Fast Document Clustering with Search Technologies

Debasis Ganguly

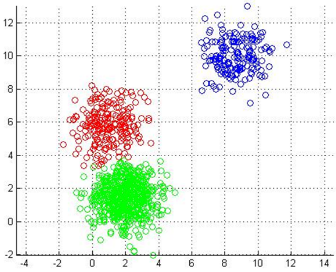
IBM Research, Dublin

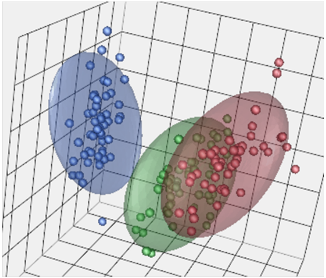
April 24, 2017

Overview

Clustering

- **Clustering**: Groups *similar* items in equivalence classes.
- Items could be images, videos, text documents, users etc.
- In general, consider each item as a *feature vector* in some feature space.





Document Clustering Usefulness

- In the era of **user driven content generation**, the number of documents in the web, including social media and community based forums, is increasing very rapidly.
- Efficient content management techniques typically involve clustering similar documents into groups.
- Clusters of documents has been used to:
  1. **Distributed indexing**, or **ranking clusters of documents** as a whole.
  2. Smooth language models for improving retrieval quality.
  3. Improve initial retrieval and relevance feedback quality.
  4. Perform document expansion to enrich the informative content of documents.
  5. Effective information presentation for exploratory browsing.

Document Clustering Fundamentals

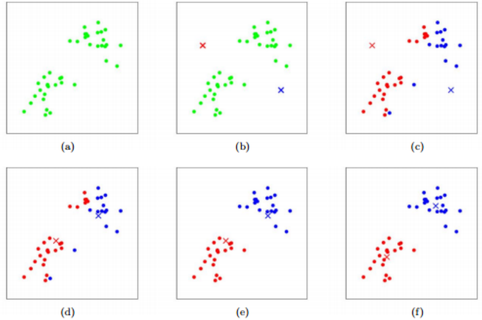
- **Term Vector space**: Each document is a point in a term vector space.
- Dimensionality of this space is the number of unique terms in a collection or the **vocabulary size**.
- Each component of this vector is the (weighted) term frequency.
- Example: Consider a three term word comprised of terms 'a', 'b' and 'c'.
- Vector for document $D_1 : (abaa)$ is (3, 1, 0), that of $D_2 : (ccb)$ is (0, 1, 2).
- A popular weighting scheme is weighting by the **idf** values to capture term importance.

Document Clustering

- For large collections, a document vector is extremely sparse.
- A convenient way to represent documents is with sparse vectors, e.g. the sparse vector representation of D_1 is $\{(a, 3), (b, 1)\}$.
- The most efficient way of storing sparse vectors is with the help of **inverted lists**.
- In an inverted list:
  - Each list head is a term.
  - Each list head points to a sorted list of document ids with the corresponding term weights. For our example:
    - a: $(D_1, 3)$
    - b: $(D_1, 1)$ $(D_2, 1)$
    - c: $(D_2, 2)$

K-means Review

- Starts with a set of **randomly chosen initial centres**.
- **E-step:** Each input point is repeatedly assigned to its nearest cluster centre.
- **M-step:** Cluster centres are then recomputed by making use of the current assignments.



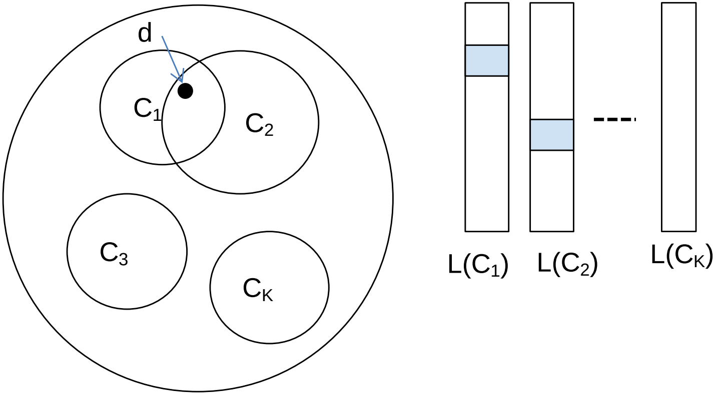
Curse of Dimensionality

- Advantage of sparse representation: Fast cluster assignments and re-computation of the cluster centres.
- Recomputation of the cluster centres results in an increase in the dimensionality of the centroid vectors **(producing denser vectors)** in subsequent iterations.
- Work with the *medoids* instead of the centroids, an algorithm called the K-medoids. This ensures that dimensionality remains fixed. **Why?**

Fast K-means

- The main bottleneck of the K-means is to assign each non-centroid document d ($d \in D - \bigcup_{k=1}^{K} C_k$) to a cluster.
- Can efficiently be achieved with the help of the inverted list data structure.
- The inverted list data structure is particularly suitable for efficiently computing the set $TOP(x)$ for a given document vector x.

Fast K-means



Key idea

- Select random cluster centres.
- Use each cluster centre document as a query to retrieve documents similar to it.
- Assign identical cluster ids to all these documents.
- Assign the longest document of every cluster group as its new cluster centre.

Fast K-means Algorithm

Algorithm 2: FPAC Heuristic Algorithm

Input: K, the number of clusters
Input: Collection of N documents D ($|D| = N$)
Input: M, the maximum number of iterations
Output: A partition of D, $\bigcup_{k=1}^{K} D_k = D$

begin
  for $j = 1, \ldots, M$ **do**
    `// Initialize the cluster centres`
    $L \leftarrow \Phi$
    for $k = 1, \ldots, K$ **do**
      Randomly initialize C_k from $D - L$
      $L \leftarrow L \bigcup TOP(C_k, \tau)$
    end
    `// Execute queries and merge ranked lists`
    for *each* $x \in \bigcup_{k=1}^{K} C_k$ **do**
      `// Assign` d `to its nearest cluster centre`
      $x_\tau \leftarrow$ **ExtractQuery**(x)
      Retrieve ranked list $L(x) \leftarrow TOP(x_\tau)$
      $L \leftarrow \bigcup L_x$
    end
    `// Assign documents to clusters based on retrieval scores`
    Sort L by normalized retrieval scores
    for *each* $d \in L$ **do**
      $D'_k = D'_k \cup d$, where the retrieval score of d is maximum in list $L(k')$
    end
    `// Consider the document with the most number of unique terms as the central one.`
    for $k = 1, \ldots, K$ **do**
      $C_k \leftarrow d \in D_k$ s.t. $|d| > |d'|, \forall d' \in D_k$
    end
  end
end

Dataset

	Train	Test
#Documents (N)	348,867	681,611
#Terms (V)	603,816	1,273,397
#Reference classes (web domains)	49	203
Avg. length of a document	404.48	525.35
Avg. #documents in a cluster (domain)	7119.55	3357.69
Max. #documents in a cluster	43,500	60,906
Min. #documents in a cluster	33	3

Table: Characteristics of the WMT '16 dataset.

Results

Clustering method	Initial centroid sel	Centroid recomp.	#iters	Train K	Train Purity	Train RI	Train Time (s)	Train Gain	Test K	Test Purity	Test RI	Test Time (s)	Test Gain
K-means	Random	Vec sum	50	49	0.5371	0.8895	161K	-	203	0.6602	0.9597	433,870	-
FPAC	Random	Vec sum	50	49	0.1388	0.9245	1780	90.57	203	0.0942	0.9670	4771	90.93
FPAC	Min. inter-sim	Vec sum	50	49	0.1471	0.9245	818	197.09	203	0.1396	0.9672	1579	274.77
FPAC	Min. inter-sim	Max_terms	50	49	0.1471	0.9245	**743**	**216.98**	203	0.1396	0.9673	**1531**	**283.83**

Table: Comparison of FPAC and its baseline variations with the K-means algorithm.