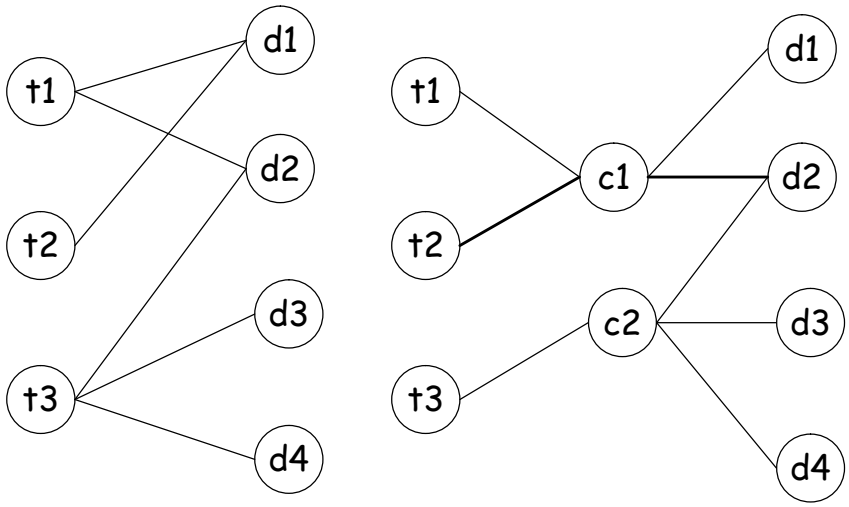
Latent Semantic Indexing

- Vector Space Retrieval might lead to poor retrieval
  - Unrelated documents might be included in the answer set
  - Relevant documents that do not contain at least one index term are not retrieved
- Reasoning
  - retrieval based on index terms is vague and noisy
  - The user information need is more related to concepts and ideas than to index terms
- Key Idea: map documents and queries into a lower dimensional space composed of higher level concepts which are fewer in number than the index terms
- Dimensionality reduction: Retrieval (and clustering) in a reduced concept space might be superior to retrieval in the high-dimensional space of index terms

Despite its success the vector model suffers some problems. Unrelated documents may be retrieved simply because terms occur accidentally in it, and on the other hand related documents may be missed because no term in the document occurs in the query (consider synonyms, there exists a study that different people use the same keywords for expressing hte same concepts only 20% of the time). Thus it would be an interesting idea to see whether the retrieval could be based on concepts rather than on terms, by mapping first terms to a "concept space" (and queries as well) and then establish the ranking with respect to similarity within the concept space. This idea is explored in the following.

Using Concepts for Retrieval



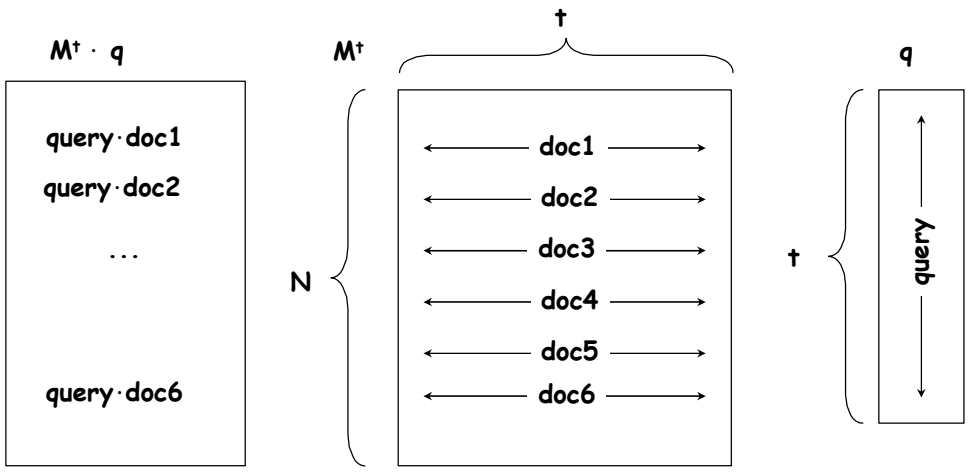
This illustrates the approach: rather than directly relating documents and terms as in vector retrieval, there exists a middle layer into which both queries and documents map. The space of concepts can be of smaller dimension. For example, we could determine that the query t3 returns d2, d3, d4 in the answer set based on the observation that they relate to concept c2, without requiring that the document contains term d3. The question is, of how to obtain such a concept space. One possible way would be to find canonical representations of natural language, but this is a difficult task to achieve. Much simpler, we could try to use mathematical properties of the term-document matrix, i.e. determine the concepts by matrix compuation.

Basic Definitions

- Problem: how to identify and compute "concepts" ?

- Consider the term-document matrix
  - Let M_{ij} be a term-document matrix with t rows (terms) and N columns (documents)
  - To each element of this matrix is assigned a weight w_{ij} associated with k_i and d_j
  - The weight w_{ij} can be based on a tf-idf weighting scheme

Computing the Ranking Using M



This figure illustrates how the term-document matrix M can be used to compute the ranking of the documents with respect to a query q. (The columns in M and q would have to be normalized to 1)

Singular Value Decomposition

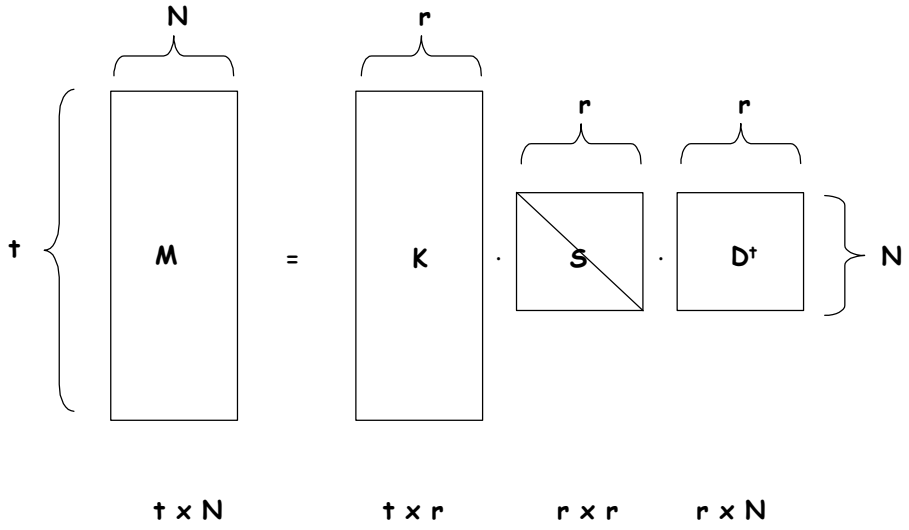
- Key Idea: extract the essential features of M^t and approximate it by the most important ones

- Singular Value Decomposition (SVD)
  - $M = K \bullet S \bullet D^t$
  - K and D are matrices with orthonormal columns
  - S is an r x r diagonal matrix of the singular values sorted in decreasing order where r = min(t, N), i.e. the rank of M
  - Such a decomposition always exists and is unique (up to sign)

- Construction of SVD
  - K is the matrix of eigenvectors derived from $M \bullet M^t$
  - D^t is the matrix of eigenvectors derived from $M^t \bullet M$
  - Algorithms for constructing the SVD of a $m \times n$ matrix have complexity $O(n^3)$ if $m \approx n$

For extracting "conceptual features" a mathematical construction from linear algebra is used, the singular value decomposition (SVD). It decomposes a matrix into the product of three matrices. The middle matrix is a diagonal matrix, where the elements of this matrix are singular values. This decomposition can always be constructed in O(n^3). Note that the complexity is considerable, which makes the approach computationally expensive. There exist however also approximation techniques to compute S more efficiently.

Illustration of Singular Value Decomposition



Assuming N iÂ t

Interpretation of SVD

- First interpretation: we can write

$$M = \sum_{i=1}^{r} s_i k_i d_i^t$$

- The s_i are ordered in decreasing size, thus by taking only the largest ones we obtain a good "approximation" of M
- Second interpretation: the singular values s_i are the lengths of the semi-axes of the hyperellipsoid E defined by

$$E = \left\{ Mx \mid \|x\|_2 = 1 \right\}$$

- Each value s_i corresponds to a dimension of a "concept space"
- Third interpretation: the SVD is a least square approximation

The singular value decomposition is an extremely useful construction to reveal properties of a matrix. This is illustrated by the following facts:

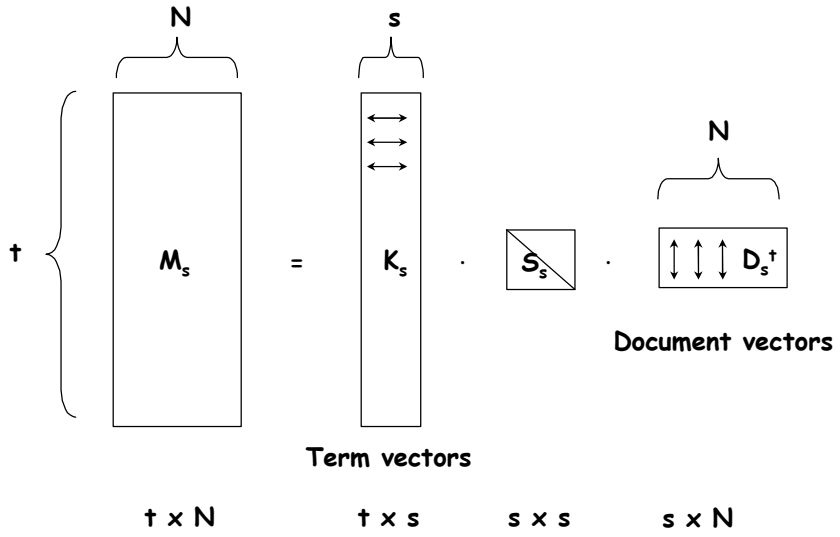
- We can write the matrix as the sum of components weighted by the singular values, thus we can obtain approximations of the matrix by only considering the larger singular values.
- The singular values have also a geometrical interpretation, as they tell us how a unit ball (||x||=1) is distorted by the multiplication with the matrix M. Thus we can view the axes of the hyperellipsoid E as the dimensions of the concept space.
- The SVD after eliminating less important dimensions (smaller singular values) can be interpreted as a least square approximation to the original matrix.

Latent Semantic Indexing

- In the matrix S, select only the s largest singular values
  - Keep the corresponding columns in K and D^t
- The resultant matrix is called M_s and is given by
  - $M_s = K_s \bullet S_s \bullet D_s^t$
    where s, $s < r$, is the dimensionality of the concept space
- The parameter s should be
  - large enough to allow fitting the characteristics of the data
  - small enough to filter out the non-relevant representational details

Using the singular value decomposition, we can now derive an "approximation" of M by taking only the s largest singular values in matrix S. The choice of s determines on how many of the "important concepts" the ranking will be based on. The assumption is that concepts with small singular value in S are rather to be considered as "noise" and thus can be neglected.

Illustration of Latent Semantic Indexing



This illustrates of how the sizes of the involved matrices reduce, when only the first s singular values are kept for the computation of the ranking. The columns in matrix K_s correspond to term vectors, whereas the columns in matrix D_s^t correspond to document vectors. By using the cosine similarity measure between columns the similarity of the documents can be evaluated.

Answering Queries

- Documents can be compared by computing cosine similarity in the "document space", i.e. comparing their rows d_i and d_j in matrix D_s^t
- A query q is treater like one further document
  - it is added like an additional column to matrix M
  - the same transformation is applied as for mapping M to D
- Mapping of M to D
  - $M = K \bullet S \bullet D^t \Rightarrow S^{-1} \bullet K^t \bullet M = D^t$ (since $K \bullet K^t = 1$) $\Rightarrow D = M^t \bullet K \bullet S^{-1}$
- Apply same transformation to q: $q^* = q^t \bullet K_s \bullet S_s^{-1}$
- Then compare transformed vector by using the standard cosine measure

$$sim(q^*, d_i) = \frac{q^* \bullet (D_s^t)_i}{|q^*| \left| (D_s^t)_i \right|}$$

$(D_s^t)_i$ denotes the i-th column of matrix D_s^t

The mapping of query vectors can be mathematically explained as follows: it corresponds to adding a new column (like a new document) to matrix M. We can use the fact that $K_s^t \bullet K_s = 1$

Since $M_s = K_s \bullet S_s \bullet D_s^t$ we obtain $S_s^{-1} \bullet K_s^t \bullet M_s = D_s^t$ or $D_s = M_s^t \bullet K_s \bullet S_s^{-1}$

Thus adding columns to D_s^t requires the transformation that is applied to obtain q*.

Example (SVD, s=2)

K_s

$$\begin{pmatrix} -0.0154227 & -0.422647 \\ -0.0242622 & -0.382996 \\ -0.178994 & -0.196573 \\ -0.603612 & 0.0992276 \\ -0.668904 & 0.135237 \\ -0.0143585 & -0.354329 \\ -0.0123052 & -0.174656 \\ -0.0063823 & -0.0905044 \\ -0.150975 & 0.119194 \\ -0.0816699 & 0.0728611 \\ -0.150975 & 0.119194 \\ -0.178994 & -0.196573 \\ -0.142064 & 0.0983069 \\ -0.00711902 & -0.144923 \\ -0.0954645 & 0.0687813 \\ -0.203256 & -0.579569 \end{pmatrix}$$

S_s

$$\begin{pmatrix} 4.52655 & 0 \\ 0 & 2.74066 \end{pmatrix}$$

D_s^t

$$\begin{pmatrix} -0.147774 & 0.0493447 \\ -0.147774 & 0.0493447 \\ -0.0568424 & -0.634717 \\ -0.312508 & 0.12142 \\ -0.00481714 & -0.187237 \\ -0.0240726 & -0.0608051 \\ -0.00815196 & -0.336379 \\ -0.36892 & 0.197629 \\ -0.0391324 & 0.0516819 \\ -0.314476 & 0.129042 \\ -0.405113 & -0.26937 \\ -0.405113 & -0.26937 \\ -0.33055 & 0.148005 \\ -0.314476 & 0.129042 \\ -0.281123 & 0.0855504 \\ -0.00271846 & -0.0637277 \\ -0.0529817 & -0.414944 \end{pmatrix}$$

SVD for Term-Document Matrix from the running example.

Mapping of Query Vector into Document Space

$$\begin{pmatrix} -0.076442 & -0.605047 \end{pmatrix} = \begin{pmatrix} 0 \\ 2.14007 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1.44692 \end{pmatrix} \begin{pmatrix} -0.0154227 & -0.422647 \\ -0.0242622 & -0.382996 \\ -0.178994 & -0.196573 \\ -0.603612 & 0.0992276 \\ -0.668904 & 0.135237 \\ -0.0143585 & -0.354329 \\ -0.0123052 & -0.174656 \\ -0.0063823 & -0.0905044 \\ -0.150975 & 0.119194 \\ -0.0816699 & 0.0728611 \\ -0.150975 & 0.119194 \\ -0.178994 & -0.196573 \\ -0.142064 & 0.0983069 \\ -0.00711902 & -0.144923 \\ -0.0954645 & 0.0687813 \\ -0.203256 & -0.579569 \end{pmatrix} \begin{pmatrix} 0.220919 & 0. \\ 0. & 0.364876 \end{pmatrix}$$

(query "application theory")

Ranked Result

s=2

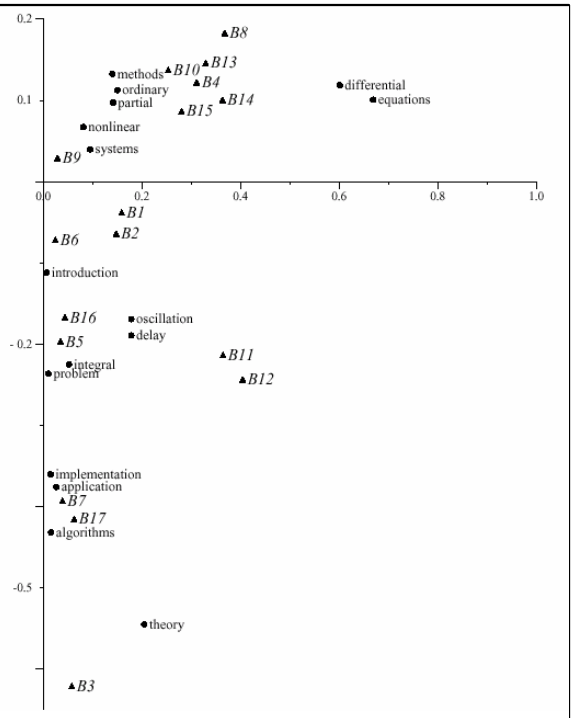
0.999999	{17}
0.999339	{3}
0.996554	{16}
0.995009	{5}
0.994859	{7}
0.968592	{6}
0.653706	{12}
0.653706	{11}
-0.168923	{15}
-0.19534	{1}

s=4

0.992173	{17}
0.970698	{16}
0.837632	{3}
0.537269	{12}
0.537269	{11}
0.434723	{7}
0.348928	{5}
-0.101838	{6}
-0.125203	{15}
-0.131291	{4}

This is the ranking produced for the query for different values of s.

Plot of Terms and Documents in 2-d Space



Since the concept space has two dimensions we can plot both the documents and the terms in the 2 dimensional space. It is interesting to observe of how semantically "close" terms and documents cluster in the same regions. This illustrates very well the power of latent semantic indexing in revealing the "essential" semantics in document collections.

Discussion of Latent Semantic Indexing

- Latent semantic indexing provides an interesting conceptualization of the IR problem
- Advantages
  - It allows reducing the complexity of the underline representational framework
  - For instance, with the purpose of interfacing with the user
- Disadvantages
  - Computationally expensive
  - Assumes normal distribution of terms (least squares), whereas term frequencies are a count

4. Classification

- Data: tuples with multiple categorical and quantitative attributes and at least one categorical attribute (the class label attribute)
- Classification
  - Predicts categorical class labels
  - Classifies data (constructs a model) based on a training set and the values (class labels) in a class label attribute
  - Uses the model in classifying new data
- Prediction/Regression
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical Applications
  - credit approval, target marketing, medical diagnosis, treatment effectiveness analysis

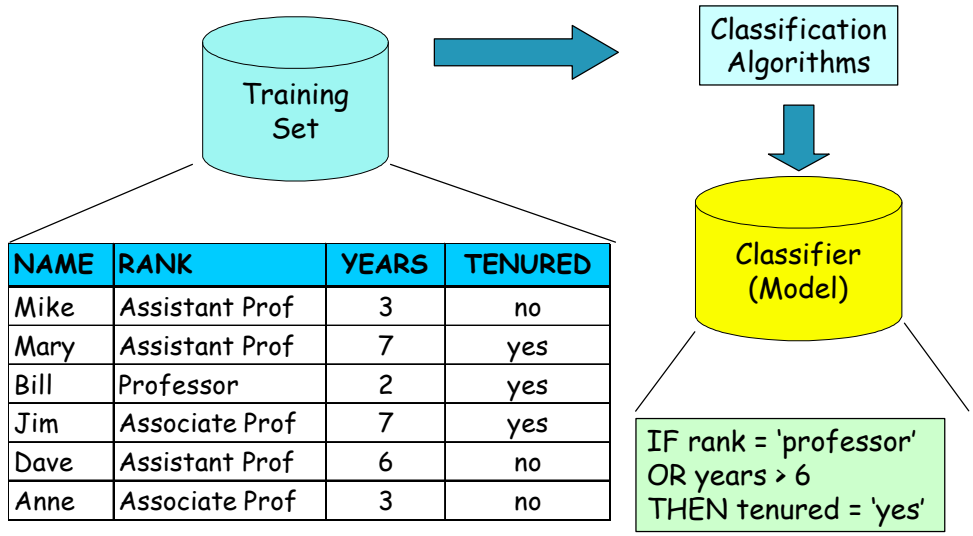
Classification creates a GLOBAL model, that is used for PREDICTING the class label of unknown data. The predicted class label is a CATEGORICAL attribute. Classification is clearly useful in many decision problems, where for a given data item a decision is to be made (which depends on the class to which the data item belongs).

Classification Process

- Model: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class based on its attribute values
  - The class is determined by the class label attribute
  - The set of tuples used for model construction: training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown data
  - Estimate accuracy of the model using a test set
  - Test set is independent of training set, otherwise over-fitting will occur
  - The known label of the test set sample is compared with the classified result from the model
  - Accuracy rate is the percentage of test set samples that are correctly classified by the model

In order to build a global model for classificiation a training set is needed from which the model can be derived. There exist many possible models for classification, which can be expressed as rules, decision trees or mathematical formulae. Once the model is built, unknown data can be classified. In order to test the quality of the model its accuracy can be tested by using a test set. If a certain set of data is available for building a classifier, normally one splits this set into a larger set, which is the training set, and a smaller set which is the test set.

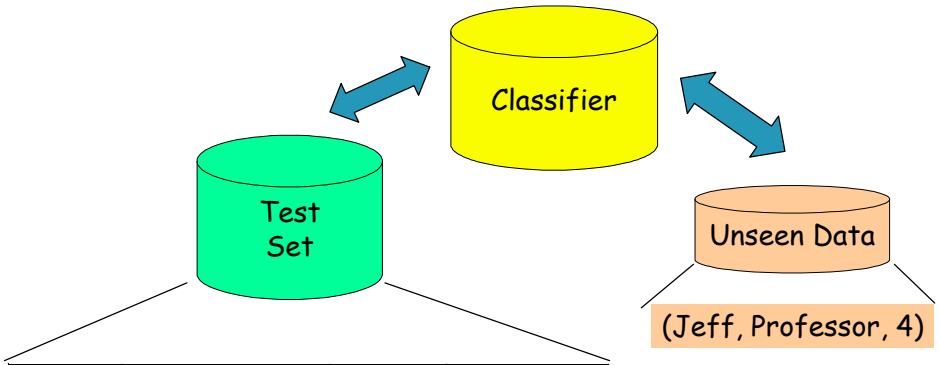
Classification: Training



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

In classification the classes are known and given by so-called class label attributes. For the given data collection TENURED would be the class label attribute. The goal of classification is to determine rules on the other attributes that allow to predict the class label attribute, as the one shown right on the bottom.

Classification: Model Usage



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

In order to determine the quality of the rules derived from the training set, the test set is used. We see that the classifier that has been found is correct in 75% of the cases. If rules are of sufficient quality they are used in order to classify data that has not been seen before. Since the reliability of the rule has been evaluated as 75% by testing it against the test set and assuming that the test set is a representative sample of all data, then the reliability of the rule applied to unseen data should be the same.

Criteria for Classification Methods

- Predictive accuracy
- Speed and scalability
  - time to construct the model
  - time to use the model
  - efficiency in disk-resident databases
- Robustness
  - handling noise and missing values
- Interpretability
  - understanding and insight provded by the model
- Goodness of rules
  - decision tree size
  - compactness of classification rules

Classification by Decision Tree Induction

- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases
  - Tree construction
    - At start, all the training samples are at the root
    - Partition samples recursively based on selected attributes
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying an unknown sample
  - Test the attribute values of the sample against the decision tree

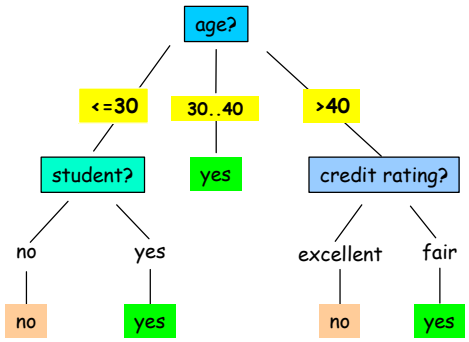
In the following we will introduce a method to construct a specific kind of classification models, namely decision trees. A decision tree splits at each node the data set into smaller partitions, based on a test predicate that is applied to one of the attributes in the tuples. Each leaf of the decision tree is then associated with one specific class label.

Generally a decision tree is first constructed in a top-down manner by recursively splitting the training set using conditions on the attributes. How these conditions are found is one of the key issues of decision tree induction. After the tree construction it usually is the case that at the leaf level the granularity is too fine, i.e. many leaves represent some kind of exceptional data. Thus in a second phase such leaves are identified and eliminated.

Using the decision tree classifier is straightforward: the attribute values of an unknown sample are tested against the conditions in the tree nodes, and the class is derived from the class of the leaf node at which the sample arrives.

Classification by Decision Tree Induction

income	student	credit_rating	buys_computer
high	no	fair	no
high	no	excellent	no
high	no	fair	yes
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
low	yes	excellent	yes
medium	no	fair	no
low	yes	fair	yes
medium	yes	fair	yes
medium	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes
medium	no	excellent	no



buys_computer ?

A standard approach to represent the classification rules is by a decision tree. In a decision tree at each level one of the existing attributes is used to partition the data set based on the attribute value. At the leaf level of the classification tree then the values of the class label attribute are found. Thus, for a given data item with unknown class label attribute, by traversing the tree from the root to the leaf its class can be determined. Note that in different branches of the tree, different attributes may be used for classification. The key problem of finding classification rules is thus to determine the attributes that are used to partition the data set at each level of the decision tree.

Algorithm for Decision Tree Construction

- Basic algorithm for categorical attributes (greedy)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training samples are at the root
  - Examples are partitioned recursively based on test attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left
- Attribute Selection Measure
  - Information Gain (ID3/C4.5)

The basic algorithm for decision tree induction proceeds in a greedy manner. First all samples are at the root. Among the attributes one is chosen to partition the set. The criterion that is applied to select the attribute is based on measuring the information gain that can be achieved, or how much uncertainty on the classification of the samples is removed by the partitioning. Three conditions can occur such that no further splits can be performed:

- all samples are in the same class, therefore further splitting makes no sense,
- no attributes are left which can be used to split. Still samples from different classes can be in the leaf, then majority voting is applied.
- no samples are left.

Which Attribute to Split ?

Maximize Information Gain

Class P: buys_computer = "yes"

Class N: buys_computer = "no"

I(p, n) = I(9, 5) =0.940

age	pi	ni	I(pi, ni)
<=30	2	3	0.971
30…40	4	0	0
>40	3	2	0.971

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.69$$

$$Gain(age) = I(p,n) - E(age) = 0.250$$

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

The amount of information, needed to decide if an arbitrary example in S belongs to P or N

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Attribute A partitions S into $\{S_1, S_2, \ldots, S_v\}$

If S_i contains p_i examples of P and n_i examples of N, the expected information needed to classify objects in all subtrees S_i is

$$E(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

The encoding information that would be gained by branching on A

$$Gain(A) = I(p,n) - E(A)$$

Here we summarize the basic idea of how split attributes are found during the construction of a decision tree. It is based on an information-theoretic argument. Assuming that we have a binary category, i.e. two classes P and N into which a data collection S needs to be classified, we can compute the amount of information required to determine the class, by I(p, n), the standard entropy measure, where p and n denote the cardinalities of P and N. Given an attribute A that can be used for partitioning further the data collection in the decision tree, we can calculate the amount of information needed to classify the data after the split according to attribute A has been performed. This value is obtained by calculating I(p, n) for each of the partitions and weighting these values by the probability that a data item belongs to the respective partition. The information gained by a split then can be determined as the difference of the amount of information needed for correct classification before and after the split. Thus we calculate the reduction in uncertainty that is obtained by splitting according to attribute A and select among all possible attributes the one that leads to the highest reduction. On the left hand side we illustrate these calculations for our example.

Pruning

- Classification reflects "noise" in the data
  - Remove subtrees that are overclassifying
- Apply Principle of Minimum Description Length (MDL)
  - Find tree that encodes the training set with minimal cost
  - Total encoding cost: cost(M, D)
  - Cost of encoding data D given a model M: cost(D | M)
  - Cost of encoding model M: cost(M)

  cost(M, D) = cost(D | M) + cost(M)
- Measuring cost
  - For data: count misclassifications
  - For model: assume an appropriate encoding of the tree

It is important to recognize that for a test dataset a classifier may overspecialize and capture noise in the data rather than general properties. One possiblity to limit overspecialization would be to stop the partitioning of tree nodes when some criteria is met (e.g. number of samples assigned to the leaf node). However, in general it is difficult to find a suitable criterion. Another alternative is to first built the fully grown classification tree, and then in a second phase prune those subtrees that do not contribute to an efficient classification scheme. Efficiency can be measured in that case as follows: if the effort in order to specify a class (the implicit description of the class extension) exceeds the effort to enumerate all class members (the explicit description of the class extension), then the subtree is overclassifying and non-optimal. This is called the priniciple of minimum description length. To measure the description cost a suitable metrics for the encoding cost, both for trees and data sets is required. For trees this can be done by suitably counting the various structural elements needed to encode the tree (nodes, test predicates), whereas for explicit classification, it is sufficient to count the number of misclassifications that occur in a tree node.

Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
  - One rule is created for each path from the root to a leaf
  - Each attribute-value pair along a path forms a conjunction
  - The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF age = "<=30" AND student = "no"
THEN buys_computer = "no"

IF age = "<=30" AND student = "yes"
THEN buys_computer = "yes"

IF age = "31…40"
THEN buys_computer = "yes"

IF age = ">40" AND credit_rating = "excellent"
THEN buys_computer = "yes"

IF age = ">40" AND credit_rating = "fair"
THEN buys_computer = "no"

A decision tree can also be seen as an implicit description of classification rules. Classification rules represent the classification knowledge as IF-THEN rules and are easier to understand for human users. They can be easily extracted from the classification tree as described.

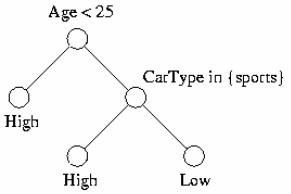
Decision Tree Construction with Continuous Attributes

- Binary decision trees
  - For continuous attributes A a split is defined by val(A) < X
  - For categorical attributes A a split is defined by a subset X $\subseteq$ domain(A)
- Determining continuous attribute splits
  - Sorting the data according to attribute value
  - Determine the value of X which maximizes information gain by scanning through the data items

continuous categorical class

rid	Age	Car Type	Risk
0	23	family	High
1	17	sports	High
2	43	sports	High
3	68	family	Low
4	32	truck	Low
5	20	family	High

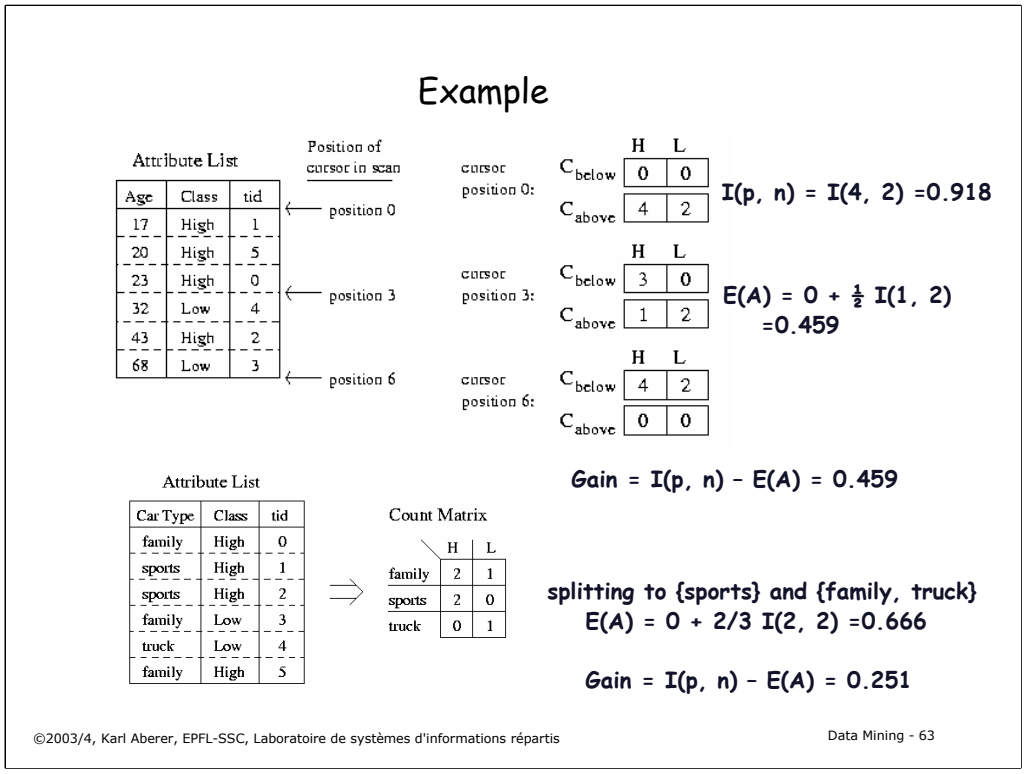
(a) Training Set



(b) Decision Tree

If continuous attributes occur the decision tree can be constructed as a binary decision tree, by finding an attribute value that splits the samples into 2 partitions. Consequently also categorical attributes are treated that way. In order to determine suitable split points for continuous attributes the samples need first to be sorted.

On the left we see a sample database with the class label attribute Risk and a continuous attribute Age and a categorical attribute Car Type used for classification.



This example illustrates the principle of how splitting is performed both for continuous and categorical attributes.

For reasons we will discuss later we construct separate attribute lists for each attribute, that is used for classification. The attribute list contains the attribute for which it is constructed (i.e. Age and Car Type), the class label attribute and the transaction identifier tid. The attribute list is sorted for continuous attributes.

Now let us see of how a split point is found for the continuous attribute Age. The distribution of the class attribute for the whole data set (4 High and 2 Low) is stored in variables C_above and a pointer is positioned on top of the attribute list. Then the pointer is moved downwards. Whenever the Age value changes the values C_below and C_above are updated (such that they always keep the distribution of H and L values above and below the pointer). Also, when the Age value changes the information gain is computed if the split were performed at that point (in the same way as done for categorical attributes before). After passing through the attribute list the optimal split value for the Age attribute is known.

For the categorical attribute we have to establish a statistics of the distribution of the classes for each of the possible attribute values and store it in a matrix. Then we check the information gain that can be obtained for each of the possible subsets of attribute values and thus determine the optimal "split" for the categorical attribute.

Finally, the attribute is chosen that results in the best (binary) split.

Scalability

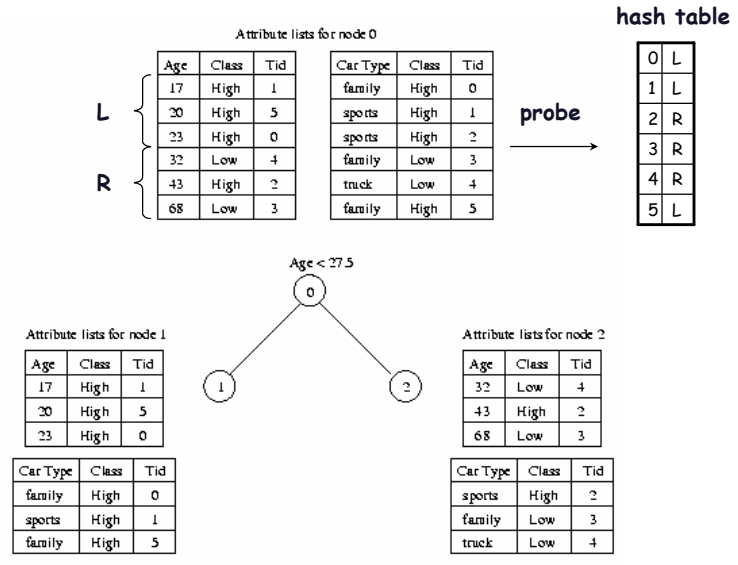
- Naive implementation
  - At each step the data set is split and associated with its tree node
- Problem with naive implementation
  - For evaluating which attribute to split data needs to be sorted according to these attributes
  - Becomes dominating cost
- Idea: Presorting of data and maintaining order throughout tree construction
  - Requires separate sorted attribute tables for each attribute
  - Attribute selected for split: splitting attribute table straightforward
  - Build Hash Table associating TIDs of selected data items with partitions
  - Select data from other attribute tables by scanning and probing the hash table

If we would associate the complete table of samples with the nodes of the classification tree during its induction, we would have to re-sort the table constantly, when searching for split points for different continuous attributes. This would become for large databases the dominating cost in the algorithm. For that reason the attribute values are stored in separate tables (as we have already seen in the example before) which are sorted once at the beginning. After a split the order in the attribute tables can be maintained:

- For the attribute table on which the split occurs the table needs just to be cut into two pieces.
- For the other tables a scan is performed, after building a hash table of the TIDs is constructed for associating the TIDs with their partition. During the scan the hash table is probed in order to redirect the tuples to their proper partition. The order of the attribute table is maintained during that process.

Remark: this is a similar idea as was employed in the construction of inverted files !

Example



This illustration demonstrates a split of attribute tables.

Summary

- What is the difference between clustering and classification ?
- What is the difference between model construction, model test and model usage in classification ?
- Which criterion is used to select an optimal attribute for partitioning a node in the decision tree ?
- How are clusters characterized ?
- When is the k-Means algorithm terminating ?

References

- Textbook
  - Jiawei Han, Data Mining: concepts and techniques, Morgan Kaufman, 2000, ISBN 1-55860-489-8
- Some relevant research literatue
  - R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD'93, 207-216, Washington, D.C.