INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, KALYANI



*Data Science Project*

DEPARTMENT: I.T

Presented by: SHUBHAM KUMAR SINGH RAJPUT (48)

ATUL SAHAY (04)

MADHU DEVI (27)

May 13, 2017

CONFERENCE PROCEEDINGS CLUSTERING

**PREFACE**

This project

**“CONFERENCE PROCEEDING CLUSTERING”**

can cluster the conference proceedings in groups in which they are more associated with each other.

* Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.
* Throughout the project the focus has been on presenting information in an easy and intelligible manner.
* The project provides facilities like measuring the cluster quality through the random index and F- measure.

Thus reducing paperwork and automating the

Process of organizing the conference papers.

**….....**

**ACKNOWLEDGEMENT**

We take this opportunity to express our sincere gratitude to all those who helped us in various capacities in undertaking this project and devising the report.

We are privileged to express our sense of gratitude to our respected teacher

**Dr. Ashit kumar Das,**

Whose unparalleled knowledge, moral fiber and judgment along with his know-how, was an immense support in completing the project.

We take this opportunity also to thank our friends and contemporaries for their co-operation and compliance.

* Atul Sahay
* Shubham Kumar Singh Rajput
* Madhu Devi

**………**

**Contents**

1. SYNOPSIS OF PROJECT
2. TECHNOLOGY OVERVIEW
3. PROJECT SURVEY
4. PROPOSED METHDOLOGY
5. FLOWCHART
6. APPLIED ALGORITHM DETAILS
7. EXPERIMENTAL RESULTS
8. CODE SNIPPET
9. CONCLUSION
10. BIBLOGRAPHY

………

1. Synopsis of project

Abstract :

CONFERENCE PROCEEDING CLUSTERING can be used by education institutes OR ORGANIZTIONS to maintain the records oF CONFERENCE easily. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.

Objective :

* Preprocessing the collected
* Normalizing the records around specified parameter
* Clustering the records
* Measuring the cluster quality

Technologies Used :

* Python 2.7.1 IDE
* Nltk library
* Snowball Stemmer
* scikit

1. Technology Overview

Python – Python has a design philosophy which emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code)

Nltk – NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, and tagging, parsing, and semantic reasoning .

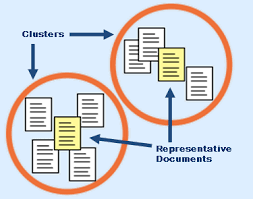
Scikit – an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified learning algorithm techniques.

1. **PROJECT SURVEY**

Background and Motivation –

Document clustering has been investigated for use in a number of different areas of text mining and information retrieval. Initially, document clustering was investigated for improving the precision or recall in information retrieval systems [Rij79, Kow97] and as an efficient way of finding the nearest neighbors of a document [BL85]. More recently, clustering has been proposed for use in browsing a collection of documents [CKPT92] or in organizing the results returned by a search engine in response to a user’s query [ZEMK97]. Document clustering has also been used to automatically generate hierarchical clusters of documents [KS97]. (The automatic generation of a taxonomy of Web documents like that provided by Yahoo! A somewhat different approach [AGY99] finds the natural clusters in an already existing document taxonomy (Yahoo!), and then uses these clusters to produce an effective document classifier for new documents. Agglomerative hierarchical clustering and K-means are two clustering techniques that are commonly used for document clustering.

Agglomerative hierarchical clustering is often portrayed as “better” than K-means, although slower. A widely known study, discussed in [DJ88], indicated that agglomerative hierarchical clustering is superior to K-means, although we stress that these results were with non-document data. In the document domain, Scatter/Gather [CKPT92], a document browsing system based on clustering, uses a hybrid approach involving both K-means and agglomerative hierarchical clustering. K-means is used because of its efficiency and agglomerative hierarchical clustering is used because of its quality. Recent work to generate document hierarchies [LA99] uses some of the clustering techniques from [CKPT92] and presents a result that indicates that agglomerative hierarchical clustering is better than K-means, although this result is just for a single data set and is not one of the major results of the paper. Initially we also believed that agglomerative hierarchical clustering was superior to K-means clustering, especially for building document hierarchies, and we sought to find new and better hierarchical clustering algorithms.

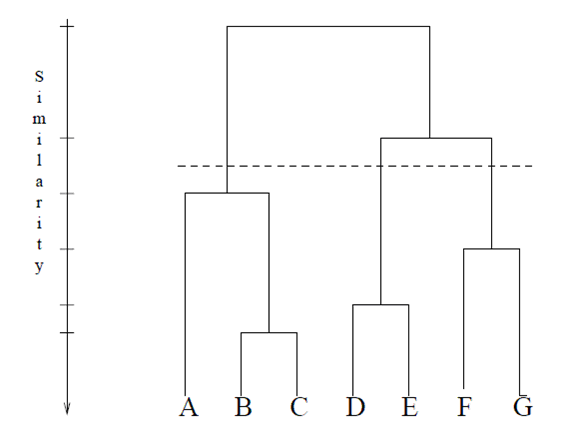
However, during the course of our experiments we discovered that a simple and efficient variant of K-means, “bisecting” K-means, can produce clusters of documents that are better than those produced by “regular” K-means and as good or better than those produced by agglomerative hierarchical clustering techniques. We have also been able to find what we think is a reasonable explanation for this behavior. The basic outline of this paper is as follows. Section 2 provides a brief review of agglomerative and hierarchical clustering techniques, while Section 3 reviews the vector space model for documents, particularly the aspects necessary to understand document clustering. Section 4 presents some measures of cluster quality that will be used as the basis for our comparison of different document clustering techniques and Section 5 gives some additional details about the K-means and bisecting K-means algorithms. Section 6 briefly describes the data sets used in our experiments, while sections 7 and 8 present our experimental results. More specifically, Section 7 compares three agglomerative hierarchical techniques, while Section 8 3 p1 p2 p3 p4 compares the best hierarchical technique to K-means and bisecting K-means. Section 9 presents our explanation for these results and Section 10 is a summary of our results.

1. PROPOSED METHDOLOGY

**Clustering Techniques**

In this section we provide a brief overview of hierarchical and partitioned (K-means) clustering techniques .Hierarchical techniques produce a nested sequence of partitions, with a single, all inclusive cluster at the top and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining two clusters from the next lower level (or splitting a cluster from the next higher level). The result of a hierarchical clustering algorithm can be graphically displayed as tree, called a genogram. This tree graphically displays the merging process and the intermediate clusters. The genogram at the right shows how four points can be merged into a single cluster. For document clustering, this genogram provides a taxonomy, or hierarchical index.

There are two basic approaches to generating a hierarchical clustering:

a) Agglomerative: Start with the points as individual clusters and, at each step, merge the most similar or closest pair of clusters. This requires a definition of cluster similarity or distance. 

b) Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide, at each step, which cluster to split and how to perform the split.

Agglomerative techniques are more common, and these are the techniques that we will compare to K-means and its variants.

**We summarize the traditional agglomerative hierarchical clustering procedure as follows:**

**Simple Agglomerative Clustering Algorithm**:

1. Compute the similarity between all pairs of clusters, i.e., calculate a similarity matrix whose i-j th entry gives the similarity between the i th and j th clusters.

2. Merge the most similar (closest) two clusters.

3. Update the similarity matrix to reflect the pairwise similarity between the new cluster and the original clusters.

4. Repeat steps 2 and 3 until only a single cluster remains

**The Vector Space Model and Document Clustering**

Many issues specific to documents are discussed more fully in information retrieval texts [Rij79, Kow97]. We briefly review a few essential topics to provide a sufficient background for understanding document clustering. For our clustering algorithms documents are represented using the vector-space model. In this model, each document, d, is considered to be a vector, d, in the term-space (set of document “words”). In its simplest form, each document is represented by the

(TF) vector, dtf = (tf1, tf2, …, tfn), where tfi is the frequency of the i th term in the document.

(Normally very common words are stripped out completely and different forms of a word are reduced to one canonical form.) In addition, we use the version of this model that weights each term based on its inverse document 5 frequency (IDF) in the document collection. (This discounts frequent words with little discriminating power.) Finally, in order to account for documents of different lengths, each document vector is normalized so that it is of unit length. The similarity between two documents must be measured in some way if a clustering algorithm is to be used. There are a number of possible measures for computing the similarity between documents, but the most common one is the cosine measure, which is defined as

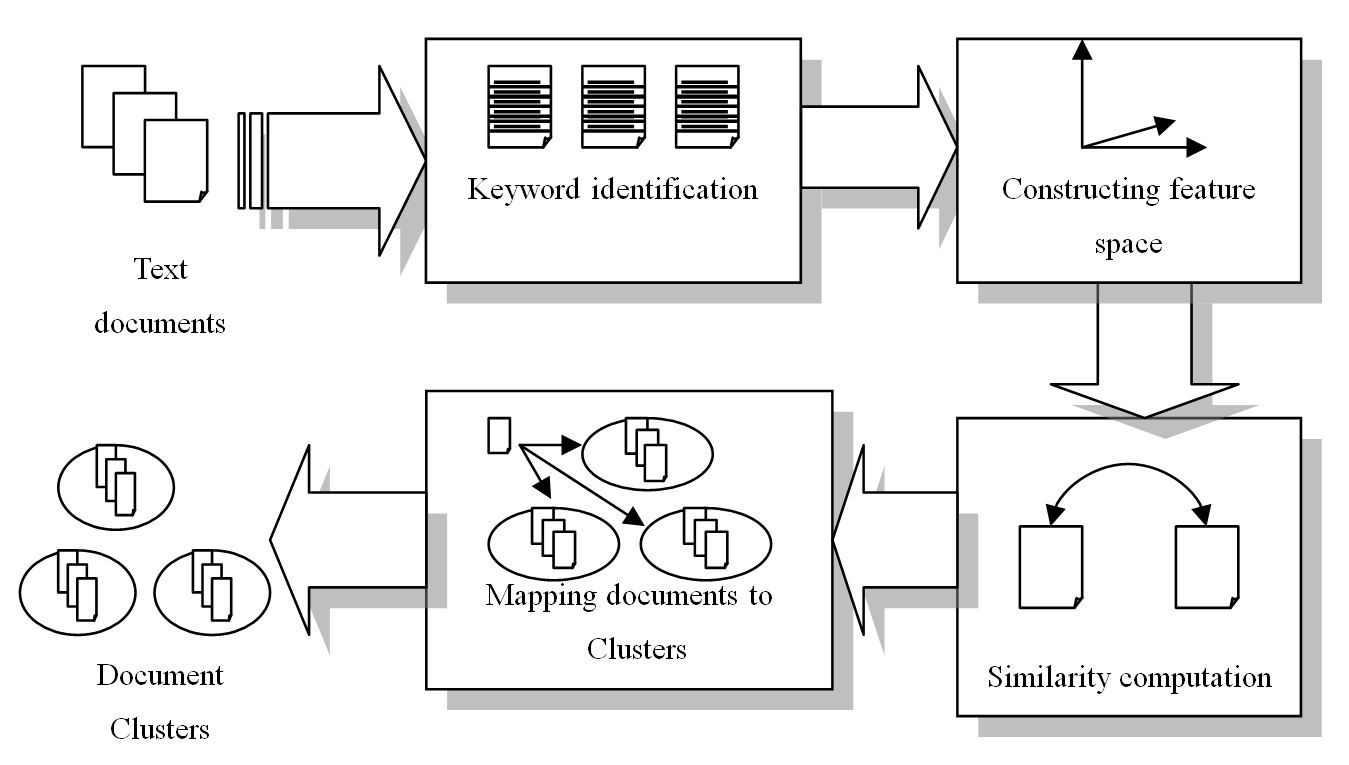
Cosine ( d1, d2 ) = (d1 • d2) / ||d1|| ||d2|| , where • indicates the vector dot product and ||d|| is the length of vector d.

Given a set, S, of documents and their corresponding vector representations, we define the centroid vector c to be ∈ = S d S c d 1 which is nothing more than the vector obtained by averaging the weights of the various terms present in the documents of S. Analogously to documents, the similarity between two centroid vectors and between a document and a centroid vector are computed using the cosine measure, i.e., cosine( d, c ) = (d • c) / ||d|| ||c|| = (d • c) / ||c|| cosine( c1, c2) = (c1 • c2) / ||c1|| ||c2||

Note that even though the document vectors are of length one, the centroid vectors will not necessarily be of unit length. (We use these two definitions in defining two of our agglomerative hierarchical techniques in Section 7, the “intra-cluster similarity and “centroid similarity” techniques, respectively.) For K-means clustering, the cosine measure is used to compute which document centroid is closest to a given document. While a median is sometimes used as the centroid for K-means clustering, we follow the common practice of using the mean. The mean is easier to calculate than the median and has a number of nice mathematical properties. For example, calculating the dot product between a document and a cluster centroid is equivalent to calculating the average similarity between that document and all the documents that comprise the cluster the centroid represents. (This observation is the basis of the “intracluster similarity” agglomerative hierarchical clustering technique in section 7.) Mathematically, 6 ( ) ∈ ∈ • = • = d S d S cosine S S d1 c d1 d d1, d 1 1 Also the square of the length of the centroid vector is just the average pairwise similarity between all points in the cluster. (This includes the similarity of each point with itself, which is just 1.) In the following section, we will use this average pairwise similarity as the basis for one of the measures for quantifying the goodness of a clustering algorithm. ( )

**d′ d = d • d = c • c ∈ ∈ ′∈ ∈ d S d S d S d S S S cosine S 1 1 , 1 2 = ||c||2**

1. **FLOWCHART**



1. **APPLIED ALGORITHM DETAILS**

**Data Sets**

In all of the data sets, we have removed stop words, i.e., common words such as “a”, “are”, “do”, and “for”. We have also performed stemming using Porter's suffix-stripping algorithm. Thus, all the words sharing the same stem are considered to be the same word. For example, words “compute”, “computing”, and “computed” are stemmed to “comput”. The summary of documents used in this paper is shown in Table 1. The details of each data set are described here.

We collected documents that have relevance judgments and then selected documents that have just a single relevance judgment. The class labels of la1, and la2 were generated according to the section names of articles, such as “Entertainment”, “Financial”, “Foreign”, “Metro”, “National”, and “Sports.” Data sets re0 and re1 are from Reuters-21578 text categorization test 10 collection Distribution 1.0 [reut]. Data set wrap is from the Web ACE project (WAP) [han98]. Each document corresponds to a web page listed in the subject hierarchy of Yahoo!.

**AGGLOMERATIVE HIERARCHICAL ALGORITHM**

**Overview**

Agglomerative hierarchical clustering is a bottom-up clustering method where clusters have sub-clusters, which in turn have sub-clusters, etc. The classic example of this is species taxonomy. Gene expression data might also exhibit this hierarchical quality (e.g. neurotransmitter gene families). Agglomerative hierarchical clustering starts with every single object (gene or sample) in a single cluster. Then, in each successive iteration, it agglomerates (merges) the closest pair of clusters by satisfying some similarity criteria, until all of the data is in one cluster.

The hierarchy within the final cluster has the following properties:

* Clusters generated in early stages are nested in those generated in later stages.
* Clusters with different sizes in the tree can be valuable for discovery.

**Process**

* Assign each object to a separate cluster.
* Evaluate all pair-wise distances between clusters.
* Construct a distance matrix using the distance values.
* Look for the pair of clusters with the shortest distance.
* Remove the pair from the matrix and merge them.
* Evaluate all distances from this new cluster to all other clusters, and update the matrix.
* Repeat until the distance matrix is reduced to a single element.

**Advantages**

* It can produce an ordering of the objects, which may be informative for data display.
* Smaller clusters are generated, which may be helpful for discovery.

**Disadvantages**

* No provision can be made for a relocation of objects that may have been 'incorrectly' grouped at an early stage. The result should be examined closely to ensure it makes sense.
* Use of different distance metrics for measuring distances between clusters may generate different results. Performing multiple experiments and comparing the results is recommended to support the veracity of the original results.

**Techniques**

**Intra-Cluster Similarity Technique (IST):**

This hierarchical technique looks at the similarity of all the documents in a cluster to their cluster centroid and is defined by

Sim(X) = d∈X cosine (d,c), where d is a document in cluster, X, and c is the centroid of cluster X.

The choice of which pair of clusters to merge is made by determining which pair of clusters will lead to smallest decrease in similarity. Thus, if cluster Z is formed by merging clusters X and Y, then we select X and Y so as to maximize Sim(Z) – (Sim(X) + Sim(Y)).

Note that Sim (Z) – (Sim(X) + Sim(Y)) is non-positive.

**Centroid Similarity Technique (CST)**: This hierarchical technique defines the similarity of two clusters to be the cosine similarity between the centroids of the two clusters.

**Complexity**

It defines the cluster similarity as follows, similarity (cluster1, cluster2) = ( 1)\* ( 2) ( , ) 2 1 1 2 2 1 size cluster size cluster cosine d cluster d cluster ∈ ∈ d d where d1 and d2 are, documents, respectively, in cluster1 and cluster2.

Time complexity of at least O(*n2log n*) is required, where *‘n’* is the number of data points.

1. EXPERIMENTAL RESULT

For clustering, two measures of cluster “goodness” or quality are used. One type of measure allows us to compare different sets of clusters without reference to external knowledge and is called an internal quality measure. As mentioned in the previous section, we will use a measure of “overall similarity” based on the pairwise similarity of documents in a cluster. The other type of measures lets us evaluate how well the clustering is working by comparing the groups produced by the clustering techniques to known classes. This type of measure is called an external quality measure. One external measure is entropy [Sha48], which provides a measure of “goodness” for un-nested clusters or for the clusters at one level of a hierarchical clustering. Another external measure is the F-measure, which, as we use it here, is more oriented toward measuring the effectiveness of a hierarchical clustering. The F measure has a long history, but was recently extended to document hierarchies in [LA99]. There are many different quality measures and the performance and relative ranking of different clustering algorithms can vary substantially depending on which measure is used. However, if one clustering algorithm performs better than other clustering algorithms on many of these measures, then we can have some confidence that it is truly the best clustering algorithm for the situation being evaluated. As we shall see in the results sections, the bisecting K-means algorithm has the best performance for the three quality measures that we are about to describe.

**Entropy** :

We use entropy as a measure of quality of the clusters (with the caveat that the best entropy is obtained when each cluster contains exactly one data point). Let CS be a clustering solution. For each cluster, the class distribution of the data is calculated first, i.e., for cluster j we 7 compute pij, the “probability” that a member of cluster j belongs to class i. Then using this class distribution, the entropy of each cluster j is calculated using the standard formula ( )ij i Ej = − pij log p , where the sum is taken over all classes. The total entropy for a set of clusters is calculated as the sum of the entropies of each cluster weighted by the size of each cluster: = = m j j j CS n n E E 1 \* , where nj is the size of cluster j, m is the number of clusters, and n is the total number of data points.

**F measure** :The second external quality measure is the F measure [LA99], a measure that combines the precision and recall ideas from information retrieval [Rij79, Kow97]. We treat each cluster as if it were the result of a query and each class as if it were the desired set of documents for a query. We then calculate the recall and precision of that cluster for each given class. More specifically, for cluster j and class i Recall( i, j ) = nij / ni Precision( i, j ) = nij / nj where nij is the number of members of class i in cluster j, nj is the number of members of cluster j and ni is the number of members of class i. The F measure of cluster j and class i is then given by F(i, j) = (2 \* Recall( i, j ) \* Precision( i, j )) / ((Precision( i, j ) + Recall( i, j )) For an entire hierarchical clustering the F measure of any class is the maximum value it attains at any node in the tree and an overall value for the F measure is computed by taking the weighted average of all values for the F measure as given by the following. F = { } F( ) i j n n i i max , where the max is taken over all clusters at all levels, and n is the number of documents.

**Overall Similarity** In the absence of any external information, such as class labels, the cohesiveness of clusters can be used as a measure of cluster similarity. One method for computing the cluster cohesiveness is to use the weighted similarity of the internal cluster similarity, 8 ( ) , . 1 2 ′∈ ∈ ′ d S d S cosine S d d . Recall that in Section 3 it was shown that this is just the squared length of the cluster centroid, ||c||2 .

1. **CODE SNIPPET**

**Clustering.py**

import os, math

def get\_distance(word\_fr1, word\_fr2):

#--- paper is already merged ---#

if (word\_fr1 is None or word\_fr2 is None):

return -float('inf')

num = 0.0

for i in range(no\_of\_words):

num += word\_fr1[i]\*word\_fr2[i]

paper1 = 0.0

for i in range(no\_of\_words):

paper1 += word\_fr1[i]\*\*2

paper2 = 0.0

for i in range(no\_of\_words):

paper2 += word\_fr2[i]\*\*2

ans = num/(paper1\*\*0.5 \* paper2\*\*0.5)

return num

fin = open("data", 'r')

temp\_data = fin.readlines()

##print "\ntemp data :", temp\_data, "has", len(temp\_data), "lines"

#--- Number of Papers ---#

no\_of\_papers = len(temp\_data)//3

#print "\nno of papers :", no\_of\_papers

no\_of\_words = len(temp\_data[-2].split())

#print "\nno of words :", no\_of\_words

titles = []

authors = []

word\_frs = []

#--- Separating Title, Author, Word-Frequency ---#

for i in range(no\_of\_papers):

title\_curr, author\_curr = temp\_data[i\*3].split('&')

title\_curr = title\_curr.split(':')[1].strip()

author\_curr = author\_curr.split(':')[1].strip()

word\_fr\_curr = map(int, temp\_data[i\*3 + 1].split())

word\_fr\_curr = word\_fr\_curr + [0]\*(no\_of\_words - len(word\_fr\_curr))

titles.append(title\_curr)

authors.append(author\_curr)

word\_frs.append(word\_fr\_curr)

#--- #printing original data ---#

#print "\noriginal word\_frs :"

for i in range(no\_of\_papers):

#print titles[i], 'by', authors[i]

for e in word\_frs[i]:

do\_nothing = True

#print ("%.2f" % e), '\t',

#print ''

#--- Finding idf for each word ---#

#--- idf(t, D) = log (N/n) ---#

#--- 'N' is no. of papers ---#

#--- 'n' is no. of papers where word 't' is present ---#

n = [0]\*no\_of\_words

for i in range(no\_of\_papers):

for j in range(no\_of\_words):

if (word\_frs[i][j] > 0):

n[j] += 1

for i in range(no\_of\_papers):

for j in range(no\_of\_words):

word\_frs[i][j] = word\_frs[i][j] \* math.log(no\_of\_papers/n[j])

#--- #printing modified data ---#

#print "\nmodified word\_frs (after tf x idf):"

for i in range(no\_of\_papers):

#print titles[i], 'by', authors[i]

for e in word\_frs[i]:

do\_nothing = True

#print ("%.2f" % e), '\t',

#print ''

#--- Finding max\_fr for each word ---#

max\_fr = [0]\*no\_of\_words

for i in range(no\_of\_papers):

for j in range(no\_of\_words):

max\_fr[j] = max(max\_fr[j], word\_frs[i][j])

#--- Normalizing words\_frs ---#

for i in range(no\_of\_papers):

for j in range(no\_of\_words):

word\_frs[i][j] = (word\_frs[i][j]\*10)/float(max\_fr[j])

#--- #printing modified data ---#

#print "\nmodified word\_frs (after normalization):"

for i in range(no\_of\_papers):

#print titles[i], 'by', authors[i]

for e in word\_frs[i]:

do\_nothing = True

#print ("%.2f" % e), '\t',

#print ''

parent = {}

for i in range(no\_of\_papers):

parent[i] = i

def find(x):

while (parent[x] != x):

x = parent[x]

return x

def union(i, j):

pi = find(i)

pj = find(j)

parent[pj] = pi

#--- Forming cluster ---#

no\_of\_papers\_temp = no\_of\_papers

while (True):

matrix = []

for i in range(no\_of\_papers\_temp):

matrix.append(['\_']\*no\_of\_papers\_temp)

#--- Finding closest pair ---#

curr\_closest\_pair = None

curr\_closest\_dist = -float('inf')

for i in range(no\_of\_papers\_temp):

for j in range(i+1, no\_of\_papers\_temp):

matrix[i][j] = get\_distance(word\_frs[i], word\_frs[j])

if (matrix[i][j] > curr\_closest\_dist):

curr\_closest\_dist = matrix[i][j]

curr\_closest\_pair = (i, j)

#print "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#print "\nclosest pair :", curr\_closest\_pair

#print "\ndistances :"

for i in range(no\_of\_papers\_temp):

for j in range(no\_of\_papers\_temp):

if (type(matrix[i][j]) is str):

do\_nothing = True

#print matrix[i][j], '\t',

else:

do\_nothing = True

#print ("%.2f" % matrix[i][j]), '\t',

#print ''

#--- Terminating condition ---#

if (curr\_closest\_dist <= 0.2):

#print "\nno two paper are close enough to cluster"

break

#--- Merge current closest pair ---#

p1 = curr\_closest\_pair[0]

p2 = curr\_closest\_pair[1]

union(p1, p2)

#no\_of\_papers\_temp -= 1 #--- commented so that matrix remains okay ---#

##print word\_frs, '->',

for i in range(no\_of\_words):

word\_frs[p1][i] = int(math.ceil((word\_frs[p1][i] + word\_frs[p2][i])/2.0))

word\_frs[p2] = None

##print word\_frs

#print "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

#--- Writing to File ---#

f = open('cluster', 'w')

count = 1

for i in range(no\_of\_papers):

title\_in\_cluster = ''

if (i == find(i)):

print "cluster", count

for j in range(i, no\_of\_papers):

if (find(j) == i):

print titles[j], 'by', authors[j]

title\_in\_cluster += titles[j]

title\_in\_cluster += ' \* '

count += 1

print ''

if (len (title\_in\_cluster) != 0):

f.write(title\_in\_cluster+'\n')

f.close()

pre-process.py

import os

can\_stem = False

if (can\_stem):

from nltk.stem.snowball import SnowballStemmer

def sort\_utility(x):

return x[1]

def get\_keywords\_from\_abstract(string, n):

#--- Returns 'n' most-frequent keywords and their frequencies from the 'string' ---#

if (string is None):

return [], []

abstract\_word\_list = string.split()

temp\_wf = {}

for word in abstract\_word\_list:

word = word.lower()

if (can\_stem):

word = str(stemmer.stem(word))

if (word not in stop\_words):

temp\_wf[word] = temp\_wf.get(word, 0) + 1

sorted\_wf = sorted(temp\_wf.items(), reverse = True, key = sort\_utility)

word\_list = []

freq\_list = []

for i in range(n):

word\_list.append(sorted\_wf[i][0])

freq\_list.append(sorted\_wf[i][1])

return word\_list, freq\_list

def get\_keywords\_listed(string):

#--- Returns all the keywords listed in the 'string' and setting frequency = 20---#

keywords = string.split(',')

for i in range(len(keywords)):

keywords[i] = keywords[i].strip().lower()

if (can\_stem):

keywords[i] = str(stemmer.stem(keywords[i]))

return keywords

#--- Cleaning old data ---#

fin = open("data", "w")

fin.close()

fin = open("words", "w")

fin.close()

#--- Creating Stemmer ---#

if (can\_stem):

stemmer = SnowballStemmer('english')

#--- Getting the list of stop-words ---#

fsw = open("stopwords", 'r')

stop\_words = fsw.readlines()

for i in range(len(stop\_words)):

#--- Cleaning stop-words ---#

stop\_words[i] = stop\_words[i].strip()

if (can\_stem):

stop\_words[i] = str(stemmer.stem(stop\_words[i]))

fsw.close()

#--- Change directory to Papers ---#

os.chdir(os.getcwd() + '/Papers')

#--- Read all files and add to data ---#

file\_names = os.listdir(os.getcwd())

no\_of\_words\_to\_extract\_from\_abstract = 5

for file\_name in file\_names:

title = ''

author = ''

abstract = ''

keywords = ''

if (file\_name.endswith('.txt')):

fin = open(file\_name, 'r')

for line in fin:

#--- Trim blank lines ---#

if (len(line) > 2):

line = line.strip()

if (line.startswith('Title')):

title = line.split(':')[1].strip()

elif (line.startswith('Author')):

author = line.split(':')[1].strip()

elif (line.startswith('Keywords')):

keywords = line.split(':')[1].strip()

else:

if (line.startswith('Abstract')):

abstract += line.split(':')[1].strip()

else:

abstract += line.strip()

fin.close()

#--- Get additional keywords ---#

keywords\_from\_abstract, frequencies\_from\_abstract = get\_keywords\_from\_abstract(abstract, no\_of\_words\_to\_extract\_from\_abstract)

keywords\_listed = get\_keywords\_listed(keywords)

#--- Changing directory going outside 'Papers' directory ---#

os.chdir(os.getcwd()[:-7])

#--- Read all words from 'words' file ---#

fin\_words = open('words', 'r')

words = fin\_words.readlines()

fin\_words.close()

#--- Combining keywords (from abstract and those listed) and frequencies ---#

keywords\_combined = []

frequencies\_combined = []

weight\_of\_keywords = 10

for word in keywords\_from\_abstract + keywords\_listed:

#--- Skipping repeated words ---#

if (word in keywords\_combined):

continue

if (word in keywords\_from\_abstract and word in keywords\_listed):

keywords\_combined.append(word)

frequencies\_combined.append(weight\_of\_keywords + frequencies\_from\_abstract[keywords\_from\_abstract.index(word)])

elif (word in keywords\_from\_abstract):

keywords\_combined.append(word)

frequencies\_combined.append(frequencies\_from\_abstract[keywords\_from\_abstract.index(word)])

else:

keywords\_combined.append(word)

frequencies\_combined.append(weight\_of\_keywords)

#--- Printing temp data ---#

print title, author, ':', keywords\_combined, '->', frequencies\_combined

#--- Calculating frequencies of existing words --- #

visited = [False]\*len(keywords\_combined) #--- To track the new words >> all the words not visited at the end are new words ---#

frequency\_for\_curr\_file = []

for word in words:

word = word.strip() #--- removing '\n' from the end ---#

if word in keywords\_combined:

pos = keywords\_combined.index(word)

visited[pos] = True

frequency\_for\_curr\_file.append(frequencies\_combined[pos])

else:

frequency\_for\_curr\_file.append(0)

#--- Adding new words and their frequencies ---#

fout\_words = open('words', 'a') #--- To add new words ---#

for i in range(len(keywords\_listed)):

if (not visited[i]):

fout\_words.write(keywords\_combined[i].strip()+'\n')

frequency\_for\_curr\_file.append(frequencies\_combined[i])

fout\_words.close()

#--- Adding data to the 'data' file ---#

fout\_data = open('data', 'a')

fout\_data.write('title : ' + title + ' & author : ' + author + '\n')

temp = map(str, frequency\_for\_curr\_file)

fout\_data.write(' '.join(temp) + '\n')

fout\_data.write('\n')

fout\_data.close()

#--- Changing directory going inside 'Papers' directory ---#

os.chdir(os.getcwd() + '/Papers')

else:

print 'ERROR :', file\_name, 'is in wrong format'

print '----------'\*5

**Quality-check.py**

import os

d\_ground = {}

d\_predicted = {}

titles = []

#--- Get PREDICTED VALUE ---#

f = open('cluster', 'r')

count = 1

for line in f:

titles\_in\_cluster = line.split('\*')[:-1]

for title in titles\_in\_cluster:

d\_predicted[title.strip()] = count

count += 1

#--- Get GROUND VALUE ---#

#--- Change directory to Papers ---#

os.chdir(os.getcwd() + '/Papers')

#--- Read all files and add to data ---#

file\_names = os.listdir(os.getcwd())

for file\_name in file\_names:

ground\_value = file\_name.split()[0]

if (file\_name.endswith('.txt')):

fin = open(file\_name, 'r')

for line in fin:

#--- Trim blank lines ---#

if (len(line) > 2):

if (line.startswith('Title')):

title = line.split(':')[1].strip()

break

titles.append(title)

fin.close()

d\_ground[title] = ground\_value

no\_of\_papers = len(titles)

#print len(d\_ground.keys()), d\_ground

#print len(d\_predicted.keys()), d\_predicted

Css = 0.0

Csd = 0.0

Cds = 0.0

Cdd = 0.0

for i in range(no\_of\_papers):

for j in range(i+1, no\_of\_papers):

if (d\_ground[titles[i]] == d\_ground[titles[j]] and d\_predicted[titles[i]] == d\_predicted[titles[j]]):

Css += 1

elif (d\_ground[titles[i]] == d\_ground[titles[j]] and d\_predicted[titles[i]] != d\_predicted[titles[j]]):

Csd += 1

elif (d\_ground[titles[i]] != d\_ground[titles[j]] and d\_predicted[titles[i]] == d\_predicted[titles[j]]):

Cds += 1

else:

Cdd += 1

#print Css, Csd, Cds, Cdd

random\_index = (Css + Cdd) / (Css + Csd + Cds + Cdd)

precision = Css / (Css + Csd)

recall = Css / (Css + Cds)

f\_measure = (2 \* precision \* recall) / (precision + recall)

print "Random Index :", random\_index

print "Precision :", precision

print "Recall :", recall

print "F-Measure :", f\_measure

1. CONCLUSION
2. REFERENCE

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, (1998), ROCK: A Robust Clustering Algorithm for Categorical Attributes, In Proceedings of the 15th International Conference on Data Engineering, 1999.

[Han98] Eui-Hong (Sam) Han, Daniel Boley, Maria Gini, Robert Gross, Kyle Hastings, George Karypis, Vipin Kumar, B. Mobasher, and Jerry Moore, WebAce: A Web Agent for Document Categorization and Exploration. Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98).

[KS97] Daphe Koller and Mehran Sahami, Hierarchically classifying documents using very few words, Proceedings of the 14th International Conference on Machine Learning (ML), Nashville, Tennessee, July 1997, Pages 170-178.

[Kow97] Gerald Kowalski, Information Retrieval Systems – Theory and Implementation, Kluwer Academic Publishers, 1997.

[KR90] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: an Introduction to Cluster Analysis, John Wiley and Sons, 1990.

[LA99] Bjorner Larsen and Chinatsu Aone, Fast and Effective Text Mining Using Linear-time Document Clustering, KDD-99, San Diego, California, 1999.

[reut] D. Lewis, Reuters-21578 text categorization text collection 1.0. http://www.research.att.com/~lewis [Rij79] C. J. van Rijsbergen, (1989), Information Retrieval, Buttersworth, London, second edition.

[SS97] Hinrich Schutze and Craig Silverstein, Projections for Efficient Document Clustering, SIGIR ’97, Philadelphia, PA, 1997. [Sha48] Claude. E. Shannon, A mathematical theory of communication, Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948. Documents, KDD ’97, Pages 287-290, 1997.