ISSN: 2321-7782 (Online)

Volume 1, Issue 5, October 2013

International Journal of Advance Research in Computer Science and Management Studies

Research Paper

Available online at: www.ijarcsms.com

# Search Results Clustering: Comparison of Lingo and K-Means

**Sri Shilpa Padmanabhuni**[1]
Computer Science & Engineering
Ramachandra College of Engineering
Eluru - India

**Hima Bindu T**[2]
Computer Science & Engineering
Ramachandra College of Engineering
Eluru - India

***Abstract: Clustering is an unsupervised learning technique and K-Means algorithm is a powerful clustering method. Search results clustering are proposed in literature to aid the novice users and Lingo algorithm is a popular method for search results clustering. This paper compares these two algorithms with an empirical analysis on standard datasets.***

***Keywords: Clustering, Search Results Clustering, Suffix Arrays, Information Retrieval, Rand Index***

## I. Introduction

When the intention behind the search query is not clear, the search engine returns a large number of results. The results are displayed in the form of a ranked list. The user need to swift through the long list of results to find the result that suits his (or) her information need. This process is a tedious job, hence search results clustering [26] is proposed to present the results in thematic groups. The aim of the search results clustering is to provide quick focus on relevant search results. The search results clustering organize the search results into different groups each group corresponding to different theme. For example, the query "engine" yields results that belong to search engine as well as car engine parts.

The ranked list technique is not efficient because documents are presented in the linear fashion, which makes the time complexity as linear search and if the query is too general, it is difficult for discovering interested topics of a user.  But there are some situations where the search engines will be used by novice people, (or) there may be search engines database, which is populated with large number of documents. Under such circumstances we need to provide low precision searches that help the users to browse this large number of documents in easy and quick manner. Clustering the results of web search engines can provide a powerful browsing tool [21]. In clustering of search results, the relationship between the results is obtained by using cluster hypothesis, which states that if there is a document from a cluster that is relevant to search request, then it is likely that other documents from the same cluster are also relevant, this is because clustering merges the documents that share many properties and the time complexity is decreased to logarithmic search.

In Search results clustering, the results are presented in groups of similar documents. This enables the user to easily scan the coherent groups rather than the scanning the individual results. By using search results clustering, the search time and effort is reduced drastically, the organization and presentation of the results is effective and allows users to find information they are looking for more easily.

Lingo [26] is a search results clustering algorithm, which identify the clusters with relevant labels. K-Means [27] is one of the popular clustering algorithms. This paper compares these two algorithms. The paper is organized as follows: section I contains Introduction. Literature survey is presented in section II. Section III contains the two algorithms compared, Lingo and K-Means. Section IV presents the experiment and results. The paper is concluded in section V.

## II. LITERATURE SURVEY

**A. *Text Mining Preliminaries:*** Text mining explores patterns in textual data by finding latent topics, topical trends, outliers and other hidden patterns[13]. There are many examples of text-based documents (all in'electronic' format…) like e-mails, corporate web pages, customer surveys, resumes, medical records, DNA sequences, technical papers, incident reports, news stories and so on. Users won't have patience (or) time to read entire collection of documents. So there is a way to gain knowledge in summarizied form from all the text, without reading (or) examining the complete document. This process is known as "Text Mining". Text mining is used in retrieval, assume that there is a large corpus of text documents, and user wants the one closest to a specified query. The text mining retrieval is done by finding *k* objects in the corpus of documents which are most similar to user query .Ex: web search, library catalogs. To find the similarities following techniques are used:

***I. Elimination of stop words:*** The usage of set of all words in collection may cause too much noise for retrieval. For example, word "the" might lead to retrieval of documents that are unrelated to user query. These words are called *stop words*. One way to reduce set of words to be used is to control the number of distinct words (index terms). The reason that stop-words should be removed from a text is that they make the text look heavier and less important for analysts and the stop-words are not necessary for the analysis. The disadvantage of stop-words is that the words which are too frequent among the documents in collection are not good discriminators. The elimination of stop-words reduces the size of indexing structuring. Some examples of stop-words are shown in Table I:

**TABLE I**
List of Stop-words

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | an | and | are | as | at | be | by | for | from |
| has | he | in | is | it | its | of | on | that | the |
| this | to | so | soon | sure | was | were | will | with | yet |

***II. Stemming:*** The goal of stemming is to reduce derivationally related forms of words of a word to a common base form by finding out the root (or) stem of a word. A stem is the portion of a word which is left after the removal of its affixes (i.e., prefixes and suffixes) [4]. Documents containing morphological variant words are reduced to a common word as shown in Table II:

**TABLE II**
Stemming code Example

| Words | Stems |
|---|---|
| user<br>users<br>used<br>using | **use** |
| engineering<br>engineered<br>engineer | **engineer** |

It is useful for improving retrieval performance. The effectiveness of searching, most obviously but not exclusively in terms of recall, would be expected to increase if it were possible to bring together the variants of a given word so that they could all be retrieved in response to a query that specified just a single variant. This not only means that different variants of a term can be conflated to a single representative form – it also reduces the dictionary size, that is, the number of distinct terms needed for representing a set of documents. A smaller dictionary size results in a saving of storage space and processing time. The suffix removal is done by using Porter's Algorithm: The process of removing the common morphological and inflexional endings from words [16]. The algorithm consists of five steps applying rules within each step. Within each step, if a suffix rule matched to a word, then the conditions attached to that rule are tested on what would be the resulting stem, if that suffix was removed, in the way defined by the rule. For example such a condition may be, the number of vowel characters, which are followed by a consonant character in the stem, must be greater than one for the rule to be applied. Once a rule passes its conditions and is accepted, the rule fires and the suffix are removed and control moves to the next step. If the rule is not accepted then the next rule in the step is tested, until either a rule from that step fires and control passes to the next step or there are no more rules in that step when control moves to the next step. This process continues for all five steps, the resultant stem being returned by the Stemmer after control has been passed from step five as shown in Fig.1.

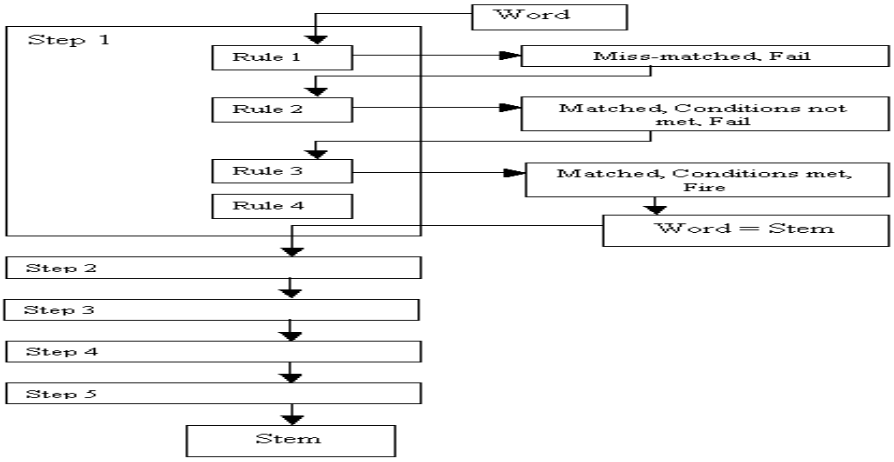

Fig 1: Stemming Process

Step 1: Gets rid of plurals and -ed or -ing suffix

Ex: a. possesses: possess

b. fated : fate

c. infuriating: infuriate

Step 2: Turns terminal y to i when there is another vowel in the stem

Step 3: Maps double suffixes to single ones:-ization, -ational, etc.

Ex: a. optional: option

b. realization: realize

Step 4: Deals with suffixes, -full, -ness etc.

Ex: a. playful: play

b. largeness: large

Step 5: Takes off -ant, -ence, etc

**III.** ***Frequent Phrase Extraction:*** The frequent phrases are list of phrases that capture the main topics present in a given document [7]. This enables the users to quickly determine whether the obtained query is related to their interest or not and search for documents that match a given query term in the keyword field yields a smaller, higher quality list of hints than a search for the same term in the full text of the documents. By extracting frequent phrases, users easily chooses which documents to read or users learn the relation among the documents in collection. The frequent phrases are constructed based on the suffix arrays.

Suffix Array *is* a simple data structure that enables lookup of any substring of a text and identification of repeated substrings. The data structure is simply an array containing all the pointers to the text suffixes sorted in lexicographical (alphabetical) order. Each suffix is a string starting at a certain position in the text and ending at the end of the text. Searching a text can be performed by binary search using the suffix array. The suffix array is constructed by calculating the Longest Common Prefix, (LCP) ranking.

*Illustrative Example:* Consider the text "java is a object oriented programming language" and n=7. The LCP's are obtained as shown in Table III:

**TABLE III**
LCP Calculation

| Index | word | lcp |
|---|---|---|
| 1 | a | 0 |
| 2 | is | 1 |
| 4 | java | 2 |
| 3 | language | 4 |
| 6 | object | 6 |
| 7 | oriented | 6 |
| 5 | programming | 8 |

Algorithm: LCP array construction

Input: Text T [0…n], suffix array SA [0...n], inverse suffix array $SA^{-1}$[0..n]

Output: LCP array LCP[1..n]

Method:

```
L ← 0
for i←0 to n-1 do
 k← SA^-1[i]  //i=SA[k]
 j←SA[k-1]
 while T[i+l]=T[j+l] do L←L+1
 LCP[k]←L
 if  L>0 then L←L-1
return LCP
```

**IV.** ***Term frequency and weighting:*** A document *d* is represented as vector of weights $[W_{t0},W_{t1},\ldots\ldots W_{tn}]$, where $t_0,t_1\ldots.t_n$ represents set of terms and $W_{ti}$ expresses the term $t_i$ in document *d*. This is termed as "term-weighting". The weight expresses importance of term in a document and also in the whole collection of documents. The weight of term is expressed as product of term frequency and inverse document frequency. The term frequency is defined as number of occurrences of a term *t* in a document *d* and represented as $tf_{t,d}$ . It is generally defined as 0, if the document does not contain the term and non-zero otherwise [20]. To normalize the term-frequency, inverse document frequency measure, this represents importance of a term t, is used. If a term *t* occurs in many documents, its importance will be scaled down and is calculated using

$$idf_t = log_{10}\frac{N}{df_t}, where\ df_t \text{ is document } d \text{ in collection that contain a term } t.$$

The *tf-idf* weight of a term is the product of its *tf* weight and its *idf* weight.

$$tf\text{-}idf_{t,d} = tf_{t,d} * idf_t$$

**V.** ***Term-document Matrix Construction:*** A popular algorithm for indexing is the *tf-idf* measure [22], which extracts keywords that appear frequently in a document. Term similarity is based on term-document relationship which is obtained as shown in Table IV.

Illustrative Example: In a (very small) database of cook books, the documents are titled as:
d1: Quick and Easy Dinners
d2: Vegetarian Cooking: Healthy Meals Made Easy
d3: Cooking Up a Storm: Preparing Meals for 100+

d4: 101 Healthy Crockpot Dinners
d5: Cook like Martha: 50 Recipes Guaranteed to Please

Each of these documents may be indexed by certain terms contained in the titles. The associated terms for this set of documents are:

t1: meal
t2: dinners
t3: recipe
t4: cooking
t5: healthy
t6: vegetarian

**TABLE IV**
Term-Document Matrix

| | D1 | D2 | D3 | D4 | D5 |
|---|---|---|---|---|---|
| **T1** | 0 | 1 | 1 | 0 | 0 |
| **T2** | 1 | 0 | 0 | 1 | 0 |
| **T3** | 0 | 0 | 0 | 0 | 1 |
| **T4** | 0 | 1 | 1 | 0 | 1 |
| **T5** | 0 | 1 | 0 | 1 | 0 |
| **T6** | 0 | 1 | 0 | 0 | 0 |

**VI. *Singular Value Decomposition:*** It is a method for transforming correlated variables into set of uncorrelated variables that better represent the relationship among the documents in the collection. It decomposes a rectangular matrix into three matrices:

a. U – an orthogonal matrix
b. ∑- a diagonal matrix
c. V- a transpose of orthogonal matrix

$$\mathbf{A=U\sum V^T}$$

The notation of term-document matrix is: M X N matrix in which each row represents terms and each column represents documents. The SVD decomposes this matrix as U, which represents orthogonal columns known as "Left singular values" of size M X M (term X term). The tf-idf weight matrix for Table IV is represented as:

$$A_{tfidf}=\begin{bmatrix} 0 & 0.50 & 0.70 & 0 & 0 \\ 1 & 0 & 0 & 0.70 & 0 \\ 0 & 0 & 0 & 0 & 0.70 \\ 0 & 0.50 & 0.70 & 0 & 0.70 \\ 0 & 0.50 & 0 & 0.70 & 0 \\ 0 & 0.50 & 0 & 0 & 0 \end{bmatrix}$$

After SVD decomposition, the obtained matrices are:

$$U=\begin{bmatrix} 0.479 & 0.189 & 0.398 & -0.474 & -0.134 & -0.577 \\ 0.283 & -0.860 & -0.260 & -0.311 & 0.132 & 0 \\ 0.213 & 0.138 & -0.664 & 0.395 & 0.074 & -0.577 \\ 0.692 & 0.327 & -0.266 & -0.079 & -0.060 & 0.577 \\ 0.353 & -0.313 & 0.414 & 0.684 & -0.371 & 0 \\ 0.203 & 0.035 & 0.303 & 0.218 & 0.94 & 0 \end{bmatrix}$$

$$\sum=\begin{bmatrix} 1.457 & 0 & 0 & 0 & 0 \\ 0 & 1.297 & 0 & 0 & 0 \\ 0 & 0 & 0.837 & 0 & 0 \\ 0 & 0 & 0 & 0.632 & 0 \\ 0 & 0 & 0 & 0 & 0.36 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

**VII. *Latent Semantic Indexing:*** Identifies the patterns in the relationships between terms and documents in the collection using a mathematical technique called SVD [24]. The aim of this technique is to decompose the original term-document matrix of the vector space model and to retain k largest singular values from the singular value matrix. The advantage of LSI technique is its ability to establish associations between the terms that occur in similar documents.

The complete collection of term-document matrix in which each entry represents weight corresponding to specific term and a specific document is then decomposed using SVD and small singular values are eliminated from the singular value matrix. The resulting matrix preserves semantic relationship from term-document matrix. LSI begins by constructing a term-document matrix, A, to identify the occurrences of the m unique terms within a collection of n documents. In a term-document matrix, each term is represented by a row, and each document is represented by a column, with each matrix cell, aij, initially representing the number of times the associated term appears in the indicated document, tf-idf. This matrix is usually very large and very sparse.

**VIII.** ***Vector Space Model:*** An algebraic model for representing text documents as vectors of identifiers. The notation of document vector helps in capturing the relative importance of the terms in a document. The representation of a set of documents as vectors in a common vector space is known as "vector space model" [14]. Documents and queries are represented as vectors. For any query, the query vector is compared to every one of the document vectors. The results can be ranked. This decides which document comes closest to the query, and ranks the others as to the closeness of the fit. Vector-space models, by placing terms, documents, and queries in a term-document space and computing similarities between the queries and the terms or documents, allow the results of a query to be ranked according to the similarity measure used. The distance between the two vectors is an indication of the similarity of the two documents. The cosine of the angle between the two vectors is the most common distance measure. Cosine measures similarities between two documents in the collection.

Method of Computation:

1. Get a query from the user.
2. Convert it to tf-idf scores.
3. Create a data structure that is indexed by documents, which will accumulate scores for the documents.
4. For each term in the query, get the posting list for the term and for each document that has that term we are going to update the entry in Scores

   *Scores[d] += tf-idf(term,query) * tf-idf(term, document)*
5. Finally, the scores must be normalized so we can compare them against each other. For this, create a new data-structure like Scores called "*Magnitude*".
6. For each term in the entire posting list and for each document represented in Scores

   *Magnitude [document] += tf-idf (term, document) ^2*
7. For each document in Scores, the cosine similarity is calculated using

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

Where,

$q_i$ is the tf-idf weight of term $i$ in the query.

$d_i$ is the tf-idf weight of term $i$ in the document.

Illustrative Example: Assume following two documents

1. Xeon goes to marry Xeonian girl, a girl.
2. Leon goes to forest to find Xeon.

From the above sentences, calculate the terms and their respective frequencies as shown in Table V.

**TABLE V**
Frequency Calculation

| | Frequency in document1 | Frequency in document2 |
|---|---|---|
| **xenon** | 1 | 1 |
| **goes** | 1 | 1 |
| **to** | 1 | 2 |
| **marry** | 1 | 0 |
| **xenonian** | 1 | 0 |
| **girl** | 2 | 0 |
| **lenon** | 0 | 1 |
| **forest** | 0 | 1 |
| **find** | 0 | 1 |

VEC1 = [1,1,1,1,1,2,0,0,0] AND VEC2 = [1,1,2,0,0,0,1,1,1]

$$\mathbf{COS\Theta}=\frac{(1*1)+(1*1)+(1*2)+(1*0)+(1*0)+(2*0)+(0*1)+(0*1)+(0*1)}{\sqrt{((1*1)+(1*1)+(1*1)+(1*1)+(1*1)+(2*2)}\ \sqrt{(1*1)+(1*1)+(2*2)+(1*1)+(1*1)+(1*1)}}$$

***B.*** **Clustering***:* A process of collecting data objects that are similar to one another with in same group and that are dissimilar to the objects in other groups [12]. In the information retrieval (IR) field, cluster analysis has been used to create groups of documents. The terms in a document collection can also be clustered to show their relationships. Classes (or) conceptually meaningful group of objects that share common characteristics play an important role in analysis [6].

***I.*** ***Clustering Techniques:*** Many algorithms exist in the literature. A partitioning algorithm organizes the objects into k partitions, where each partition represents a cluster [1]. A hierarchical method works by grouping data objects into tree of clusters. Hierarchical method is classified as agglomerative and divisive methods. The quality of a hierarchical method suffers from its inability to perform adjustment once a merge (or) a split decision has been executed. To discover clusters with arbitrary shape, density-based clustering methods are proposed. These methods regard clusters as dense region of objects in the data space that are separated by regions of low density. DBSCAN grows clusters according to a density-based connectivity analysis. DENCLUE clusters objects based on a set of density distribution functions.

***II.*** ***Measures:*** To access the ability of our clustering algorithms to diversify the top results returned by a search engine, calculate [22].

*Recall (R):* The percentage of documents that are relevant to the query and were, in fact, retrieved.

$$recall = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Relevant\}|}$$

*Precision (P): The* percentage of retrieved documents that are in fact relevant to the query (i.e., “correct” responses).

$$precision = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Retrieved\}|}$$

***C.*** ***Search Results Clustering*****:** Document clustering was proposed mainly as a method of improving the effectiveness of document ranking following the hypothesis that closely associated documents will match the same requests. There exist various categories of clustering algorithms, as follows: The scatter/Gather is an example for data-centric algorithm. The system works by performing an initial clustering of a collection of documents into a set of k clusters (a process called scattering). This provides an initial overview of the collection of documents. Then, at query time, the user selected clusters of interest and the system re-clusters the indicated sub collection of documents dynamically (the gathering step).The query re-cluster cycle is obviously very similar to search results clustering systems, with the difference that instead of querying a back-end search engine, Scatter/Gather slices its own collection of documents. Description- aware algorithms carefully selects the features so that they are immediately recognizable to the user as something meaningful. If the features are meaningful and precise then they can be used to describe the output clusters accurately and sufficiently. The algorithm that first implemented this idea was Suffix Tree Clustering (STC). Description-centric algorithm designed specifically for clustering search results. The quality of cluster titles is crucial to the usability of any clustering algorithm. The commercial search engine Vivisimo is the example for Description-centric algorithm [26].

***I.*** ***Clustering Quality:*** The Rand index measures the percentage of decisions that are accurate [14]. Two documents can be assigned to the same cluster, if they are similar. A true positive (TP) decision assigns two similar documents same cluster. A true negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors that are committed. A (FP) decision assigns two dissimilar documents to same cluster. A (FN) decision assigns two similar documents to different clusters.

$$\mathrm{RI}=\frac{(\mathrm{TP+TN})}{(\mathrm{TP+TN+FP+FN})}$$

## III. LINGO AND K-MEANS ALGORITHM

The algorithms compared in this paper are described in this section with illustration

***A.*** ***K-means:*** A database ***D*** of ***n*** objects is partitioned into ***k*** clusters, such that the sum of squared distances is minimized

$$\mathrm{E}=\sum_{i=1}^{k}\sum_{\mathrm{p}\in\mathrm{C_i}}(\mathrm{d}(\mathrm{p},\mathrm{C_i}))^2,\text{ (where } \mathrm{c_i} \text{ is the centroid or medoid of cluster } \mathrm{C_i})$$

The clustering method is initiated by selecting k seeds randomly, where k represents number of clusters (k is a user specified parameter). The remaining objects are assigned to the clusters to which they are most similar, depending on the distance between the objects and means of the clusters. The centroid of each cluster is updated based on the points assigned to the cluster. This approach moves objects between clusters until the sum of squared distances cannot be decreased further.

Algorithm: k-means: The k-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

1. k:the number of clusters
2. D: a dataset containing n objects

Output: A set of k clusters.

Method:

1. Arbitrarily choose k objects from D as the initial cluster centers;
2. Repeat
3. Assign each object to the cluster to which the object is most similar, based on the mean value of the objects in the cluster;
4. Update the cluster means, i.e., calculate the mean value of the objects for each cluster;
5. Until no change;

*Illustrative Example:* Randomly we choose following two centroids (k=2) for two clusters. Table VI presents the dataset used for illustrating k-means clustering.

**TABLE VI**

Dataset for K-Means Clustering

| Subject | A | B |
|---|---|---|
| 1 | 1.0 | 1.0 |
| 2 | 1.5 | 2.0 |
| 3 | 3.0 | 4.0 |
| 4 | 5.0 | 7.0 |
| 5 | 3.5 | 5.0 |
| 6 | 4.5 | 5.0 |
| 7 | 3.5 | 4.5 |

As a first step in finding initial sensible partitioning, the subjects which are farthest apart (by using the Euclidean distance) are considered. Subject 1 and subject 2 are the two centroids based on this observation. Table VII presents the mean vector centroids initially.

**TABLE VII**

Mean Calculation (1st Iteration)

| | Subject | Mean Vector (centroid) |
|---|---|---|
| **Cluster 1** | 1 | (1.0, 1.0) |
| **Cluster 2** | 4 | (5.0, 7.0) |

The mean vector is recalculated each time a new member is added. This leads to the following iteration of steps as in Table VIII:

**TABLE VIII**

Mean Calculation and addition of each member to a cluster

| Subject | Centroid1 | Centroid2 |
|---|---|---|
| 1 | 0 | 7.21 |
| 2 | 1.12 | 6.10 |
| 3 | 3.61 | 3.61 |
| 4 | 7.21 | 0 |
| 5 | 4.72 | 2.5 |
| 6 | 5.31 | 2.06 |
| 7 | 4.30 | 2..92 |

Thus, the obtained two clusters are:{1,2,3} and {4,5,6,7}. Table IX presents the mean vector centroids. The mean vectors are calculated as:

$$m1= (\tfrac{1}{3}(1.0+1.5+3.0),\tfrac{1}{3}(1.0+2.0+4.0)) =(1.8,2.3)$$

$$m2= (\tfrac{1}{4}(5.0+3.5+4.5+3.5),\tfrac{1}{4}(7.0+5.0+5.0+4.5)) =(4.1,5.4)$$

**TABLE IX**

Mean of Each Cluster (2nd Iteration)

| | Subject | Mean Vector (centroid) |
|---|---|---|
| Cluster 1 | 1, 2, 3 | (1.8, 2.3) |
| Cluster 2 | 4, 5, 6, 7 | (4.1, 5.4) |

Now using these centroids compute the Euclidean distance of each object, as shown in Table X.

**TABLE X**

Calculation of distance to Mean

| Subject | Distance to mean (centroid) of Cluster 1 | Distance to mean (centroid) of Cluster 2 |
|---|---|---|
| 1 | 1.5 | 5.4 |
| 2 | 0.4 | 4.3 |
| 3 | 2.1 | 1.8 |
| 4 | 5.7 | 1.8 |

| | | |
|---|---|---|
| 5 | 3.2 | 0.7 |
| 6 | 3.8 | 0.6 |
| 7 | 2.8 | 1.1 |

Only individual 3 are nearer to the mean of the opposite cluster (Cluster 2) than its own (Cluster 1). In other words, each individual's distance to its own cluster mean should be smaller that the distance to the other cluster's mean (which is not the case with individual 3). Thus, individual 3 is relocated to Cluster 2 resulting in the new partition is obtained as shown in Table XI. Therefore, the new clusters are: {1,2} and {3,4,5,6,7}.

**TABLE XI**
Clusters Formed by K-Means

| | Subject | Mean Vector (centroid) |
|---|---|---|
| **Cluster 1** | 1, 2 | (1.3, 1.5) |
| **Cluster 2** | 3, 4, 5, 6, 7 | (3.9, 5.1) |

The iterative relocation would now continue from this new partition until no more relocation occurs. However, in this example each individual is now nearer its own cluster mean than that of the other cluster and the iteration stops, choosing the latest partitioning as the final cluster solution as shown in Fig.2

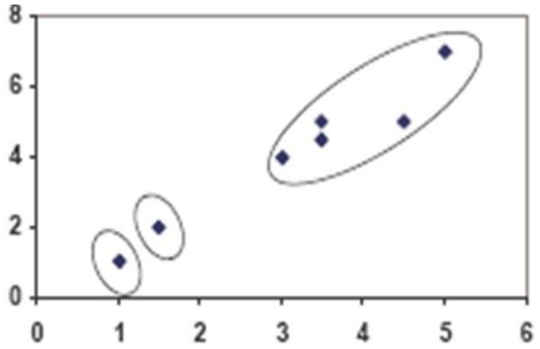

Fig 2: K-Means Clustering

**B.** ***Lingo:*** The key characteristic of the Lingo algorithm is that it first identifies cluster labels and then assigns documents to the labels to form final clusters [3]. To find the labels, Lingo builds a term-document matrix for all input documents and decomposes the matrix to obtain a number of base vectors. Each such vector gives rise to one cluster label. To complete the clustering process, each label is assigned documents that contain the label's words.

Lingo identifies meaningful cluster labels using the Singular Value Decomposition (SVD), and then assigns documents to these labels to form proper clusters [25]. For this reason this algorithm could be considered as an example of a description-comes-first approach. The algorithm consists of five phases.

1. Preprocessing of the input snippets, which includes tokenization, stemming and stop-word marking.
2. It identifies words and sequences of words frequently appearing in the input snippets.
3. A matrix factorization is used to induce cluster labels.
4. Snippets are assigned to each of these labels to form proper clusters. The assignment is based on the Vector Space Model (VSM) and the cosine similarity between vectors representing the label and the snippets.
5. Post processing, which includes cluster merging and pruning?

The key component in label induction is an approximate matrix factorization, which is used to produce a low-dimensional basis for the column space of the term-document matrix. In linear algebra, base vectors of a linear space can be linearly combined to create any other vector belonging that space Therefore, in Lingo, each vector of the low-dimensional basis gives rise to one cluster label.

The frequent word sequences or even single words appearing in the input snippets can also be expressed as vectors in the same vector space. Thus, the well-known measures of similarity between vectors, such as the cosine similarity, can be used to determine which frequent word sequence or single word best approximates the dominant verbal meaning of a base vector.

Algorithm: Lingo

```
Input: A set of documents D
Output: A set of k clusters.
Method:
D ← input documents (or snippets)
{STEP 1: Preprocessing}
for all d ∈ D do
   perform text segmentation of d; {Detect word boundaries etc.}
   if language of d recognized then
        apply stemming and mark stop-words in d;
   end if
end for

{STEP 2: Frequent Phrase Extraction}
concatenate all documents;
Pc ← discover complete phrases;
Pf ← p : {p ∈ Pc ∧ frequency(p) > Term Frequency Threshold};
{STEP 3: Cluster Label Induction}
A ← term-document matrix of terms not marked as stop-words and
    with frequency higher than the Term Frequency Threshold;
Σ,U, V ← SVD(A); {Product of SVD decomposition of A}
k ← 0; {Start with zero clusters}
n ← rank(A);
repeat
     k ← k + 1;
     $q \leftarrow (\Sigma^{k}_{i=1} \sum_{ii})/(\sum^{n}_{i=1} \Sigma_{ii})$;
until q < Candidate Label Threshold;
P ← phrase matrix for Pf ;
for all columns of $U^{T}_{k}$ P do
  find the largest component mi in the column;
      add the corresponding phrase to the Cluster Label Candidates set;
      labelScore ← $m_i$ ;
end for
calculate cosine similarities between all pairs of candidate labels;
identify groups of labels that exceed the Label Similarity Threshold;
for all groups of similar labels do
          select one label with the highest score;
end for
{STEP 4: Cluster Content Discovery}
for all L ∈ Cluster Label Candidates do
      create cluster C described with L;
      add to C all documents whose similarity
       to C exceeds the Snippet Assignment Theshold;
end for
put all unassigned documents in the “Others” group;
{STEP 5: Final Cluster Formation}
for all clusters do
       clusterScore ← labelScore × C;
end for
```

## IV. EXPERIMENTS AND RESULTS

The standard datasets for Search Results Clustering, AMBIENT [2], MORESQUE [19] are taken for comparing Lingo and k-means algorithm.

I. ***Ambient Dataset (AMBIguous ENTries):*** It is a dataset which contains 44 ambiguous queries for evaluating subtopic information retrieval. The topics were selected from the list of ambiguous Wikipedia entries. For ex: Given the Bronx query, its disambiguation page on Wikipedia provides results as Bronx zoo, real estate, school, service e.t.c., The ambient dataset is available at [2].

II. ***Moresque Dataset (MORE Sense-tagged QUEry):*** It is designed as a complement of AMBIENT dataset. The dataset consists of 114 topics, each with a set of subtopics and a list of 100 top-ranking documents. , the AMBIENT dataset is composed mostly of single-word queries. MORESQUE provides dozens of queries of length 2, 3 and 4, together with the 100 top results from Yahoo! for each query annotated as in the AMBIENT dataset(overall, we tagged 11,400 snippets).

The average values of the measures Recall, Precision and Rand Index on AMBIENT dataset are given in Table XIII.

**TABLE XIII**
Results on Ambient Dataset

| | K- Means | Lingo |
|---|---|---|
| **Recall** | 79.41 | 85.73 |
| **Precision** | 70.27 | 75.27 |
| **Rand Index** | 34.69 | 39.78 |

The same measure values on MORESQUE dataset are reported in Table XIV.

**TABLE XIV**
Results on Moresque Dataset

| | K-Means | Lingo |
|---|---|---|
| **Recall** | 80.95 | 87.64 |
| **Precision** | 73.45 | 79.62 |
| **Rand Index** | 37.85 | 39.89 |

The Rand Index values for each query of the AMBIENT dataset are reported in Table XV

**TABLE XV**
Rand Index Calculation on Ambient Dataset

| S.NO | Topics | K-Means | Lingo |
|---|---|---|---|
| 1 | Aida | 35.3 | 38.21 |
| 2 | B-52 | 37.75 | 38.43 |
| 3 | Beagle | 36.55 | 38.30 |
| 4 | Bronx | 30.4 | 32.4 |
| 5 | Cain | 35.03 | 37.24 |
| 6 | Camel | 24.15 | 34.04 |
| 7 | Coral Sea | 30.9 | 39.85 |
| 8 | Cube | 31.28 | 41.17 |
| 9 | Eos | 30.91 | 40.07 |
| 10 | Excalibur | 34.96 | 38.73 |
| 11 | Fahrenheit | 29.75 | 35.5 |
| 12 | Globe | 33.78 | 35.72 |
| 13 | Hornet | 35.75 | 39.84 |
| 14 | Indigo | 32.8 | 37.75 |
| 15 | Iwo Jima | 40.05 | 39.97 |
| 16 | Jaguar | 30.6 | 31.5 |
| 17 | La Plata | 32.78 | 42.27 |
| 18 | Labyrinth | 30.35 | 31.70 |
| 19 | Landau | 34.68 | 37.5 |
| 20 | Life on Mars | 34.86 | 34.82 |
| 21 | Locust | 30.64 | 32.16 |
| 22 | Magic Mountain | 35.49 | 38.15 |

| | | | |
|---|---|---|---|
| 23 | Matador | 37.03 | 36.95 |
| 24 | Metamorphosis | 25.96 | 35.89 |
| 25 | Minotaur | 33.75 | 42.69 |
| 26 | Mira | 27.03 | 36.97 |
| 27 | Mirage | 28.05 | 35.53 |
| 28 | Monte Carlo | 25.56 | 35.45 |
| 29 | Oppenheim | 29.28 | 38.15 |
| 30 | Out of Control | 32.53 | 41.49 |
| 31 | Pelican | 32.43 | 39.43 |
| 32 | Purple Haze | 33.51 | 34.60 |
| 33 | Raam | 34.12 | 36.43 |
| 34 | Rhea | 32.16 | 33.07 |
| 35 | Scorpion | 37.28 | 39.42 |
| 36 | The Little Mermaid | 39.08 | 41.66 |
| 37 | Tortuga | 38.95 | 40.01 |
| 38 | Urania | 37.9 | 39.86 |
| 39 | Wink | 38.72 | 40.31 |
| 40 | Xanadu | 29.08 | 32.17 |
| 41 | Zebra | 33.11 | 34.77 |
| 42 | Zenith | 39.49 | 40.30 |
| 43 | Zodiac | 39.05 | 39.37 |
| 44 | Zombie | 36.81 | 35.68 |

Due to space limitations, the individual query values of MORESQUE dataset are not reported in this paper (114 queries).

***III.*** ***Observation:*** The results presented reveals that Lingo algorithm is out performing for all the measures in both the datasets. This is perhaps due to the ability of Lingo in identifying the latent relationships between the documents (search results). K-means is a powerful technique but it is a general approach that suits any data. Hence k-means did not capture the latent relationships between the documents.

## V. CONCLUSION

This paper compares Lingo algorithm and k-means algorithm using standard Search Results Clustering datasets. It is observed that Lingo outperformed on both the datasets due to its capability to identify the hidden relationships between the search results.

## References

1. A. K. Jain and R. C. Dubes. Algorithms for Clustering Data. Printice Hall, 1988.
2. C. Carpineto , G. Roman . “Ambient dataset”, http://credo.fub.it/ambient/, (2008).
3. “Carrot2 Clustering Engine”, http://search.carrot2.org/stable/search.
4. D.A. Hull, Stemming algorithms: A case study for detailed evaluation. Journal of the American Society for Information Science, , 1996, pp:70-84
5. D. Cai, X. He, J.-R.Wen, andW.-Y.Ma. Block-level link analysis. In Proc. Int. 2004 ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR’04), Sheffield, UK, July 2004 pp: 440–447.
6. D. Christopher , Manning, Prabhakar Raghavan, Hinrich Schutze, Introduction to Information Retrieval, Cambridge Press, (2008).
7. F.Beil, M.Ester.: "Frequent Term-Based Text Clustering", KDD'02. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, 2002, pp: 436-442
8. G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. COMPUTER, 1999.
9. G. Salton and M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.
10. Hull, David. Using statistical testing in the evaluation of retrieval performance.In Proc.1993. SIGIR, pp. 329–338.
11. J. A. Hartigan. Clustering Algorithms. JohnWiley & Sons, 1975.

12. Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques, Second Edition, , Morgan Kaufmann.
13. J.J. Rocchio. Document retrieval systems - optimization and evaluation.Ph.D. Thesis, Harvard University, 1966.
14. J. Stefanowski. and D.Weiss. Carrot2 and language properties in Web search results clustering. In Proceedings of the 1st International Atlantic Web Intelligence Conference. Lecture Notes in Computer Science, vol. 2663. Springer, 2003, pp: 240–249.
15. L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. JohnWiley & Sons, 1990.
16. M.F.Porter. An algorithm for suffix stripping. In Readings in Information Retrieval, K. S. Jones and P. Willett, Eds. Morgan Kaufmann, 1997, pp: 313–316.
17. M. Hearst, “Untangling Text Data Mining,” in the Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, 1999
18. M.Mehta, R. Agrawal and J. Rissanen. SLIQ: A fast scalable classifier for datamining. InProc. 1996 Int. Conf. Extending Database Technology (EDBT’96), Avignon,France, Mar. 1996, pp: 18–32.
19. “Moresque dataset”, http://lcl.uniroma1.it/moresque/
20. N. Belkin and B. Croft. Information filtering and information retrieval: Two sides of the same coin? Comm. ACM, 1992 , pp:29–38.
21. Oren E. Zamir. Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results. Doctoral Dissertation, University of Washington, 1999
22. O. Zamir, O.Etzioni. Grouper: A Dynamic Clustering Interface to Web Search Results. In Proceedings of the Eighth International World Wide Web Conference (WWW8), Toronto, Canada, May 1999.
23. Salton, Gerard. Cluster search strategies and the optimization of retrieval effectiveness. In The SMART Retrieval System – Experiments in Automatic Document Processing Salton (1971), pp. 223–242. 351, 372, 530
24. S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. J. American Society for Information Science, 1990, pp: 391–407.
25. S. P. Lloyd. Least SquaresQuantization in PCM. IEEE Trans. Information Theory, 1982, (original version: Technical Report, Bell Labs, 1957.) pp: 128–137.
26. S. Osinski, D.Weiss, : A concept-driven algorithm for clustering search results. IEEE Intelligent Systems 20(3), 2005, pp: 48–54
27. T. Dasu and T. Johnson. Exploratory Data Mining and Data Cleaning. John Wiley & Sons, 2003
28. Y.S.Maarek, R.Fagin, I.Z.Ben-Shaul, D.Pelleg, Ephemeral document clustering for web applications, Technical Report RJ 10186, IBM Research, 2000.
29. Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Mining and Knowledge Discovery, 1998 pp: 283–304

## AUTHOR(S) PROFILE

**Sri Silpa Padmanabhuni,** received the B.Tech. degree in Computer Science & Engineering from Sir CRR college of Engineering in 2009 . She is pursuing her M.Tech from Ramachandra College of Engineering.

**Hima Bindu T,** received the B.Tech. and M.Tech. degrees in Computer Science & Engineering from NIT, Warangal and Jawaharlal Nehru Technology University, Hyderabad respectively. She is pursuing her PhD from university of Hyderabad. Her areas of interest are Information Retrieval and Knowledge Discovery.