International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

Clustering based Forgy's Algorithm using Java

Y. Ratna Rao & Dr. D.B.K. Kamesh
1Schlor, Shri Venkateshwara University, 2Professor, St. Martin's Engineering College, Secunderabad

Abstract:

As the number of available Web pages grows, it becomes more difficult for users finding documents relevant to their interests. Clustering is the classification of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often proximity per some defined distance measure. It can enable users to find the relevant documents more easily and help users to form an understanding of the different facets of the query that have been provided for web search engine.

Keywords: Cluster, web search, K-means, Forgy's algorithm

Introduction

One of the simplest clustering algorithms is Forgy's algorithm. K-means algorithm is like Fogy's algorithm. Clustering based on k-means is closely related to several other clustering and location problems. These include the Euclidean k-medians, in which the objective is to minimize the sum of distances to the nearest center, and the geometric k-center problem in which the objective is to minimize the maximum distance from every point to its closest center. In is presented an asymptotically efficient approximation for the k-means clustering problem, but the large constant factors suggest that it is not a good candidate for practical implementation. The fast greedy k-means algorithm overcomes the major shortcomings of the k-means algorithm discussed above, possible convergence to local minima and large time complexity with respect to the number of points in the dataset. One group of experiments aimed to assess the ability of the implementation to bring together topically related documents. Implementation included a procedure of term selection for document representation which preceded the clustering process and a procedure involving cluster representation for users viewing following the clustering process. After some tuning of the implementation parameters for the databases used, several different types of experiments were designed and conducted to assess whether clusters could group documents in useful ways. Document clustering analysis plays an important role in document mining research. A widely adopted definition of optimal clustering is a partitioning that minimizes distances within a cluster and maximizes distances between clusters. In this approach the clusters and, to a limited degree, relationships between clusters are derived automatically from the documents to be clustered, and the documents are subsequently assigned to those clusters.

Related Work:

Linear time clustering algorithms are the best candidates to comply with the speed requirement of on-line clustering. These include the K-Means algorithm [5], and the Single-Pass method [3], which has the advantage of being incremental and as such is popular in the event detection domain

[6] Buckshot and Fractionation [2]. In contrast to STC, all the mentioned algorithms treat a document as a set of words and not as an ordered sequence of words, thus losing valuable information. Phrases have long been used to supplement word-based indexing in IR systems [1]. Phrases generated by simple statistical approaches have also been successfully used [4]. Yet these methods have not been widely applied to document clustering. The only example known to the authors is the use of the co-appearance of pairs of words as the attributes of the documents' vector representations [7]. On the Web, there are some attempts to handle the large number of documents returned by search engines. Many search engines provide query refinement features. AltaVista, for example, suggests words to be added or to be excluded from the query. These words are organized into groups, but these groups do not represent clusters of documents. The Northern Light search engine (www.nlsearch.com), provides "Custom Search Folders", in which the retrieved documents are organized. Each folder is labeled by a single word or a two-word phrase, and is comprised of all the documents containing the label. Northern Light does not reveal the method used to create these folders nor its cost.

The Lucene Search Engine

The Lucene search engine is an open source, Jakarta project used to build and search indexes. Lucene can index any text-based information you like and then find it later based on various search criteria. Although Lucene only works with text, there are other add-ons to Lucene that allow you to index Word documents, PDF files, XML, or HTML pages. Lucene has a very flexible and powerful search capability that uses fuzzy logic to locate indexed items. Lucene is not overly complex. It provides a basic framework that you can use to build full-featured search into your web sites.

The easiest way to learn Lucene is to look at an example of using it. Let's pretend that we are writing an application for our university's Physics department. The professors have been writing articles and storing them online and we would like to make the articles searchable. (To make the example simple, we will assume that the articles are stored in text format.) Although we could use google, we would like to make the articles searchable by various criteria such as who wrote the article, what branch of physics the article deals with, etc. Google could index the articles but we wouldn't be able to show results based on questions such as, "show me all the articles by Professor Henry that deal with relativity and have superstring in their title."

Let's look at the key classes that we will use to build a search engine.

- **Document** - The *Document* class represents a document in Lucene. We index *Document* objects and get *Document* objects back when we do a search.
- **Field** - The *Field* class represents a section of a *Document*. The *Field* object will contain a name for the section and the actual data.
- **Analyzer** - The *Analyzer* class is an abstract class that used to provide an interface that will take a *Document* and turn it into tokens that can be indexed. There are several useful implementations

of this class but the most commonly used is the *StandardAnalyzer* class.

- **IndexWriter** - The *IndexWriter* class is used to create and maintain indexes.
- **IndexSearcher** - The *IndexSearcher* class is used to search through an index.
- **QueryParser** - The *QueryParser* class is used to build a parser that can search through an index.
- **Query** - The *Query* class is an abstract class that contains the search criteria created by the *Query Parser*.
- **Hits** - The *Hits* class contains the *Document* objects that are returned by running the *Query* object against the index.

Existing Algorithm

Implementation

Apache Tomcat (or Jakarta Tomcat or simply Tomcat) is an open sourceservlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTPweb server environment for Java code to run. Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files. In many production environments, it is very useful to have the capability to deploy a new web application, or undeploy an existing one, without having to shut down and restart the entire container. In addition, you can request an existing

Results

application to reload itself, even if you have not declared it to be reloadable in the Tomcat 5 server configuration file.

To support these capabilities, Tomcat 5 includes a web application (installed by default on context path /manager) that supports the following functions:

Deploy a new web application from the uploaded contents of a WAR file.

Deploy a new web application, on a specified context path, from the server file system.

List the currently deployed web applications, as well as the sessions that are currently active for those web apps.

Reload an existing web application, to reflect changes in the contents of /WEB-INF/classes or /WEB-INF/lib.

List the OS and JVM property values.

List the available global JNDI resources, for use in deployment tools that are preparing <ResourceLink> elements nested in a <Context> deployment description.

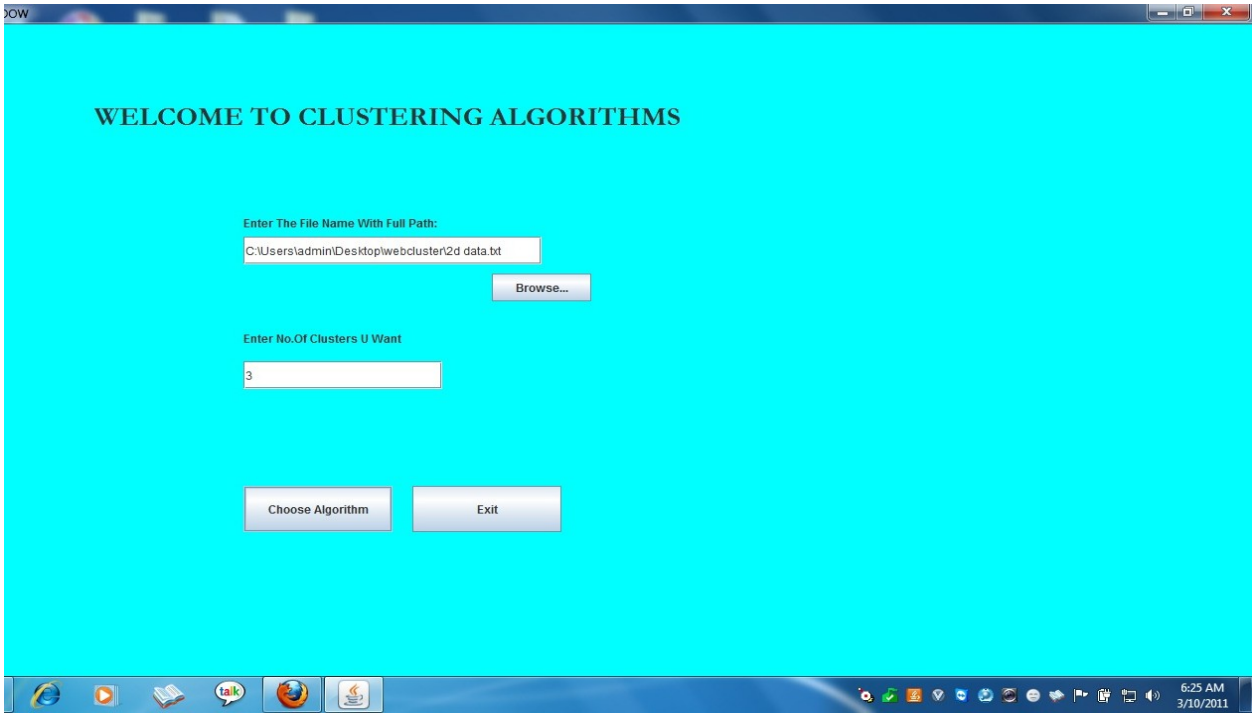
List the available security roles defined in the user database.

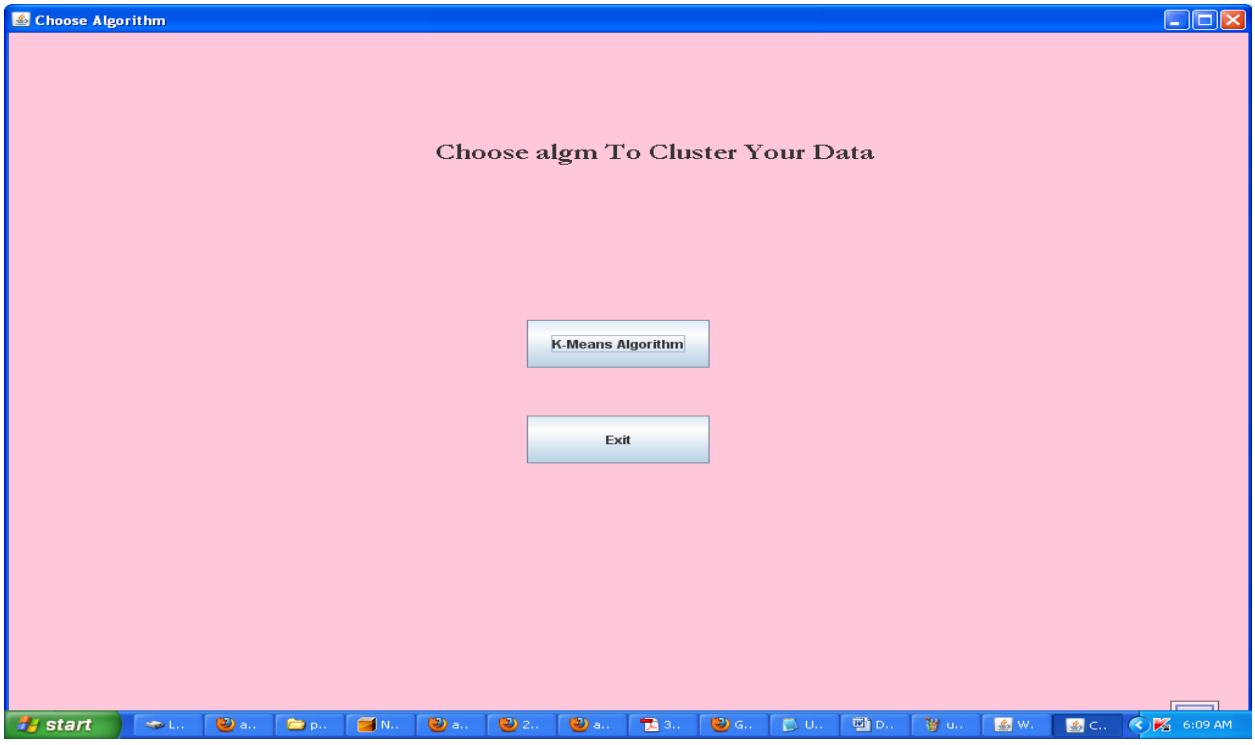
Start a stopped application (thus making it available again).

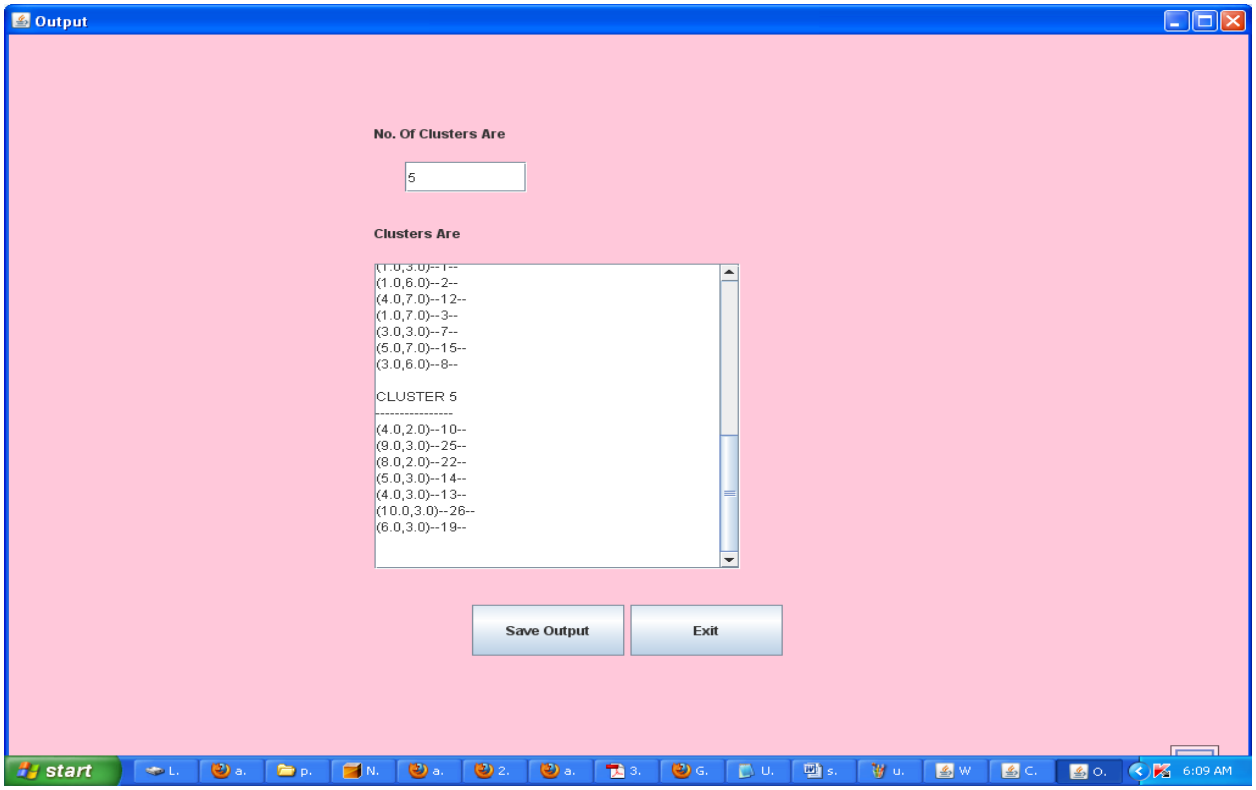
Stop an existing application (so that it becomes unavailable), but do not undeploy it.

Undeploy a deployed web application and delete its document base directory (unless it was deployed from file system).

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018







References:

[1] C. Buckley, G. Salton, J. Allen and A. Singhal. Automatic query expansion using SMART: TREC-3. In: D. K. Harman (ed.), The Third Text Retrieval Conference (TREC-3). U.S. Department of Commerce, 1995.

[2] D. R. Cutting, D. R. Karger, J. 0. Pedersen and J. W. Tukey. Scatter/Gather: a cluster-based approach to browsing large document collections. In Proceedings of the 15th International ACM SIGIR Conference on Research and Development in information Retrieval, pages 318-29, 1992.

[3] D. R. Hill. A vector clustering technique. In Samuelson (ed.), Mechanised Information Storage, Retrieval and Dissemination, North-Holland, Amsterdam, 1968.

[4] L. Fagan. Experiments in automatic phrase indexing for document retrieval: a comparison of syntactic and non- syntactic methods. Ph.D. Thesis, Cornell University, 1987.

[5] J. J. Rocchio, Document retrieval systems - optimization and evaluation. Ph.D. Thesis, Harvard University, 1966

[6] Proceedings of the TDT Workshop, University of Maryland, College Park, MD, October 1997.

[7] Y. S. Maarek and A. J. Wecker. The Librarian's Assistant: automatically organizing on-line books into dynamic bookshelves. In Proceedings of RIAO '94, 1994.