, 1)$
- $(x, x - \sigma)$ - **Worse than random classifier** - any point below the random classifier, this yields more errors than correct predictions
- **Area under the curve (AUC)** - the area under the ROC curve, a measure of performance. 0.5 is random, 1 is perfect.

Multiple Labels

What happens in case of multiple labels? We can build one ROC curve per label, setting it as positive, and all others as negative. Then, we can average the AUCs of each label to get a final performance measure.

10.5 Regression

Unlike classification tasks, regression tasks aim to predict continuous labels. Thus, we need different performance measures, we cannot count "positive" vs "negative" predictions.

Typical regression performance measures are:

- Mean Squared Error (MSE) - Mean error per instance, squared
- Max Squared Error (MaxSE) - Maximum error among all instances, squared
- R Squared (R^2) - Error over default model, where for default model we mean the average of all labels, so the error is the variance of the labels. Proportion of variance explained by the model

$$R^2 = 1 - \frac{e^2}{\sigma^2} == 1 - \frac{\sum_{i=1}^{n}(Y_i - f(x^i))^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

Part IV

Classification

Chapter 11

Decision Trees

11.1 Classification

Classification is a data mining task that assigns a class label to a record based on its attribute values.

We start off with **training set** of records, each characterized by a tuple (x, y), where x is the attribute set and y is the class label

- x: attribute, predictor, independent variable, input
- y: class, response, dependent variable, output

The goal of classification is to learn a function (often called "**model**") $f : X \to Y$ that maps each attribute set x into one of the predefined class labels y.

The **goal** of classification is to accurately predict the class labels of **unseen records** (i.e., records not in the training set).
A test set is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

- **Base classifier**: a classifier built from the training set
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-neighbor
  - Neural Networks
  - Deep Learning
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines
- **Ensemble classifier**: a classifier that combines multiple base classifiers to improve accuracy
  - Bagging
  - Boosting
  - Random Forests

11.2 Classification with Decision Trees

A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents the outcome of the test, and each leaf node holds a class label.

There are various algorithms to build decision trees, such as Hunt's algorithm (among the earliest ones), ID3, C4.5, CART, etc.

11.2.1 Hunt's Algorithm

Let D_t be the set of training records at node t. The general procedure of Hunt's algorithm is as follows:

- If D_t contains records that belong the same class y_t, then t is a leaf node labeled as y_t

- ◇ If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

11.2.1.1 Splitting data

The **split** consists in selecting an attribute A and partitioning the set of records D_t into subsets based on the values of A.
The idea is that if the value of an attribute is strongly correlated with the class label, then splitting on that attribute will create child nodes that are more homogeneous in terms of class distribution. In other terms, if the majority of records where $A = a_i$ belong to class C_j, then it is likely that an unseen record with $A = a_i$ also belongs to class C_j.

11.2.1.1.1 Example: Understanding Splits

Consider a simple dataset with car types and classes:

ID	CarType	Class
1	Sports	C1
2	Family	C2
3	Sports	C1
4	Luxury	C2
5	Family	C2

Table 11.1: Example dataset for splitting

Before the split: All 5 records are together in one node (the parent node), with mixed classes.

After splitting on "CarType = Sports":
- ◇ **Group 1 (CarType = Sports):** Records 1, 3 → majority class C1
- ◇ **Group 2 (CarType ≠ Sports):** Records 2, 4, 5 → majority class C2

We have therefore **divided/separated** (split!) the original dataset into two more homogeneous groups. This split is useful because it creates subsets where the class distribution is more uniform, making predictions more reliable.

Why is it called a "split"? Because we literally *split/divide/separate* the dataset into smaller subsets based on a condition on an attribute, similar to separating a deck of cards into red cards vs black cards.

Purpose of splitting:
1. **Reduce impurity:** Create groups where data is more homogeneous with respect to the target class
2. **Build the tree:** Each split creates branches in the decision tree
3. **Make predictions:** By following splits, we reach leaf nodes that predict the class

The goal is to find the **best split**, i.e., the one that:
- ◇ Best separates the classes (reduces impurity)
- ◇ Makes the resulting groups as "pure" as possible (all or most from the same class)

This is why we compute measures like the *Gini Index* (discussed later): to quantify how well a split separates the classes.

11.2.1.2 Example: Building a Decision Tree Step-by-Step

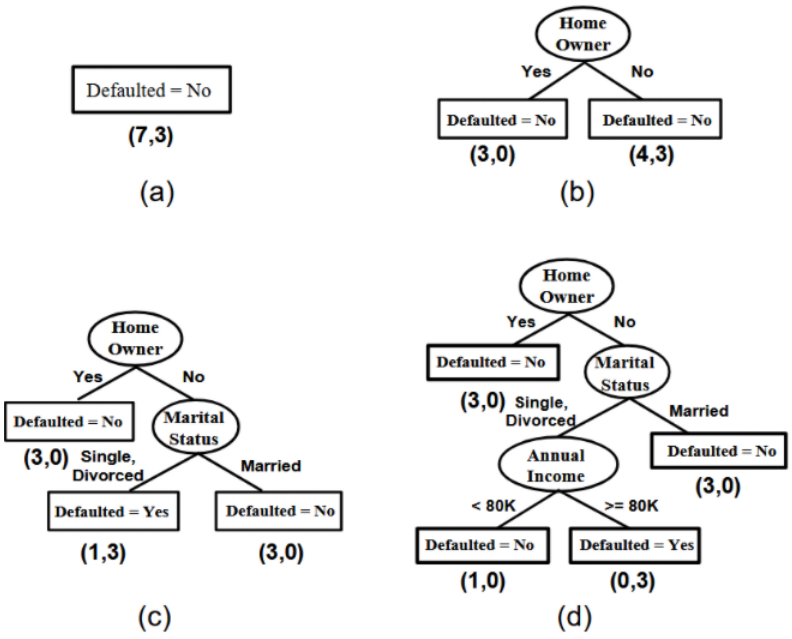


Fig. 11.1: Hunt's algorithm steps applied to Table 11.2

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Table 11.2: Example dataset for classification

Using the dataset in Table 11.2, we can illustrate how Hunt's algorithm builds a decision tree iteratively:

Step (a): The algorithm starts with all training records at the root node. The initial dataset contains 10 records with 7 instances of "No" and 3 instances of "Yes" for the Defaulted Borrower class. Since the node contains records from both classes, we need to split the data.

Step (b): The algorithm selects *Home Owner* as the first splitting attribute. This creates two branches:

- ◇ **Home Owner = Yes:** Contains 3 records, all with class label "Defaulted = No" (3,0). Since all records belong to the same class, this becomes a leaf node.
- ◇ **Home Owner = No:** Contains 7 records with 4 instances of "No" and 3 instances of "Yes" (4,3). Since this node contains mixed classes, further splitting is required.

Step (c): The algorithm continues by splitting the *Home Owner = No* branch using *Marital Status* as the splitting attribute:

- ◇ **Home Owner = No, Marital Status = Single or Divorced:** Contains 4 records with 1 instance of "No" and 3 instances of "Yes" (1,3). The majority class is "Yes", but the node is not pure.
- ◇ **Home Owner = No, Marital Status = Married:** Contains 3 records, all with class label "Defaulted = No" (3,0). This becomes a leaf node.

Step (d): The algorithm makes a final split on the remaining impure node (*Home Owner = No, Marital Status = Single or Divorced*) using *Annual Income* with threshold 80K:

- ◇ **Home Owner = No, Marital Status = Single/Divorced, Annual Income < 80K:** Contains 1 record with class "Defaulted = No" (1,0). This becomes a leaf node.

◇ **Home Owner = No, Marital Status = Single/Divorced, Annual Income ≥ 80K:** Contains 3 records with class "Defaulted = Yes" (0,3). This becomes a leaf node.

At this point, all leaf nodes contain records from a single class (pure nodes), and the decision tree is complete. The tree can now be used to classify new instances by traversing from the root to a leaf based on the attribute values of the test record.

11.2.1.3 Splitting Strategies

When building decision trees, there are different strategies for splitting nodes based on categorical attributes:

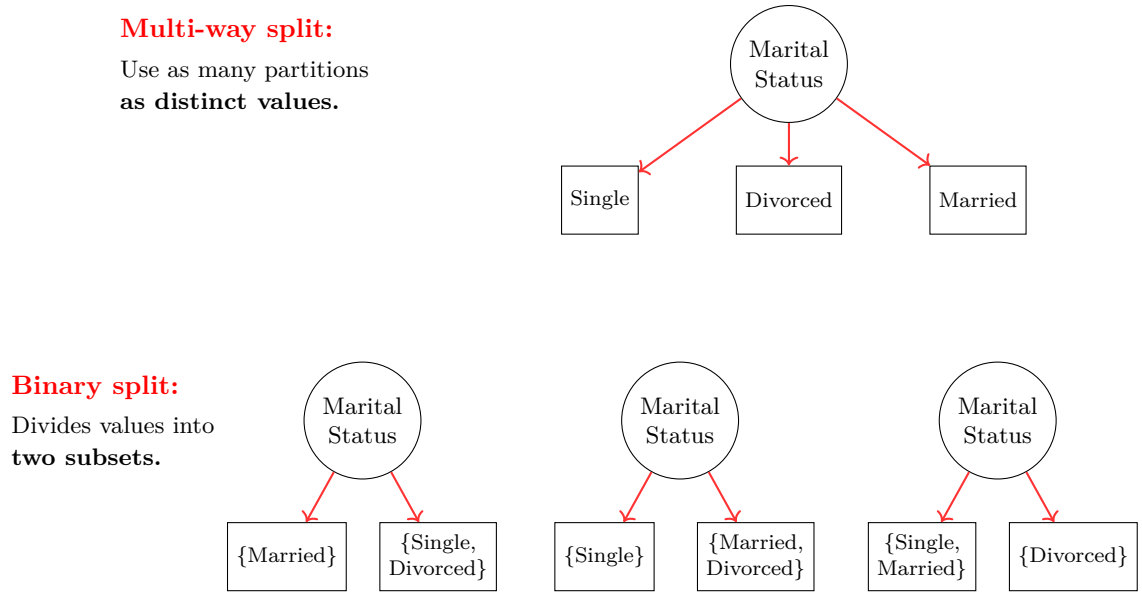


Fig. 11.2: Splitting strategies for categorical attributes: Multi-way vs Binary splits

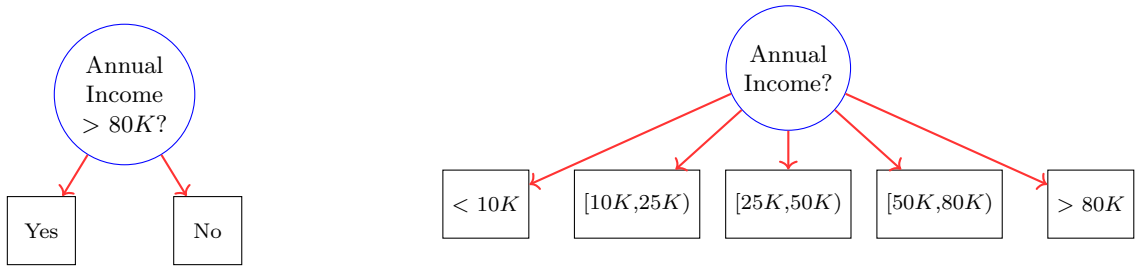
Multi-way split: Creates as many child nodes as there are distinct values in the attribute. For example, splitting on Marital Status with three values (Single, Divorced, Married) creates three branches. This approach is intuitive but can lead to data fragmentation, especially when attributes have many distinct values.

Binary split: Divides the attribute values into two subsets, creating only two child nodes. There are multiple ways to partition the values (e.g., {Married} vs {Single, Divorced}, or {Single} vs {Married, Divorced}, etc.). Binary splits are preferred in algorithms like CART as they create more balanced trees and are computationally more efficient, though they may require more splits to achieve the same separation.

11.2.1.4 Continuous attributes

There are two main approaches to handle continuous attributes in decision trees:

◇ **Discretization** to form an ordinal categorical attribute based on a set of **intervals**. Such intervals may be determined using domain knowledge or algorithms like equal-width or equal-frequency binning (percentiles), or clustering.
The discretization may be static, so performed only once at the beginning of the tree construction, or dynamic, so performed at each node during tree construction.
◇ **Binary splits** based on thresholding, e.g., Income $\leq$ 100K vs Income $>$ 100K.
All possible splits should be considered to find the best threshold that optimizes a certain criterion (e.g., information gain, Gini index, etc.). This may result in an intensive computation.



(i) Binary split

(ii) Multi-way split

Fig. 11.3: Splitting strategies for continuous attributes

11.2.1.5 Choosing the best split

Nodes with purer homogeneous class distributions are preferred. To measure the **impurity** of a node, we can use metrics such as **Gini Index**,**Information Gain**(?), **Entropy** or **Misclassification error**.

$$
\begin{aligned}
\textit{given:} \quad & p(i|t) = \text{proportion of records in node } t \text{ that belong to class } i \\
& \text{relative frequency of class } i \text{ at node } t \\
\textbf{Gini Index:} \quad & Gini(t) = 1 - \sum_{i=1}^{c} p(i|t)^2 \\
\textbf{Entropy:} \quad & Entropy(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t) \\
\textbf{Misclassification Error:} \quad & ME(t) = 1 - \max_i p(i|t)
\end{aligned}
$$

// TODO add when Misclass error has minimum/maximum

Practically, we compute the chosen impurity measure P before splitting attributes, and then we compute it again (M) after the split for each child node. M is usually a weighted average of the impurity measures of the child nodes.

At this point we may choose the attribute test condition that produces the highest gain $G = P - M$, or, equivalently, the lowest impurity measure after splitting (M).

11.2.2 Gini Index

11.2.2.1 Gini Index Examples

The following examples illustrate how the Gini Index measures node impurity:

C1	**0**
C2	**6**

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$
$$Gini = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$$

This represents a **pure node** where all instances belong to class C2. The Gini Index is 0, indicating no impurity.

C1	**1**
C2	**5**

$$P(C1) = 1/6 \quad P(C2) = 5/6$$
$$Gini = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

This node has low impurity with most instances belonging to C2.

This node has higher impurity with a less uniform distribution. Note that the Gini Index reaches its maximum value of 0.5 (for two classes) when the distribution is perfectly balanced, e.g., $P(C1) = P(C2) = 0.5$.

C1	2
C2	4

$P(C1) = 2/6 \quad P(C2) = 4/6$

$Gini = 1 - (2/6)^2 - (4/6)^2 = 0.444$

11.2.2.2 Gini for a collection of nodes

To compute the Gini Index for a collection of nodes after a split, we use a weighted average based on the number of instances in each child node.

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} Gini(i)$$

where:

- ◇ k: number of child nodes after the split
- ◇ n_i: number of instances in child node i
- ◇ n: total number of instances in the parent node before the split
- ◇ $Gini(i)$: Gini Index of child node i

Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT

There are some slides concerning optimizations on how the Gini index is computed when dealing with continuous attributes. These optimizations are not included here for brevity. By the way, they probably are already included in some Python library implementing decision trees ☺.

We may build a *count matrix* counting the occurences of each class for each possible split position, allowing efficient computation and comparison of Gini for all possible splits. The key idea for building the matrix is to sort the attribute values, get split positions which are inbetween consecutive values and then iterate through them to update class counts incrementally.

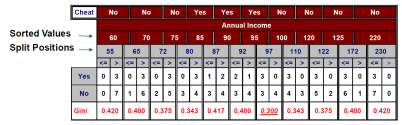


Fig. 11.7: Count Matrix with split positions

11.2.2.3 Entropy

$$Entropy(t) = -\sum_{j} p(j|t) \log_2 p(j|t)$$

Entropy is another impurity measure used in decision trees, particularly in the ID3 and C4.5 algorithms. It quantifies the uncertainty or randomness in the class distribution of a node.
It is pretty similar to Gini index, but it tends to be more sensitive to changes in the class distribution. For both of them, the lower the value, the purer the node.

11.2.2.4 Information Gain

$$GAIN_{split} = Entropy(parent) - \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$$

with n_i number of records in partition i, n total number of records before the split, and k number of partitions after the split.

This clearly tends to prefer splits that result in large number of partitions, each being small but pure

Used by ID3 and C4.5 algorithms

11.2.2.5 Overcoming impurity limitations

Note impurity limitations

Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure. This may lead to overfitting, as the model captures noise in the training data rather than the underlying patterns.

To mitigate this, we may consider also the number of child nodes created by a split when choosing the best attribute to split on. Or, as it happens in CART, we may use binary splits only, leading to have at most two child nodes per split.

To overcome this limitation, we may use **Gain Ratio**, which normalizes the information gain by the intrinsic information of a split.

$$GainRatio_{split} = \frac{GAIN_{split}}{SplitInfo_{split}}$$

$$SplitInfo_{split} = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

Adjusts Information Gain by the entropy of the partitioning itself, penalizing splits that create many small partitions.

This is used by C4.5 algorithm.

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

SplitINFO = 1.52

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

SplitINFO = 0.72

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

SplitINFO = 0.97

Fig. 11.4: Gain Ratio example

11.2.2.6 Comparing measures

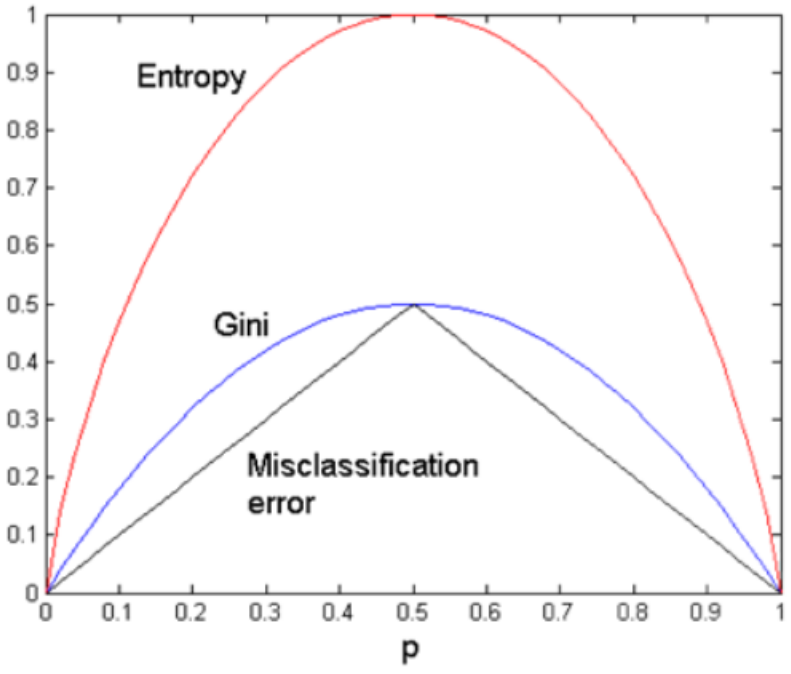


Fig. 11.5: Comparing Entropy, Gini and Misclassification error

The image displays a 2-class problem. We can easily see that there is consistency among the three measures: they all reach their minimum (0) when the node is pure (i.e., all instances belong to one class) and their maximum when the

classes are evenly distributed (i.e., $P(C1) = P(C2) = 0.5$). Furthermore, if a node N_1 has lower entropy than node, N_2, then the Gini index and error rate of N_1, will also be lower than that of N_2

In some cases Gini may get better, but the misclassification error may stay exactly the same.

11.3 Decision Trees Wrap Up

- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant or irrelevant attributes
  - **Redundant** attributes: provide no additional information because they are highly correlated with other attributes
  - **Irrelevant** attributes: provide no useful information for predicting the target class
- Inexpensive to construct
- Extremely fast at classifying unknown record
  - Construction cost: $O(MNlogN)$ having M attributes and N records
  - Testing cost: $O(\log N) = O(w)$ per record, where w is the depth of the tree
- Handle Missing Values

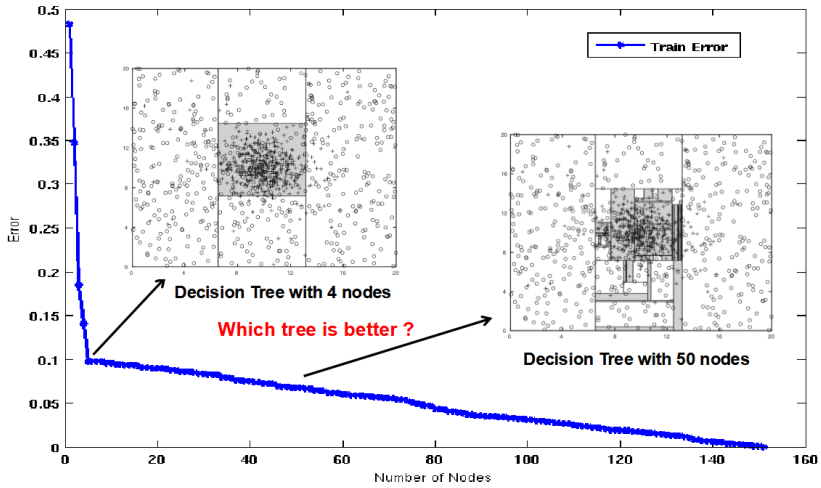


Fig. 11.6: The model on the right with 50 nodes probably overfits the data. This may lead to a high error rate on a test set

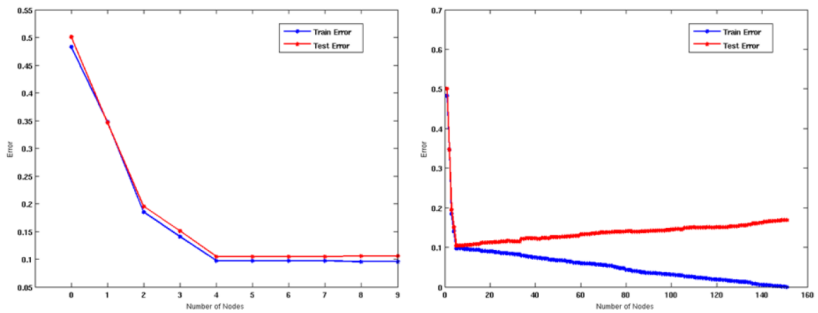


Fig. 11.7: Underfitting and overfitting

11.4 Model selection

Divide training data into two parts: a **Training set** used for model building and a **Validation set** used for estimating generalization error.

Note that the validation set is not the same as the test set. This is because the test set is used only at the very end to get an unbiased estimate of the generalization error.

This is a nice way to avoid overfitting, but the drawback is that there is less data available for training.

Definition 11.1 (Occam's Razor) *Given two models of similar generalization errors, one should prefer the simpler model over the more complex model.*

This is due to a complex model having a greater chance of being fitted accidentally by errors in data

Occam's Razor suggests to include a penalty for model complexity in the estimate of generalization error.

11.4.1 Evaluating Trees

Definition 11.2 (Pessimistic Error Estimate of decision tree T) *Pessimistic Error Estimate of decision tree T with k leaf nodes:*

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

- ◇ *$err(T)$: error rate on all training records*
- ◇ *Ω: Relative cost of adding a leaf node*
- ◇ *k: number of leaf nodes*
- ◇ *N_{train}: total number of training record*

Minimum Description Length (MDL) is another way to implement Occam's Razor.

Definition 11.3 (Minimum Description Length) *The best model is the one that minimizes the total description length:*

$$DL(Model) + DL(Data|Model)$$

where:

- ◇ *$DL(Model)$: number of bits to describe the model*
- ◇ *$DL(Data|Model)$: number of bits to describe the data when the model is known, i.e. to indicate which labels are predicted incorrectly by the model, the ones correctly predicted do not need to be specified, are already known and present in the labelled training set.*

11.4.1.1 Estimating Statistical Bounds

We can apply a statistical correction to the training error rate of the model that is indicative of its model complexity.

Doing so, we can obtain a statistical upper bound on the true error rate of the model.

To compute this statistical bound, we need the probability distribution of the training error, which can be either available from data or assumed.

In decision trees, the number of errors committed by a leaf node can be assumed to follow a **binomial distribution**. This is because:

- ◇ Each instance in the leaf is classified as either correct or incorrect (binary outcome)
- ◇ We assume instances are independent
- ◇ The probability of error is constant for all instances in the leaf

Given a leaf node with N instances and observed error rate e, we can compute a corrected error estimate $e'(N, e, \alpha)$ using the binomial confidence interval:

$$e'(N, e, \alpha) = \frac{e + \frac{z^2_{\alpha/2}}{2N} + z_{\alpha/2}\sqrt{\frac{e(1-e)}{N} + \frac{z^2_{\alpha/2}}{4N^2}}}{1 + \frac{z^2_{\alpha/2}}{N}}$$

where $z_{\alpha/2}$ is the critical value from the standard normal distribution at confidence level α (e.g., $z_{0.125} \approx 1.15$ for $\alpha = 0.25$).

The total generalized error for a tree T is then:

$$e'(T) = \sum_{\text{leaves } L} N_L \times e'(N_L, e_L, \alpha)$$

This statistical correction penalizes small leaf nodes (which have higher uncertainty) and helps decide whether a split actually improves the model or just fits noise in the training data.

11.4.1.2 Address overfitting

- ◇ Pre-pruning: stop growing the tree earlier
  - – Typical stopping conditions for a node are:
    - ○ Stop if all instances belong to the same class
    - ○ Stop if all the attribute values are the same
  - – More restrictive conditions:
    - ○ Stop if number of instances is less than some user-specified threshold
    - ○ Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
    - ○ Stop if estimated generalization error falls below certain threshold
- ◇ Post-pruning: grow the full tree, then remove nodes that do not help, following a bottom-up approach
  - – If generalization error improves after trimming, replace sub-tree by a leaf node.
  - – Class label of leaf node is determined from majority class of instances in the sub-tree
  - – Can use MDL for post-pruning

11.4.1.3 Observations

The number of possible decision trees can be very large, many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space.
That is **splitting** the records based on an attribute test that optimizes a certain criterion (e.g., information gain, Gini index, etc.) at each node.

" But then, how should training records be split at each node?
And, how should the splitting procedure stop? " —

We need a method to specify a **test condition** and a measure to evaluate the quality of a split, as well as a stopping criterion to prevent overfitting, which could be that if all records belong to the same class or if further splitting does not improve the model significantly.

11.4.2 Test conditions

Defining the test conditions to split the attributes depends on the type of attributes, whether they are categorical, binary, nominal, or continuous.

In the slides there are some examples of test conditions for different attribute types. They are not reported here for brevity.

11.5 Classification Model Evaluation

Model selection is performed during model building to ensure not to choose an overly complex model that would most likely overfit data.

11.5.1 Metrics for Performance evaluation

To evaluate the performance of a classification model we must focus on the predictive capability of a model. Rather than how fast it takes to classify or build models, scalability, etc.

We exploit a **confusion matrix** to evaluate the performance of a classification model, built like the one on the left.

- ◇ **True Positive (TP):** Correctly predicted positive instances

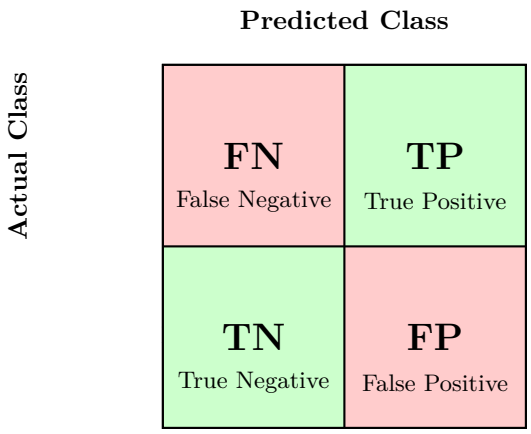


Fig. 11.8: Confusion Matrix structure

- ◇ **True Negative (TN):** Correctly predicted negative instances
- ◇ **False Positive (FP):** Incorrectly predicted as positive (Type I error)
- ◇ **False Negative (FN):** Incorrectly predicted as negative (Type II error)

The most widely used metric based on the matrix is the **accuracy**.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Note that accuracy may be misleading when dealing with imbalanced classes. Consider a dataset where 95% of instances belong to class A and only 5% to class B. A model that always predicts class A will have an accuracy of 95%, but it fails to identify any instances of class B.

Another metric is $C(i|j)$, the **cost of erroneously classifying** an instance of class j as class i.

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of misclassifying class j example as class i

Fig. 11.9: Cost Matrix: $C(i|j)$ represents the cost of misclassifying class j as class i

The cost matrix allows us to assign different penalties to different types of misclassifications. Typically, correct classifications have zero cost, while misclassifications have positive costs that reflect their real-world impact.

We have also:

- ◇ **Precision**:

$$Precision = \frac{TP}{TP + FP}$$

- ◇ **Recall**:

$$Recall = \frac{TP}{TP + FN}$$

- ◇ **F-measure**:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- ◇ **Weighted accuracy**:

$$Weighted\ Accuracy = \frac{TP + TN}{TP + TN + \beta \times FP + (1 - \beta) \times FN}$$

or

$$Weighted\ Accuracy = \frac{w_1 \cdot TP + w_2 \cdot TN}{w_1 \cdot TP + w_2 \cdot TN + w_3 \cdot FP + w_4 \cdot FN}$$

where β is a user-defined parameter that adjusts the weight of false positives and false negatives.

11.6 Methods for Performance evaluated

It is key that test data is **not** used in any way to create the classifier.

Having **training**, **validation** and **test** sets is a good practice to avoid overfitting and get an unbiased estimate of the model's performance.

After having built the basic strucutre of the model, validation data may be used to tune hyperparameters and select the best model.

Finally, the test set is used only once at the very end to get an unbiased estimate of the generalization error.

In general, the more data we have for training, the better the model will perform, but there is a threshold after which adding more data is not that impactful

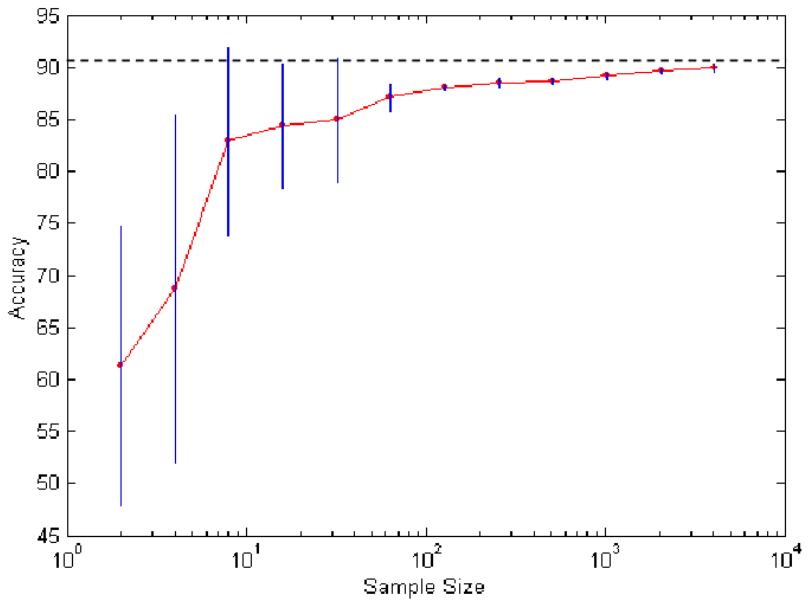


Fig. 11.10: Learning Curve displaying how accuracy changes with varying sample size.

11.6.1 Estimation methods

The following methods can be used to estimate the performance of a classification model. They consists in splitting the available dataset into training and test sets in different ways, leading to different trade-offs between bias and variance of the performance estimate.

- ◇ The **holdout** method involves splitting the dataset into two separate sets: 2/3 for training set and 1/3 for a test set.
- ◇ **Random subsampling** involves randomly splitting the dataset into training and test sets multiple times, training the model on each training set, and evaluating it on the corresponding test set. The final performance is averaged over all iterations.
  This is essentially a repeated holdout method.
- ◇ **Cross-validation** involves partitioning the dataset into k subsets (folds). The model is trained on k-1 folds and tested on the remaining fold. This process is repeated k times, with each fold serving as the test set once. The final performance is averaged over all k iterations.
- ◇ **Stratified sampling** is a variation of cross-validation that ensures each fold has a representative distribution of classes, which is particularly important for imbalanced datasets.
  Instead of blindly assigning 2/3 and 1/3 of instances to training and test sets, stratified sampling aims also at maintaining the same class proportions in both sets as in the original dataset.

- ◇ **Bootstrap** involves creating multiple training sets by sampling with replacement from the original dataset. Each bootstrap sample is used to train the model, and the instances not included in the sample (out-of-bag instances) are used for testing. The final performance is averaged over all bootstrap samples.

11.7 Methods for Model Comparison

To compare two models, we can use statistical tests to determine if the observed differences in performance are significant or due to random chance.

Comparing models is key when tuning hyperparameters, selecting features, or choosing between different algorithms.

11.7.1 ROC - Receiver Operating Characteristic

ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
A model provides a probability of an instance belonging to the positive class, and by varying the threshold for classification, we can obtain different TPR and FPR values.

If threshold $t = 0$ implies that, in order to classify an instance as positive, the predicted probability must be equal to 1, then all instances will be classified as negative, leading to $TPR = 0$ and $FPR = 0$; explaining why all ROC curves start at the origin $(0, 0)$.
Conversely, if threshold $t = 1$ implies that all instances will be classified as positive, leading to $TPR = 1$ and $FPR = 1$; explaining why all ROC curves end at the point $(1, 1)$.
When varying the threshold from 0 to 1, we move from the point $(0, 0)$ to $(1, 1)$ on the ROC curve. Depending on the model, different thresholds may yield different TPR and FPR values, resulting in different ROC curves.
Note that lowering the threshold increases FPR, they are essentially inversely related. This is due to the fact that as the threshold gets closer to zero, all instances are classified as positive, so the number of false positives can only increase.

The area under the ROC curve (**AUC**) provides a single measure of overall model performance; the bigger, the better.
Every point on the ROC curve represents the performance of each classifier, each having different trade-off between sensitivity and specificity.

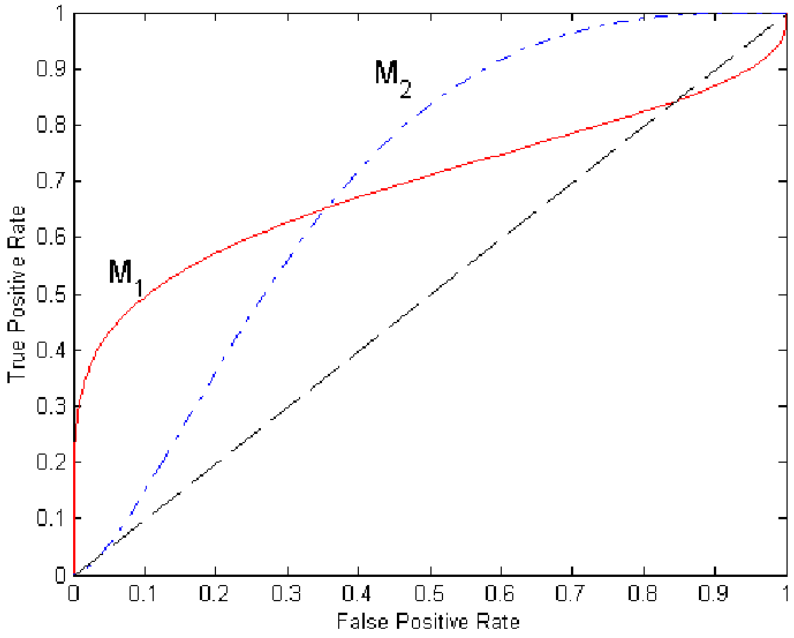


Fig. 11.11: ROC curve example

Here the diagonal line represent random guessing, so on average both TPR and FPR are equal. If a model performs below the diagonal line is worse than random guessing, it's essentially predicting more wrong classes than correct ones.

The two models in the figure perform similarly, they are better for different things. M_1 has high TPR for high thresholds (so for low FPR), while M_2 has high TPR for low thresholds (so for high FPR).

11.7.2 Significance Testing

Given two models:

- ◇ Model M1: $accuracy = 85\%$, tested on 30 instances
- ◇ Model M2: $accuracy = 75\%$, tested on 5000 instances

Is M1 really better than M2? Can the difference in performance measure be explained as a result of random fluctuations in the test set?
We can use significance testing to answer these questions.

11.7.2.1 Confidence Interval for accuracy

We can determine a confidence interval for accuracy, in order to estimate the range within which the true accuracy of the model lies, with a certain level of confidence.

Prediction can be regarded as a Bernoulli trial, where each prediction is either correct (success) or incorrect (failure). A collection of Bernoulli trials has a Binomial distribution. There are some practical formulas in the slides which ain't reported here.

Chapter 12

Rule-based Classification

12.1 Introduction to Rule-based Classifiers

Rule-based classifiers classify records by using a collection of "if...then..." rules. A **rule** has the form $(Condition) \rightarrow y$, where **Condition** is a conjunction of tests on attributes and **y** is the class label.

Examples of classification rules include $(Blood\ Type = Warm) \wedge (Lay\ Eggs = Yes) \rightarrow Birds$ and $(Taxable\ Income < 50K) \wedge (Refund = Yes) \rightarrow Evade = No$.

12.1.1 Example of Rule-based Classifier

Consider the following rule set:

- R1: $(Give\ Birth = no) \wedge (Can\ Fly = yes) \rightarrow Birds$
- R2: $(Give\ Birth = no) \wedge (Live\ in\ Water = yes) \rightarrow Fishes$
- R3: $(Give\ Birth = yes) \wedge (Blood\ Type = warm) \rightarrow Mammals$
- R4: $(Give\ Birth = no) \wedge (Can\ Fly = no) \rightarrow Reptiles$
- R5: $(Live\ in\ Water = sometimes) \rightarrow Amphibians$

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

Fig. 12.1: Mammals example table

12.1.2 Application of Rule-Based Classifier

A rule r **covers** an instance x if the attributes of the instance satisfy the condition of the rule. For example, the rule R1 covers a hawk (classifying it as a Bird), while the rule R3 covers the grizzly bear (classifying it as a Mammal).

Rule Coverage and Accuracy

The **coverage** of a rule is the fraction of records that satisfy the antecedent of the rule, while the **accuracy** is the fraction of records that satisfy the antecedent that also satisfy the consequent of the rule. For example, the rule $(Status = Single) \rightarrow No$ might have a coverage of 40% and an accuracy of 50%.

12.2 Characteristics of Rule Sets

When applying a rule-based classifier, different scenarios can occur. A lemur triggers rule R3 and is classified as a mammal. A turtle triggers both R4 and R5, which requires a conflict resolution strategy. A dogfish shark triggers none of the rules, necessitating the use of a default class.

12.2.1 Mutually Exclusive and Exhaustive Rules

A classifier contains **mutually exclusive rules** if the rules are independent of each other, meaning every record is covered by at most one rule.

The classifier has **exhaustive coverage** if it accounts for every possible combination of attribute values, ensuring that each record is covered by at least one rule.

When rules are **not** mutually exclusive, a record may trigger more than one rule. This conflict can be resolved using either an **ordered rule set** or an unordered rule set with **voting** schemes.

When rules are not exhaustive, a record may not trigger any rules, and the solution is to use a default class for such cases.

12.2.2 Ordered Rule Set

In an ordered rule set, rules are rank ordered according to their priority. Such an ordered rule set is known as a **decision list**. When a test record is presented to the classifier, it is assigned to the class label of the highest ranked rule it has triggered. If none of the rules fired, it is assigned to the default class (typically the majority class).

12.2.2.1 Rule Ordering Schemes

Rule-based ordering: Individual rules are ranked based on their quality measured by accuracy, coverage, or size (number of attribute tests in the rule antecedent).
The resulting rule set is known as a **decision list**, where the record X is classified by the rule with the highest priority and any other rule that satisfies is ignored. Each rule in a decision list implies the negation of the rules that come before it in the list, making rules in a decision list more difficult to interpret.

Class-based ordering: Rules that belong to the same class appear together. The classes are sorted in order of decreasing "importance", such as by decreasing order of prevalence, or alternatively based on the misclassification cost per class. Within each class, the rules are not ordered.

Unordered rules: These use a voting schema to resolve conflicts.

12.3 Building Classification Rules

There are two main approaches for building classification rules. The **direct method** extracts rules directly from data (examples include RIPPER, CN2, and Holte's 1R), while the **indirect method** extracts rules from other classification models such as decision trees or neural networks (e.g., C4.5rules).

12.3.1 Direct Method: Sequential Covering

Algorithm:

1. Start from an empty rule
2. For each class:
    i. Grow a rule using the Learn-One-Rule function
    ii. Remove training records covered by the rule
    iii. Repeat Step until stopping criterion is met

12.3.1.1 Learn-One-Rule Function

The goal of the Learn-One-Rule function is to extract a classification rule covering many positive records and none (or few) negative ones. Since finding the optimal rule requires high computational time, a greedy strategy is employed by refining an initial rule based on some evaluation measure.

Rules are extracted one class at a time, and the criterion for deciding the order of the class to consider depends on class prevalence and misclassification error for a given class.

12.3.1.2 Rule Growing Strategies

There are two common strategies exist for growing rules.

The **general-to-specific** approach starts with an empty rule { } and iteratively adds conjuncts like $Refund = No$, $Status = Single$, $Income > 80K$ until the rule is sufficiently specific.
Conversely, the **specific-to-general** approach starts with a specific rule covering a selected record (e.g., $Refund = No, Status = Single, Income = 85K$ with Class=Yes) and then generalizes it by removing conditions.

12.4 Rule Evaluation for Growing Rules

12.4.1 Based on Rule Coverage

Evaluation measures can be based on the coverage of the rule with respect to records with class c.

$$Accuracy = \frac{n_c}{n}$$
$$Laplace = \frac{n_c + 1}{n + k}$$
$$M - estimate = \frac{n_c + k \cdot p}{n + k}$$

- n_c: number of records covered by the rule with class c
- n: total number of records covered by the rule
- k: number of classes
- p: prior probability (of class c?)

12.4.2 Based on Support Count: FOIL's Information Gain

FOIL's Information Gain compares an initial rule $R_0 : \{ \} \Rightarrow class$ with a rule after adding a conjunct $R_1 : \{A\} \Rightarrow class$:

$$Gain(R_0, R_1) = p_1 \times \left[\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right]$$

where p_0 and n_0 are the number of positive and negative instances covered by R_0, respectively, and p_1 and n_1 are the corresponding counts for R_1. FOIL (First Order Inductive Learner) is an early rule-based learning algorithm.

12.4.3 Based on Statistical Test: Likelihood Ratio Statistic

Given a rule, we can compute the Likelihood ratio statistic $R = 2 \sum_i f_i \log \frac{f_i}{e_i}$, where f_i is the number of records covered by the rule and e_i is the expected frequency of a rule that makes random predictions. A large R value suggests that the number of correct predictions made by the rule is significantly larger than that expected by random guessing.

Example:

Dataset: 60 positive records and 100 negative records

Rule 1: covers 50 positive records and 5 negative examples

- Expected frequency for the positive class is $e_+ = 55 \times 60/160 = 20.625$
- Expected frequency for the negative class is $e_- = 55 \times 100/160 = 34.375$

Rule 2: covers 2 positive records and no negative examples

- Expected frequency for the positive class is $e_+ = 2 \times 60/160 = 0.75$
- Expected frequency for the negative class is $e_- = 2 \times 100/160 = 1.25$

12.5 Direct Method: RIPPER

12.5.1 For 2-class Problem

For a 2-class problem, RIPPER chooses one of the classes as the positive class and the other as the negative class. It then learns rules for the positive class, while the negative class becomes the default class.

12.5.2 For Multi-class Problem

For multi-class problems, RIPPER orders the classes according to increasing class prevalence (the fraction of instances that belong to a particular class). It learns the rule set for the smallest class first, treating the rest as the negative class, then repeats the process with the next smallest class as the positive class.

12.5.3 Growing a Rule in RIPPER

- ◇ RIPPER starts from an empty rule and adds conjuncts as long as they improve FOIL's information gain.
- ◇ It stops when the rule no longer covers negative examples and then immediately prunes the rule using incremental reduced error pruning.
- ◇ The measure for pruning is $v = (p - n)/(p + n)$, where p is the number of positive examples covered by the rule in the validation set and n is the number of negative examples.
- ◇ The pruning method deletes any final sequence of conditions that maximizes v.

12.5.4 Building a Rule Set in RIPPER

RIPPER uses a **sequential covering algorithm** that finds the best rule covering the current set of positive examples and eliminates both positive and negative examples covered by the rule.

Each time a rule is added to the rule set, the algorithm computes the new description length and stops adding new rules when this description length is d bits longer than the smallest description length obtained so far.

12.5.5 Minimum Description Length (MDL)

The Minimum Description Length principle is expressed as $Cost(Model, Data) = Cost(Data|Model) + \alpha \times Cost(Model)$, where Cost is the number of bits needed for encoding. The goal is to search for the least costly model. Here, $Cost(Data|Model)$ encodes the misclassification errors, while $Cost(Model)$ uses node encoding (number of children) plus splitting condition encoding.

12.6 Indirect Method: C4.5rules

12.6.1 Algorithm

C4.5rules extracts rules from an unpruned decision tree.
For each rule $r : A \to y$:

- ◇ considers an alternative rule $r' : A' \to y$ where A' is obtained by removing one of the conjuncts in A.
- ◇ The algorithm compares the pessimistic error rate for r against all r's and prunes the rule if one of the alternative rules has a lower pessimistic error rate.
- ◇ This process repeats until generalization error can no longer be improved.
- ◇ After removing duplicate rules, C4.5rules uses class ordering instead of ordering individual rules.
- ◇ For multi-class problems, rather than assigning a default class to test records not covered by any rule, C4.5rules looks for the rule that most closely matches the record.

12.6.2 Pessimistic Error Estimate

The Pessimistic Error Estimate of a rule set T with k rules is computed as

$$err(T) + W \times \frac{k}{N_{train}}$$

where $err(T)$ is the error rate on all training records, W is a trade-off hyper-parameter representing the relative cost of adding a rule, k is the number of rule nodes, and N_{train} is the total number of training records.

This measure can be used to evaluate whether pruning a rule improves the overall performance of the rule set.

12.6.3 Class Ordering in C4.5rules

C4.5rules orders subsets of rules rather than individual rules, using class ordering. Each subset is a collection of rules with the same rule consequent (class).

The algorithm computes the description length of each subset as

$$L(error) + g \times L(model)$$

where g is a parameter that takes into account the presence of redundant attributes in a rule set (with a default value of 0.5).

12.7 Advantages of Rule-Based Classifiers

Rule-based classifiers have characteristics quite similar to decision trees: they are as highly expressive as decision trees, easy to interpret, have comparable performance, and can handle redundant attributes.

Additionally, they are better suited for handling imbalanced classes, though they are harder to handle missing values in the test set.

12.8 Example: C4.5 vs C4.5rules vs RIPPER

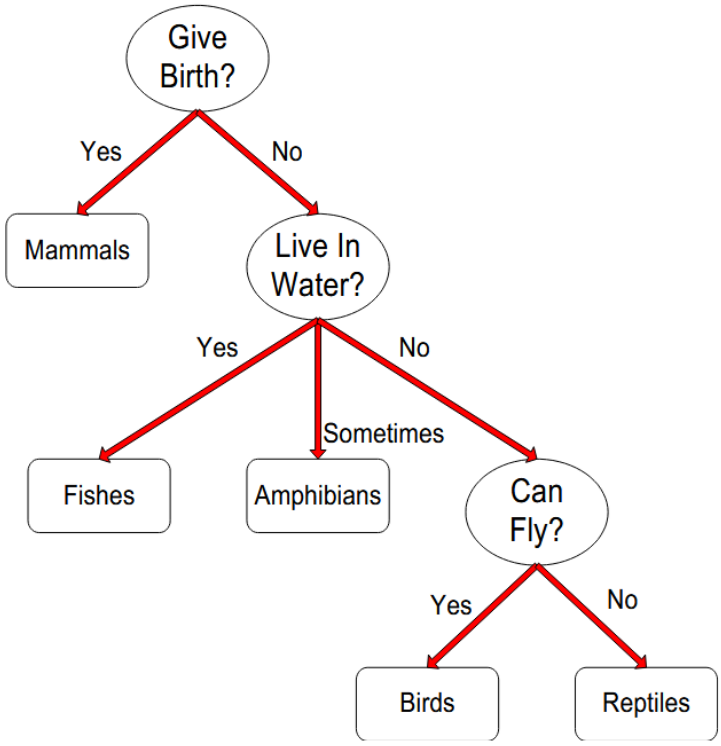


Fig. 12.1: Decision tree for Table **??**

12.8.1 C4.5rules

- $(Give\ Birth = No, Can\ Fly = Yes) \rightarrow Birds$
- $(Give\ Birth = No, Live\ in\ Water = Yes) \rightarrow Fishes$
- $(Give\ Birth = Yes) \rightarrow Mammals$
- $(Give\ Birth = No, Can\ Fly = No, Live\ in\ Water = No) \rightarrow Reptiles$
- $() \rightarrow Amphibians$

12.8.2 RIPPER

- $(Live\ in\ Water = Yes) \rightarrow Fishes$
- $(Have\ Legs = No) \rightarrow Reptiles$
- $(Give\ Birth = No, Can\ Fly = No, Live\ In\ Water = No) \rightarrow Reptiles$
- $(Can\ Fly = Yes, Give\ Birth = No) \rightarrow Birds$
- $() \rightarrow Mammals$

12.8.3 Performance Comparison

When comparing C4.5, C4.5rules, and RIPPER, the algorithms produce different classification results. The confusion matrices reveal that RIPPER may have different error patterns, particularly in distinguishing between classes like Amphibians, Reptiles, and Mammals. While C4.5 and C4.5rules tend to produce similar results due to their shared origin from decision trees, RIPPER's sequential covering approach leads to a different rule structure and potentially different classification behavior.

Chapter 13

Decision Tree Simulation

index	state	contract	sex	calls	churn
1	it	C	F	>10	Y
2	ge	T	M	<=10	N
3	it	T	M	>10	Y
4	ge	Y	F	<=10	N
5	ge	T	F	>10	N
2	ge	C	M	>10	Y
2	ge	C	M	<=10	Y
2	ge	Y	F	<=10	Y
2	ge	Y	M	>10	N
2	ge	T	F	<=10	N
2	ge	Y	F	>10	Y
2	it	C	F	>10	N

Table 13.1: Customer churn dataset

13.1 Simulating DT

Initial dataset statistics:

- Y = 6
- N = 6
- ME = 6/12

The formula for calculating the expected misclassification error after a split is given by:

$$E = \sum_{i=1}^{k} ME(S_i) \cdot \frac{|S_i|}{|S|}$$

where:

- k is the number of subsets created by the split,
- $ME(S_i)$ is the misclassification error of subset S_i,
- $|S_i|$ is the number of instances in subset S_i,
- $|S|$ is the total number of instances before the split.

The misclassification error for a subset is calculated as:

$$ME(S_i) = 1 - \frac{\max(Y_i, N_i)}{|S_i|}$$

where:

- Y_i is the number of positive instances in subset S_i,
- N_i is the number of negative instances in subset S_i.

	IT	GE
Y	2	4
N	1	5

Table 13.2: Split by STATE attribute

13.1.1 Splitting by STATE

$$ME(IT) = 1/3 = 1 - \max\left(\frac{2}{3}, \frac{1}{3}\right)$$
$$ME(GE) = 4/9 = 1 - \max\left(\frac{5}{9}, \frac{4}{9}\right)$$
$$E = ME(IT) \cdot p(IT) + ME(GE) \cdot p(GE)$$
$$E = \frac{1}{3} \cdot \frac{3}{12} + \frac{4}{9} \cdot \frac{9}{12} = \frac{5}{12}$$

13.1.2 Splitting by CONTRACT

13.1.2.1 3-way split: C, T, Y

	C	T	Y
Y	3	1	2
N	1	3	2

Table 13.3: 3-way split by CONTRACT attribute

$$ME(C) = 1/4$$
$$ME(T) = 1/4$$
$$ME(Y) = 2/4$$
$$E = \frac{1}{4} \cdot \frac{4}{12} + \frac{1}{4} \cdot \frac{4}{12} + \frac{2}{4} \cdot \frac{4}{12} = \frac{4}{12}$$

13.1.2.2 2-way split: CT, Y

	CT	Y
Y	4	2
N	4	2

Table 13.4: 2-way split by CONTRACT: CT vs Y

$$ME(CT) = 4/8$$
$$ME(Y) = 2/4$$
$$E = \frac{4}{8} \cdot \frac{8}{12} + \frac{2}{4} \cdot \frac{4}{12} = \frac{6}{12}$$

13.1.2.3 2-way split: C, TY

$$ME(C) = 1/4$$
$$ME(TY) = 3/8$$
$$E = \frac{1}{4} \cdot \frac{4}{12} + \frac{3}{8} \cdot \frac{8}{12} = \frac{4}{12}$$

13.1.2.4 2-way split: CY, T

$$ME(CY) = 3/8$$
$$ME(T) = 1/4$$
$$E = \frac{3}{8} \cdot \frac{8}{12} + \frac{1}{4} \cdot \frac{4}{12} = \frac{4}{12}$$

	C	TY
Y	3	3
N	1	5

Table 13.5: 2-way split by CONTRACT: C vs TY

	CY	T
Y	5	1
N	3	3

Table 13.6: 2-way split by CONTRACT: CY vs T

13.1.3 Splitting by SEX

$$ME(F) = 3/7$$
$$ME(M) = 2/5$$
$$E = \frac{3}{7} \cdot \frac{7}{12} + \frac{2}{5} \cdot \frac{5}{12} = \frac{5}{12}$$

13.1.4 Splitting by CALLS

$$ME(> 10) = 2/5$$
$$ME(<= 10) = 3/7$$
$$E = \frac{2}{5} \cdot \frac{5}{12} + \frac{3}{7} \cdot \frac{7}{12} = \frac{5}{12}$$

13.2 Sequential Pattern Mining

13.2.1 Exercise Setup

Given the sequential rule: $\{B\} \rightarrow \{CD\}$ with constraint MIN-GAP = 1.

For each of the sequences below, list all the subsequences that satisfy the constraint (if any).

13.2.2 Sequence 1

{ABF}	{C}	{CDF}	{E}	{CD}
t_0	t_1	t_2	t_3	t_4

No constraints:

- $\langle t_0, t_2 \rangle$
- $\langle t_0, t_4 \rangle$

With min-gap = 1:

- $\langle t_0, t_2 \rangle$
- $\langle t_0, t_4 \rangle$

13.2.3 Sequence 2

{AB}	{C}	{AB}	{CD}
t_0	t_1	t_2	t_3

No constraints:

- $\langle t_0, t_2 \rangle$
- $\langle t_0, t_3 \rangle$

With min-gap = 1:

- $\langle t_0, t_3 \rangle$

	F	**M**
Y	3	3
N	4	2

Table 13.7: Split by SEX attribute

	>10	**<=10**
Y	2	4
N	3	3

Table 13.8: Split by CALLS attribute

13.2.4 Sequence 3

{AF}	{BC}	{AB}	{E}	{D}
t_0	t_1	t_2	t_3	t_4

No constraints:

- ◇ None

With min-gap = 1:

- ◇ None

13.2.5 Sequence 4

{A,B,F}	{A,C}	{A,B,D}	{C}	{C,D}
t_0	t_1	t_2	t_3	t_4

Rule: $\{A\} \rightarrow \{C, D\}$

No constraints:

- ◇ $\langle t_0, t_4 \rangle$
- ◇ $\langle t_1, t_4 \rangle$
- ◇ $\langle t_2, t_4 \rangle$

Rule: $\{A\} \rightarrow \{D\}$

Chapter 14

Supervised Tasks

Supervised learning tasks can be grouped into two main categories:

- **Classification:** the target variable is categorical (e.g., spam vs. not spam)
- **Regression:** the target variable is continuous (e.g., house price)

We will focus on **classification** tasks.

14.1 Splitting trees

14.1.1 Introduction to classification

There are three main approaches to classification.

- **Trees** - Learning a space partitition separating data according to its label
- **Linear** - Predicting the label through a linear combination of the input features
- **Neural** - Predicting the label through **computational graphs**

14.1.2 Split function

Inducing Decision Trees depends roughly on two factors:

- Split function f_i: how do I route instances in my subtrees?
- Split loss L: how do I measure the quality of a split?

Univariate split functions are the most common, they test a single attribute at a time. When plotted, they correspond to axis-aligned splits. They are highly interpretable and their complexity is easily manageable, but they have limited expressiveness.

$$f_i(x) \equiv x_i \leq \Theta_i$$

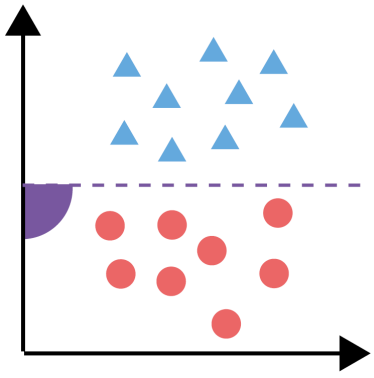


Fig. 14.2: Univariate split

Multivariate split functions test multiple attributes at a time. When plotted, they correspond to oblique splits; in fact, they are also called **oblique** splits. They are more expressive than univariate splits, but less interpretable and more complex to manage.

$$f_i(x) \equiv x_\Theta + \Theta^0$$

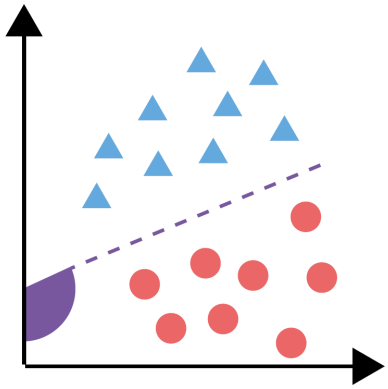


Fig. 14.3: Multivariate split

Probabilistic split functions route instances to subtrees with a certain probability. They are mostly used in **soft** decision trees, where instances can be routed to multiple subtrees at once.
Instances are routed to all leaves in the tree, accumulating probability density at each node. The prediction can then be given by the leaf with higher probability mass, or average class probabilities across leaves.

$$f_i(x) \equiv p_i(x; \theta_i)$$

p_i is the probability function parametrized by θ_i at node i which we apply on the input x. The output represents the probability of routing x to one of the two subtrees. Parameters θ_i are learned during training.

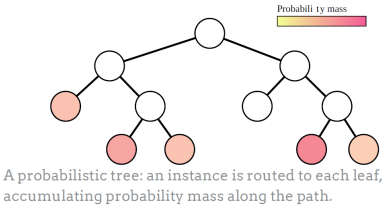


Fig. 14.4: Probabilistic split

These work best on non-relational data, such as images. However, they suffer from interpretability issues, and are more complex to train.

In probabilistic trees (*soft* trees) every node is, for instance, the probability that x matches a Normal,Gaussian, Binomial, etc. distribution.
Every edge has a probability of being traversed by instance x, i.e. probability for each edge, hence probability masses for each leaf change for each instance!
Every leaf represents a probability distribution over classes, that is "if you're here, you have $p(A)$ of being in class A, and $p(B)$ of being in class B", because each leaf —i suppose— represents a set of examples belonging to different classes.
There are two ways to predict the class of an instance:

- Pick the leaf with the highest probability mass, and use its class distribution to predict. In other words, follow the most probable path, and then check $max(p_l(A), p_l(B))$ in that leaf.
- Instead of picking a path, consider all possible endings. Average the class distributions of all leaves, weighted by their probability mass for the instance.

$$P(A) = \sum_{l \in leaves} p_l(A) \cdot mass_l(x)$$
$$P(B) = \sum_{l \in leaves} p_l(B) \cdot mass_l(x)$$

And then pick the class with maximum probability $max(P(A), P(B))$.

14.2 Linear Models

Linear models are simple, generally interpretable, family of models. Typically used for tasks on

- **Representation**, e.g., PCA, PLA: can I find an alternative, linear representation of my data?
- **Classification**, e.g., Linear *regression*: can I predict a variable as a linear model of my data?

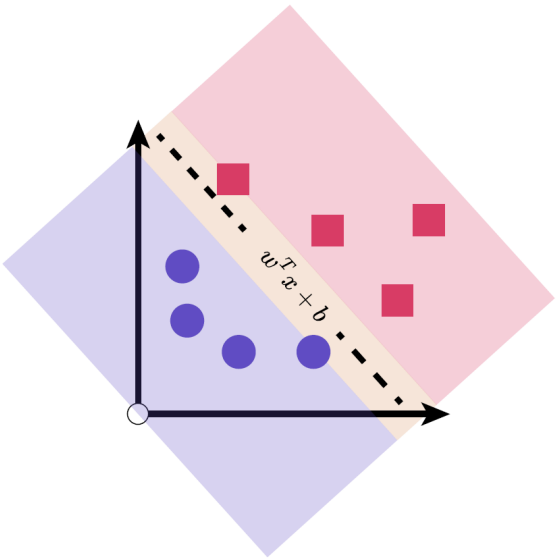


Fig. 14.1: Two classes dataset and a linear model separating them

14.2.1 Formally

A linear model makes predictions according to a linear combination of the input features. When parametrized by ω it has the form:

$$f \equiv \omega^T x$$

It can be leveraged for classification by applying an activation function (threshold) to the output:

$$f(x) \equiv \sigma(\omega^T x)$$

They are interpretable, actionable (each feature may be tuned independently), and easily optimizable (the loss function is convex, allowing for efficient optimization and few local minima) but have limited expressiveness.

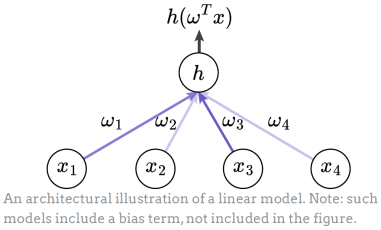


Fig. 14.2: An architectural illustration of a linear model. Note: such models include a bias term, not included in the figure.

The interpretability of linear models comes from their additive nature: each component has a measurable and *independent*[1] contribution $\omega_i x_i$. Their limitation also comes from this, as there is no real learned representation of the data.

[1] Indenpendency here refers to the model, there still may be actual dependency between data

Generalized Addictive Models (GAMs) extend linear models by applying non-linear functions to each feature before combining them additively.

$$f(x) \equiv \sum^{m} \omega_i f_i(x_i) + \omega_0$$

Each feature enjoys a learned representation given by a parametric *shape* function f_i of arbitrary complexity.

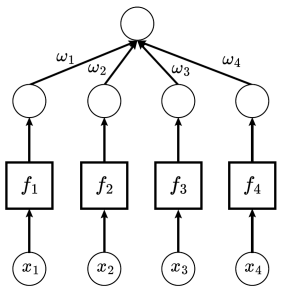


Fig. 14.3: Generalized Addictive Model

14.3 Bagging

As highlighted by the bias-variance decomposition, learning algorithms can incur in a variance in terms of generalization. We have seen two strategies to mitigate this:

- ◇ **Cross validation**: reduce variance by increasing samples

◇ **Regularization**: reduce variance by reducing capacity

There exists a third way, somewhat related, approach: leverage variance.

A "bag" essentially is a bootstrap sample of the original dataset: a dataset created by sampling with replacement from the original data.

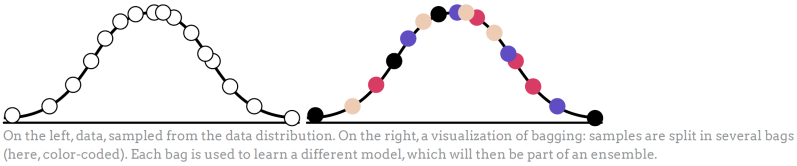


Fig. 14.2: Bagging process illustration

Bagging (Bootstrap Aggregating) is an ensemble method which aims to reduce variance by combining multiple models trained on different samples of the data. The idea is to create multiple datasets by sampling with replacement from the original dataset (bootstrapping), train a model on each dataset, and combine their predictions (aggregating).
This works because the variance of the average of multiple independent models is lower than the variance of a single model.

$$f_\Theta = \sum_{i=1}^{T} \eta f_i(x) \qquad \eta \in \mathbb{R}$$

Bagging is particularly effective for high-variance, low-bias models, such as decision trees. In fact, the underlying assumption is that models retain a low-enough bias to be accurate on their respective bags.

The overall computational complexity depends on the complexity of each model, and the number of models T. For a uniform learning time of $\mathcal{O}(c)$ of k bags, the cost is $\mathcal{O}(k \cdot c) \cong \mathcal{O}(c)$.
For non-uniform costs, there is an upper bound of $\max_{c \in C} \mathcal{O}(c)$.

14.3.1 Random Forests

Decision Trees show a moderate variance, and thus are a perfect candidate. **Random Forests** sample a set of bags by sampling both *instances* and *features*, training a decision tree on each bag, and aggregating their predictions (by majority vote for classification, or averaging for regression).

Note that Random Forests sample bags with **replacement**, so some instances may appear multiple times in a bag, while others may not appear at all.

As a bagging model, the computational complexity is given by the complexity of the single tree.

Pros
- ◇ Simple definition with high variance models
- ◇ Interpretable-ish results
- ◇ Fast learning

Cons
- ◇ Weak to high degrees of covariance
- ◇ Sampling with replacement: risk of high correlation
- ◇ Random bagging

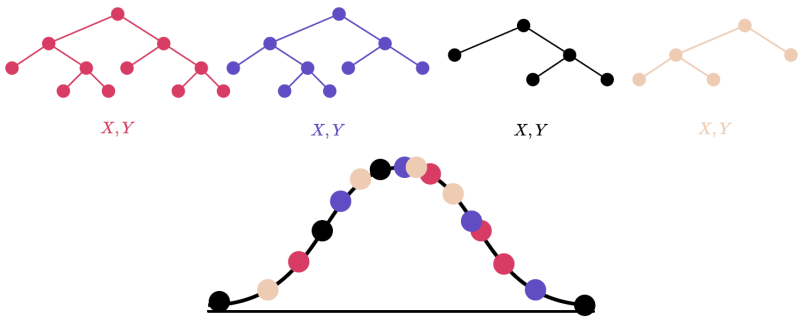


Fig. 14.3: Random Forests process illustration
On the bottom, the bagged data, each bag indicated by a different color. On top, a set of learned trees, color-coded as the bag they have been learned from: the red tree has been learned on the red bag, and so forth.

14.3.2 Boosting

Boosting creates fuzzy soft bags, wherein instances are jointly predicted by all models, thus decomposing the task, rather than the data! Each model is trained to correct the mistakes of the previous one, by focusing more on the instances that were misclassified. The final prediction is made by combining the predictions of all models, typically through a weighted vote or sum.

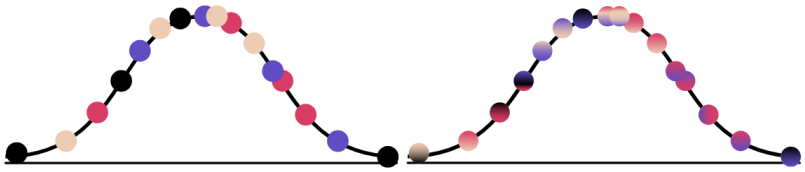


Fig. 14.4: Bagging VS Boosting. In bagging, each bag is used to learn a model. In boosting, bags are fuzzy, and the task is decomposed so that each model predicts a subset of it.

In boosting, the model has the following form:

$$f_T(x) = \sum_{i=1}^{T} \eta_i f^i(x), \eta_i \in \mathbb{R}$$

where each model f^i is trained to correct the mistakes of the previous models, hence it is iteratively built(learned).

14.3.2.1 Towards linear algebra

Boosting can be interpreted as a gradient descent in function space, where each model is a step in the direction of the negative gradient of the loss function with respect to the current ensemble prediction.

Let's address this step by step.

Since we are operating with finite datasets, we can interpret functions as vectors $\vec{f_i} = [f_i(x_1), \ldots, f_i(x_n)]$ in a finite-dimensional vector space, where each dimension corresponds to an instance in the dataset. Hence, we can interpret operations on functions as operations on vectors:

- ◇ Function addition: $(f + g)(x) \equiv f(x) + g(x)$ corresponds to vector addition $\vec{f} + \vec{g}$
- ◇ Scalar multiplication: $(\eta f)(x) \equiv \eta f(x)$ corresponds to scalar multiplication $\eta \vec{f}$
- ◇ Function multiplication: $(f \cdot g)(x) \equiv f(x) \cdot g(x)$ corresponds to element-wise vector multiplication $\vec{f} \circ \vec{g}$

As said before, boosting models are learned iteratively, each additional model f_{t+1} looking to reduce the loss

$$L_{t+1} = l(Y_i, f_t(x) + \eta_{n+1} f_{t+1})$$

The additional function f_{t+1} is one of possibly many (possibly infinite) directions we can take in functional space. We want to pick the direction minimizing loss.

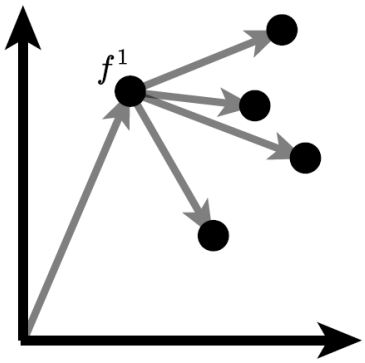


Fig. 14.5: f_1 and some of the infinite possible directions f_2

To get the direction of maximum loss reduction, we can compute the gradient of the loss with respect to the current model's predictions, i.e. the gradient of L^{t+1} w.r.t. f^t : $\nabla_{f_T} L^{t+1}$. The negated gradient indicates the direction in which we should adjust our predictions to minimize the loss.

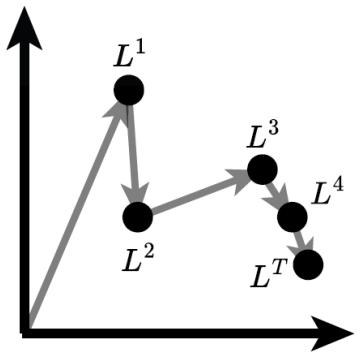


Fig. 14.6: Boosting and iterative optimization: we construct the model one loss improvement at a time, exploring the loss space.

Note that the space of models $\mathcal{F}$ to which f_{t+1} belongs may not be continuous, e.g., decision trees. In such cases, we cannot directly follow the gradient direction, and we choose the closest match $f_{t+1} in \mathcal{F}$ maximizing its similarity to the negative gradient direction:

$$f_{t+1} = arg\max_{f \in \mathcal{F}} f \cdot -\nabla_f^T L^{t+1}$$

In boosting models, gradient of the loss gives us a search direction, but also a **residual**: practically, since each model is additive, gradients define directions as much as residuals we want to optimize. Thus, we fit models on datasets $(X, -\nabla L^t)$

14.3.2.2 Algorithm

We can define a generic algorithm for boosting models as follows:

Algorithm 11 Generic Boosting Procedure

1: Initialize f_1
2: Find optimal direction $-\nabla_{f_t} L^t$
3: Find admissible direction f_t (choose $f_t \in \mathcal{F}$ approximating the negative gradient)
4: Find learning step η_t
5: Update ensemble: $f_{t+1} \leftarrow f_t + \eta_t f_t$
6: Go to step 2 (until stopping criterion)

Notes

- ◇ The function space could be anything: the space of Decision Trees (Gradient-Boosted Trees), of Logistic Regression (Logitboost), etc.
- ◇ Regularization is usually applied to f_t: allows to have weak learners, which helps with decreasing overfit

- ◇ **Adaboost** - Uses an exponential loss, adapts with a closed form search
- ◇ **Gradient-Boosted Trees** - Leverage Decision Trees as models
- ◇ **XGBoost** - Leverages trees, and uses a more robust loss approximation through the Hessian, rather than the gradient

Regularization is the process of constraining models to reduce their variance, at the cost of increasing their bias. This is particularly important in boosting, where each model is trained to fit the residuals of the previous models, which can lead to overfitting if the models are too complex.
Regularization is performed directly on the weak learners (to make sure we keep variance high), but can be applied post-hoc too through pruning!

Unlike bagging, boosting models are learned iteratively. Thus the computational complexity for k models is given by $\mathcal{O}(k \cdot c)$, where c is the cost of learning a single model.

Pros

- ◇ Model-agnostic
- ◇ Given some relatively likely theoretical assumptions, has extremely low bias
- ◇ Unlikely to overfit
- ◇ Computationally quick

Cons

- ◇ Largely un-interpretable

Chapter 15

Gradient-based optimization

15.1 Initial notions - Computational graphs

Definition 15.1 (Computational graph) *A computational graph is a directed acyclic graph where:*

- ◇ *nodes represent operations or variables*
- ◇ *edges represent the flow of data between them*

In a computational graph there are

- ◇ *Parameters Θ given by the weights on edges*
- ◇ *Structure given by the graph's architecture and function.*

We can generalize several gradient-based optimization to computational graphs, made up of objects which compute a generic function $f(x)$. Functional form: $h(\omega^T x)$. Note: $\omega^T x$ is simply a linear combination of x.

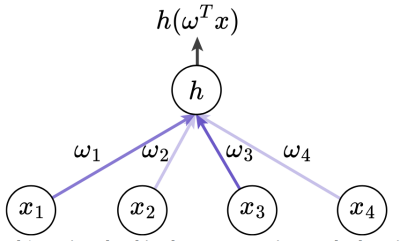


Fig. 15.1: A computational graph: nodes indicate objects involved in the computation, and edges indicate their flow in the graph. **Structure** given b y the operations $(h, \cdot)$ while the parameters are the weights $\omega_{1\ldots4}$.

15.1.1 Forward and backward passes

Forward pass: given input x, compute $f(x)$ by following the graph from input to output, thus **bottom-up**. Recall that nodes are either functions or variables, so the forward pass consists in applying functions to variables as we traverse the graph, with the edges indicating the flow of data along with a **weight** (parameter) associated to each edge.

Backward pass: given a —differentiable— loss function $l(f(x), y)$, compute the gradient of the loss wrt the parameters of the graph.

15.2 Scaling up - Layering graphs

The backward pass is made possible by the differentiability of the loss... but this can be applied also to the functions within g: if a function h_i within g is differentiable, then I can **recursively compute gradients** on it.

Definition 15.2 (Chain Rule of Calculus) *If* $f(x) = h(g(x)) = h \circ g$, *then*

$$\frac{\vartheta f}{\vartheta x} = \frac{\vartheta h}{\vartheta g} \cdot \frac{\vartheta g}{\vartheta x}$$

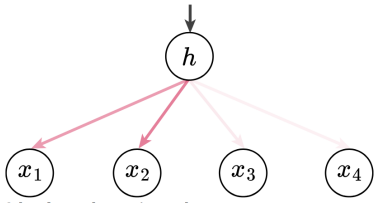


Fig. 15.2: A backward pass in g: change parameters indicated by the gradient indicates the direction towards which move parameters to minimize the loss l

Applying the chain rule recursively, and going down the computational graph, we can compute optimization directions for all parameters! The algorithm chaining back the loss gradient is called **backpropagation**.

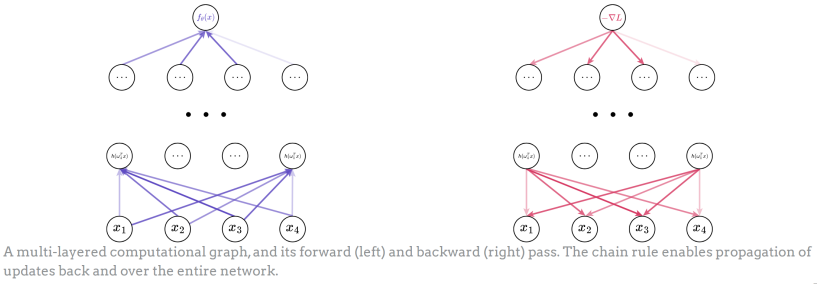


Fig. 15.1: A multi-layered computational graph, and its forward (left) and backward (right) pass. The chain rule enables propagation of updates back and over the entire network.

15.2.1 Neural Networks

Neural networks implement computational graphs.

- ◇ **Layer**: set of all adjacent nodes
- ◇ **Block**: collection of consecutive layers
- ◇ **Activation function**: functions found in nodes
- ◇ **Hidden layer**: a layer, except the first or last one
- ◇ **Output layer/nodes**: layer/nodes yielding the computed output $f_\theta(x)$

Fitting a neural network consists in, at a very high level, repeatedly computing forward and backward passes, each pair improving on the current loss.

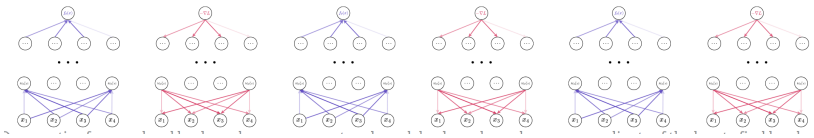


Fig. 15.2: Consecutive foreward and backward passes on a network, each backward pass leverages gradients of the loss to find local directions of loss minimization, which are then used to fit the model.

15.2.2 Learning - Stochastic learning

Backpropagation takes care of defining optimal directions for the parameters, but how do we estimate and leverage these directions?

Estimating directions: Through stochastic gradient, by computing gradient on a batch of data, rather than the whole dataset.

Provides a good approximation and a much faster computation.

Batches can be aggregated, the directions they yield combined: a training-specific approach to combat overfit.

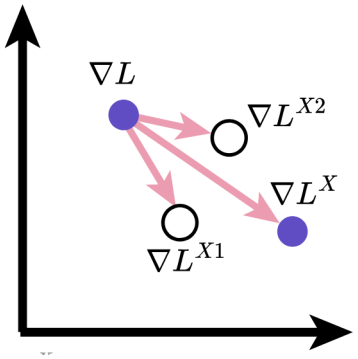


Fig. 15.3: The gradient ∇L^X estimated on some data X, and gradients ∇L^{X_1} and ∇L^{X_2} estimated on two batches X_1 and X_2 (subsets of X). The batch gradients can be combined to approximate the full gradient.

15.2.2.1 Momentum

Backpropagation takes care of defining optimal directions for the parameters, but how do we estimate and leverage these directions?

Estimating weight update: Update directions are weighted by a possibly decaying learning rate η, and possibly weighted by past updates (Adam, RMSProp, Nesterov momentum).

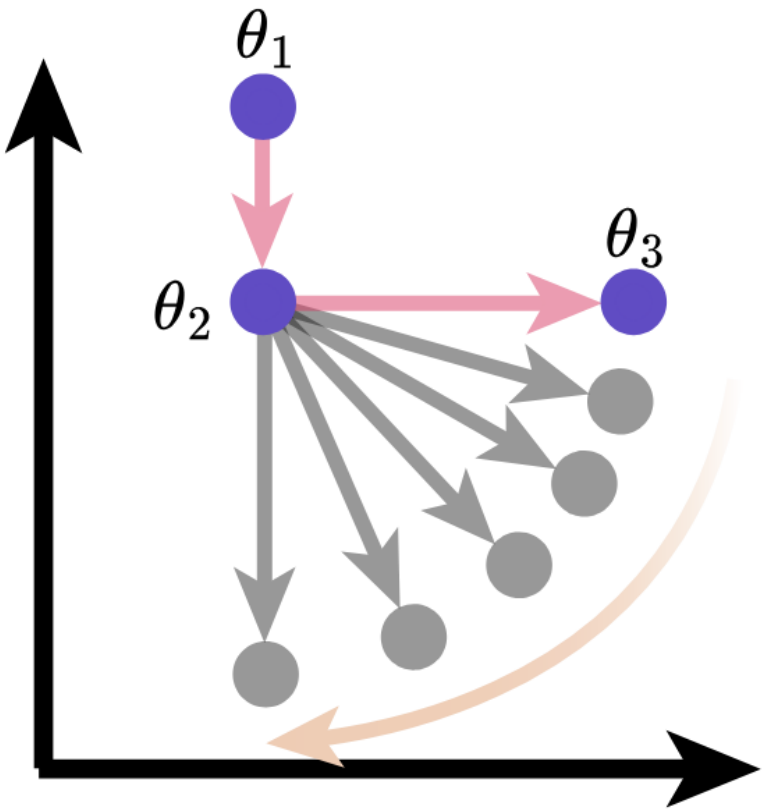


Fig. 15.3: Momentum

15.2.2.2 Regularization

Regularization refers to techniques aimed at reducing overfitting during training. Regularization of neural networks can take many forms.

- ◇ **Weight Regularization** - The loss includes a term $||\Omega||_2$ to penalize large weights in the netork, as higher weights tend to increase the network capacity.
  Weight regularization constrains the capacity of the network by discouraging large weights.
- ◇ **Dropout** - Random deletion of network connections, aims to create networks less reliant on a small number of neurons.
  In other terms, dropout forces the network to be more robust, as it cannot rely on specific paths in the network.
- ◇ **Early Stopping** - Stop training when the validation loss starts increasing, even if the training loss is still decreasing. Early stopping prevents overfitting by halting training before the model starts to memorize the training data.
  More specifically, there are two rolling metrics:
  1. The gap between train and validation error
  2. Train error itself

  Early stopping halts training when the gap starts increasing (indicating overfitting) or when the train error does not decrease significantly anymore (indicating convergence, i.e., the model has learned as much as it can from the training data, so no further training is beneficial).

15.3 Structure

Activation functions impact the flow of data throughout the network, and their outputs are called **activations**. Different activations define different **representations** of the data, which, unlike in PCA, are dependent on learned parameters.

Formulation	**Heads**	**Task**	
Identity	$\omega^T x$	1	Regression
Logistic	$\frac{1}{1+e^{-\omega^T x}}$	1	Classification*
ReLU	$\max\{0, \omega^T x\}$	1	Regression*
Softmax	$\frac{exp(\omega_i x_i)}{exp(\omega^T x)}$	≥ 1	Classification*
...	...	...	

Table 15.1: Common activation functions in neural networks
**Indirectly*: classification is often rendered as a continuous value (to turn into discrete), while regression may be bounded, e.g., to be positive by ReLU.
Remember, $\omega^T x$ is the sum of the incoming edges in the node.

15.4 Architectures

Given their high capacity, and innate ability to encode data, networks are often not fit from scratch. Rather, a network is fit on a task, then adapted to other tasks.

- ◇ **Fine tuning**: use a pre-trained network as initialization for a new specialized task, then continue training on the new task.
- ◇ **Adapters**: insert small layers (adapters) between pre-trained layers, freeze the pre-trained layers, and only train the adapters on the new task. Adapters are fit to "steer" the parameters of the larger pre-trained network towards the new task, while keeping the original knowledge intact.

15.4.1 ResNet

Architectures have proven to be extremely important, and in several cases, the application dictates the network architecture. On tabular dataset, **residual networks** (ResNets) are a particularly strong baseline. They have a functional form

$$f(x) = x + h(x)$$

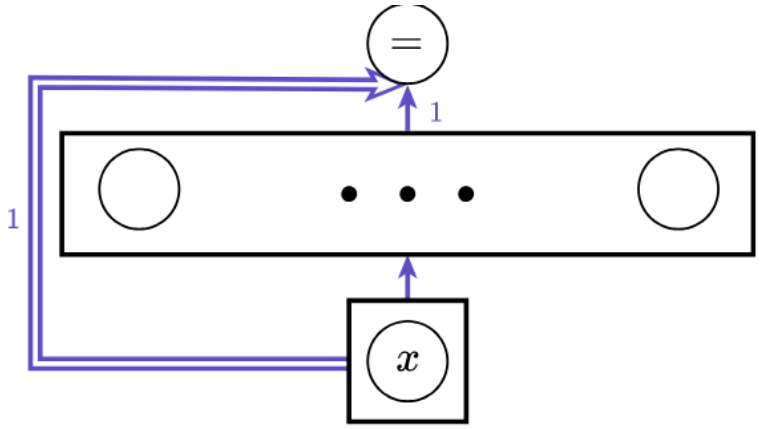


Fig. 15.4: A residual block: a node is forwarded to the next layer, and to the one after it as well. Residual (also called skip) connections allow a more effective backpropagation. Weights on the skip connection are set to to preserve data.

15.4.2 Feature Transformer

Blocks constructs circuits of similarity, computing degrees of "attention" between features, and representations. Similarity then weighs on skip connections. Functional form:

$$f(x^i) = f(x^i) + \sum_{i \neq j} \alpha_{i,j} x^j$$

where $\alpha_{i,j}$ is the attention weight between features i and j.

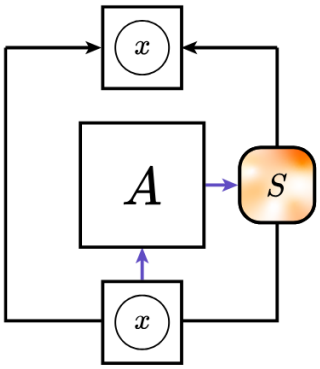


Fig. 15.5: An attention block: similarities between representations are computed through a (scaled) multiplication. Addition through a residual connection allows representations to explicitly influence each other

Part V

Time Series Analysis

Chapter 16

Time series

16.1 Definitions

Definition 16.1 (Univariate series) *A **univariate series** $s \in \mathcal{X}^n$ is a sequence $s = [s_1, \dots, s_n]$ of n values belonging to a domain $\mathcal{X}$ ordered in time, where each value $s_i \in \mathcal{X}$ is a real number.*
A series has five properties:

- ◇ ***Type*** *- discrete or continuous (e.g nucleotide bases vs temperature values)*
- ◇ ***Sampling rate*** *- how often values are sampled*
- ◇ ***Amplitude*** *- the range of values taken by the series*
- ◇ ***Seasonality*** *- periodic patterns that repeat over fixed intervals*
- ◇ ***Period*** *- the length of the interval over which patterns (or the overall series) repeat*

A ***multivariate* time series**, instead, generalizes time series to multiple variables. Each instance is comprised of multiple time series, each representing a different feature.

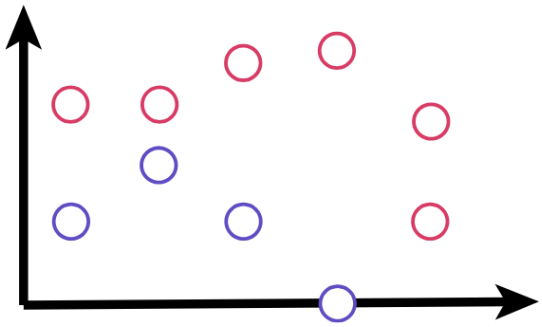


Fig. 16.1: A multivariate time series, with a variable in red and one in blue

16.1.1 Metrics for Time Series

- ◇ Mean - Expected value $\mathbb{E}[s]$ of the series s

$$\mu = \mathbb{E}[s] = \frac{1}{n}\sum_{i=1}^{n} s_i$$

- ◇ Variance

$$Var(s) = \frac{1}{n}\sum_{i=1}^{n}(s_i - \mathbb{E}[s])^2$$

- ◇ Trend - slope Δ of a linear model modeling the series

$$s_i = \Delta \cdot i + b$$

- ◇ Interquantile Range - difference between the a^{th} and b^{th} percentiles

$$IQN(s) = q_a(s) - q_b(s)$$

◇ Skewness - asymmetry of the distribution of values in the series

$$Skewness(s) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{s_i - \mathbb{E}[s]}{\sigma}\right)^3 = \mathbb{E}\left[\left(\frac{s-\mu}{\sigma}\right)^3\right]$$

◇ Kurtosis - "tailedness" of the distribution of values in the series

$$Kurtosis(s) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{s_i - \mathbb{E}[s]}{\sigma}\right)^4 = \mathbb{E}\left[\left(\frac{s-\mu}{\sigma}\right)^4\right]$$

16.1.2 Local analysis

16.1.2.1 Rolling statistics

In order to capture Seasonality and Trend, we must analyze time series over **windows** of time, thus locally. **Rolling statistics** compute metrics over a sliding window of fixed size w. In general **rolling** indicates the act of extracting a series of consecutive subsequences of given dimensionalities $w^1, \dots, w^k$, named **windows** given a different view of the original series s.

◇ **Rolling mean** - mean over a sliding window of size w
◇ **Rolling variance** - variance over a sliding window of size w
◇ **Rolling min/max** - minimum/maximum value over a sliding window of size w
◇ quantiles, skewness, kurtosis and so on…

16.1.2.2 Sliding statistics

Unlike rolling statistics, which compute metrics over non-overlapping windows, hence a subseries of given series s, **sliding statistics** slide a series t over s computing statistics between the two. The act of sliding a windows through a function is called **convolution**.

◇ **Autocovariance** - covariance of a series with itself at different lags. High autocovariances may indicate seasonality in the series.
"How much does a component of a time series correlate with previous and future components?"

$$\gamma(k) = Cov(s_i, s_{i+k}) = \mathbb{E}[(s_i - \mu)(s_{i+k} - \mu)] \qquad \textit{Copilot}$$

$$cov(s_{t:}, s_{:t+\Delta:}) = \frac{1}{n-\Delta}\sum_{i=1}^{n-\Delta} s_i \cdot s_{i+\Delta} \qquad \textit{Slides}$$

The $-\Delta$ in the summation upper bound is to avoid exceeding the bounds of the series.

◇ **Cross-correlation** - Shifted pointwise correlation of the two series s, t , measured as a sliding inner product. For univariate time series, the inner product is simply a multiplication.

$$CC_\Delta(s,t) = \sum_{i=1}^{n-\Delta} s_i \cdot t_{i+\Delta}$$

◦ There might be more, but they are not presented in the slides.

16.2 Segmentation

Definition 16.2 (Segmentation) *Given a set of time series $S = \{s^1, s^2, \dots, s^N\}$, let $S^\subset$ be a set of k subseries $s^1_\subset, \dots, s^k_\subset$ extracted from the series in S. We define an alphabet Σ of symbols, each symbol representing a subseries in $S^\subset$. We can now **segment** each series into a sequence of subseries, each represented by a symbol in Σ^*.*

$$s^i \to \underbrace{[s^{i,1}|\dots|s^{i,k_i}]}_{\in\Sigma^*}$$

" How to split the series into segments? What symbols do we use to represent each segment? " —

16.2.1 Segmentation methods

16.2.1.1 Fixed window

The simplest segmentation method is to use a fixed-size sliding window of size w to partition the series into segments. Each series s^i is divided into n/m non-overlapping segments of length w.

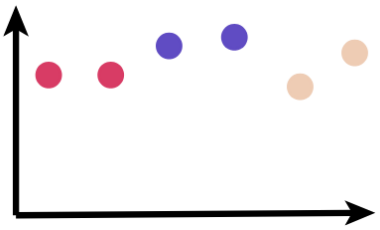


Fig. 16.2: $w = 2$ here

16.2.1.2 Learned segmentation

We can learn segmentation which minimize an approximation error. **Piecewise Linear Approximation** (PLA) approximates each segment with a linear function. It defines a segmentation minimizingg segment-local linear models of the data. It works by either providing a maximum number of segments k or a maximum error tolerance ϵ.

- ◇ Given k, it iteratively adds segments until reaching k, distributing approximation error as evenly as possible among segments (approximation error refers to the difference between the original series and its linear approximation within each segment).
- ◇ Given ϵ, it generates the minimum number of segments such that the approximation error within each segment does not exceed ϵ.

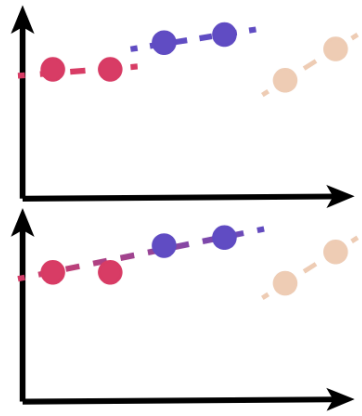


Fig. 16.3: Segmentation on given error bound (top) and on given number of segments (bottom)

Method	Description	Type	$\mathbf{f^{\Sigma}}$	$\mathbf{\Sigma}$
Piecewise Average (PA)	Average value of the segment	Continuous	$\mu^c_{s_i}$	$\mathbb{R}$
Piecewise Linear (PL)	Slope (gradient) of the segment	Continuous	$\nabla^c_{s_i}$	$\mathbb{R}$, $\mathbb{R}^2$

Table 16.1: Segment-based feature functions (PA = Piecewise Average, PL = Piecewise Linear)
$\mathbb{R}^2$ is given by storing both slope and intercept of the linear model.

16.2.1.3 Learned Equidistributional windows

We *learn* windows (segmentation) by their likelihood: given a desired number of segments k, we search for $k-1$ cutting points which maximize the likelihood that the segments are equiprobable as possible, ideally following a uniform distribuition.

16.2.1.4 Two-tier segmentations

Windows segmentations create a series of symbols themselves. We can learn representations through a two-tier algorithm. Symbols may be either categorical or ordinal.

1. continous transformation, yielding segments $S^{\subset}$ with symbols in $\Sigma^{\subset}$
2. transformation partitioning, mapping symbols in $\Sigma^{\subset}$ to discrete symbols in Σ

Symbolic aggregate approximation (SAX) implements a two-tier transformation to create discrete *and* ordinal symbols.

- ◇ fixed-windows segmentation followed by piecewise average transformation in sybsymbols $(\Sigma^{\subset})^*$
- ◇ aggregation of subsymbols into equiprobable symbols in Σ: partition the subsymbol distribution into equiprobable buckets, each defined by a symbol in Σ.

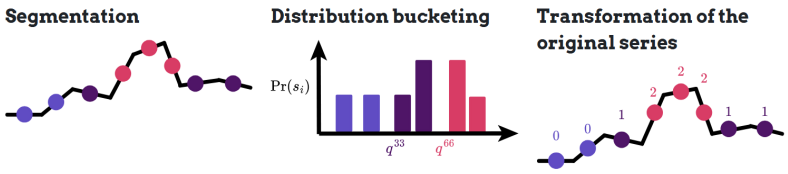
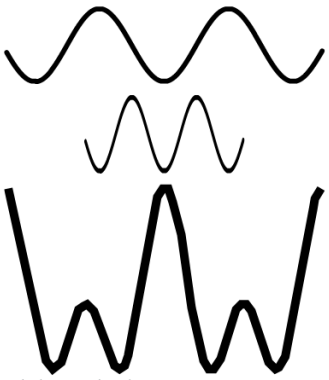


Fig. 16.2: The three steps of SAX. First, fixed-window average segmentation creates a set of segments and segment-wise averages. Then, their empirical distribution is estimated, and a set of buckets of equal density is extracted by considering the distribution's quantiles. Finally, each bucket is assigned a symbol, and the series is mapped to the symbols through substitution.

16.3 Signals

Segmentations can be tricky to handle, and simply offer a representation in a domain quite different from the original. **Signal representations**, on the other hand, aim to define a series in terms of other series. To stick with the signal processing literature, where they are most prevalent, we'll refer to series as signals.

Fourier analysis tackles **periodic** (or *"stationary"*) signals. Note that we can linearly combine sine and cosine signals to obtain any periodic signal. In the figure we have two signals, $cos(x)$ and $cos(2x)$ and their linear combination $cos(x) + cos(2x)$.
As a lineare transformation we need to learn a set of coefficients α which map the basis to the signal s. In Fourier analysis, we constrain $\alpha \in \mathbb{R}^+$. Basis signals are sinusoids of different frequencies. The ones not contributing to the signal have coefficient 0, otherwise it's positive real-valued.



16.4 Sinusoids

Fourier analysis is a method to decompose a time series into a sum of sinusoidal functions, each characterized by a specific frequency, amplitude, and phase. This decomposition allows us to analyze the frequency components of the time series, which can be useful for identifying periodic patterns, trends, and noise.

Sinusoids are mathematical functions that describe smooth, periodic oscillations. They are sin/cos functions defined by a **phasor** ψ. Sinusoids define **amplitude**, which is the height of the wave, and **frequency**, which is how many cycles occur in a unit of time. At a time t, the phasor defines a series component with amplitude

$$s_t = \underbrace{\alpha}_{\text{amplitude scaling}} \cos(\overbrace{2\pi f_0 t}^{\text{angle } \theta_{0,t}} \underbrace{+\phi}_{\text{shifting}}).$$

Where:

- α is the amplitude scaling factor
- $f_0 = t_0^{-1}$ is the frequency of the sinusoid, the reciprocal of the period t_0
- ϕ is the phase shift, determining the horizontal shift of the wave

By Euler, we can map complex numbers to the complex unit circle

$$e^{-i\Theta} = \cos(\Theta) + i\sin(\Theta)$$

In Figure 16.3, the complex unit circle and an imaginary number $z = e^{-i\Theta}$. As in linear algebra, we can define a vector $(z_{\mathbb{R}}, z_{\mathbb{C}})$ through the standard basis $(1, 0), (0, i)$.

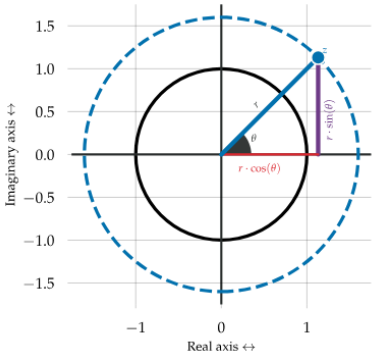


Fig. 16.3

Slides from 28 to 30 display some formulas concerning the computation of coefficients α. The takeaway is that we can compute coefficients through inner products between the signal and basis sinusoids.
The **Discrete Fourier Transform** (DFT) computes the coefficients for all frequencies up to the Nyquist frequency $f_N = \frac{1}{2\Delta t}$, where Δt is the sampling interval of the time series.

Pros

- ◇ Quick: $\mathcal{O}(n \log n)$ with FFT
- ◇ Exact decomposition in separate and different signals

Cons

- ◇ Decomposition defined exclusively for sinusoidal series
- ◇ Decomposition of periodic series
- ◇ Decomposition exclusively in terms of frequency, not time

16.5 Wavelets

Wavelets (little waves) aim to tackle weaknesses of Sinusoids, namely the fact that they are defined only for periodic series, and that they decompose signals only in terms of frequency, not time. Signals are decomposed in *mother wavelets* through scaling and shifting operations. Wavelets are functions that can be used to analyze signals at different scales and resolutions. Unlike sinusoids, which are infinite in extent and have a fixed frequency, wavelets are localized in both time and frequency, allowing for a more flexible representation of signals.

A wavelet $\psi_{\kappa,\tau} : \mathbb{R} \to \mathbb{R}$ is a function such that:

$$\int_{-\infty}^{\infty} \psi_{\kappa,\tau}(t)\, dt = 0 \quad \textit{zero mean}$$

$$\int_{-\infty}^{\infty} |\psi_{\kappa,\tau}(t)|^2\, dt \neq \infty \quad \textit{finite energy} \text{ or } \textit{compact support}$$

Finite energy makes it so a wavelet, unlike a sinusoidal function, is bounded: thus, by construction, wavelets can be **localized in time**.

A mother wavelet $\psi_{K,T}(t)$ defines a family of *daughter wavelets* through scaling (K) and shifting (T) operations:

- ◇ A frequency κ: shrink or stretch the daughter wavelet
- ◇ A shift τ: pushes or pulls the daughter wavelet along the time axis

Hence, a daughter wavelet is defined in function of ψ and the two parameters κ, τ as:

$$\psi_{\kappa,\tau}(t) = \frac{1}{\sqrt{\kappa}}\psi\left(\frac{t-\tau}{\kappa}\right)$$

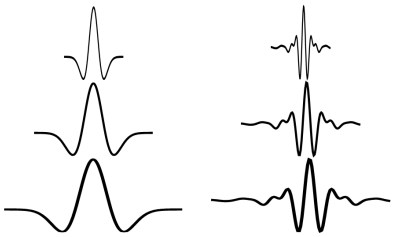


Fig. 16.4: Mexican Hat wavelet (left column), and Meyer wavelet (right)

Sinusoid to Wavelets

Moving from Sinusoids to Wavelets is pretty easy if we think in terms of basis functions.

$$\underbrace{\sum_{\rho=1}^{n} \alpha\psi_{\kappa}(t)}_{\text{Discrete Fourier}} \rightarrow \underbrace{\sum_{\rho=1}^{n} \alpha\psi_{\kappa,\tau}(t)}_{\text{Wavelet Transform}}$$

Wavelets are convoluted across the series, producing a list of coefficients. κ and τ define the scale and position of each wavelet, and are *dyadic*, i.e. they take values which are powers of 2, e.g. $\tau = 2^{-i}, \tau = k2^{-i}$.

16.6 Motif

Motifs tie strongly with subseries extraction for discretization: if representation algorithms extracted to represent, and thus describe, motif extraction algorithms instead extract subseries to represent, and thus **discriminate**. In other words, a motif is a subseries characteristic of a set of series.

A motif is defined as a **subseries** which is **frequent** across a set of time series.

There are two approaches to motif discovery:

- ◇ **Descriptive** - a motif is a reoccurring subseries in the set S
- ◇ **Discriminative** - a motif is a subseries which is frequent in a class of series S, and infrequent (does not occur) in another class/set $S^{\neq}$.
  This is also called a **shapelet**.

Discriminative *power* and descriptive *power*, respectively.

16.7 Alignment

An **alignment** (or "matching") $A = [(i,j)]^{\max n,m}$ of two time series l, u with components $[l_1, \ldots, l_n]$ and $[u_1, \ldots, u_m]$ is a set of pairs of indices (i,j), assigning each component of l to one or more components of u, and vice versa.
An alignment A induces an alignment cost C_A quantifying how unaligned the two series are.

We have two costs to consider:

- ◇ **Local cost** - cost of aligning two components l_i and u_j

$$c^a_{i,i} = ||l_i - u_j||_p$$

- ◇ **Match cost** - cost of foregoing matching i with j in favor of a lower cost $i' \neq i$ (i.e. "How much would I pay for another alignment")

16.7.1 Warping

Note that a simple **straight match** alignment is not always possible, time series must not be shifted or stretched, and series must have the same length.

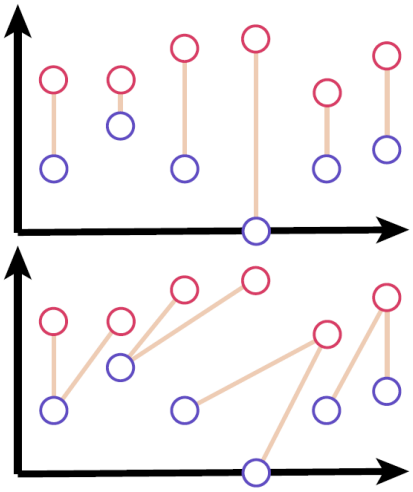
Definition 16.3 (Shifted alignment) *Assuming some subseries of either l or u are shifted, under a shifted alignment we must ensure:*

- ◇ ***Warp, Replication*** *- Each component l_i can be assigned to any u_j*
- ◇ ***Planarity*** *- If l_i is assigned to u_j, then l_{i+1} can only be assigned to $u_{j'}$ with $j' \geq j$*

$$\forall i, j.(i,j) \in A \Rightarrow (l_{i+1}, u_{j-1} \notin A)$$

By warping, we replicate a component, warping it also further in the series. This allows us to replicate components, and emulate a straight match alignment.
Replication refers to assigning the same component multiple times in the alignment.

- ◇ **No Warp** - component l_i is assigned to exactly one component u_j



Two possible alignments (in beige) of an *upper* series s^1 (in red), and a *lower* series s^2 (in blue). The bottom alignment is $A = [(0,0),(1,0),(2,1),(3,1),(4,2),(4,3),(5,4),(5,5)]$.

Fig. 16.3

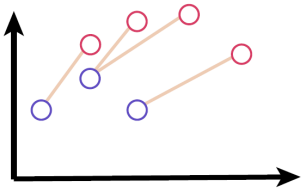
- ◇ **Upper Warp** - (l_i, u_j) and (l_i, u_{j+1}) can be in A, replicating the component on the upper series l
  That is, l_i is aligned to both u_j and u_{j+1}.
- ◇ **Lower Warp** - (l_i, u_j) and (l_{i+1}, u_j) can be in A, replicating the component on the lower series u

16.7.2 DTW - Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm to compute the optimal shifted alignment between two time series l and u. It uses dynamic programming to compute the optimal alignment cost C_A. The algorithm builds a cost matrix $C^\Sigma \in \mathbb{R}^{n \times m}$, where each entry $C^\Sigma(i,j)$ represents the minimum cost of aligning the first i components of l with the first j components of u. The cost matrix is filled using the following recurrence relation:

$$C^\Sigma(i,j) = c^a_{i,j} + \min \begin{cases} C^\Sigma(i-1,j) & \text{(insertion)} \\ C^\Sigma(i,j-1) & \text{(deletion)} \\ C^\Sigma(i-1,j-1) & \text{(match)} \end{cases}$$

Where $c^a_{i,j}$ is the local cost of aligning components l_i and u_j. The final alignment cost C_A is found in the bottom-right cell of the matrix, i.e., $C^\Sigma(n,m)$.



$$C^\Sigma \begin{bmatrix} 0.495 & 0.613 & 0.429 & 0.618 \\ 0.524 & 0.350 & 0.536 & 0.885 \\ 0.458 & 0.214 & 0.511 & 0.194 \end{bmatrix}$$

A DTW alignment (top), and a cumulative alignment cost matrix C^Σ (bottom).
The (i,j) component holds the cumulative cost of aligning the l_i, u_j.

Fig. 16.4

This is Copilot generated

In the slides, the algorithm is presented as follows:

C_Σ computes a minimal and cumulative cost of aligning the two series up to components i and j respectively. Each entry in the matrix is computed using a cumulative cost and the local cost of aligning l_i and u_j.

$$C^\Sigma(i,j) = c^a_{i,j} + c^\Sigma_{i,j} = \underbrace{||l_i + u_j||_p}_{alignmentcost} + \min \underbrace{C^\Sigma(i-1,j-1)}_{NoWarp}, \underbrace{C^\Sigma(i-1,j)}_{LowerWarp}, \underbrace{C^\Sigma(i,j-1)}_{UpperWarp}$$

- ◇ Base case - the alignment (l_1, u_1) has minimal cost. This is trivially always true. $C^\sigma(1,1) = ||l_1 - u_1||_p$
- ◇ Inductive case - i, j end up in three warping cases:
  - **No Warp** - both l_i and u_j are matched together, the accumulated cost is $c^\Sigma(i,j) = C^\Sigma(i-1,j-1)$
    Note that is only the cumulative cost! To get the actual cost, the one in the matrix C^Σ we must add the local cost

$c^a_{i,j}$ to it to get the total cost.
- **Upper Warp** - l_i is matched to both u_j and u_{j-1}, the accumulated cost is $c^\Sigma(i,j) = C^\Sigma(i,j-1)$
- **Lower Warp** - u_j is matched to both l_i and l_{i-1}, the accumulated cost is $c^Sigma(i,j) = C^\Sigma(i-1,j)$

As a matrix of minimal accumulated alignment cost, we know that for each component i,j in C^Σ, we have, by construction, the minimal cost to align up to i,j. Thus, we can simply start from the last alignment, and follow $C^\leftarrow$ backwards for the warps of minimal cost!

$$C^\leftarrow = \begin{bmatrix} \leftarrow & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \leftarrow & \leftarrow \\ \uparrow & \uparrow & \leftarrow & \leftarrow \end{bmatrix}, \quad A = [(1,1),(2,2),(3,2),(3,3),(3$$

Directions of minimal accumulated cost $C^\leftarrow$ over C^Σ: the red path from the last entry indicates the alignment of minimal cost. The alignment A is given by the indices of the path.

$$C^\Sigma \begin{bmatrix} 0.495 & 0.613 & 0.429 & 0.618 \\ 0.524 & 0.350 & 0.536 & 0.885 \\ 0.458 & 0.214 & 0.511 & 0.194 \end{bmatrix}$$

$$C^\leftarrow = \begin{bmatrix} - & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \uparrow & \leftarrow \\ \uparrow & \uparrow & \leftarrow & \leftarrow \end{bmatrix}$$

Directions of minimal accumulated cost $C^\leftarrow$ over C^Σ: entries indicate which alignment choice has produced the minimal cost.

Fig. 16.5

Sakoe-Chiba band and Itakura Parallelogram are two techniques to limit the warping path in DTW, constraining the alignment to a band around the diagonal of the cost matrix, reducing computational complexity and preventing pathological alignments.

Chapter 17

Time Series

17.1 Classification

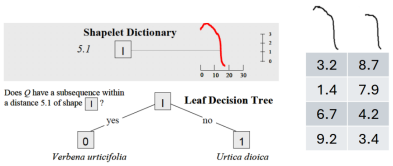
Given a set X of n time series, $X = x_1, x_2, \ldots, x_n$, each time series has m ordered values $x_i = \langle x_{t1}, x_{t2}, \ldots, x_t m \rangle$ and a class value c_i.

The objective is to find a function f that maps from the space of possible time series to the space of possible class values.

Generally, it is assumed that all the TS have the same length m.

17.1.1 Shapelet-based classification

Definition 17.1 (Shapelet-based classification) *First represent a TS as a vector of distances with representative subsequences, namely* ***shapelets****. Then, use it as input for machine learning classifiers.*



When it comes to time series, **shapelets** are *TS subsequences* which are maximally representative of a class. Shapelets can provide interpretable results,which may help domain practitioners betterunderstand their data. Furthermore, shapelets can be significantly more accurate/robust because they are local features, whereas most other state-of-the-art TS classifiers consider global features.

Algorithm 12 GenerateCandidates(**dataset** D, $MAXLEN$, $MINLEN$)

```
pool ← ∅
l ← MAXLEN
while l ≥ MINLEN do
    for each time series T in D do
        pool ← pool ∪ S_T^l                ▷ add all subsequences of length l from T
    l ← l − 1
return pool
```

Algorithm 13 FindingShapeletBF(**dataset** D, $MAXLEN$, $MINLEN$)

```
candidates ← GenerateCandidates(D, MAXLEN, MINLEN)
bsf_gain ← 0
bsf_shapelet ← null
for each shapelet S in candidates do
    gain ← CheckCandidate(D, S)
    if gain > bsf_gain then
        bsf_gain ← gain
        bsf_shapelet ← S
return bsf_shapelet
```

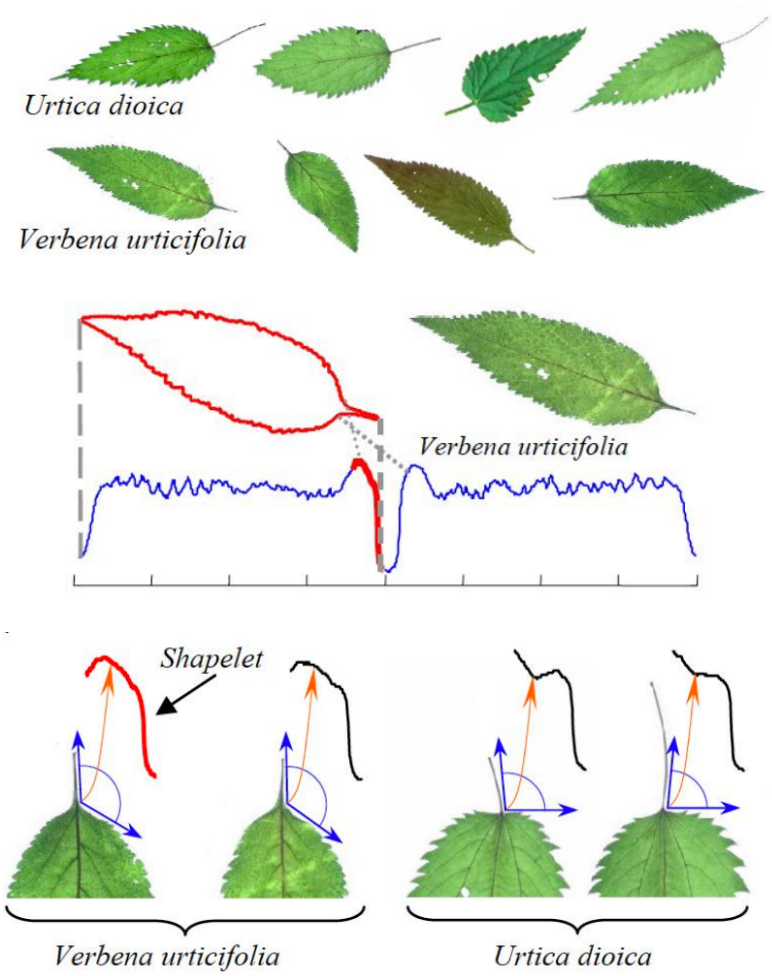


Fig. 17.1: Shapelet application example

Algorithm 14 CalculateInformationGain(**distance histogram** *obj_hist*)

```
split_dist ← OptimalSplitPoint(obj_hist)
D1 ← ∅,   D2 ← ∅
for each entry d in obj_hist do
    if d.dist ≤ split_dist then
        D1 ← D1 ∪ d.objects
    else
        D2 ← D2 ∪ d.objects
return I(D) − Î(D)                    ▷ return information gain after split
```

17.1.2 Information Gain issues

Distance from the TS to the subsequence SubsequenceDist(T, S) is a distance function that takes time series T and subsequence S as inputs and returns a non- negative value d, which is the distance from T to S.

The idea behind information gain is: given a splitting rule *sp* dividing the dataset in two parts D_1 and D_2, we want to measure how much uncertainty in the class labels is reduced by the split, i.e., how much more "pure" the two resulting datasets are compared to the original one. We use entropy $I(D)$ to measure uncertainty in the class labels of dataset D. The information gain is then computed as the difference between the entropy of the original dataset and the weighted average entropy of the two resulting datasets:

$$\text{InformationGain}(D, D_1, D_2) = I(D) - \hat{I}(D)$$

where

$$\hat{I}(D) = \frac{|D_1|}{|D|} I(D_1) + \frac{|D_2|}{|D|} I(D_2)$$

We may use the distance from T to S as a splitting rule: all time series with distance less than or equal to a threshold d_{th} go to D_1, and the rest go to D_2.

The problem is that we have many candidates, and for each candidate we need to compute the distance from each time series sample to the candidate, which is expensive.

$$\#candidates = \sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (|T_i| - l + 1)$$

That is, having 200 instances of length 275 each, we have about 7.480.200 candidates for some $MINLEN = \ldots$ and $MAXLEN = 200$. This results in a very high computational cost.

There are two main optimizations:

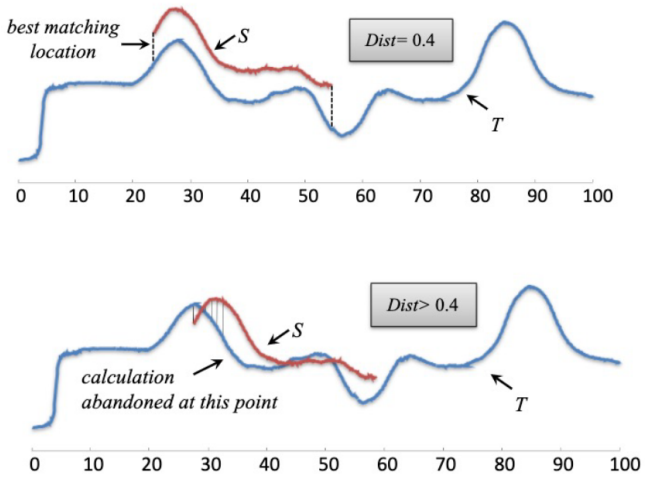


Fig. 17.2: Early abandoning

1. Early abandoning distance calculations
   If we know that the current best minimum distance found so far is d_{best}, and while computing the distance between a time series T and a candidate shapelet S we find that the partial distance already exceeds d_{best}, we can abandon the computation early. This can save a lot of computation time, especially when the shapelet is not similar to the time series.

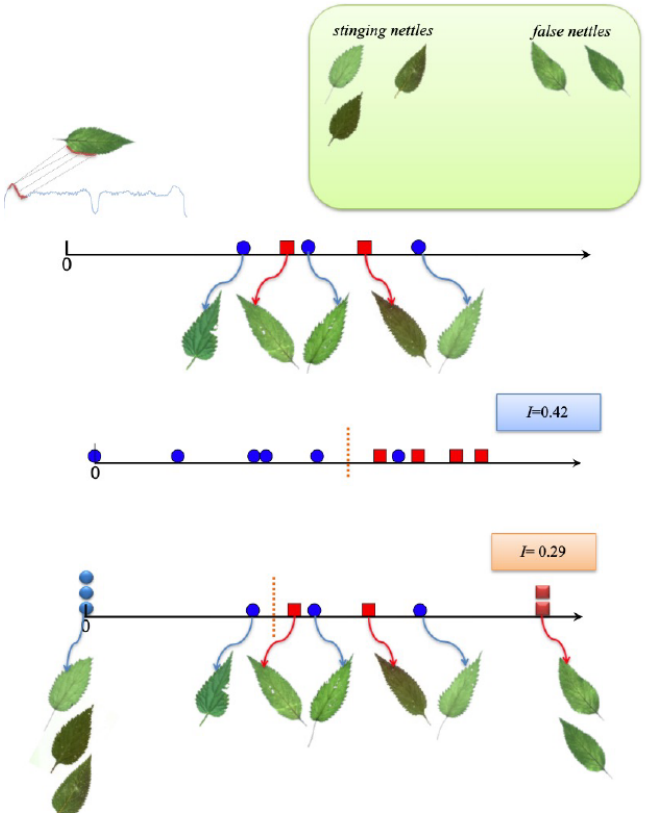


Fig. 17.3: Admissible Entropy pruning

2. Pruning candidates based on an upper bound on information gain
   Given a candidate shapelet S, we can compute an upper bound on the information gain that can be achieved by splitting the dataset based on distances to S.
   If this upper bound is less than the best information gain found so far, we can prune S without computing the actual information gain.
   So, in other words, for a candidate shapelet S, we do not necessarily need to calculate the distance for each training sample. After calculating some training samples, if the upper bound of information gain (corresponding to the optimistic scenario) *is less* than the best candidate shapelet, we stop calculation for that candidate and try the next one, since there is no way it can be a better fit than the current best.

17.2 Motif

Motif discovery essentially is finding repeated patterns, i.e., pattern mining. That is

" Are there any repeated patterns, of length m in the TS? " —

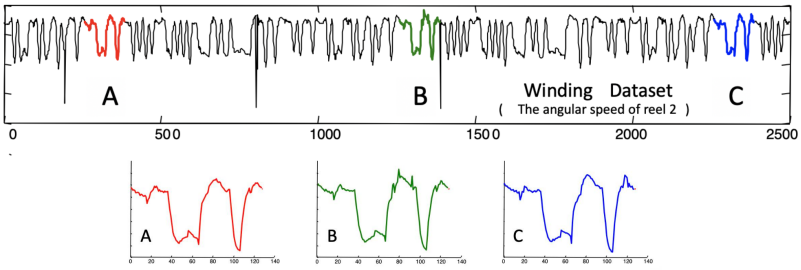


Fig. 17.4: Repeated patterns example

There are several reasons why motif discovery is useful:

- Mining **association rules** in TS requires the discovery of motifs. These are referred to as *primitive shapes* and *frequent patterns*.
- Several **TS classifiers** work by constructing typical *prototypes* of each class. These prototypes may be considered motifs.
- Many **TS anomaly detection** algorithms consist of modeling *normal behavior* with a set of typical shapes (which we see as motifs), and detecting future patterns that are dissimilar to all typical shapes.

17.2.1 Algorithm

The Matrix Profile (MP) is a data structure that annotates a TS and can be exploited for many purposes, e.g. efficient Motif Discovery.

Given a time series, T and a desired subsequence length, m.

Note that m is fixed and decided apriori. The algorithm may be applied multiple times with multiple m values to find the best motifs.

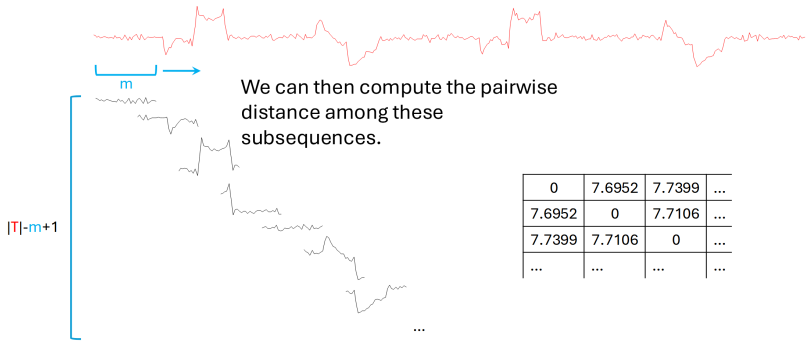


Fig. 17.5: Finding subsequences of length m, along with distance matrix

For each subsequence we keep only the distance with the closest **nearest neighbor**. The **distance** to the corresponding nearest neighbor of each subsequence can be stored in a vector called **matrix profile** P.
The matrix profile value at location i is the distance between T_i and its nearest neighbor

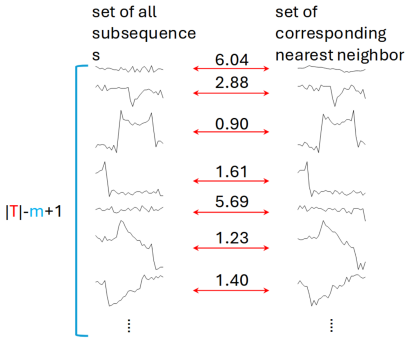


Fig. 17.6: Finding the closest neighbor for each pattern

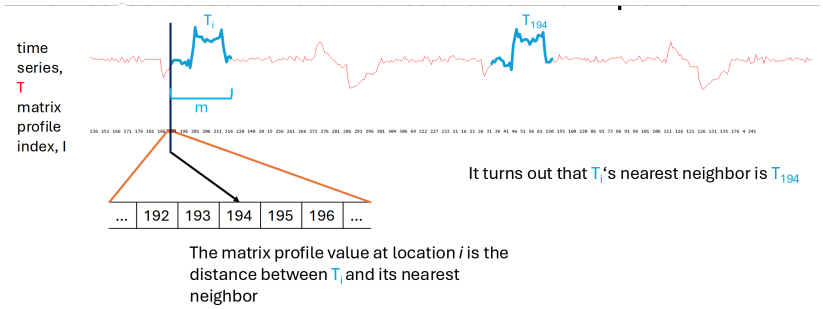


Fig. 17.6: Matrix profile **index**

The index of corresponding nearest neighbor of each subsequence is also stored in a vector called matrix profile index.

The MP index allows to find the nearest neighbor to any subsequence in constant time.
Note that the pointers in the matrix profile index are not necessarily symmetric. If A points to B, then B *may or may not* point to A.

The classic TS motif: the two smallest values in the MP must have the same value, and their pointers must be mutual.

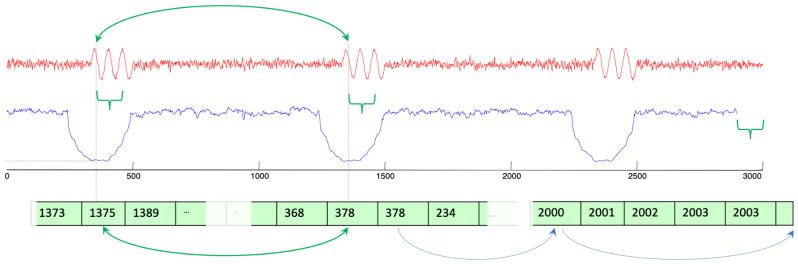


Fig. 17.7: Matrix Profile and what its index represents

17.2.2 Reading the Matrix Profile

- ◇ For relatively low values, you know that the subsequence in the original TS must have (at least one) relatively similar subsequence elsewhere in the data (such regions are "motifs")
- ◇ For relatively high values, you know that the subsequence in the original TS must be unique in its shape (such areas are anomalies).

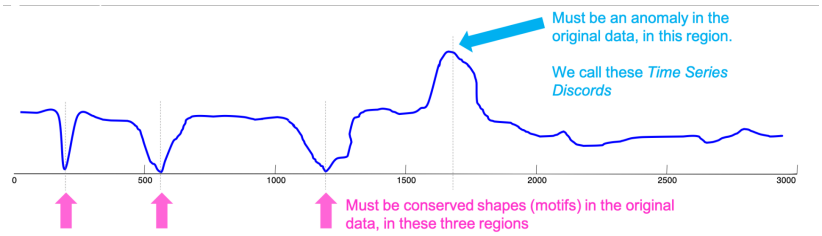


Fig. 17.8: Reading the matrix profile

17.2.3 Computing the Matrix Profile

Part VI

XAI and Responsible AI

Chapter 18

XAI

Explainable-AI explores and investigates methods to produce or complement AI models to make accessible and interpretable the internal logic and the outcome of the algorithms, making such process understandable by humans.

Explicability, understood as incorporating both intelligibility("how does it work?") for non-experts, e.g., patients or business customers, and for experts, e.g., product designers or engineers) and accountability("who is responsible for").

To interpretmeans to give or provide the meaning or to explain and present in understandable terms some concepts. In data mining and machine learning, interpretability is the ability to explain or to provide the meaning in understandable terms to a human.

18.1 Interpretability and Black box

A black box is a model, whose internals are either unknown to the observer or they are known but uninterpretableby humans. A model is interpretable when there is a direct and understandable mapping between the internal mechanism of the model and the human understanding of the same mechanism.

XAI is a set of techniques and methods to make the behaviour and the outcome of black box AI systems understandable by humans.

- ◇ Global and Local Interpretability:
  - Global: understanding the whole logic of a model
  - Local: understanding only the reasons for a specific decision
- ◇ **Time Limitation**: the time that the user can spend for understanding an explanation.
- ◇ **Nature of User Expertise**: users of a predictive model may have different background knowledge and experience in the task. The nature of the user expertise is a key aspect for interpretability of a model.
- ◇ **Interpretability** (or comprehensibility): to which extent the model and/or its predictions are human understandable. Is measured with the complexity of the model.
- ◇ **Fidelity**: to which extent the model imitatea black-box predictor.
- ◇ **Accuracy**: to which extent the model predicts unseen instances.
  Fairness: the model guarantees the protection of groups against discrimination.
- ◇ **Privacy**: the model does not reveal sensitive information about people.
- ◇ **Respect Monotonicity**: the increase of the values of an attribute either increase or decrease in a monotonic way the probability of a record of being member of a class.
- ◇ **Usability**: an interactive and queryable explanation is more usable than a textual and fixed explanation.

Complexity is a measure of how difficult is to understand a model, it is inversely proportional to interpretability. Typically it is related to the **size** of the interpretable model, e.g., number of nodes in a decision tree, number of rules in a rule-based model, number of features used in a linear model.

18.2 XAI techniques

We may have a *transparent model* that is interpretable by design, or we may have a *post-hoc explanation* that is an explanation generated after a black box model has been built.

We have some categorization depending on:

- ◇ **Problem** to be solved
- ◇ Type of **black-box** model (NN, SVM, Ensemble, …)
- ◇ **Data** type used for building the model (Image, Text, Tabular data, …)
- ◇ Type of **explanator** used (DT,Decision Rules,Feature importance, …)

In case of black-box models, an explainer may feed the black-box model with controlled instances and get the output to better understand its behaviour. The explainer may be **model-agnostic** or **model-specific**.

18.3 Explainers

18.3.1 SHAP

SHAP (SHapley Additive exPlanations) is a unified approach to explain the output of any machine learning model. It connects game theory with local explanations, uniting several previous methods. SHAP assigns each feature an importance value for a particular prediction. The SHAP values represent the average contribution of a feature value to the prediction across all possible combinations of features.
SHAP is based on the concept of Shapley values from cooperative game theory. The Shapley value is a way to fairly distribute the total gain (or "payout") among the players (features) based on their contributions to the total gain.
The Shapley value is the average marginal contribution of a feature value across all possible coalitions. A feature coalition is a subset of features used to make a prediction. The marginal contribution of a feature is the difference between the prediction made with and without that feature in the coalition.

Computing a Shapley Value

Let's address an example. For each coalition, we compute the predicted apartment price with and without the feature value cat-banned and take the difference to get the marginal contribution.

The Shapley value is the average of all marginal contributions.

We replace the feature values of features that are not in a coalition with random feature values from the apartment dataset to get a prediction from the machine learning model.

SHAP assigns each feature an importance value for a particular prediction by means of an additive feature attribution method.
It assigns an importance value to each feature that represents the effect on the model prediction of including that feature.

18.3.2 LIME

- ◇ LIME **turns** an image x to a vector x' of interpretable superpixels expressing presence/absence.
- ◇ It **generates** a synthetic neighborhood Z by randomly perturbing x' and labels them with the black box.
- ◇ It **trains** a linear regression model (interpretable and locally faithful) and assigns a weight to each superpixel.

LIME does not really generate images with different information: it randomly removes some superpixels, i.e. it suppresses the presence of an information rather than modifying it.

On tabular data LIME generates the neighborhood by changing the feature values with other values of the domain:

$$x = age = 24, sex = male, income = 1000$$
$$z = age = 30, sex = male, income = 800$$

Given the perturbated instances, LIME queries the black-box model to get the predictions and then it trains a weighted linear regression model to approximate the black-box behaviour in the neighborhood of the instance to be explained.

18.3.3 Other models

Other models include:

- ◇ LORE
- ◇ Adversarial Black Box explainer
- ◇ Adversarial autoencoder
- ◇ Local classifier rule Extraction
- ◇ Saliency Maps

18.3.4 Time Series

For time series data, an explainability method is LASTS (Local Agnostic Subsequence-based Time Series explainer). It generates a synthetic neighborhood by randomly removing subsequences from the time series to be explained. Then it trains an interpretable model (e.g., decision tree) on the neighborhood to explain the black-box prediction. The interpretable model provides insights into which subsequences of the time series are most influential in the black-box model's prediction.

Chapter 19

Responsible AI

We produce an enormous amount of data while running our daily activities. How can we manage all these data? Can we get an added value from them?

Data is gathered from domotics, streaming services, mobile apps, social media, audio sources, images, and so on. Such data is used in a *Gather, Analyze, Gain insight, Influence, Optimize experiences,* (*Sell* ☺), **loop**.

AI is used in Healthcare, Big Data analytics, Social mining. It's the main tool for a Data Scientist to measure, understand and possibly predict **human behaviour**.
Data Scientist needs to take into account **ethical** and **legal** aspects and social impact of data science and AI.

19.1 European guidelines

2019 European guidelines aim at setting the standard for AI use and development. The ambition/goal is a **trustworthy** AI with three key components.

- **Lawful AI** complying with all applicable laws and regulations
- **Ethical AI** ensuring adherence to ethical principles and values
- **Robust AI** perform in a safe, secure and reliable manner, both form technical and a social perspective, with safeguards to foresee and prevent unintentional harm

- Human agency and oversight
  - Fundamental rights
  - Human agency
  - Human oversight
- Technical robustness
  - Resilience to attack and security
  - Safety
  - Accuracy
  - Reliability and reproducibility
- Privacy and data governance
  - Privacy and data protection
  - Quality and integrity of data
  - Access to data
- Transparency
  - Traceability
  - Explainability
- Diversity, non-discrimination and fairness
  - Avoidance of unfair bias
  - Accessibility and universal design
  - Stakeholder Participation
- Societal and environmental well-being
  - Sustainable and environmentally friendly AI
  - Social impact
  - Society and Democracy
- Accountability

Fig. 19.1: AI requirements according to EU

- Minimisation and reporting of negative impacts
- Auditability
- Minimisation and reporting of negative impacts
- Trade-offs

19.2 Privacy and Data Protection

19.2.1 Types of Data

Personal data is defined as any information relating to an identity or identifiable natural person. It includes name, address, photo, email, bank account, ….

***Sensitive* personal data** is a specific set of "special categories" that must be treated with extra security:

- ◇ Racial or ethnic origin
- ◇ Political opinions
- ◇ Religious or philosophical beliefs
- ◇ Trade union membership
- ◇ Genetic data
- ◇ Biometric data