Distributed Information Systems

Lucía Montero Sanchis

Tuesday 24th April, 2018

1 Distributed Information Systems Overview

1.1 What is an Information System

w1 slides 6-18

1.2 Data Management

w1 slides 20-48

1.3 Information Management

w1 slides 50-58

1.4 Distributed Information Management

w1 slides 60-99

2 Information Retrieval

2 slide 2

2.1 Information Retrieval

w2 slides 6-30

2.2 Text-based Information Retrieval

w2 slides 32-47

2.3 Vector Space Retrieval

w2 slides 49-62

2.4 Probabilistic Information Retrieval

w2 slides 64-78

2.5 Query Expansion

w3 slides 2-20

2.6 Inverted Index

w3 slides 22-47

2.7 Distributed Retrieval

w3 slides 49-58

2.8 Latent semantic indexing

2.8.1 Introduction

Vector Space Retrieval model problems: It's based on **index terms**, so there are problems due to *synonyms* ($\Rightarrow$ miss relevant documents, poor recall) and *homonyms* ($\Rightarrow$ include unrelated documents, poor precision).

A **solution** is to use higher-level **concepts** instead, with each concept represented by a combination of terms in the *concept space*. There are much fewer concepts than terms, and the dimensionality of the term space is given by the *vocabulary* size. Both query terms and documents are mapped to the concept space.

Similarity in concept space is computed by scalar product of *normalized concept vectors*. *Concepts* are represented by sets of terms. *Documents* are represented by *concept vectors* that count the number of *concept terms* in the document. The *concept vectors* are then normalized, and we compute the *cosine similarities* among the resulting concept vectors.

Concepts identification and computation can be done manually by users annotating documents using terms of an *ontology*, or can be done *automatically*:

- M_{ij} is a term-document matrix with m rows (terms) and n columns (documents). M^t has a document in each row, and these rows should be normalized to 1.
- Each element of M_{ij} is assigned a weight w_{ij} associated with t_i and d_j. The weight can be based on TF-IDF.

The **ranking** can be computed as $M^t \cdot q$, with q being the *query*.

2.8.2 Identifying top concepts

Key idea: Extract the essential features of M^t and approximate it by the most important ones.

Singular Value Decomposition (SVD) (always exists and is unique):

- $M = K \cdot S \cdot D^t$
- S a $r \times r$ diagonal matrix of the singular values in decreasing order, $r = \min(m, n)$ (rank of M)
- K is the matrix of eigenvectors of $M \cdot M^t$
- D is the matrix of eigenvectors of $M^t \cdot M$

- Algorithms have complexity $O(n^3)$ if $m \leq n$
- *Interpretation*:
  - Rewrite M as *sum of outer vector products*: $\sum_{i=1}^{r} s_i k_i \times d_i^t$ (sum of components weighted by the singular values)
  - s_i ordered in decreasing size.
  - *Least square approximation* is obtained by taking the largest s_i
  - d_i is the i-th row of D
  - *Geometrical interpretation of singular values* – lengths of sexi-axes of hyperellipsoid $E = \{Mx \quad | \quad ||x||_2 = 1\}$. We can interpret the axes of E as the dimensions of the concept space.

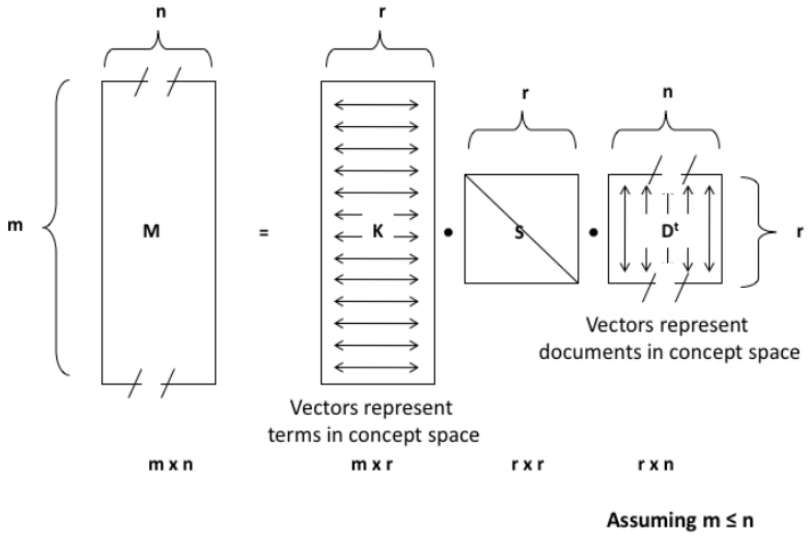

Figure 1: SVD illustration 1

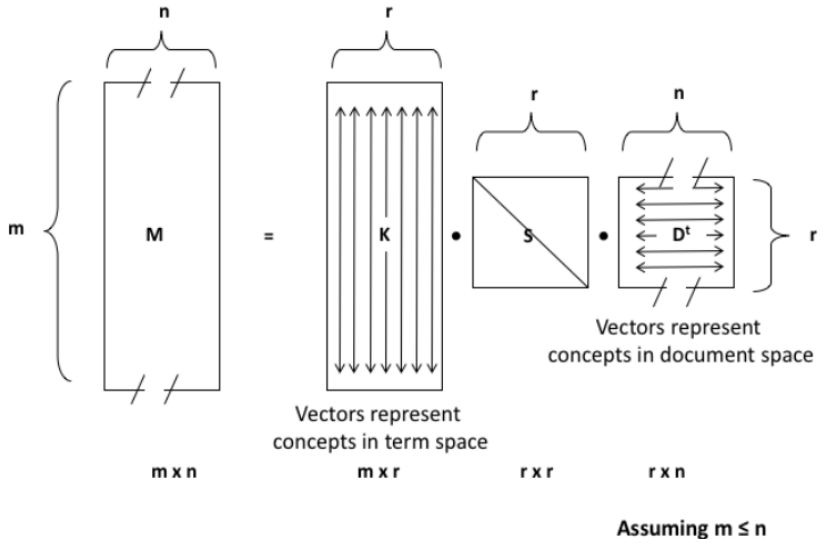

Figure 2: SVD illustration 2

Latent Semantic Indexing (LSI): $M_s = K_s \cdot S_s \cdot D_s^t$ with $s < r$ the dimensionality of the concept space (large enough to allow fitting the data, small enough to filter details). Rows in K_s correspond to *term vectors*, columns in D_s^t correspond to document vectors. By using *cosine similarity* between columns of D_s^t the *similarity of documents* can be computed.

Answering queries: After SVD, documents can be compared by computing *cosine similarity* in the *concept space* (comparing columns $(D_s^t)_i$ and $(D_s^t)_j$ of matrix D_s^t). Queries are treated like another

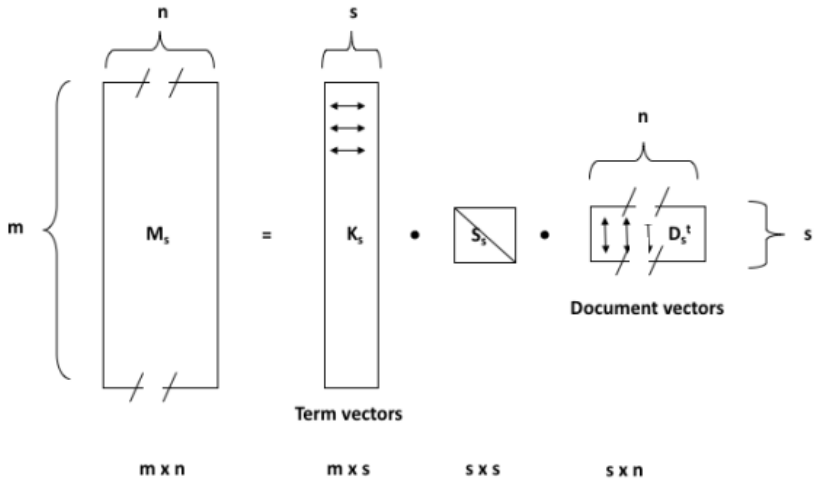

Figure 3: LSI illustration

document (is added as an additional column to M, and then the same transformation is applied as for mapping M to D).

To map M to D:

1. $M = K \cdot S \cdot D^t \Rightarrow D = M^t \cdot K \cdot S^{-1}$
2. Apply same transformation to query q: $q^* = q^t \cdot K_s \cdot S_s^{-1}$
3. Compare using standard cosine measure:

$$sim(q^*, d_i) - \frac{q^* \cdot (D_s^t)_i}{|q^*| \cdot |(D_s^t)_i|}$$

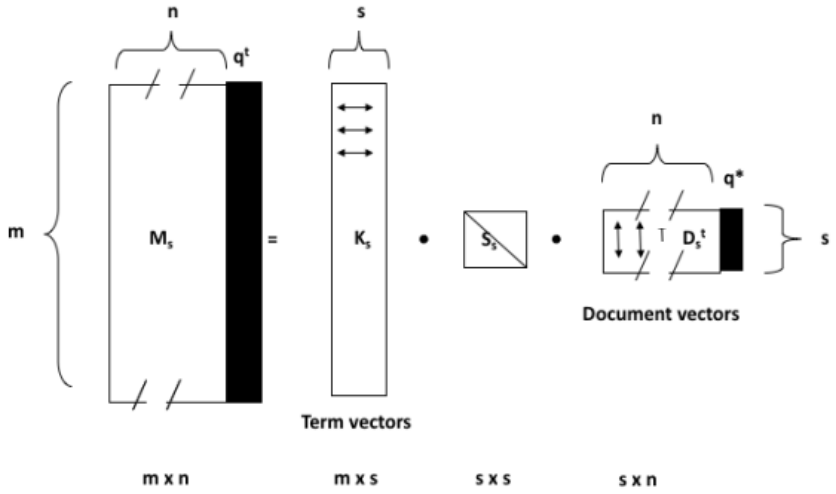

Figure 4: LSI querying illustration

Discussion of LSI: Allows reducing complexity of concept representation and facilitates interfacing with user. Computationally expensive and poor statistical explanation (i.e. poor explanatory power, because the LS approximation assumes normally distributed term frequencies, but term frequencies are power law distributed). **Alternatives to LSI**:

- Probabilistic Latent Semantic Analysis – based on Bayesian Networks
- Latent Dirichlet Allocation (LDA) – based on Dirichlet Allocation, state-of-the-art for concept extraction. Better explained mathematical foundation and better experimental results.

Latent Dirichlet Allocation (LDA): Assumes a document collection is (randomly) generated from a known set of topics (probabilistic generative model, similar to probabilistic language model used in information retrieval).

LDA Document generation using a probabilistic process – (1) For each topic, choose a mixture of topics. (2) For each word position, sample a topic from the topic mixture. (3) For every word position, sample a word from the chosen topic.

LDA Topic Identification – Given a document collection, reconstruct the (probabilistic) topic model that has generated the document collection. Distributions follow a Dirichlet distribution.

Use of Topic Models: *Unsupervised Learning* (understand main topics of a topic collection, organize a document collection); *Information retrieval* (use topic vectors instead of term vectors to represent documents and queries); *Supervised learning* (document classification using topics as features).

2.9 Word embeddings

The neighborhood of a word expresses a lot about its meaning (Firth, 1957). We can consider all the neighborhoods that we can find in all documents of a large document collection.

For each word w we can identify its *context* $C(w)$ ("neighboring words" choosing a certain number of preceding and succeeding words, e.g. 5).

Similarity based representation – Two words are *similar* if they have similar contexts. Advantages compared to co-occurrence methods like LSI:

- The context captures *synctactic* and *semantic* similarity.
- It distinguishes if a word occurs as the *center of a context* and as a *member of a context*.

Approach:

- Words and context-words are mapped into the same low-dimensional space (e.g. $d = 200$). Their representations in the space are similar if word and context-word often occur together.
- The *vector distance* (product) measures how likely the word and its context are likely to occur together. Similar words and contexts should be close (thanks to dimensionality reduction).

Model overview:

- The two W matrices map words and context words
- Mapping a word (word vector w): $V_w = W^{(w)} \cdot w$
- Mapping a context (context vector c): $V_c = W^{(c)} \cdot c$
- Model parameters (θ) are the coefficients of $W^{(w)}$ and $W^{(c)}$
- $W^{(w)}$ is a $m \times d$ matrix with m rows and d columns, with each row representing a word of the vocabulary in the concept space of dimension d.
- In $W^{(c)}$, each column represents a dimension in the concept space – how relevant word c is for each concept, how often context-word c occurs with all words, word c represented in concept space.

The model is learned from the data. Intuition: convert similarity value in vector space into a probability. Consider a pair (w, c) and get probability that it comes from the data

$$p(D = 1|w, c; \theta) = 1/(1 + e^{-v_c \cdot v_w}) = \sigma(v_c \cdot v_w)$$

Problem: finding parameters θ

- Formulate an optimization problem
- Assume we have positive examples D for (w, c) and negative examples $\hat{D}$ (not in document collection).
- Find θ that maximizes overall probability

$$\theta = \text{argmax}_\theta \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \hat{D}} P(D = 0|w, c, \theta)$$

- The previous can be expressed as

$$\text{argmax}_\theta \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w)$$

- Loss function to be minimized: $J(\theta) = 1/m \cdot \sum_{t=1}^{m} J_t(\theta)$, with $J_t(\theta) = -\log(\sigma(v_c \cdot v_w)) - \sum_{k=1}^{K} \log(\sigma(v_{c_k} \cdot v_w)$
- v_{c_k} are negative examples of context words taken from a set $P_n(w) = V$ $C(w)$. Negative examples can be obtained by choosing any word from vocabulary V that is not contained in the context. For p_w the probability of word w in collection, choose the word with probability $p_w^{3/4}$. Less frequent words are sampled more often. In practice the probability is approximated by sampling a few non-context words.
- Given the loss function, optimal θ is obtained using SGD iterating for every $(w, c) \in D$, using

$$\theta^{(t+1)} = \theta^t - \alpha \Delta_\theta J_t(\theta)$$

Result: Matrices $W^{(w)}$ and $W^{(c)}$ that capture information on word similarity. Words appearing in similar contexts generate similar contexts and viceversa ⇒ mapped to similar representations in lower dimensional space. Use $W = W^{(w)} + W^{(c)}$ as the low-dimensional representation.

Alternative approaches, with similar approaches – first model observation on how co-occurrence can capture semantic relationship, then use gradient descent methods to learn parameters.

- CBOW (Continuous Bag of Words) Model – Predict word from context
- GLOVE – *Ratios of probabilities* can capture *semantic relationships* among terms. E.g, `solid` co-occurs more with `ice` than with `water`; `steam` co-occurs with `gas` than with `ice`; `water` co-occurs with both ⇒ captures semantics on the meaning of solid and gas, beyond co-occurrence.

Properties of word embeddings:

- Cluster similar terms (i.e. group similar words together)
- Capture *syntactic* relationships (singular-plural, comparative-superlative...)
- Capture *semantic* relationships (enable *word analogies* as linear mappings, e.g. king - man + woman = queen)
- Capture *semantic meaning in dimensions*

Use of word embedding models: Document search (word embedding vectors as document representation); Thesaurus construction and taxonomy induction (search engine for semantically analogous or related terms); Document classification (word embedding vectors of document terms as features).

Web documents are connected through hyperlinks: *anchor text* describes content of referred document, and *hyperlink* is a quality signal.

2.9.1 Indexing anchor text

Anchor text is the text surrounding a hyperlink, that can contain valuable information on the referred page. For example, `epfl.ch` is pointed by many 'reputed' organization pages, so we can trust `epfl.ch`.

Scoring of anchor text with a weight depending on the authority of the anchor page's website (i.e. trust more anchor text from 'authorative' websites). Also use *non-nepotistic scoring* to avoid *self-promotion* (i.e. score anchor text from other domains higher than text from the same site).

Indexing anchor text can lead to unexpected effects – it's *easily spammable*. Malicious users can create spam pages. This is seen from log-log representation of in-degree versus frequency of pages representation, because spammers violate power laws.

2.9.2 Pagerank

Citation analysis can be used for web document collections with hyperlinks. **Ideas**:

- *Citation frequency* – Popularity/visibility of author
- *Co-citation analysis* – Articles that co-cite same articles are related.
- *Citation indexing* – Explore kind of researchers that are citing a certain author (indicator of *quality* and *discipline*)
- *Impact factor* – Authority of sources, such as journals. Can be used to weight the relevance of publications.

In particular, in the web we are interested in **incoming links** and **number of referrals with high relevance**. Spamming is widespread, e.g. *link farms*.

Link-based ranking idea, that fights spam: based on a *random walker* in the long run.

- *Random walker model*: $P(p_i) = \sum_{p_j \mid p_j \to p_i} P(p_j)/C(p_j)$ where $C(p)$ is the number of outgoing links of page p and $P(p_i)$ is the probability to visit page p_i (i.e. relevance).
  - R is the *transition probability* matrix, and its eigenvectors are the long-term visiting probabilities s.t. $\hat{p} = R \cdot \hat{p}$.
  - L contains the links from one page to another. First *column* are the links *from* first page, first *row* are the links *to* first page.
  - It takes into account the number of referrals and the relevance of referrals and is recursive.
  - $P(p_i)$ defined as a matrix equation:

    slides 14
- *PageRank method*: Adds a 'source of rank', teleporting with probability $1 - q$ according to E.

  slides 17-23

2.9.3 Hyperlink-Induced Topic Search (HITS)

Finds two sets of inter-related pages:

- **Hubs** – Point to many/relevant authorities
- **Authorities** – Pointed to by many/relevant hubs

Computation, in practice 5 iterations are enough:

- $H(p_i) = \sum_{p_j \in N \mid p_i \to p_j} A(p_j)$
- $A(p_i) = \sum_{p_j \in N \mid p_j \to p_i} H(p_j)$
- Normalize values (scaling) to avoid that the scores grow continuously:
  $\sum_{p_j \in N} A(p_j)^2 = 1 \qquad \sum_{p_j \in N} H(p_j)^2 = 1$

slides 28-33

2.9.4 Link indexing

slides 34-41

3 Data Mining

3.1 Introduction to Data Mining

Information systems create a model of reality based on data. **Key tasks**:

- **Data mining** (*data* to *models*) – Given data, find a model that matches the data; provide algorithms that reveal hidden structures in data
- **Retrieval** (*models* to *data*) – Given a model, find some specific data

Growing data collections: the rate at which digital data is produced now exceeds the rate at which storage space grows. Most data is useless, but *data mining* (or *data analytics*) allow to extract **actionable insights** (*understandable* and *useful*).

Challenges:

- *Practical*: Data access and ownership (legal and political reasons, different systems); Domain knowledge and expertise (what we're looking for)
- *Technical*: Data management (Big Data); Data mining algorithms

Classes of Data Mining Problems:

- *Local Properties* – Patterns (association rules, pattern mining)
- *Global model* – Descriptive model (clustering, information retrieval) and Predictive model (classification, regression)
- *Exploratory Data Analysis* – used when no clear idea exists and as preprocessing

Components of Data Mining algorithms:

1. Pattern structure / Model representation (*what we look for*), e.g. dependencies, clusters, decision trees
2. Scoring function (*how well the model fits the data set*), e.g. similarity functions
3. Optimization (*parameter tuning*) and search (*find data satisfying a pattern*)
4. Data management (*implement algorithm for large datasets*), e.g. use of inverted files

Data Mining System: Data mining algorithms are part of larger data mining system that support the pre- an post-processing of the data. A data mining systems performs the following typical tasks (each step can influence the preceding steps):

1. Data extraction / integration / transformation / cleaning – The integrated data is kept in *data warehouses* (databases replicating and consolidating the data). Data cleaning consists on removing inconsistent and faulty data. Data integration and data cleaning are supported by *data warehousing systems*.
2. Data selection – Subsets of the data can be selected from the *data warehouse* for performing specific data mining tasks targeting a specific question. Task-specific data collections are called *data-marts*.
3. Data analytics – The *data mining* algorithm is applied to the *data-mart*. Data mining detects patterns in the data (e.g. association rule mining).
4. Pattern / model assessment – Once specific patterns are detected they can be further processed, e.g. for evaluating how interesting the patterns are.

3.2 Association Rule Mining

3.2.1 Pattern structure

w6 slides 21-24

3.2.2 Scoring function

w6 slides 25-28

3.2.3 Building trees

w6 slides 30-49

3.2.4 Sampling and partitioning

w6 slides 50-57

3.2.5 FP Growth

Apriori is original and most popular rule mining algorithm but others provide better performance under certain circumstances

FP Growth is a frequent itemset discovery *without* candidate itemset generation that aims at main memory implementations (thus less suitable for distributed implementation).

Steps:

1. Build the FP-tree datastructure. It requires 2 passes over the dataset.
2. Extract frequent itemsets directly from the FP-tree.

FP-Tree

w6 slides 58-72

3.3 Clustering

w7 slides 1-33

3.4 Classification

Given a dataset of objects described by d attributes, build a model $X^d \rightarrow C$ that assigns classifies accurately the objects in the *training* set and generalizes well for *test* set. Classification is *supervised* ML (clustering is *non-supervised*).

Characteristics: Predictive accuracy; Speed and scalability (Time to build/use the model, in memory vs. on disk processing); Robustness (Handling noise, outliers and missing values); Interpretability (black box vs. white box, compactness of the model).

Classification Algorithms:

- **Basic Methods** (Naive Bayes, kNN, logistic regression)
- **Ensemble methods** (random forest, gradient boosting) – Collection of simple/weak learners, combine their results. They *lower the probability of making a wrong prediction*: $P(\text{wrong prediction}) = \sum_{i=1}^{L} \binom{L}{i} \epsilon^i (1-\epsilon)^{L-i}$, with L base classifiers. Types:
  - *Bagging* – train learners parallelly on different data samples and combine them with *voting* or *averaging*
  - *Stacking* – combine model outputs with secondstage learner (e.g. linear regression)
  - *Boosting* – train on filtered output of other learners
- **Support Vector Machines**

- **Neural Networks** (CNN, RNN, LSTM)

3.4.1 Decision Trees

Nodes are tests on a *single* attribute. **Branches** are attribute values. **Leaves** contain a *class* label. We can extract **classification rules** from trees, in the form of IF-THEN: one rule created for each path from the root to a leaf, each attribute-value pair along a path forms a conjuction and the leaf node holds the class prediction.

Algorithm:

1. *Tree construction* (top-down, divide-and-conquer)
   - Partition recursively based on *most discriminative* attribute, based on *information gain* (**ID3/C4.5**)
   - *Entropy*: max. (1) if $P = N$, min. (0) if $P = 0$ or $N = 0$

     $$H(P, N) = -(\frac{P}{P+N} \log_2 \frac{P}{P+N}) - (\frac{N}{P+N} \cdot \log_2 \frac{N}{P+N})$$

     Entropy of attribute A (to be minimized):

     $$H(A) = \sum_{i=1}^{V} \frac{P_i + N_i}{P+N} H(P_i, N_i)$$

     Gain of attribute A (to be maximized):

     $$Gain(A) = H(P, N) - H(A)$$

   - *Continuous Attributes*:
     - Use binary decision trees, based on $val(A) < A$.
     - Steps: (1) sort data according to attribute value, (2) compare the *relevant decision points* where we can split (determined by points where class label changes).
     - *Scalability*: With naive implementation, at each step data set is split in subsets associated with a tree node; when evaluating attribute to split by, data needs to be sorted according to these attributes.
       Solution: Create separate sorted attributes tables for each attribute (for presorting of data and maintaining order throughout construction) → Splitting attribute table straightforward, build hash table associating tuple identifiers (TIDs) of data items with partitions, select data from other attribute tables by scanning and probing the hash table.
   - *Categorical Attributes* — Split defined by a subset $X \subseteq domain(A)$ (you need to compute the entropy for each subset)
2. *Partitioning* stops if: (1) All samples belong to the same class (class label of leaf), (2) No attributes left (majority voting) or (3) No samples left
3. *Pruning* – Construction phase does not filter out noise. Strategies:
   - Stop partitioning a node when large majority of samples is positive or negative: $N/(N+P) > (1-\epsilon)$ or $P/(N+P) > (1-\epsilon)$
   - Build the full tree, then replace nodes with leaves labelled with the majority class if the classification accuracy does not change
   - Apply **Minimum Description Length** (MDL) principle:
     Let M_1, M_2, ... M_n be a list of candidate models (i.e. trees), the best one is the one that minimizes $L(M) + L(D|M)$ where $L(M)$ is the length in bits of the description of the model (nb. nodes, nb. leaves, nb. arcs) and $L(M|D)$ is the length in bits of the description of the data when encoded with the model (nb. misclassifications).

Strengths: Automatic feature selection; minimal data preparation; non-linear model; easy interpretation.

Weaknesses: Sensitive to small perturbations in data; tend to overfit; have to be re-trained from scratch with new data.

Properties: the model is a flow-like structure, the score function is the *classification accuracy*, for data management we avoid sorting during splits, and the optimisation is with top-down tree construction + pruning.

3.4.2 Random Forests

Learn K different decision trees from independent samples of the data (*bagging*), they should not be similar since then they'll do **majority vote**.

Sampling strategies:

- *Sampling data* – Each tree is trained on a different data subset
- *Sampling attributes* – Different trees consider different attributes subsets, hence usually split with different features

Algorithm:

1. Draw K bootstrap samples of size N from original dataset, with replacement (*bootstrapping*), $K \approx 500$
2. While constructing the decision tree, select a random set of m attributes out of the p attributes available to infer split (*feature bagging*), $m \lessapprox \sqrt{p}$

Strengths: can model extremely complex decision boundaries without overfit; most popular classifier for dense data (≈ 1000 features); easy implementation; easy parallelization (good for MapReduce)

Weaknesses: Worse than Deep Neural Nets; Needs many passes over the data (at least tree max. depth); easy to overfit; hard to balance accuracy/fit tradeoff

3.5 Mining Social Graphs

pendiente

3.6 Classification Methodology

pendiente

3.7 Document Classification

pendiente

3.8 Recommender Systems

pendiente