Methods for Intelligent Systems

Lecture Notes on Clustering (IV)
2009-2010

Davide Eynard

eynard@elet.polimi.it

Department of Electronics and Information

Politecnico di Milano

Course Schedule [*Tentative*]

Date	Topic
11/03/2010	Clustering: Introduction
18/03/2010	Clustering: K-means & Hierarchical
25/03/2010	Clustering: Fuzzy, Gaussian & SOM
08/04/2010	Clustering: PDDP & Vector Space Model
15/04/2010	Clustering: Limits, DBSCAN & Jarvis-Patrick
29/04/2010	Clustering: Evaluation Measures

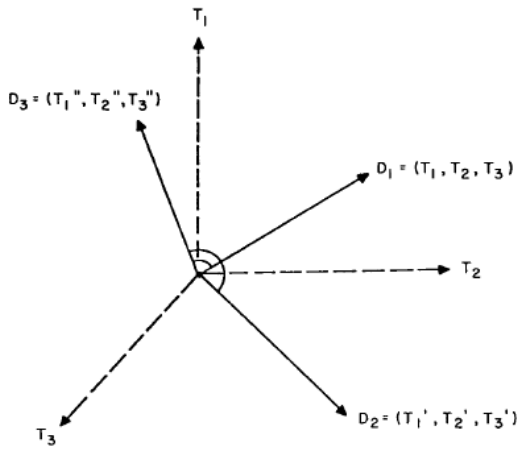
Search Engines

How do search engines work?

- document retrieval and indexing
- query language that allows to search for Web pages that contain (or don't contain) specified words and phrases
- SE have their roots in *information retrieval* systems, which prepare a keyword index for the given corpus and respond to keyword queries with a ranked list of documents
- some queries:
  - docs containing the word "Java"
  - docs containing "Java" but not "coffee"
  - docs containing the phrase "Java Beans" but not "API"
  - docs where "Java" and "island" occur in the same sentence

VSM

In the Vector Space Model, documents are represented as vectors in a multidimensional Euclidean space.



VSM

The coordinate of document d in the direction corresponding to the term t is determined by two quantities:

- *Term Frequency* $TF(d,t)$: this is simply $n(d,t)$, the number of times term t occurs in document d, scaled to normalize document length
- *Inverse Document Frequency* $IDF(t)$: this is a weight factor used to scale down the coordinates of terms which occur in many documents

$$IDF(t) = log\frac{1+|D|}{|D_t|}$$

VSM

TF and IDF are combined into the complete vector-space model in the obvious way: the coordinate of document d in axis t is given by

$$d_t = TF(d,t)IDF(T)$$

Then, the *cosine distance* between the two vectors

$$v_{d1} = [w_{1,d1}, w_{2,d1}, \ldots, w_{N,d1}]^T$$
$$v_{d2} = [w_{1,d2}, w_{2,d2}, \ldots, w_{N,d2}]^T$$

is calculated as

$$cos\theta = \frac{v_1 \cdot v_2}{||v_1||||v_2||}$$

Note: one of the two vectors might be the query itself!

Limits of VSM

- Long documents are poorly represented, because they have poor similarity values (a small scalar product and a large dimensionality)
- Search keywords must precisely match document terms; word substrings might result in a "false positive" match
- Semantic sensitivity: documents with similar context but different term vocabulary won’t be associated, resulting in a "false negative" match

PDDP

PDDP stands for "Principal Direction Divisive Partitioning"

- **Principal Direction**, because the algorithm is based on the computation of the leading principal direction at each stage of the partitioning (NOTE: similarity measure is not used to do the actual splitting)
- **Partitioning**, because we start placing all data into one cluster, so that at every stage clusters are disjoint and their union equals the entire set of documents
- **Divisive**, because it’s a *hierarchical divisive* (vs. *hierarchical agglomerative*) clustering algorithm

PDDP Algorithm Description

- Sample space of m samples in which each sample (document) is an n-vector containing a numerical value
- Each document is represented by a column vector of attribute values

$$\mathbf{d} = (d_1, d_2, \ldots, d_n)^T$$

whose i-th entry, d_i, is the relative frequency of the i-th word.

- Each document vector is normalized to have a euclidean length of 1:

$$d_i = \frac{TF_i}{\sqrt{\sum_j (TF_j)^2}}$$

where TF_i is the number of occurrences of word i in the particular document **d**.

PDDP Algorithm Description

- The entire set of documents is represented by an $n \times m$ matrix $\mathbf{M} = (\mathbf{d}_1, \ldots, \mathbf{d}_m)$ whose i-th column, $\mathbf{d}_i$, is the column vector representing the i-th document
- The algorithm proceeds by separating the entire set of documents into two partitions by using principal directions
  - Each of the two partitions will be splitted into two subpartitions using the same process recursively
  - The result is a hierarchical structure of partitions arranged into a binary tree

What method is used to split a partition into two subpartitions?
In what order are the partitions selected to be split?

Splitting a partition

- The *mean* or *centroid* of the document set is

$$\mathbf{w} = \frac{\mathbf{d}_1 + \ldots + \mathbf{d}_m}{m} = \mathbf{M} \cdot \mathbf{e} \cdot \frac{1}{m}$$

- In the general case, the covariance matrix is

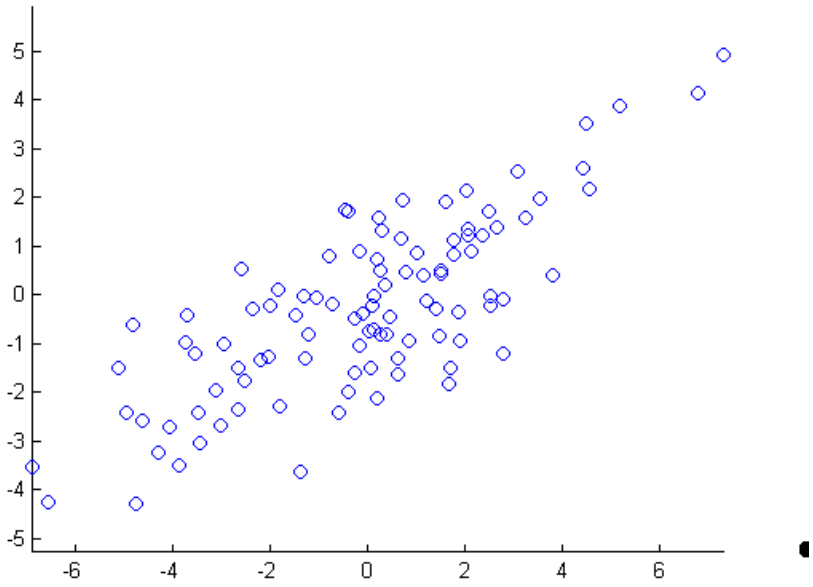
$$\mathbf{C} = (\mathbf{M} - \mathbf{w}\mathbf{e}^T) \cdot (\mathbf{M} - \mathbf{w}\mathbf{e}^T)^T = \mathbf{A} \cdot \mathbf{A}^T$$

  where $\mathbf{e} = (1, 1, \ldots, 1)^T$ is a vector of appropriate dimension.
- **A** is symmetric and positive definite, so all its eigenvalues are real and non-negative
- The eigenvectors corresponding to the k largest eigenvalues are called the *principal components* or *principal directions*

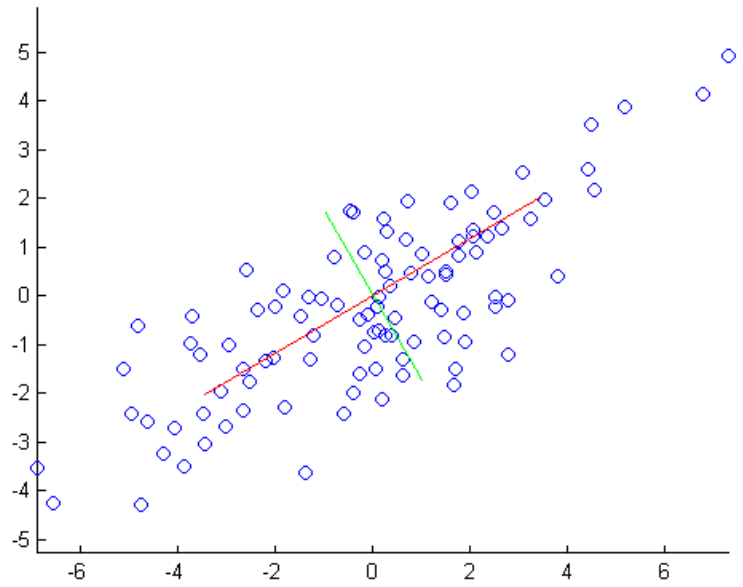
Splitting a partition

- A partition of p documents is represented by an $n \times p$ matrix $\mathbf{M}_p = (\mathbf{d}_1, \ldots, \mathbf{d}_p)$ where each $\mathbf{d}_i$ is an n-vector representing a document.
  - The matrix $\mathbf{M}_p$ is a submatrix of **M** consisting of some selection of p columns of **M**, not necessarily the first p in the set!
- The principal directions of the matrix $\mathbf{M}_p$ are the eigenvectors of its sample covariance matrix **C**
- We're interested in temporarily projecting each document onto the single leading eigenvector **u** (the *principal direction*)
  - Besides reducing the dimensionality, the transformation often has the effect of removing much noise present in the data

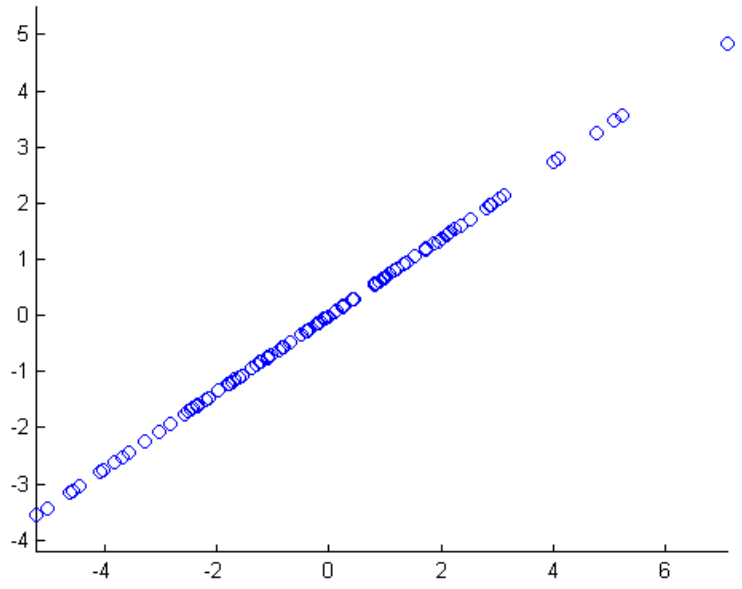
Principal Component Analysis



Principal Component Analysis



Principal Component Analysis



Splitting a partition

- The projection of the i-th document $\mathbf{d}_i$ is given by the formula

$$\sigma v_i = \mathbf{u}^T(\mathbf{d}_i - \mathbf{w})$$

  where σ is a positive constant.
  - All the documents are translated so that their mean is at the origin, then they're projected on the principal direction
  - The values $v_1, \ldots, v_k$ are used to determine the splitting (accordingly to their sign)
- We still have to decide at each stage which node should be split next:
  - A "scatter" value is used to measure the distance between each document in the cluster and the overall mean of the cluster (a measure of its *cohesiveness*)

The algorithm

Start with $n \times m$ matrix **M** of (scaled) document vectors, and a desired number of clusters c_{max}.

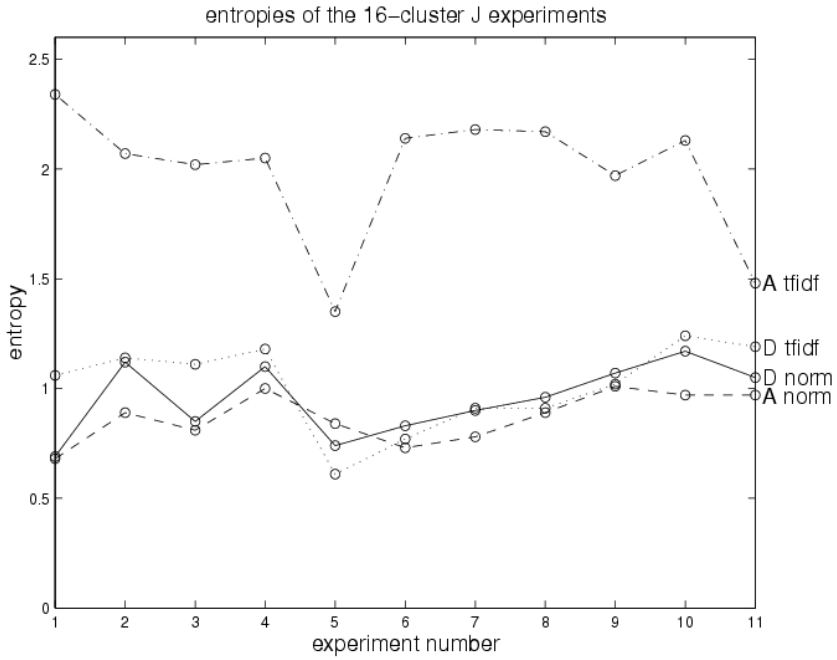
1. Initialize Binary Tree with a single Root Node
2. For $c = 2, 3, \ldots, c_{max}$ do
3. Select node K with largest scat value
4. Create nodes L:=leftchild(K) and R:=rightchild(K)
5. Set indices(L):=indices of the non-positive entries in rightvec(K)
6. Set indices(R):=indices of the positive entries in rightvec(K)
7. Compute all the other fields for the nodes L,R
8. end.

Result: A binary tree with c_{max} leaf nodes forming a partitioning of the entire data set.

Conclusions

- PDDP algorithm is effective at least as well as an agglomeration algorithm, but is much faster
- Its expected running time is linear in the number of documents, whereas unmodified agglomeration algorithms typically have $O(m^2)$ running time

Experimental results



Experimental results

expe- riment	word count	contents summary
J1	10536	all words
J6	5106	words with TF $>$ 1
J4	2951	Top 5+ words + HTML-tagged words
J3	1763	Top 20+ words
J7	1328	Top 20+ with TF $>$ 1
J8	1105	Top 15+ with TF $>$ 1
J2	946	words accounting for 25% of document
J9	805	Top 10+ with TF $>$ 1
J10	474	Top 5+ with TF $>$ 1
J5	449	words appearing in a-priori word clusters
J11	183	hypergraph word clusters.

Table 3: Experimental data set summary, sorted by number of words. TF refers to Text Frequency (number of occurrences of a word).

Experimental results

data set	divisive algorithm 8 clusters	16 clusters	32 clusters	agglomerative algorithm 16 clusters	32 clusters
J1	1:40	1:54	2:10	–	–
J6	0:39	0:44	0:50	53:15	52:57
J4	0:24	0:27	0:30	29:23	29:14
J3	0:15	0:17	0:20	16:54	16:48
J7	0:13	0:14	0:17	11:35	11:31
J8	0:21	0:23	0:25	11:31	11:23
J2	0:11	0:12	0:14	7:49	7:45
J9	0:10	0:11	0:14	6:19	6:15
J10	0:08	0:09	0:11	3:51	3:49
J5	0:11	0:13	0:16	3:38	3:35
J11	0:07	0:08	0:11	1:45	1:43

Table 5: Comparative times (min:sec) for divisive and agglomerative algorithms, in Matlab 5 on an SGI Challenge 194 MHz processor.