Distributed Information Systems

Lucía Montero Sanchis

Sunday 17th June, 2018

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

Association rule mining is a technique for discovering unsuspected data dependencies and is one of the best known data mining techniques.

3.2.1 Pattern structure

$\texttt{Body} \rightarrow \texttt{Head} \quad [\texttt{support}, \texttt{confidence}]$

Single-dimensional rules and multi-dimensional rules: Single-dimensional: $\texttt{buy}(x, \{\texttt{diapers}\}) \rightarrow \texttt{buy}(x, \{\texttt{beer}\}) \quad [0.5, 0.6]$
Multi-dimensional:

$$\texttt{age}(x, \{19-25\}) \wedge \texttt{buy}(x, \{\texttt{diapers}\}) \rightarrow \texttt{buy}(x, \{\texttt{beer}\}) \quad [0.5, 0.6]$$

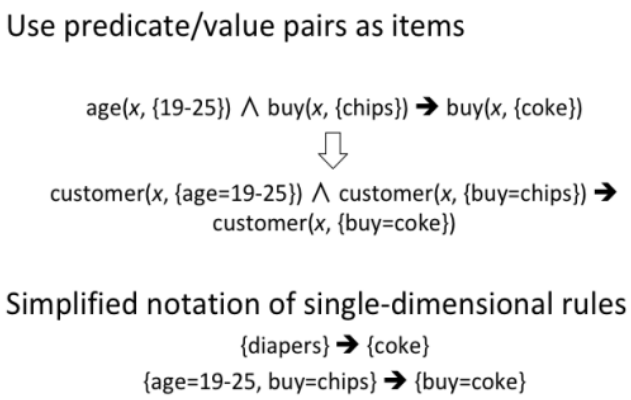


Figure 1: Multi-dimensional rules can be transformed into single-dimensional.

3.2.2 Scoring function

Support (probability that body and head occur in transaction – $P(A \cup B)$) and **confidence** (probability that if body occurs, head also occurs – $P(B|A)$). We want both of them to be high.

Association rule mining approach:

1. Find frequent itemsets (i.e. with enough support)
2. Select pertinent rules (i.e. with enough confidence)

PART 1: FIND FREQUENT ITEMSETS

Some *properties*:

- $A \rightarrow B$ can be an association rule only if $A \cup B$ is a frequent itemset.
- Any subset of a frequent itemset is also a frequent itemset.
- To reduce the search space, find frequent itemsets with increasing cardinality.

Exploiting the *apriori* property:

- Any $(k-1)$ itemset that is a subset of X must be frequent.
- Any ordered $(k-1)$ itemset that is subset of X must be frequent.
- Two ordered $(k-1)$-itemsets that are subsets of X and differ only in their last position must be frequent.

- The union of two $(k-1)$-itemsets that differ only by one item is a **candidate** (i.e. *potential*) frequent k-itemset.

Exploiting the Apriori property:

- **JOIN** step – Given the frequent $(k-1)$-itemsets (L_{k-1}) we can build the candidate set C_k by joining two $(k-1)$-itemsets that only differ by the item in the last position.
  - Sort itemsets in L_{k-1}.
  - Find all pairs with the same first $k-2$ items but different last item.
  - Join the two itemsets in all pairs and add them to C_k.
- **PRUNE** step – A k-itemset in C_k might still contain $(k-1)$-itemsets that are not frequent. Eliminate (*prune*) them by checking if every $(k-1)$ sub-itemset of the candidate itemset is a frequent $(k-1)$-itemset.
- Final step – Check with the DB if the remaining candidates are indeed frequent.

Apriori Algorithm

```
k := 1, L_k := { J ⊆ I : |J|=1 ∧ p(J) > s_min }        //frequent items
while L_k != ∅
        C'_{k+1} := JOIN(L_k)              // join itemsets in L_k that differ by one element
        C_{k+1} := PRUNE(C'_{k+1})                 // remove non-frequent itemsets
        for all (id,T) ∈ D                         // compute the frequency of candidate
                for all J ∈ C_{k+1} , J ⊆ T
                        count(J)++
                end for
        end for
        L_{k+1} := { J ⊆ C_{k+1} : p(J) > s_min }  // add candidate frequent itemsets
        k := k+1
end while
return ∪_k L_k
```

Figure 2: Summary of the complete apriori algorithm.

En resumen:

1. Busco los candidatos para k
2. Compruebo los candidatos para k → PRUNE
3. Busco los candidatos para k+1 a partir de los candidatos para k
4. Compruebo los candidatos para k+1 → PRUNE

   Al final, comparo con la base de datos

PART 2: SELECT PERTINENT RULES

Check for every frequent itemset whether there is a subset A that can occur as the body of a rule (using the support count, i.e. the frequency of the itemset in the database which was obtained during the execution of the Apriori algorithm, to compute the confidence as a conditional probability).

Not all high-confidence rules are interesting or useful.

Alternative measures of interest:

- Added Value $AV(A \to B) = confidence(A \to B) - support(B)$
  We want the ones that have high positive or negative interest values (above 0.5).
- Lift $Lift(A \to B) = confidence(A \to B)/support(B)$

Quantitative attributes: Usually transformed into categorical ones. The rules depend on the chosen discretization.

- *Static discretization* into predefined bins
- *Dynamic discretization* based on data distribution

```
R := ∅                                          // initial set of rules
L := Apriori(D)                                 // set of frequent itemsets
for all J ∈ L
        for all A ⊆ J, A != ∅
                r := A ➔ J \ A                  // create candidate rule
                if c(A ➔ J \ A) = s(J)/s(A)> c_min
                        R := R ∪ r
                end if
        end for
end for
```

Figure 3: Select pertinent rules. Note that $c(A \rightarrow J \backslash A) = s(J \backslash A \cup A)/s(A) = s(J)/s(A)$

3.2.3 Sampling and partitioning

Improvinf Apriori for large datasets:

1. Transaction reduction – if a transaction does not contain any frequent k-itemset, it is useless (and can be ommitted from subsequent DB scans).
2. Sampling – Mining on a sampled subset of DB, with a lower support. Approach:
   (a) Randomly sample transactions with probability p.
   (b) Detect frequent itemsets with support $p \cdot s$.
   (c) Eliminate false positives by counting frequent itemsets on complete data after discovery. False negatives can only be reduced by decreasing the support threshold, at the cost of testing more candidate itemsets when scanning the complete dataset.
   * Refinement: Assume that the m transactions are randomly sorted and just choose the first $p \cdot m$ ones. False negatives can be reduced by choosing a lower threshold (e.g. $0.9 \cdot p \cdot s$).
3. Partitioning (SON algorithm) – Any itemset that is potentially frequent in a DB must be frequent in at least one of the partitions of the DB.
   (Proof: if for all partitions support is below $p \cdot s$, the total support is less than $(1/p) \cdot p \cdot s$).
   Approach:
   (a) Divide transactions in $1/p$ partitions and repeteadly read partitions into main memory.
   (b) Detect in-memory algorithm to find all frequent itemsets with support threshold $p \cdot s$.
   (c) An itemset becomes a candidate if it is frequent in at least one partition.
   (d) On a second pass, count all the candidate itemsets and determine which are frequent in the entire set of transactions.

Partitioning lends itself to distributed data mining – MapReduce implementation (compute frequent itemsets at each node, distribute candidates to all nodes, accumulate the counts of all candidates).

3.2.4 Frequent Pattern (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).

Nodes in the tree correspond to items; paths from the root correspond to itemsets. Counters at the nodes indicate how frequent the itemset corresponding to the path from the root to the node is. Different nodes for the same item are connected by a linked list.

Steps:

1. Build the FP-tree data structure. It requires 2 passes over the dataset.

(a) Sort items. This keeps the FP-Tree compact (more frequent items tend to be on the top of the tree, increasing likelihood of sharing a common prefix).

(b) FP-Tree construction. Requires two passes:

Pass 1 – Compute support for each item:

- Pass over the transaction set.
- Sort items in order of decreasing support.

Pass 2 – Construct the FP-Tree data structure. Tree is expanded one itemset at a time.

2. Extract frequent itemsets directly from the FP-tree.

Another explanation of the steps:

1. Find the support of each item, by looking at the available transactions. Then, remove the ones that have a support lower than the threshold considered.
2. Obtain the *ordered frequent items*: In every transaction, keep only the frequent items and order them in frequency decreasing order.
3. FP Tree construction: (1) Insert the first transaction, adding each item in a 'bubble' with the label `item_id:1`; (2) Insert the second transaction, if there are common items, then increase those items' counters by one; (3...) Repeat for all transactions.
4. Extract the frequent itemsets from the FP-Tree.

w6 - Frequent itemsets, slides 58-72

3.3 Clustering

w7 - Clustering

We want to partition a database into k clusters s.t. intra-cluster similarity is high and inter-cluster similarity is low. Quantitative characteristics: *Scalability* (high n, number of elements) and *dimensionality* (high d, number of dimensions). Qualitative characteristics: Different *types of attributes* (numerical, categorical...) and types of (cluster) *shapes* (spheres, hyperplanes).

Partitioning methods – Heuristic algorithms:

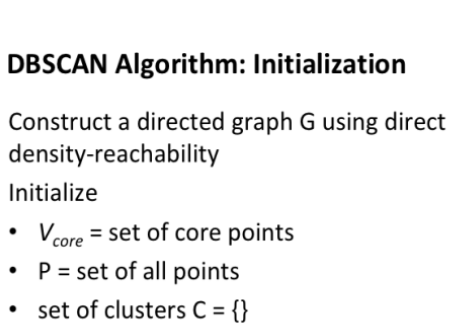
- K-means – Cluster is represented by the point whose mean distance with the objects in the cluster is minimal. It is efficient and converges fast. Gets stuck at local optimum, needs a distance function to compute the mean, k needs to be predefined, does not handle noisy data and outliers, clusters are convex.
  Algorithm:
  1. Initialise k random points as cluster centers. (We often want the smallest k after which the within-cluster sum of squares decreases slowly.)
  2. Assign each object to the nearest cluster center.
  3. While partitioning changes, calculate the centroid of the points for each cluster and set it as the new cluster's center. Then assign each point to the nearest new cluster center.
- K-medoids – Cluster is represented by the object whose mean distance with the objects in the cluster is minimal.
- K-medians – Cluster is represented by the point whose median distance with all the objects in the cluster is minimal.

Advanced clustering:

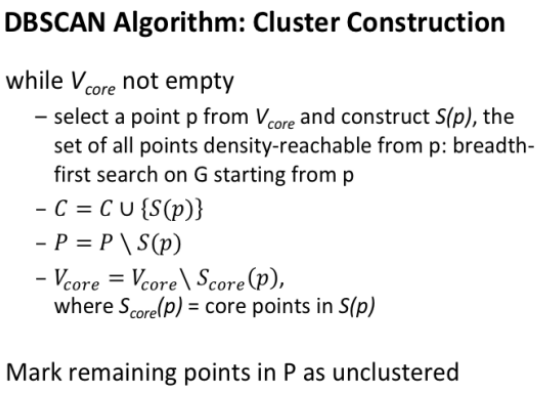
- Density-based clustering (DBSCAN) – Clustering based on a local, density-based criterion.
  - Handles noise, allows arbitrary-shape clusters, clusters in one scan, no predetermined number of clusters. Model parameters: *density parameters*.

- Epsilon-neighborhood: $N_\epsilon(q) = \{p | d(p,q) < \epsilon\}$
- A point q is **core** if $|N_\epsilon \geq \mu|$
- A point p is **directly density-reachable** from q if $p \in N_\epsilon(q)$ and $|N_\epsilon(q) \geq \mu|$ (i.e. if it is in the neighborhood of a core point). Direct density-reachability induces a *directed graph* on the points.
  - A point that is directly density-reachable from q but not a core point is a **border point**.
  - A point that is not directly density-reachable is an **outlier**.
- A point p is **density-reachable** from a point q with ϵ, μ if there is a chain of points $p_1 ... p_n$ ($p_1 = q$, $p_n = p$) s.t. p_{i+1} is directly density-reachable from p_i (asymmetric relationship).
- A point p is **density-connected** to a point q with ϵ, μ if there is a point r s.t. both p and q are density-reachable from r with ϵ, μ (symmetric relationship).
- Clusters satisfy **maximality** (if q in C is a core point and p is density-reachable from q, then p is also in C) and **connectivity** (any two points in C must be density-connected). Clusters may not be disjoint. The set of clusters is unique. Connectivity implies that a cluster contains at least one core point.
- **Complexity**: Construct direct graph ($O(n^2)$); Construct clusters ($O(n^2)$). Worst case complexity for dimension $d = 3$ is $\Omega(n^{4/3})$

- Hierarchical clustering
- Online incremental clustering



(a) DBSCAN algorithm (1)



(b) DBSCAN algorithm (2)

DBSCAN in practice, steps:

1. **Identify core points** – They are the ones that have a "number of neighbors" $\geq t$, with t a predefined threshold (the "number of neighbors" also considers the actual point! e.g. for $t = 3$, a point with two neighbors would be a core point). The neighbors are the points that are within a radial distance of r.
2. **Identify border points** – They are, out of the non-core points, the ones that are within a distance $\leq r$ to (at least) one core point. The rest are outlier/noise points and are ignored in the following.
3. Considering only the **core points**, draw lines connecting pairs of core points that are within a distance $\leq r$ from each other. The resulting "graph" represents the clusters.
4. Now consider each **border point** and find the core point(s) that are within a distance $\leq r$ of the border point. The border point will then be assigned to that core point(s)'s cluster(s).

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 Classification methodology

Credibility evaluation: Data is sometimes questionable, misleading or erroneous.

Data collection and preparation:

1. Feature identification – domain knowledge required.
2. Labelling
   - Ask experts or DIY

- Crowd-sourcing – users can be truthful (expert/normal) or untruthful (sloppy, uniform spammer or random spammer). Aggregation is then required:
  - Non-iterative aggregation algorithms – take the matrix of answers, pre-process it and produce an estimate of the probability of the answer most likely to be correct.
    Examples: *Majority decision (MD), Honey Pot (HP)*
  - Iterative aggregation algorithms – take the matrix, produce an estimate of the probability that a page X is labelled L (E), update the expertise of each worker from the consistency of answers (M), estimate the probability that a page X is labelled L (E)... (continue until convergence).
- Distant learning (i.e. find complementary information sources)

3. Discretization
   - Unsupervised
     - Equal width – Divide into a predefined number of bins.
     - Equal frequency – Divide the range into a predefined number of bins so that every interval contains the same number of values.
     - Clustering – use a clustering method, then assign one feature value per cluster.
   - Supervised – Check whether the class labels depend on the interval or not. Independence test: χ^2 statistics. Approaches:
     - Filtering – Considers features as independent. Ranks features according to predictive power (using χ^2 statistics for the n feature values; $P(\chi^2|DF = n-1)$ gives a rank measure) and selects the best ones. Ignores interaction with the classifier and is independent of the classifier.
       Alternatively, measure the *mutual information* between a class label C and a feature F. 0 means that F doesn't give any information about 0; 1 means that by knowing F we can know C.
     - Wrapper – Considers feature dependence. Iteratively adds features: At each iteration it creates a classifier for each new feature and evaluates its performance, and then adds the best feature or stops when no further improvement. It interacts with the classifier and doesn't assume independence, but is computationally intensive.
4. Feature selection – Different types of features (numerical, ordinal, categorical). Some require categorical features. Consists on reducing the number of N features to an optimal subset of M features ($M < N$).
5. Feature normalization
   - Standardisation (maps to a normal distribution) $x'_i = \frac{x_i - \mu_i}{\sigma_i}$
   - Scaling (mapts to interval 0-1) $x'_i = \frac{x_i - m_i}{M_i - m_i}$ (m_i and M_i are the minimal and maximal values of x_i, respectively).

Expectation Maximisation (EM)

(E) step: estimate $P(x_j = \ell)$ as

$$P(x_j = \ell) = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i=1}^{N} (w_i \mid a_i(x_j) = \ell)$$

(M) step: update the expertise w_i as

$$w_i = \frac{1}{M} \sum_{j=1}^{M} (1 \mid a_i(x_j) = \arg\max_{\ell} P(x_j = \ell))$$

w_i expertise of worker i

M number of webpages to label

Figure 5: Expectation Maximization for aggregating labels obtained through crowd-sourcing.

	Class1	Class2	sum
Interval 1	$O_{11} = n_{11}$ $E_{11} = (R1 \times C1)/N$	$O_{12} = n_{12}$ $E_{12} = (R1 \times C2)/N$	R1
Interval 2	$O_{21} = n_{21}$ $E_{21} = (R2 \times C1)/N$	$O_{22} = n_{22}$ $E_{22} = (R2 \times C2)/N$	R2
sum	C1	C2	N

O_{ij} observed frequency
E_{ij} expected frequency

$$\chi^2 = \sum_{i=1,2} \sum_{j=1,2} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Figure 6: Supervised discretization – Independence test (χ^2 statistics). If

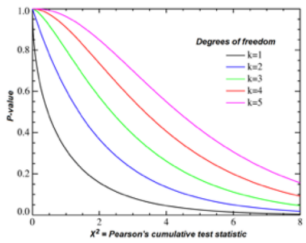


Figure 7: Supervised discretization – Independence test (χ^2 statistics). If `p-value` > 0.05 (independent) then merge the intervals. The number of degrees of freedom DF is computed as $(\texttt{nb.rows} - 1) \cdot (\texttt{nb.cols} - 1)$

$$I(F;C) = H(C) - H(C \mid F) = H(F) + H(C) - H(F,C)$$
$$H(F) = -\sum_{i} P(f_i) \log_2 P(f_i)$$
$$H(F,C) = -\sum_{i} \sum_{j} P(f_i, c_j) \log_2 P(f_i, c_j)$$

Figure 8: Mutual information between feature F and class label C.

Model training:

1. Select performance metrics
   - Model assessment: Having chosen a model, estimate the prediction error on new data.
   - Accuracy = $(TP + TN)/N$
   - Precision = $TP/(TP + FP)$
   - Recall = $TP/(TP + FN)$
   - F-score (or F1-score) = $2 \cdot (P \cdot R)/(P + R)$
   - F-beta score = $(1 + \beta^2 \cdot P \cdot R)/(\beta^2 \cdot P + R)$
2. Model selection – Estimating performances of different models to select the best one (see examples of loss functions). Error is evaluated on training set D, model is tested on independent test set T.
3. Organize training and test data (k-fold cross validation and also its extreme form, leave-one-out cross validation). If skewed distributions, use stratification (select validation set as random sample but making sure each class is approximately proportionally represented) or over- and under-sampling (over-proportional number of samples from smaller class, under-proportional number from larger class).

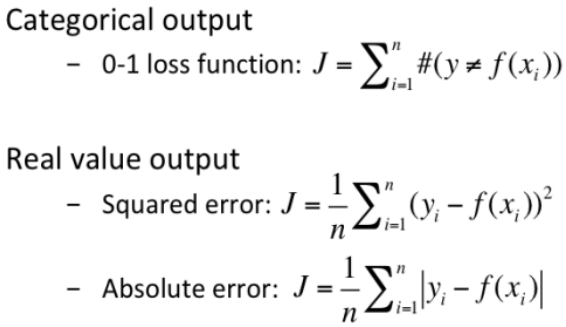


Figure 9: Loss function (error function).

3.6 Recommender systems

Given a user model and a set of items, match users with items by ranking the items in order of decreasing relevance. The user-item ratings matrix has one user per row and one item per column.

- Collaborative recommender system / Collaborative filtering – *tell me what other people like*
  We need to define:
  - Metric to compute users similarity: Pearson correlation coefficient (-1,1) or cosine similarity (considers users as two N-dimensional vectors, if they are identical then the angle between both is 0 and the cosine similarity is 1; the max. angle is $\pi/2$ and thus min. similarity is 0).
  - Number of neighbors for $N_U(u)$.
  - Way to aggregate the ratings.

  It can be user or item based. Both have the cold-start problem but item-based has better scalability (item similarities are more stable, neighborhood considered is small, pair-wise item similarities can be calculated in advance):
  - User-based collaborative filtering – Users give ratings to items, and users with similar tastes in the past will have similar tastes in the future. Given a user u and an item i not rated by u, the estimated rating $r_u(i)$ is obtained by finding a set of users $N_U(u)$ (*neighbors of* u) who rated the same items as u in the past and that have rated i.
  - Item-based collaborative filtering – find a set of items $N_I(i)$ that are similar to i and that have been rated by u, and aggregate them for predicting $r_x(i)$.

- Content-based recommender system – *show me more of what I liked*
  Requires information about the items, to find items that are similar to the ones already rated by the user (see Figs. 14, 15).
  It is the most scalable because tf-idf of items can be computed offline. Tends to recommend similar stuff. Content can be limited or impossible to extract.
- Advanced methods: Matrix factorization – transform the user-item matrix into a latent factor model, describing each user and item in a d-dimensional space (can be understood as a combination of user-based and item-based collaborative filtering).

Pearson correlation coefficient

$$sim(x,y) = \frac{\sum_{i=1}^{N}\left(r_x(i)-\bar{r_x}\right)\left(r_y(i)-\bar{r_y}\right)}{\sqrt{\sum_{i=1}^{N}\left(r_x(i)-\bar{r_x}\right)^2}\sqrt{\sum_{i=1}^{N}\left(r_y(i)-\bar{r_y}\right)^2}}$$

Cosine Similarity

$$sim(x,y) = \cos(\vartheta) = \frac{\sum_{i=1}^{N} r_x(i)\cdot r_y(i)}{\sqrt{\sum_{i=1}^{N} r_x(i)^2}\sqrt{\sum_{i=1}^{N} r_y(i)^2}}$$

x, y: users
N : number of items i rated by both x and y
$r_x(i)$: rating of user x of item i
$\bar{r_x}$: average ratings of user x

Figure 10: Similarity between users (pearson correlation coefficient and cosine similarity).

A common aggregation function is

$$r_x(a) = \bar{r_x} + \frac{\sum_{y\in N_U(x)} sim(x,y)(r_y(a) - \bar{r_y})}{\sum_{y\in N_U(x)} |sim(x,y)|}$$

$N_U(x)$: neighbours of user x
a : item not rated by x

Figure 11: Aggregate the ratings – aggregation function for user-based collaborative filtering.

$$r_x(a) = \frac{\sum_{b\in N_I(a)} sim(a,b) r_x(b)}{\sum_{b\in N_I(a)} |sim(a,b)|}$$

$N_I(a)$: neighbours of item a
b : item rated by x

Figure 12: Aggregate the ratings – aggregation function for item-based collaborative filtering.

Given an item and a textual description (e.g. movie synopsis, book review), compute the tf-idf weights

$$w(t,a) = tf(t,a) \cdot idf(t) = \frac{freq(t,a)}{\max_{s \in T} freq(s,a)} \cdot \log\left(\frac{N}{n(t)}\right)$$

N : number of items to recommend

$n(t)$: number of items where term t appears

Figure 13: Content-based recommendation, TF-IDF.

$$sim(a,b) = \cos(\vartheta) = \frac{\sum_{t=1}^{T} w(t,a) \cdot w(t,b)}{\sqrt{\sum_{t=1}^{T} w(t,a)^2} \sqrt{\sum_{t=1}^{T} w(t,b)^2}}$$

a, b: items

T : number of terms t that appear in all items

$w(t, a)$: tf-idf of term t in item a

Figure 14: Similarity between items – cosine similarity.

$$r_x(a) = \frac{\sum_{b \in N_I(a)} sim(a,b) r_x(b)}{\sum_{b \in N_I(a)} |sim(a,b)|}$$

Figure 15: Making predictions – using the ratings of the nearest neighbors to predict the rating of a with the aggregation function seen before.

$$\min_{q,p} \sum_{(u,i) \in M} (r_u(i) - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

Figure 16: Advanced methods – matrix factorization.

3.7 Mining social graphs

Hierarchical clustering iteratively identifies groups of nodes with high similarity. Types of community detection algorithms:

- Agglomerative algorithms – merge nodes and communities with high similarity.
  Louvain modularity algorithm: based on **modularity** (measure for community quality, the higher the better because it means many mutual connections and less connections to the outside $\sum_{c \in \text{Communities}}$ (nb. edges within c − expected nb. edges within c)); greedily optimizes modularity. The *expected number of edges* is computed assuming m edges and k_i outgoing edges of node i, and that edges connect to $2m$ nodes. See Fig 17 for how to compute modularity with this.
  All possibilities are tested and the best one is chosen. It's complexity is $O(n \cdot \log n)$

  1. Find communities by optimizing modularity locally on all nodes.
  2. Group each community into one new community node.
  3. Repeat until no more new communities are formed.

- Divisive algorithms – split communities by removing links that connect nodes with low similarity.
  Girvan-Newman algorithm: based on **betweenness** measure for edges (that measures how well edges separate communities). Network is decomposed by splitting along edges with highest separation capacity.

  1. Calculate betweenness of edges: fraction of number of shortest paths passing over the edge betweenness$(v) = \sum_{x,y} \frac{\sigma_{xy}(v)}{\sigma_{xy}}$, with σ_{xy} is the number of shortest paths from x to y and $\sigma_{xy}(v)$ is the number of shortest paths from x to y passing through v.
  2. Remove edges with betweenness higher than the threshold specified.
  3. Resulting connected components are communities.

  It can be computed using BFS: (1) Path counting – count the number of shortest paths from first node to all other nodes in the network (2) Edge flow – one unit of flow from first node to all other nodes. The sum of the flows from all nodes equals the betweenness value. This is repeated, each time considering a different "first node". Then they are all summed and divided by 2.

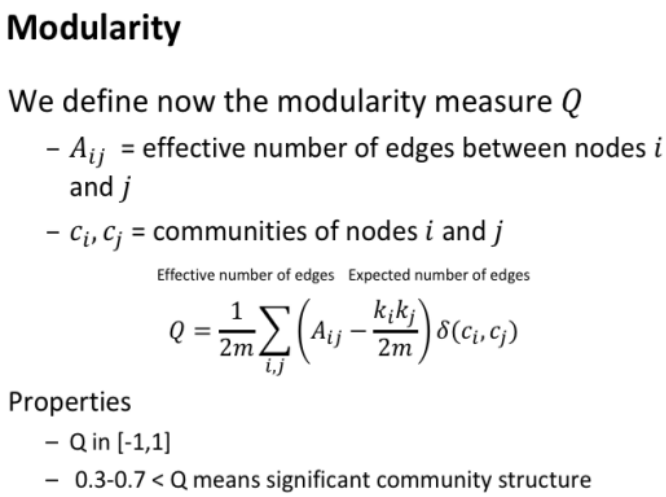


Figure 17: Computing modularity. $\delta(c_i, c_j)$ is 1 if i and j are in the same "community".

3.8 Document classification

Given a training set of labelled documents, build a classifier to decide the label/class for an unlabelled document.

Features:

- Words of the document (bag of words, document vector)

- More detailed information on words (phrases, word fragments, grammatical features)
- Any metadata known about the document and its author

The feature space is very high-dimensional, which can be dealt with using feature selection, dimensionality reduction or classification algorithms that scale well.

Simple method: kNN (uses vector space model). (1) retrieve the k nearest neighbors in the training set according to the vector space model (2) choose the majority class label as the class of the document.

Naive Bayes Classifier applies Bayes law. Feature used is all words and their counts in the document (i.e. bag of words model). The probability the class is C for document D is

$$P(C|D) \; \alpha \; P(C) \prod_{w \in D} P(w|C)$$

Training

How characteristic is word w for class C?

$$P(w|C) = \frac{|w \in D, D \in C| + 1}{\sum_{w'} |w' \in D, D \in C| + 1}$$

How frequent is class C?

$$P(C) = \frac{|D \in C|}{|\mathcal{D}|}$$

Classification: Thus the most probable class is the one with the largest probability

$$C_{NB} = \underset{C}{\operatorname{argmax}} \left(\log P(C) + \sum_{w \in D} \log P(w|C) \right)$$

Figure 18: Naive Bayes Classification Method.

Probability that word w occurs with context word c

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

Figure 19: Classification using word embeddings (e.g. for Fasttext) (1)

Consider instead of (word, context) pairs, (class, paragraph) pairs

- Word w → Class Label C
- Context words c → Paragraph p

Learn $P(C|p) = \frac{e^{v_p \cdot v_C}}{\sum_{C'} e^{-v_p \cdot v_{C'}}}$ (SGD)

Paragraph feature: $v_p = \sum_{w \in p} v_w$

Then predict label from paragraph features

Figure 20: Classification using word embeddings (e.g. for Fasttext) (2)

Fasttext

Classifier based on word embeddings

Extensions

- Using word n-grams (phrases)
- Subword information (character n-grams)

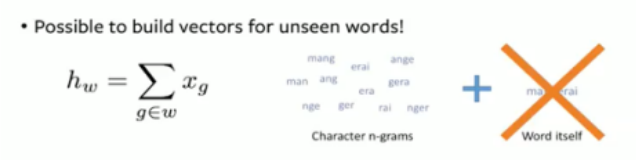


Figure 21: Fasttext.

Document Classification Summary

Widely studied problem with many applications

- Spam filtering, Sentiment Analysis, Document Search
- Increasing interest
  - Powerful methods using very large corpuses
  - Information filtering on the Internet

Rank	Data Mining Methods	# Papers
1	SVM	55
2	KNN	31
3	NB	23
4	ANN	10
5	Rocchio Algorithm	9
6	Association Rule Mining	4

Rank	Feature Selection Methods	# Papers
1	Chi-squared test(CHI)	21
2	Information Gain (IG)	20
3	Mutual Information(MI)	16
4	Latent Semantic Indexing (LSI), Singular Value Decomposition (SVD)	10
5	Document Frequency (DF)	8
6	Term Strength (TS)	3
7	Odds Ratio(OR)	3
8	Linear Discriminant Analysis (LDA)	3

Figure 22: Document classification summary.

Quiz questions

A clique is a subset of vertices, all adjacent to each other – it is also called complete subgraph.

w6 Introduction to data mining, association rule mining (frequent itemsets)

- Multi dimensional reduced to single dimension: 5 attributes, each with 3 possible values, results in 15 items when reduced to single dimension.
- FP trees do not need to have counts ≤ 1 at leaf level.

w7 Clustering

- For clustering a dataset of pictures, k-medoids is best.
- When executing DBSCAN, the result is independent on the order of choosing intial core points. Also, the number of clusters is not independent from the model parameters.

w8 Classification

w9 Classification methodology

- *The training error is* **not** *always higher than the test error.* The higher the data volume, the higher the training error.

w10 Recommender systems, mining social graphs, document classification

- For a user that hasn't rated any item, no method can make a prediction (I think we assume that we do not know anything about the user). For an item that has never been rated, content-based RS can make a prediction (I think it's because item similarity can be computed beforehand).
- Modularity clustering only ends up with a single community at the top level for connected graphs.
- Modularity clustering does not always end up with the same community structure.

4 Knowledge Modeling

4.1 Semantic Web

4.1.1 Semi-structured data

Database schemas define fixed data structures for databases, labeling data values with names (*well understood semantics*), and impose integrity constraints (for data consistency). This results in structurally and semantically coherent data.

Structured layouts (e.g. HTML tables) provide a context, allowing to data values interpretation. Limitations of HTML include: no schemas, no constraints, data semantics difficult to analyse and share, structure of data expressed as layout.

Application-specific markup (tags) are used to provide domain-specific meaning to the data in a document. The data that contains tags, markup, etc to specify the semantics of data values and to relate different data values is called **semi-structured data** (e.g. email, microformats, JSON, Extensible Markup Language XML).

An XML can be viewed as (1) *document model* (hierarchically nested tags enclosing textual content) and (2) *data model* (semi-structured data model). This is good for exchanging data over communication networks.

Schema-less data is more flexible and self-contained, but lacks consistency and certain optimizations are no longer feasible.

4.1.2 Semantic web

User defined markup (*schemas*) allows to share data interpretations across applications, resulting in **semantic hetereogeneity** (which is a problem for Web *interoperability*). The W3C initiated the *Semantic Web* initiative for this purpose, and it includes the XML framework.

Semantic hetereogeneity can be overcome by:

- Standardization – agree on schemas, not useful for existing applications (solution a priori).
- Translation – create mappings among existing schemas/DBs. Useful in small/controlled domains.
- Annotation – reason over the conceptualization and reach an agreement for it (*ontology).*

Ways to relate entities together, according to ISO 14258 (Concepts and rules for enterprise models):

- Integrated approach – common format for all models (**Standardization**).
- Federated approach – no common format; for interoperability, accommodate on the fly. They must share a common ontology. (**Translation**).
- Unified approach – there exists a common format but only at a meta-level, that provides smeantic equivalence to allow model mapping. (**Annotation**).

Ontologies are an explicit representation of a conceptualization of the real world. They should be encoded in a standardized form. They are created following either of these approaches:

- Ontology engineering – manual effort; tools for editing and checking consistency.
- Automatic induction of ontologies – from large document collections or existing structured resources.

Model requirements for ontologies are:

- **Simplicity**, to encourage wide-spread use. Some existing ontologies (e.g. Cyc) are expressed in fairly complex knowledge representation models.
- **Exchangeability**: any kind of data that is processed must be easily exchangeable. This motivated the use of XML as data representation format in the first place.

- **Non-intrusive annotation**: data interpretation is associated with data a-posteriori, and there may be multiple interpretations for the same data. This excludes, for example, the approach to use XML elements for annotation.
- **Domain-specific vocabularies**: the model must provide a mechanism that allows to specify schemas for different domains.
- **Modeling primitives**: they need to offer a sufficiently rich set of possibilities to model complex situations.
- **Reasoning Capabilities**: even simple forms of reasoning within the ontology layer can make the interpretation of the data much more powerful (and thus the processing in the Semantic Web).

The Semantic Web standards RDF and OWL are positioned in the Semantic Web architecture in top of the syntactic layer

4.1.3 Resource Description Framework (RDF)

RDF consists of two parts:

- Language for representing metadata instances, which allows to annotate Web resources with statements (**RDF (instances)**). The web resources are addressed by *Universal Resource Identifiers (URI)* (e.g. URLs).
- Language for specifying schemas for RDF instances (**RDF-schema**). This language enables specification of the vocabulary and grammar that is used for forming statements for annotation.

Since RDF statements can be created also without using RDF schemas, RDF is a semi-structured data model, similar as with well-formed XML (instances) and XML-DTD (schemas).

The RDF model is similar to the entity-relationship (ER) model. Entities correspond to resources and relationships correspond to properties. The main difference is that RDF requires that relationships are directed, and have a specific semantics: the resource from which the (directed) relationship emerges is assigned a property with the value to which the relationship points.

4.1.4 Semantic web resources

4.2 Information Extraction

4.3 Graph Databases