1. Objectives
2. Our approach
3. Modules Description
   1. Preprocessing
   2. Calculating Tf-idf
   3. Computation of distance between two texts
   4. Clustering Procedure
4. Algorithms
   1. Term-frequency / Inverse document frequency (TFIDF)
   2. Clustering

4.2.1 Introduction

4.2.2 Comparison of clustering algorithms

4.2.3 DBSCAN

4.2.3.1 Introduction

4.2.3.2 Advantages

5. Comparison of tf-idf files for clustering

5.1. Method1

5.2. Method2

5.3. Method3

5.4.Single Cluster Explosion

5.5. Cluster Explosion

5.6. Unified Cluster formation

5.7. Estimation of best threshold for clustering

5.7.1. Clustering Performance Evaluation

5.7.1.1. Adjusted Rand Index

5.7.1.2. Mathematical Formulation

5.7.1.3. Mutual Information based scores

5.7.2. Estimating initial threshold from average scores:

6. Project Implementation

7. Results

8. Conclusion

9. References

**CHAPTER 1**

**Objectives**

**Objectives of the system:**

The primary objective of the system is ‘document clustering’ i.e. to discover natural grouping among documents in such a way that similar documents form a cluster, and documents in one cluster are quite dissimilar to documents in other clusters. The greater the similarity or homogeneity within a group and the greater the dissimilarity between groups the “better” or more distinct is the clustering.

Hence, in the following sections we shall describe in detail our approach for doing so.

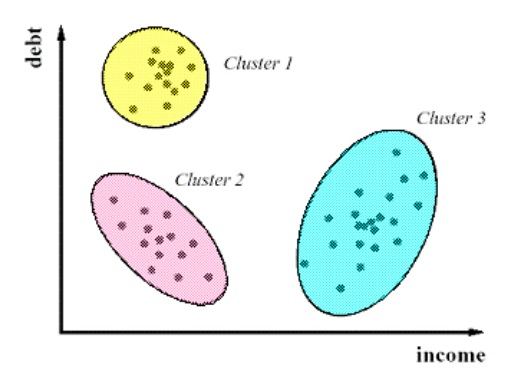


Figure 0.1. Examples of Clusters

Figure 0.1: shows an example of clustering which grouped objects into 3 clusters.

**<change this picture, give something more appealing>**

**CHAPTER 2**

**SYSTEM DESIGN**

**Our Approach:**

Since, our purpose is clustering of documents, however clustering can be applied to numerical data only. Hence, we have to devise a method by which we could quantify the textual information present in the documents in a certain way such that we can measure the similarity or dissimilarity between documents. In this project we have tried to do just that. We have first quantified the text data in documents and obtained a range of corresponding numerical data, from which we used with a popular clustering algorithm to cluster the documents.

The stages of the system is can be listed sequentially as follows.

Stage 1 : Text Preprocessing

Stage 2 : Term Weighting

Stage 3 : Computation of similarity between documents based on their term weights.

Stage 4 : Clustering

Stage 5 : Cluster performance evaluation

In Stage 1 we do some basic pre-processing on our initial text data. Here we remove punctuations, stopwords, numbers etc from the initial data.

In Stage 2 we find the term weights for each document i.e. we weight each word in accordance to their relevance to the document.

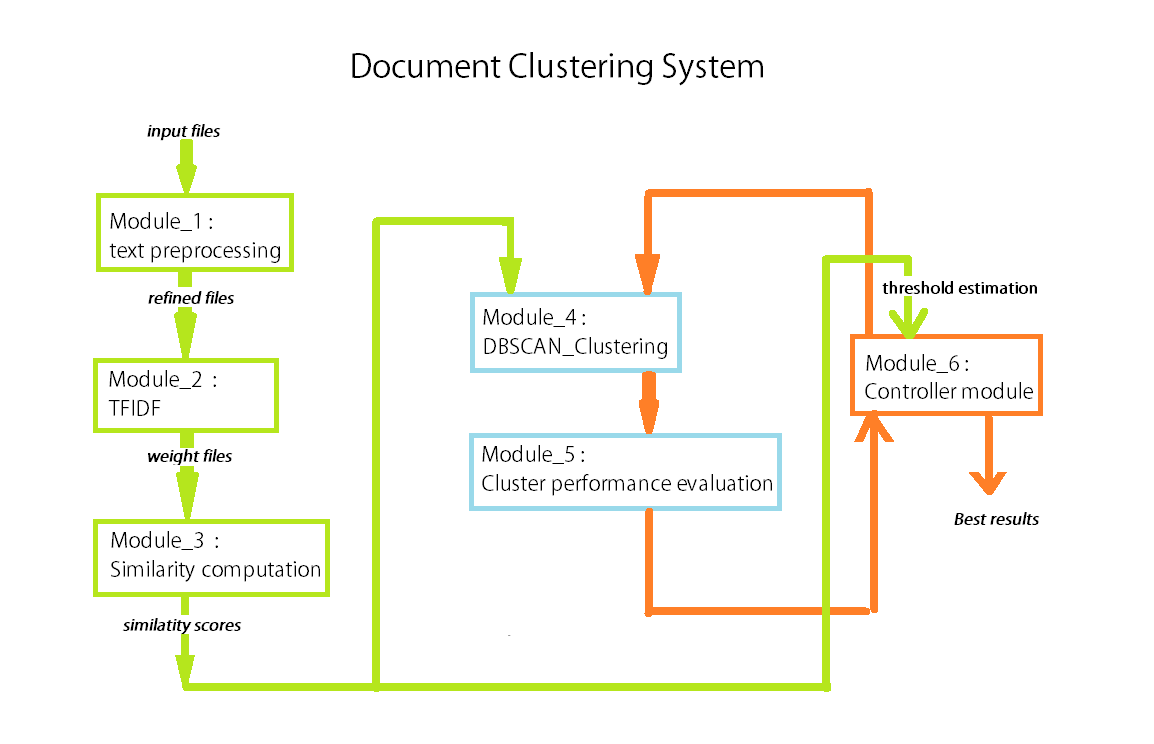
In Stage 3, we compute the similarity between texts based on the computed weights on the previous stage.

In Stage 4, we perform the main task of the project i.e. the clustering of the text.

In Stage 5, we evaluate the performance of the clusters formed. We designed the system to automatically find the best parameters for clustering resulting in the best performance in clustering. That is being done in this stage.

The work done and the method used in each of the stages will be described in the following chapters.

The detailed architecture of the system with respect to the modules present is shown in the figure below. The description of individual modules will be provided in the code documentation in a later part of this report:



**CHAPTER 3**

**SYSTEM ANALYSIS**

**Modules Description:**

A detailed description of each of the modules with their functions and characteristics are provided sequentially below. Here, only the theoretical aspects of the modules are described in detail. The implementation details of the modules are available in the documentation appended with this report.

**3.1. Preprocessing:**

**Once you are sure that all documents loaded properly, go on to pre-process your texts.** This step allows you to remove numbers, capitalization, common words, punctuation, and otherwise prepare your texts for analysis. This can be somewhat time consuming and picky, but it pays off in the end in terms of high quality analyses.

**1. Removal of Stopwords:**

What are stopwords?

Stopwords are words which are filtered out before or after processing of text. Stopwords are the words which are very common and are present in any text document regardless of their domain or subject. Hence, the presences of these words in a document do not provide any information regarding the topic or subject of the document.

Why do we need to remove stopwords?

Stopwords do not add any relevant information to any task in NLP and hence they are not desired. In fact, they create a lot of problems in many tasks. In case of text clustering too, stopwords create problems as they are present in every document, and hence documents which belong to different topics also are likely to have a large number of stopwords in common. This, in case raises up the similarity score between documents, even if the documents may be from entirely different genres or topics. Hence, stopwords need to be removed.

**2. Decapitalization:**

Why decapitalization is necessary?

If clustering is done without decapitalizing the text then two words if present in two different documents, however one has a capitalized starting letter and the other is entirely lowercase, then the words are not recognized as the same, thereby increasing the actual distance between the two documents. For e.g. suppose document1 contains the word ‘Flight’ and document2 contains the word ‘flight’. However, the computer does not recognize the two words as the same. Hence, in order to proceed with any further activity, it is best to decapitalize all the words, else some very vital information may be lost.

**3. Removal of Punctuation :**

Same reason as ‘stopword removal’. Copy, paste from the stopwords paragraph and EDIT as necessary.

**4. Removal of Numbers:**

Numbers inherently do not contain any domain specific information. For, eg, they occur in the case of amount of money, or date etc. Such things occur in almost any document regardless of their domain or topic. Hence, they are entirely useless for our task and need to be removed.

**3.2. Calculating TF-IDF:**

In order to proceed with finding the distance between texts, we need to find the features and their weights for each file in the dataset as found in previous step. Which means we shall find the distance between documents by finding the difference between the weights of the words in the documents. The features in each files are the words and they are weighted accordingly using a weighting scheme. It also contains the weight assigned to the word for each file. The weighting scheme employed is the basic term frequency – inverse document frequency (TF-IDF), which is explained in detail in section 4.1 of this report.

**3.3. Computation of semantic distance between two documents:**

After the TFIDF module is over and the weights of words in each document is found, we have to find the similarity between two documents. We shall find the similarity based on the weights of the words in each document.

There can be various ways of measuring the distance i.e., Manhattan distance, Euclidean distance, Cosine Similarity etc. We employed the Euclidean distance method. However, the procedure is not entirely simple and the detailed description of our distance measure and the approach by which we find the similarity is described **in the chapter ‘Computation of similarities between documents’**

**3.4. Clustering procedure:**

After the similarity between the documents are computed, we move on to the primary task, i.e. the clustering of the documents based on the similarity between documents. As described in section 4.2.3, we have used the DBSCAN clustering procedure for this stage. We shall describe in detail the reason and procedure why we used DBSCAN clustering and not and other algorithm. The DBSCAN clustering algorithm is explained in section 4.2.3 of this report.

**CHAPTER4**

**ALGORITHMS**

**Algorithms used in this system:**

In this section we describe the primary algorithms used in the system which are:

1. Term frequency - Inverse document frequency(TF-IDF) used for finding the weights of words inside a cluster in the characteristic file formation step.
2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) which is used for the clustering of text.

In order to describe the above two topics, some other topics do need to be introduced.

**4.1 Term Frequency – Inverse Document Frequency (TF-IDF):**

In this system, the TF-IDF has been calculated for each files by using the files from stop\_remove folder . Before, we explain how that is being done, a basic introduction is given here, about TF-IDF scheme we used in the system.

Tf-idf is the most common and most effective term weighting scheme used in information retrieval. tf–idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

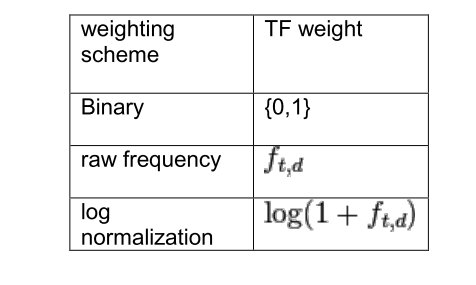
**4.1.1 Term frequency:**

Suppose we have a set of English text documents and wish to determine which document is most relevant to the query "the brown cow". A simple way to start out is by eliminating documents that do not contain all three words "the", "brown", and "cow", but this still leaves many documents. To further distinguish them, we might count the number of times each term occurs in each document and sum them all together; the number of times a term occurs in a document is called its term frequency.

The first form of term weighting is due to Hans Peter Luhn (1957) and is based on the Luhn Assumption:

● The weight of a term that occurs in a document is simply proportional to the term frequency.

The simplest variants of term-frequency weightage scheme are mentioned below:



In the case of the term frequency tf(t,d), the simplest choice is to use the raw frequency of a term in a document, i.e. the number of times that term t occurs in document d. If we denote the raw frequency of t by f(t,d), then the simple tf scheme is tf(t,d) = f(t,d). Other possibilities include.

● Boolean "frequencies": tf(t,d) = 1 if t occurs in d and 0 otherwise;

● logarithmically scaled frequency: tf(t,d) = 1 + log f(t,d), or zero if f(t, d) is zero;

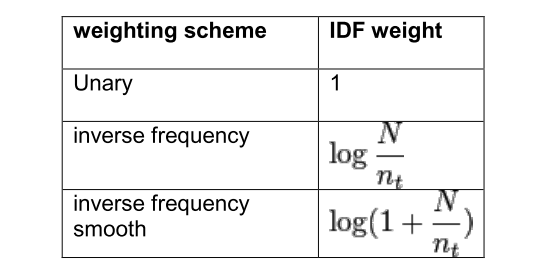
**4.1.2. Inverse document frequency**

However, because the term "the" is so common, this will tend to incorrectly emphasize documents which happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms "brown" and "cow". The term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms, unlike the less common words "brown" and "cow". Hence an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

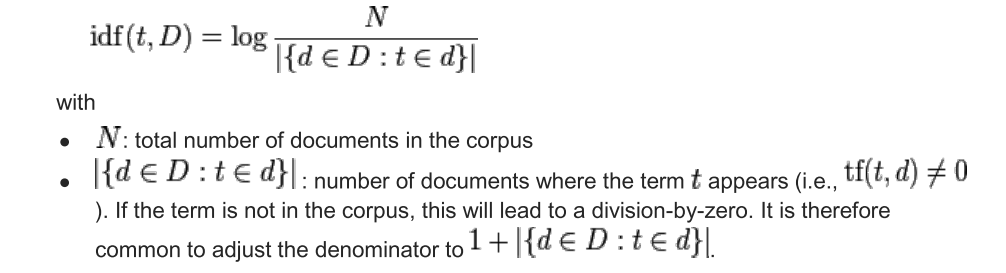
Karen Spärck Jones (1972) conceived a statistical interpretation of term specificity called IDF, which became a cornerstone of term weighting:

● The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs.

The simplest variants of inverse document frequency weightage scheme are mentioned below:



The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

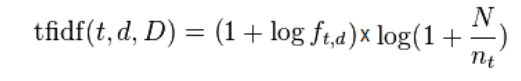


Mathematically the base of the log function does not matter and constitutes a constant multiplicative factor towards the overall result.

Then tf–idf is calculated as



In our system, we have used a logarithmically scaled term frequency weighting scheme and the inverse frequency smoothing as the inverse document frequency weighting scheme, and so, the final score is calculated as:



**4.2 Clustering:**

In this section we will give an introduction to clustering and a few of its methods. We will then mention which is the best suitable approach to solve our problem and the reasons for selecting the approach.

**4.2.1 Introduction:**

Clustering or cluster analysis is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It will often be necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

**4.2.2 Comparison of clustering algorithms:**

Clustering algorithms are plenty. So, in order to find the best clustering algorithm suited for our purpose we provide a comparison between some of the most common clustering algorithms that we have considered, with respect to our work, and why we ended up choosing a particular method. For our problem, we needed an algorithm that can cluster a group of texts which are near to each other, as per the distance between them.

Here we try to debate the suitability of various standard clustering algorithms for our purpose.

**k-means**: k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. In our case, k-means would cluster n texts into k different clusters, where k is a parameter set by the user. It recursively finds k most suitable cluster centroids, by finding their mean squared errors to the points near to it. Any point lies in the cluster of the centroid nearest to it.

However, K-means is good only for clustering linearly separable data, which is not so suitable for our case. The ability to separate out non-linear data is the primary requirement of the system. However, for a better comparison of the clustering algorithms, we take the help of a graphic from scikit-learn as in Fig.03.

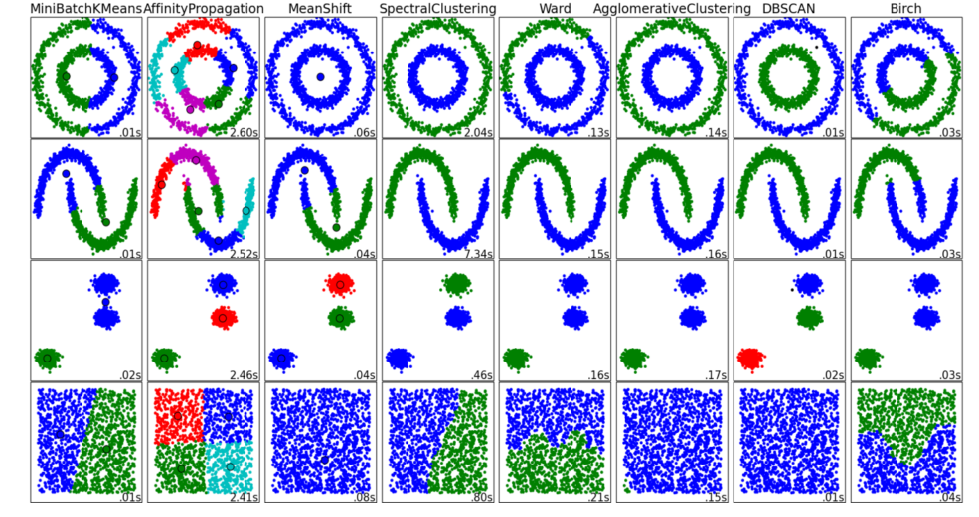


Fig.03: Graphical comparison of different clustering algorithms

From the diagram it is clear that DBSCAN is the best suitable algorithm for our purpose. Hence, we select it as the algorithm used for clustering, in our system.

**4.2.3 DBSCAN:**

DBSCAN is a density based clustering algorithm. Here we give a detailed introduction to DBSCAN.

**4.2.3.1 Introduction:**

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

Consider a set of points in some space to be clustered. For the purpose of DBSCAN clustering, the points are classified as core points, (density-) reachable points and outliers, as follows:

● A point p is a core point if at least minPts points are within distance ε of it, and those points are said to be directly reachable from p. No points are reachable from a non-core point.

● A point q is reachable from p if there is a path p1, ..., pn with p1 = p and pn = q, where each pi+1 is directly reachable from pi (so all the points on the path must be core points, with the possible exception of q).

● All points not reachable from any other point are outliers.

Now if p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points. This concept is illustrated in Fig 04. Below:

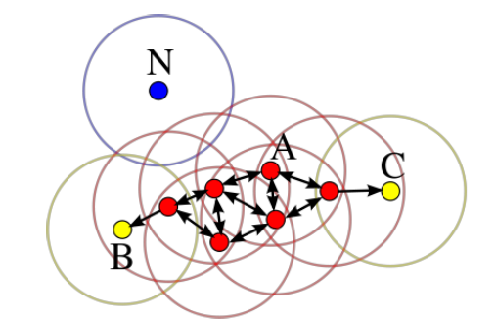


Fig.04 : In this diagram, minPts = 3. Point A and the other red points are core points, because at least three points surround it in an ε radius. Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor density-reachable.

Reachability is not a symmetric relation since, by definition, no point may be reachable from a non-core point, regardless of distance (so a non-core point may be reachable, but nothing can be reached from it). Therefore a further notion of connectedness is needed to formally define the extent of the clusters found by DBSCAN. Two points p and q are density-connected if there is a point o such that both p and q are density-reachable from o. Density connectedness is symmetric.

A cluster then satisfies two properties:

1. All points within the cluster are mutually density-connected.

2. If a point is density-reachable from any point of the cluster, it is part of the cluster as well.

**4.2.3.2 Advantages of DBSCAN:**

1. DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.

2. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points)is reduced.

3. DBSCAN has a notion of noise, and is robust to outliers.

4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)

5. The parameters minPts and ε can be set by a domain expert, if the data is well understood.

**CHAPTER 5**

**TECHNIQUES FOR SIMILARITY COMPUTATION BETWEEN DOCUMENTS**

**Comparison of tf-idf files for clustering:**

Although a small step, the stage is the most significant stage in the entire system. Comparison of clusters is one of the areas of primary consideration in the design of the entire system. Weighting schemes used during characteristic file comparison may change the results to a great extent.

After the formation of the characteristic files from each cluster that is extracted, each characteristic file need to be uniquely stored in a central location, which we will refer to as the ‘unique folder’ for the time being. By ‘uniquely’, we mean that, if a new cluster (i.e. the characteristic file) is similar to another cluster which had previously occurred, once or more times in previous document, then, the new cluster encountered need not be stored in the unique folder. This is the reason why every unique cluster encountered is given a unique ‘cluster ID’ and stored inside the vault, and any document where the cluster has occurred, the unique cluster ID is inserted into the pattern, as mentioned in the previous section.

This is the backbone of the continuous learning system i.e. extracting clusters and storing them uniquely i.e. if any new clusters are extracted then push it into the unique folder. Now the primary task for this phase is the comparison of clusters, after which we can decide if a new cluster is similar to a previously occurring one. For comparison of two cluster files, we need two characteristic file. As previously described, characteristic files contain the words as features, their corresponding weights and the likelihood of occurrence for each word. We have tried out a few methods for comparison of clusters, till we decided on using the method which provided us the ideal results. The methods are explained in brief below:

**5.1 Method 1:**

Suppose two clusters need to be compared, let them be ‘Cluster A’ and ‘Cluster B’, then the words contained in the two clusters can be categorized into two different classes : words which are common to both the clusters and words which are present in just one of the clusters.

Suppose,

CW = [cw1, cw2, cw3,….] represent the set of words which are common to both the clusters

WCA = [wca1, wca2, wca3, ….] represent the set of words which are present just in cluster A

WCB = [wcb1, wcb2, wcb3, ….] represent the set of words which are present just in cluster B

The comparison score between the two clusters are the sum of scores of the words of both common and the words present in one cluster.

a)Scoring for common words.

For common words, i.e. words present in both the clusters which need to be compared, a word will have weightage and likelihood valued in both the clusters. In that case, the score is calculated as follows,

Score(cw1) = abs(weight\_A(cw1) – weight\_B(cw1)) + abs (likelihood\_A(cw1) –

likelihood\_B(cw1))

b) For words present in just one cluster.

For words present in one cluster, the word has just one weight and likelihood pair. In that case the score is calculated in the same way as above, however, the other weight and likelihood both are taken as 0.

Score(wca1) = weight\_A(wca1) + likelihood\_A(wca1)

Score(wcb4) = weight\_B(wcb4) + likelihood\_B(wcb4)

Now, we find the sum of scores for all the words in both the clusters, whether it is present in one cluster or both the clusters. The obtained sum is the final similarity score of the two clusters which are compared.

When the total score from the comparison of two clusters is obtained, we compare it to a cluster similarity threshold, which is a parameter which can be modified during usage. If the obtained score is below cluster similarity threshold, we declare that the two compared clusters as similar, otherwise they are not similar and the characteristic file of the new cluster gets inserted into the unique folder. The threshold is decided as a number near to the median of the scores obtained on training the model on a few similar clusters.

The above method is effective upto a certain extent, but it has a lot of drawbacks, as listed below:

Sometime two clusters which might be identical, but they might have a few number of dissimilar words in each cluster and some or all of those words may have a high weightage (and a high likelihood as well), then that is enough to take the score high enough for the system to proclaim the two clusters as dissimilar, although many words and their corresponding weightages and likelihoods may almost be similar. This can be said to be a case of outlier dependency as the singleton words may be said to be outliers.

In order to solve this problem, we propose Method2.

**5.2 Method 2:**

a) Scoring for common words.

Score(cw1) = abs(weight\_A(cw1) \* likelihood\_A(cw1) - weight\_B(cw1)\* likelihood\_B(cw1))

b) For words present in just one cluster.

Score(wca1) = weight\_A(wca1) + likelihood\_A(wca1)

Score(wcb4) = weight\_B(wcb4) + likelihood\_B(wcb4)

In this method, while comparison of two clusters, words which have low likelihood of occurrence are likely to appear in just one cluster characteristic file, and they are more likely to have a low score. Even if they have a high score, the high score is likely to be counterbalanced by their low likelihood. In this way it improves on the previous method. However, satisfactory results have still not been obtained, as still there may be outliers. Hence, to improve on these above problems we propose method 3.

**5.3 Method 3:**

In this method, all weights and likelihoods are normalized. The normalization of the weights are done by dividing each weight by the total weights in that particular cluster, and similarly, the normalization of the likelihood are also done.

For eg: norm\_weight\_A(cw1) = weight\_A(cw1) / total\_weight\_A

norm\_likelihood\_A(cw1) = likelihood\_A(cw1) / total\_likelihood\_A

a)Scoring for common words.

Score(cw1) = abs(norm\_weight\_A(cw1) \* norm\_likelihood\_A(cw1) -norm\_weight\_B(cw1)\*norm\_ likelihood\_B(cw1))

b) For words present in just one cluster.

Score(wca1) =norm\_ weight\_A(wca1) + norm\_likelihood\_A(wca1)

Score(wcb4) = norm\_weight\_B(wcb4) + norm\_likelihood\_B(wcb4)

Using method 3, we obtained the ideal results during the training phase. All the documents of a particular category had matching clusters extracted in almost identical pattern, which means that the method is providing satisfactory results.

**5.XX Problems resulting in similarity computation.**

However, there are a couple of problems that we faced during the comparison of documents. The problems are related to the selection of the similarity threshold during similarity computation between documents. The problems have been described below:

**5.4 Single Cluster Explosion:**

In every unique characteristic file (i.e. the characteristics files stored in the unique folder), the last column denotes the cluster occurrence frequency (cof) i.e. the number of times the file has been worked on i.e. the number of times a similar cluster to the cluster has been encountered. This value is same for every word in the characteristics file. This number is required for the moving average calculation for the computation of the weightage and likelihood scores. After the training is done on a few samples of documents, every unique characteristic file will have a ‘cluster occurrence frequency’ which is quite less than the number of documents trained. However, sometimes there occurs one characteristic file which have a very high cof, i.e. if we tested on a total of 10 documents, with an average of 20 clusters each which results in around 200 clusters trained, then this particular characteristic file will have a cof of more than 100, which is quite aberrant. After detailed analysis, the reason for the single cluster explosion is found as follows:

1) Whenever inside a single document, two clusters are more or less similar and their similarity score is less than the ‘similarity threshold’, then the two clusters in the same document get matched to the same unique characteristic file.

2) When two clusters of the same document get matched to the same characteristic file, their characteristics are normalized and appended to the characteristic files that they got matched to.

3) Now, when another variant of the same document is trained, two clusters corresponding to the two identical clusters get matched again to the same characteristic file. Likewise the characteristic file starts growing bigger and bigger, with small additions from matched clusters.

4) As the characteristic file grows bigger, with the addition of new words, its probability of getting matched with any other cluster gets increased, and hence if any different cluster gets matched to it, likewise it subsequently leads to the explosion of words inside the characteristic file. It has been found that the single cluster explosion may be prevented. It has a simple solution: just do not let two clusters match inside the same document. The cluster similarity threshold can be fixed in such a way. The threshold should be less than the similarity score between two identical (as per the early threshold) clusters found previously, so that they cannot be identical again. Such a threshold works perfectly and will provide us with ideal results.

**5.5 Cluster Explosion:**

This is a trivial problem. Sometimes, when the cluster similarity threshold is too small, then even somewhat or almost similar clusters are not found to be similar by the system. In such a case, almost all the clusters formed from different documents form new unique clusters, which results in maximum number of unique clusters formed. We call such an event as ‘cluster explosion’. This problem can be solved by simply altering the cluster similarity threshold.

**5.6 Unified Cluster formation:**

This is just the reverse of the previous problem. When the cluster similarity threshold is too large, then most of the characteristics files are found to be similar to each other, as their similarity score is less than the threshold. In such a case, due to the occurrence of too less similar matches, very less number of unique clusters are formed. This problem too can be solved by simply altering the cluster similarity threshold. The best cluster similarity threshold will be one which will ensure that none of the above mentioned problems occur in the cluster similarity phase. So, although we didn’t have enough time to design an algorithm which could decide the optimal cluster similarity threshold, we pretty much solved the problem by manually altering the cluster similarity threshold while executing the system, and obtained thresholds which resulted in quite optimal patterns.

**5.7. Estimation of best threshold for clustering:**

We have seen some of the problems which come up because of a poorly selected similarity threshold. In order to prevent such a happening, we have to select the best threshold for clustering the documents. The estimation of best threshold will be discussed in a later stage. However, at first we shall discuss how to quantify the quality of clusters formed i.e. how we would could recognize a good or a bad cluster result.

**5.7.1. Clustering Performance Evaluation:**

Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm. In particular any evaluation metric should not take the absolute values of the cluster labels into account but rather if this clustering define separations of the data similar to some ground truth set of classes or satisfying some assumption such that members belong to the same class are more similar that members of different classes according to some similarity metric. Hence, here we discuss two methods for the

**5.7.1.1. Adjusted Rand Index:**

**Advantages:**

* **Random (uniform) label assignments have a ARI score close to 0.0** for any value of n\_clusters and n\_samples(which is not the case for raw Rand index or the V-measure for instance).
* **Bounded range [-1, 1]:** negative values are bad (independent labelings), similar clusterings have a positive ARI, 1.0 is the perfect match score.
* **No assumption is made on the cluster structure:**  can be used to compare clustering algorithms such as k-means which assumes isotropic blob shapes with results of spectral clustering algorithms which can find cluster with “folded” shapes.

**Drawbacks:**

Contrary to inertia,  **ARI requires knowledge of the ground truth classes** while is almost never available in practice or requires manual assignment by human annotators (as in the supervised learning setting).

However ARI can also be useful in a purely unsupervised setting as a building block for a Consensus Index that can be used for clustering model selection (TODO).

**5.7.1.2. Mathematical Formulation:**

If C is a ground truth class assignment and K the clustering, let us define a and b as:

* a, the number of pairs of elements that are in the same set in C and in the same set in K
* b, the number of pairs of elements that are in different sets in C and in different sets in K

The raw (unadjusted) Rand index is then given by:

\text{RI} = \frac{a + b}{C_2^{n_{samples}}}

Where C_2^{n_{samples}} is the total number of possible pairs in the dataset (without ordering).

However the RI score does not guarantee that random label assignments will get a value close to zero (esp. if the number of clusters is in the same order of magnitude as the number of samples).

To counter this effect we can discount the expected RI E[\text{RI}] of random labelings by defining the adjusted Rand index as follows:

\text{ARI} = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]}

**5.7.1.3. Mutual Information based scores:**

Given the knowledge of the ground truth class assignments  labels\_true and our clustering algorithm assignments of the same samples labels\_pred, the **Mutual Information** is a function that measures the **agreement** of the two assignments, ignoring permutations. Two different normalized versions of this measure are available, **Normalized Mutual Information(NMI)** and **Adjusted Mutual Information(AMI)**. NMI is often used in the literature while AMI was proposed more recently and is **normalized against chance**

**Advantages:**

* **Random (uniform) label assignments have a AMI score close to 0.0** for any value of n\_clusters and n\_samples(which is not the case for raw Mutual Information or the V-measure for instance).
* **Bounded range [0, 1]**: Values close to zero indicate two label assignments that are largely independent, while values close to one indicate significant agreement. Further, values of exactly 0 indicate **purely** independent label assignments and a AMI of exactly 1 indicates that the two label assignments are equal (with or without permutation).
* **No assumption is made on the cluster structure**: can be used to compare clustering algorithms such as k-means which assumes isotropic blob shapes with results of spectral clustering algorithms which can find cluster with “folded” shapes.

**Drawbacks:**

* Contrary to inertia, **MI-based measures require the knowledge of the ground truth classes** while almost never available in practice or requires manual assignment by human annotators (as in the supervised learning setting).

However MI-based measures can also be useful in purely unsupervised setting as a building block for a Consensus Index that can be used for clustering model selection.

* NMI and MI are not adjusted against chance.

**5.7.2. Estimating initial threshold from average scores:**

**<I will write this later>**

**CHAPTER 6**

**System Implementation**

**Implementation:**

In our project, we have discussed about six modules. The design of the various modules are discussed below:

**6.1. Module 1:**

**Name: 1\_rempunct**

**Module description:**

This module extracts all the text files from the dataset named source. For each input files it removes the stopwords, punctuation, numbers and other special characters, and also decapitalize the words present in the original text file. The procedure is explained as follows:

Input files: 1.txt, 2.txt,…………….,60.txt

Output files: 1.txt, 2.txt,……….,60.txt in the folder named stops\_removed

**Functions in this module:**

**Allfiles(foldername):**

1.finds the name of all the source files available inside the 'source/' folder.

2. Returns the names of all the segment files found.

Return format = array of strings

**Rem\_stop(fname, ofilename):**

1.removes punctuation from a string

Input format = string

Output format = string

**lower(liney):**

1.Makes a string lowercase

Input format = string

Output format = string

**Allremove():**

**6.2. Module 2**

**Name: 2\_alltfidf**

**Module Description:**

In this module we create the corresponding characteristic file for each text file. The characteristic file consists of the weights and likelihood of occurence of each word in the stop\_removed folder. The weights of the words are obtained by term-frequency/inverse document frequency (TF-IDF).

Input files: stop\_removed/1.txt, stop\_removed/2.txt,…….etc

Output files: dest/1.txt, dest/2.txt,……….etc

**Functions in this module:**

**allfiles():**

1. finds the name of all the files available inside the 'stop\_removed/' folder.

Return format = array of strings

**extract\_text(filex):**

1. This function extracts the text from the input filename and returns.

Input format = filename (string)

Output format = string

**find\_count(texts,word):**

1.This function finds the count of the particular word in the text.

Input format = string, string

Output format = integer

**find\_tf(text1,word):**

1. This function finds the term frequency score of a word in a text.

Input format = string, string

Output format = float

**find\_idf(infile,word):**

1. This function finds the inverse document frequency of a word in a document.

Input format = string, string

Output format = float

**find\_tfidf(infile,text1,word):**

1. This function calls find\_tf() and find\_idf() functions for a word and calculates the tfidf by

multiplying the results obtained from both the functions.

Input format = string, string, string

Output format = float

**cluster\_to\_characteristic(infile,outfile):**

1. This function finds the text present in 'infile' through the function extract\_text()

2. It finds all the uniqiue words in the extracted text.

3. It gets the tfidf score of each unique word by the function find\_tfidf() and writes them to

outfile.

Input format = filename, filename

**allconvert():**

1. This function calls allfiles() and finds all the segment files.

2. It then calls clusters\_to\_characteristic() and converts each segment file into a

characteristic file

**6.3. Module 3**

**Name: 3\_distance**

**Module Description:**

This module finds the distance between two files. The distance is measured from the tfidf scores of two files.

Input files: dest/1.txt,dest/2.txt,………..etc

Output files: scores.txt

**Functions in this module:**

**allfiles():**

1. Finds the name of all the segmen files available in the 'segments/' folder.

2. Returns the names of all the segment files found.

Return format = array of strings

**positive(n):**

1. Returns the absolute value of a number

Input format = signed floating point number

Output format = unsigned floating point number

**handle\_one(tot1,p,q,sumweight,sumlikelihood):**

1. calculates and returns the new score of a word which occurs across just one among the

compared characteristic files.

2. The algorithm used is,

score = current weight \* current count / (sum of weights \* sum of likelihood)

Here,

tot1 = current similarity score between two files

p = current weight obtained in the document where the word is found

q = current count obtained in the document where the word is found

sumweight = sum of weights of all the words in the document.

sumlikelihood = sum of likelihood of all the words in the document.

Input format = float, float, float, float, float

Output format = float

**handle\_both(tot2,p,q,r,s,sumweight1,sumlikelihood1,sumweight2,sumlikelihood2):**

1. calculates and returns the new score of a word which is common for both the two characteristic files.

2. Here,

tot2 = current similarity score between two files

p = current weight obtained in the first document where the word is found

q = current count obtained in the first document where the word is found

r = current weight obtained in the second document where the word is found

s = current count obtained in the second document where the word is found

sumweight1 = sum of weights of all the words in the first document.

sumlikelihood1 = sum of likelihood of all the words in the first document.

sumweight2 = sum of weights of all the words in the first document.

sumlikelihood2 = sum of likelihood of all the words in the first document.

3. The score is calculated as

score = p\*r/float(sumweight1\*sumlikelihood1) - q\*s/float(sumweight2\*sumlikelihood2)

Input format = float, float, float, float, float

Output format = float

**sum\_weight\_likelihood(file):**

1. Finds the sum of the weights of all the words in the document

2. Finds the sum of the likelihood of all the words in the document

Input format = file

Return format = float, float

**matchscore(str1,str2):**

1. this function compares two characteristic files. Here str1 and str2 are the filenames of the two files to be compared.

2. it categorized the words found in the two files into three categories, based on which are common in both the files and words which are unique to each of the two files.

3. As score computation algorithm is different for common and unique words, for unique words handle\_one() is called and for common words handle\_both() is called.

4. The sum of scores across all words (i.e. unique and common) is returned.

Input format = filename, filename

Return format = float

**6.4. Module 4**

**Name: 4\_cluster**

**Module Description:**

Functions in this module:

**Findscore(fid1,fid2):**

**Find\_near(fidx,noff,near\_threshold):**

**Find\_diff(array1, array2):**

**Find\_near\_final(listx, noff,near\_threshold):**

**Find\_find(listy,done,noff,near\_threshold):**

**Main\_prog(noff,near\_threshold):**

**6.5. Module 5**

**Name: 5\_result\_evaluation**

**Module Description:**

It performs the cluster measurement using Adjusted Rand Index.

Functions in this module:

**Check\_items\_correct(p,q,arrax):**

**Check\_items\_computed(p,q,corrx):**

**Find\_all(nof,arrax,corrx):**

**6.6. Module 6**

**Name: 6\_main\_module**

**Module Description:**

Functions in this module:

**Findscore(fid1,fid2):**

**Find\_near\_threshold():**

**6.7. Implementation Setup:**

We simulated and ran our algorithm on a system having following specifications:

Environmental Variable Value

Processor Pentium Quad-Core

Operating System 32 Bit Windows 7

RAM 2 G.B.

Language Python 2.5

**CHAPTER 7**

**RESULTS AND ANALYSIS**

**7.1. Results:**

In order to check the results of our algorithm and check the validity of the output we tested our algorithm on various datasets and have obtained successful results.

**7.1.1 Sample Files:**

We have used 60 different data files for obtaining the results. The dataset contains 10 files from actor-actress, 10 files from politics, disasters, culture, sports and technology respectively.

Figure 05 represents one of the file taken from dataset for preprocessing. It contains content of actress named 1.txt.

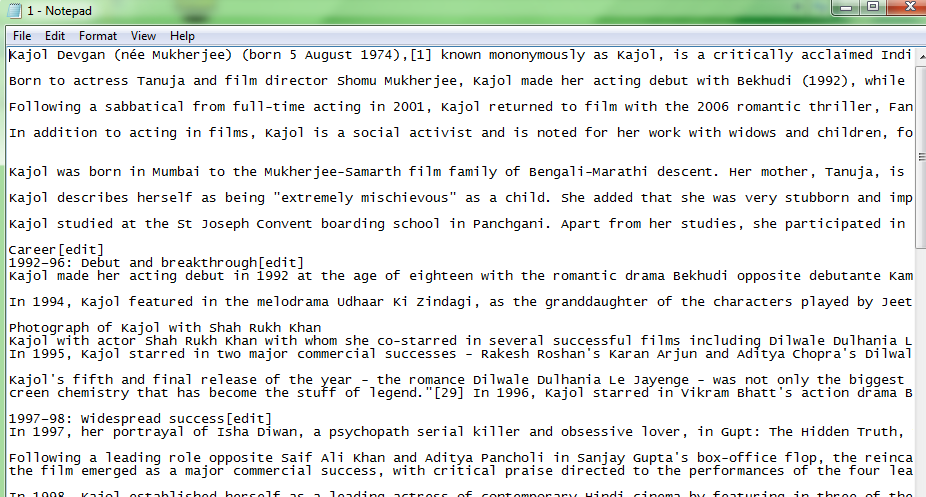


Fig. 05: 1.txt file before preprocessing

Figure 06 represents data of 1.txt file after preprocessing i.e. after removing stopwords, punctuation, numbers, special characters and decapitalization.

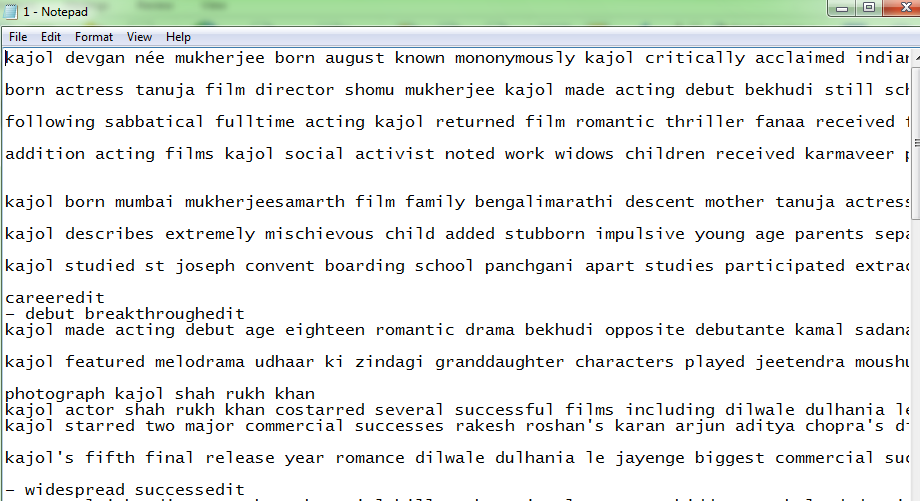


Fig. 06. 1.txt file after preprocessing

Figure 07 represents the tf-idf weight for each term present in the text file named 1.txt.

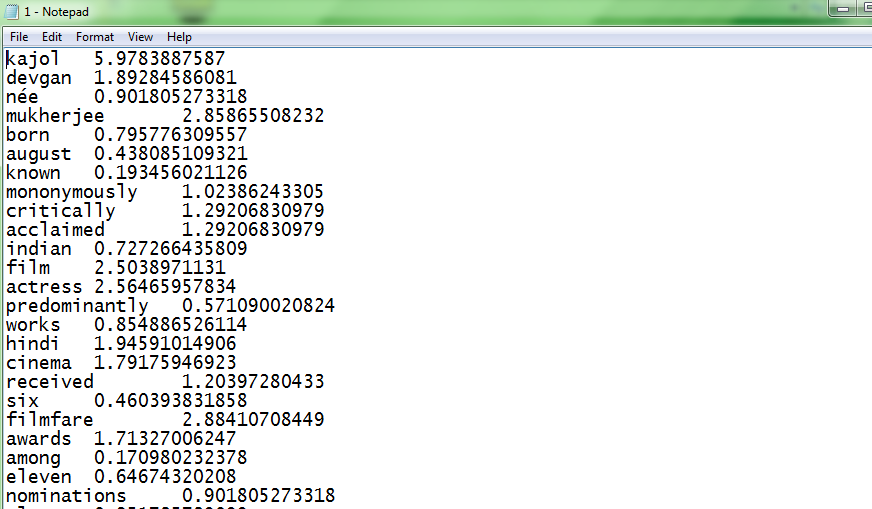


Fig 07. Representing tf-idf weight of 1.txt file

Figure 08 represents the distance of one file to all other files.

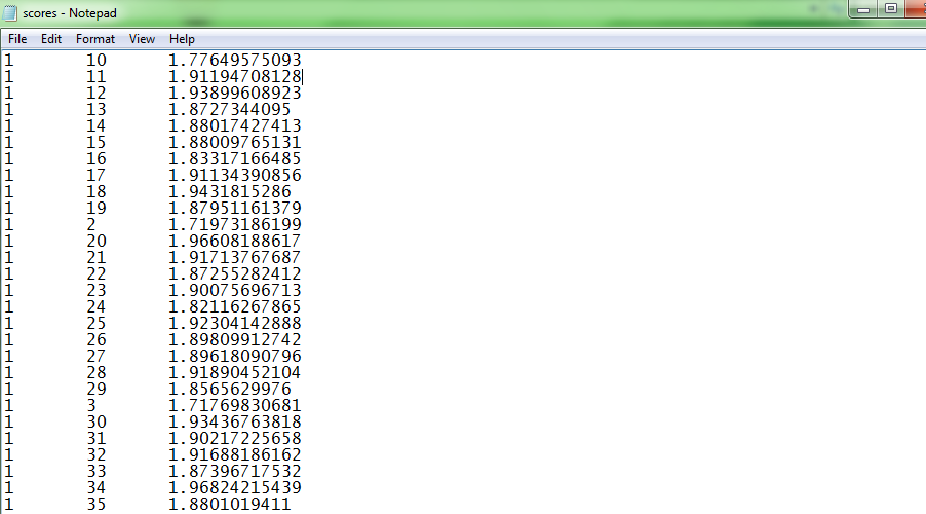


Fig. 08. Calculating scores of one file to all other files.

Figure 09 represents file running in python.exe for formation of clusters and representing the percentage of effectiveness for formation of clusters.

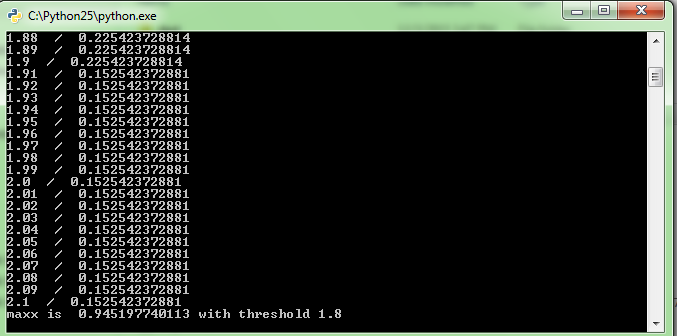


Fig. 09. File running in python.exe for formation of clusters.

Figure 10 represents the final output after formation of clusters.

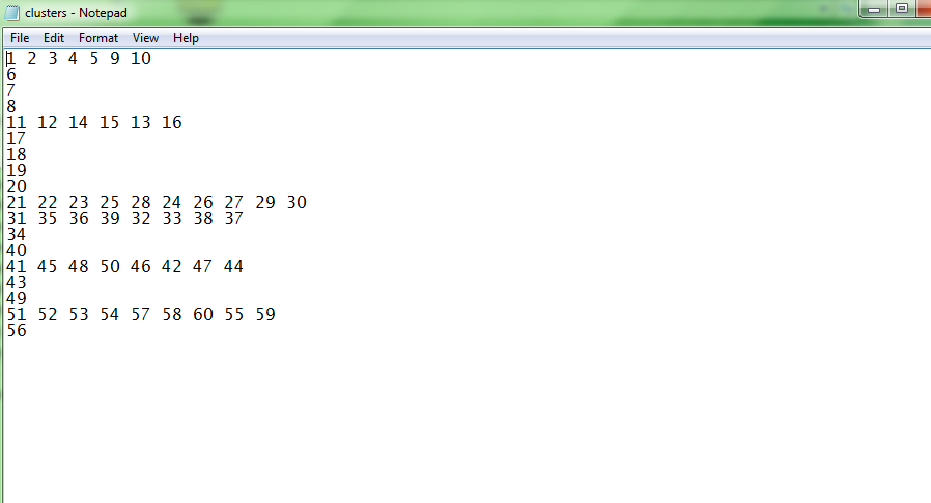


Fig. 10. Formation of clusters of all files.

**CHAPTER 8**

**CONCLUSION**

From the results obtained after implementation of the system and detailed tests on various documents, we come to a conclusion that our system can perform the clustering of all the documents effectively. So, hereby we conclude that the objective of this project has been fulfilled.

**BIBILIOGRAPHY:**

1. Natural Processing Language with Python by O’Reilly, Steven Bird, Ewan Klein & Edward Loper.
2. Term-frequency inverse document frequency, Wikipedia
3. Cluster analysis, Wikipedia
4. DBSCAN, Wikipedia
5. http://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation