CS568 : Data Mining

Review of related work and Code Architecture

Kushal Sangwan (180101096)
Milind Prabhu (180101091)
Varhade Amey Anant (180101087)

Overview of the selected algorithm, variants of the algorithm, related work review, code architecture and class design, implementation details

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

Contents

1 PROBLEM DEFINITION AND OVERVIEW

1.1 Bottleneck issue this clustering problem resolves

This algorithm solves the co-clustering problem for words and documents together. The existing algorithms either cluster the documents based on word distributions or cluster words based on common occurrence in documents. This algorithm simultaneously clusters both words and documents through partitioning the corresponding bipartite graph. This is possible because of the duality between the word clustering and the document clustering i.e, related documents have common words and related words occur in the same document.

1.2 Existing algorithms

Also, existing graph theoretic algorithms for the document-clustering problem usually choose the vertices of the graph to represent documents and the weights on edges between vertices to be some measure of similarity between the documents. These algorithms take quadratic time in terms of the number of documents to construct the similarity graph which is computationally prohibitive. The algorithm proposed in the paper chooses the bipartite graph as the similarity graph between words and documents which can be constructed in linear time with respect to the number of documents.

2 IDEA, MATHEMATICAL INTUITION

2.1 Mathematical Representation

The main idea of the algorithm is to exploit the duality between the document clustering and word clustering problem, i.e, one induces the other and thus better word clustering leads to better document clusters. The "best" for both cases will correspond to finding a partitioning of their bipartite graph with total weight of "cross" edges between partitions being minimum while maintaining the balance between partitions (i.e Normalized cut) for which the following algorithm is proposed.

Considering this, the intuition is to come up with a combined 'cost function' which captures both the aspects:

$$Q(V_1, V_2) = \frac{cut(V_1, V_2)}{weight(V_1)} + \frac{cut(V_1, V_2)}{weight(V_2)}$$

Using, the properties of the Laplacian **L**, the Weight matrix **W** and a generalized partition vector $\boldsymbol{q}$ we discover a equivalence between the Q function and a standard expression from linear algebra

$$\frac{\mathbf{q^T L q}}{\mathbf{q^T W q}} = Q(V_1, V_2)$$

The discreteness condition on q makes this optimization problem NP-hard. Finding the minimum for a real relaxation to expression on the left is what the following algorithm tries to do , but instead of using eigenvalues and eigenvectors, it tries to use the second left and right singular vectors of an appropriately scaled word-document matrix to yield good bipartitionings.

2.2 Algorithm implementation details

The **Bipartitioning algorithm** involves the following three steps. Although the last two steps might seem very obscure they can be justified by some neat mathematics.

(1) **Graph Representation:** Represent the data as a bipartite graph $G(U, V, E)$, the two disjoint sets U,V being the set of documents and the set of words appearing in these documents respectively. An edge between a document and word is assigned a weight which is a measure

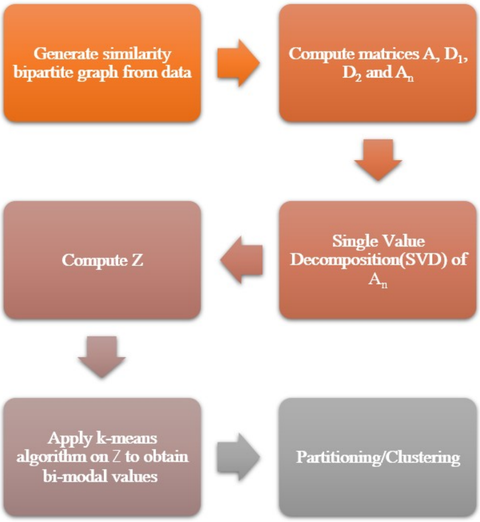

Generate similarity bipartite graph from data
Compute matrices A, D1, D2 and An
Single Value Decomposition(SVD) of An
Compute Z
Apply k-means algorithm on Z to obtain bi-modal values
Partitioning/Clustering

Fig. 1. Overview of the bipartitioning algorithm

of the word's relation to the document (say, frequency of the word in the document). For convenience we let $|U| = w$ (the number of documents) and $|V| = d$ the number of words.

(2) **Singular Value Decomposition (SVD):** Let A be the the word by document matrix. Also, D_1 be a $w \times w$ diagonal matrix such that $D_1(i,i) = \sum_j A_{(i,j)}$ and D_2 be a $d \times d$ diagonal matrix D_2 such that $D_2(j,j) = \sum_i A_{(i,j)}$. The second step of the algorithm involves the computation of the second singular vectors u_2 and v_2 of the normalized matrix A_n where $A_n = D_1^{-1/2} A D_2^{-1/2}$. Here while u clusters the words , v clusters the documents.

(3) **Using k-means**: As the above vectors correspond to a real relaxation to the optimal generalized partition vector, one needs to find two bi-modal values to cluster using u and v. Run k-means on the one dimensional data $[D_1^{-1/2}u_2, D_2^{-1/2}v_2]$ to obtain these values and then finally the clustering.

The **Multipartitioning algorithm** is similar to the Bipartitioning algorithm. In step (2), $l = \lceil \log_2(k) \rceil$ singular vectors of A_n, $u_2, u_3 \cdots u_{l+1}$ and $v_2, v_3 \cdots v_{l+1}$ are computed. The l value corresponds to the fact in the optimal partitioning vector,we can have $2^l = k$ different values corresponding to the k different clusters. To obtain these values for the real relaxation solution obtained from step (2),in step (3) the k means algorithm is executed on the l-dimensional data $\begin{bmatrix} D^{-1/2}U \\ D^{-1/2}V \end{bmatrix}$ where $U = [u_2, u_3 \cdots u_{l+1}]$ and $V = [v_2, v_3 \cdots v_{l+1}]$.

3 EVALUATION OF THE ALGORITHM

The algorithm is evaluated both for its bipartitioning and multipartitioning variants on large as well as small datasets. The paper shows the confusion matrices for the results of these tests and mentions that the goodness was gauged by the computation of the confusion matrix as well as by **purity** and **entropy** calculated from the confusion matrix.

Purity is an external evaluation criterion of cluster quality. It is the percentage of the total number of objects(data points) that were classified correctly. Entropy is another measure of performance of the algorithm. A higher entropy indicates that the cluster outputs by the algorithm in reality contains data points which belong to different labels or classes thereby saying that the algorithm is not doing well.

For the evaluation of the bipartitioning algorithm, the datasets Medline(medical abstracts), Cranfield(Aeronautical abstracts) and Cisi(information retrieval abstracts) were used. The tests were performed by mixing two of the above three datasets. In some cases care was taken to discard stop words and words that occurred either in a large fraction of the documents or in an insignificant number of documents. The authors however point out that the algorithm did well even when stop words were included.

For the multipartitioning algorithm, the evaluation was performed on a combination of the previous three datasets and also on a Yahoo datasets which consisted of news articles 6 categories: Business, Entertainment, Health, Politics, Sports and Technology. The documents in the Yahoo dataset were html pages and were pre-processed to discard HTML tags.

4 EXISTING IMPLEMENTATIONS, DATASETS

Primarily, two types of datasets are used, one for the bipartitioning and one for the multipartitioning varaint of the algorithm each. We were able to find some of the datasets used by the author:

- Bipartitoning Algorithm
  Cranfield, *Medline* and *Cisi*
- Multipartitoning Algorithm
  The above three datasets and the Yahoo dataset [Not Available].

The specific code used by the author was not available however we did find a *python implementation* of the algorithm in the sci-kit learn library

5 REAL WORLD APPLICATIONS

The applications of the document clustering problem are:

- Biomedical applications to classify patient symptoms and medical diagnoses
- A customer relationship management (CRM) application where you want to co-cluster customers and items purchased.
- Movie recommendation engines engines co-cluster accumulated movie ratings from viewers. When a new viewer submits a score for a film she liked, the engine recommends other movies based on classifying the rating she provided to a cluster of audience movie ratings.
- Document Clustering

5.1 Standard Library Implementation

The sci-kit learn python package has an implementation of the spectral co-clustering algorithm and it also cites this paper itself as the primary reference.

sci-kit learn 2.4.1 Special Co-Clustering

6 EXISTING INCREMENTAL ALGORITHMS

To the best of our knowledge, there is no incremental version of the selected algorithm. In case of algorithms for partitioning dynamic bipartite graph, they function on evolutionary data, which means they take into account the historical relationship between the data points into consideration and want smoother transitions in clusters[3], which is not what we want for incremental clustering. The two main steps involved in our algorithm are finding the second eigenvector of the Laplacian and running k-means on it. We therefore thought it might be worthwhile to check if each of these steps can be made incremental to make the overall algorithm incremental.

The step which finds the second eigenvector involves performing the SVD. Therefore we checked to see what work had been done on making SVD incremental. Existing algorithms used for incremental SVD [8] just handle the scenario of addition of data points assuming arrival in chunks while completely ignoring the deletion as well as updation possibility.

Although we did not find incremental algorithms for SVD, we did find some work [5] which discusses how changing the similarity matrix changes the eigenvalue system of the Laplacian. At the end we anyway calculate the second eigenvector of L using right and left singular vectors of A_n (normalized incidence matrix), and the modifications in this vector is only the thing that we require which can be given by the following algorithm. Incremental Spectral Co-Clustering [5] computes changes $\delta\lambda$ and δq in eigenvalues and eigenvectors. It models addition and deletion of data points as similarity changes, and differentiates $Lq = \lambda Dq$ for getting the formulas for changes in eigen spectrum, which are used to change λ and q until both converge.

7 RELATED VARIANTS OF THE ALGORITHM

7.1 Variant 1: Co-clustering documents and words using Bipartite Isoperimetric Graph Partitioning [6]

Details: It also solves the co-clustering problem by partitioning the bipartite graph but formulates normalized cut problem as that of minimizing the isoperimetric ratio for the clusters. The spectral SVD algorithm in general fails to get solutions for some families of problems for example, roach graphs and also, eigenvalues in the bulk of the spectrum are typically harder to approximate with the popularly used lanczos method for SVD. Instead of doing the SVD it solves a system of linear equations to get the two partitions for a bipartitioning problem. It gets the linear equation system by doing three things: using a Cost function and Lagrange multiplier for the optimization problem, differentiating for getting the optimum and thus getting equation $Lz = d$, where d is nothing but vector of outgoing degree of each vertex, then it eliminates one vertex to make L-non singular and solves the formed linear equations. k-means is used on the obtained solution to form the clusters.

For multipartitioning it recursively calls the bipartitioning algorithm until the isoperimetric ratio falls below a certain value. This ICA algorithm is proved to be computationally faster,better in quality and stable then SVD based algorithm based on experimental results.

7.2 Variant 2: Learning A Structured Optimal Bipartite Graph for Co-Clustering [4]

Details: This paper also proposes an algorithm which takes as input a bipartite similarity graph as in the selected paper and outputs k clusters. The main motivation for this approach was that algorithms such as those proposed in [2] first process the original graph and later on apply the k-means algorithm (or some other similar algorithm) to obtain the final clustering. The k-means algorithm is sensitive to the initialization and might possibly make the clustering performance unstable and sub-optimal. The algorithm attempts to find the bipartite graph with k connected components which is closest to the input graph. The new bipartite graph learned maintains an explicit cluster structure, from which clustering results can be obtained without post-processing.

7.3 Variant 3: Efficient semi-supervised Spectral Co-clustering with Constraints [7]

Details: The SCM algorithm proposed here solves the optimization problem of minimizing the normalized cut along along with maximizing the inclusion of the knowledge that some rows or columns are known to be in the same cluster a priori. For example in document co-clustering, it may already be known that papers of conferences KDD and ICDM belong to same group as both conferences are based on data science, while the words' clustering and co-clustering can be taken to be similar. Most co-clustering algorithms do not use this information and hence may produce inaccurate clusterings. Such info is taken into account by modifying the normalized matrix A_n taken in our algorithm to also include the incidence and degree matrix of the constraints. The purity and normalized mutual information obtained for word-document and graph-pattern co-clustering datasets were much higher than our selected algorithm in presence of 5% constraints.

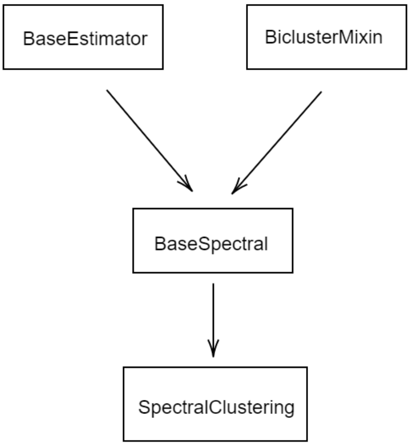


Fig. 2. The class hierarchy of the SpectralClustering class in the scikit-learn library.

8 CODE ARCHITECTURE USED IN THE SCIKIT-LEARN LIBRARY

8.1 Data Structures

8.1.1 Input Data Representation.

1. X: an $m \times n$ matrix representing the bipartite graph where $X[i, j]$ denotes the association of row i with column j.
2. D_1: $m \times m$ diagonal matrix with row sums
3. D_2: $n \times n$ diagonal matrix with column sums
4. A_n: $m \times n$ The normalized matrix

8.1.2 Data generated.

1. u: an $l \times m$ matrix of left singular vectors where $l = \lceil \log_2 k \rceil$
2. v: an $l \times n$ matrix of right singular vectors
3. z: $[D_1^{-1/2}u \quad D_2^{-1/2}v]$ matrix of dimension $l \times (m + n)$

8.2 Class Design

The code architecture is explained through the below classes and their properties and methods (see Figure 2)

1. SpectralCoClustering :
   Properties
   - `n_clusters` The number of clusters. Has value 2 for the bipartitioning algorithm and higher values for multipartitioning.
   - `svd_method` To decide which algorithm out of ’randomized’ and ’arpack’ to use to compute SVD.
   - `n_svd_vecs` Number of vectors used in calculating the SVD .
   - `mini_batch` Boolean specifying whether to use mini-batch k-means or not.
   - `init` Initialization method of k-means algorithm.
   - `n_init` Number of random initialization for k-means

Attributes
- `row_labels_, column_labels_` The results of clusterings are stored in these arrays

Methods
- `__init__` Used for parameter initialization.
- `_fit(self, X)` Computes the SVD of normalized matrix and finds the `row_labels_`, `column_labels_` array values.

(2) BaseSpectral :

Methods
- `_check_parameters` Verifies the parameters used for initialization
- `_svd` Returns the first ‘n_components‘ of left and right singular vectors u and v, discarding the first `n_discard` components
- `_k_means` The k-means algorithm is run and returns the centroid and the labels

(3) BaseEstimator :
- `_validate_data` Used to validate the training data.

(4) BiClusterMixin :
- Mixin class for all bicluster estimators in scikit-learn.

(5) ABCMeta :
- This is a meta class defining the properties of BaseSpectral class. For example, parents may be defined by ABCmeta (just a hyphothetical example) and as BaseSpectral is an instance of ABCmeta, it has the same parents. Basically classes are instances of meta classes.

8.3 Summary

The SpectralCoclustering Class implements the algorithm. First, the `scale_normalize` method obtains the `normalized_data(`A_n`)` , `row_diag(`D_1`)` , `col_diag(`D_2`)` from the incidence matrix `X` . Then the `svd` method takes the arguments `normalized_data`, number of singular vectors ($l = log_2 k$), and vectors to discard which is 1(we start from second singular vector) and returns matrix `u` and `v` containing left and right singular vectors. Then `row_diag*u` and `col_diag*v` are stacked together to obtain the matrix `z`. `k_means` method is run on the matrix `z` to get list the of labels. These labels are used to construct indicator vectors for the clusters and stack them together for both rows and columns. The `k_means` and `svd` method belong to the parent class BaseClustering. In `k_means`, either the mini-batch k-means or batch k-means is executed depending on the given option and similarly in svd, one out of `randomized` and `arpack` is executed.

9 IMPLEMENTATION IN C++

We did not find any ready to use implementation of the algorithm in C++ language. We have implemented the complete algorithm in C++ using the *Armadillo* and MLpackpackage for linear algebra and other standard matrix methods. We have implemented some parts while some are left

9.1 Input Format and Datasets

The inputs to the algorithm are a file containing the word by document matrix and another which contains the true labels. The second file is used only to print the confusion matrix and is not necessary to obtain the clustering.

We have only tested the code on very simple test cases to verify its correctness(see figure **??**). The test files have been provided in the example_code folder. We plan to test this on standard datasets such as the medline,cranfield and cisi datasets available at *Common IR Test Collections*. They were available in Harwell-Boeing compressed format and we have converted them to matrix market format using our converter program.

These datasets are to be further pairwise combined to get matrices for MedCran, MedCisi datasets. These matrices are the ones which are fed to the algorithm. Further for multipartitioning combination of all three is needed.

9.2 Implementation

We have implemented the Spectral Co clustering class,Base Clustering class ,normalize_Data classes of our implementation plan . We are also getting cluster assignments and the corresponding confusion matrix on small inputs.

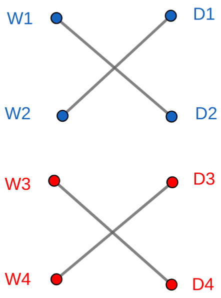
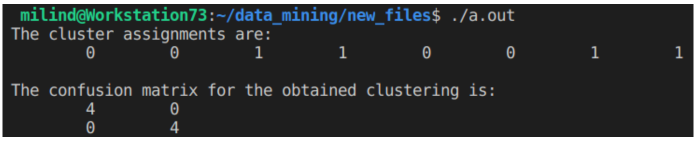


Fig. 3. The sample test case. The algorithm partitions the graph into the red and blue clusters.



```
milind@Workstation73:~/data_mining/new_files$ ./a.out
The cluster assignments are:
         0         0         1         1         0         0         1         1

The confusion matrix for the obtained clustering is:
         4         0
         0         4
```

Fig. 4. The output of the algorithm

9.3 File Structure

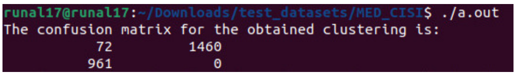
The primary file having implementation of classes mentioned is `kpartition.cpp`. The file used for the conversion of matrix formats is `converter.cpp`. We have created a sample input matrix, which is read from the file `input.txt`.

9.4 Future Work

The MedCisi, MedCran and Classic3 combination datasets have not been created by us yet from the converted matrix market format. We have hence, not completed the testing and evaluation of the algorithm on the datasets used by the author in the publication.

9.5 Results

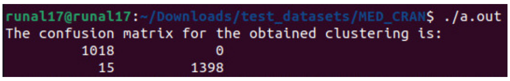
9.5.1 MedCisi. Purity: 97.11%



```
runal17@runal17:~/Downloads/test_datasets/MED_CISI$ ./a.out
The confusion matrix for the obtained clustering is:
          72          1460
         961             0
```

Fig. 5. The output for the MedCisi dataset

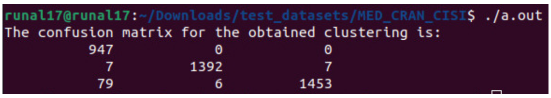
9.5.2 MedCran. Purity: 99.38%



```
runal17@runal17:~/Downloads/test_datasets/MED_CRAN$ ./a.out
The confusion matrix for the obtained clustering is:
        1018             0
          15          1398
```

Fig. 6. The output for the MedCran dataset

9.5.3 MedCisiCran. Purity: 97.45%



```
runal17@runal17:~/Downloads/test_datasets/MED_CRAN_CISI$ ./a.out
The confusion matrix for the obtained clustering is:
         947             0             0
           7          1392             7
          79             6          1453
```

Fig. 7. The output for the Classic3[MedCisiCran] dataset

10 CODE WALKTHROUGH

For explanation of the code, the output and the intermediate results were obtained from the following toy dataset

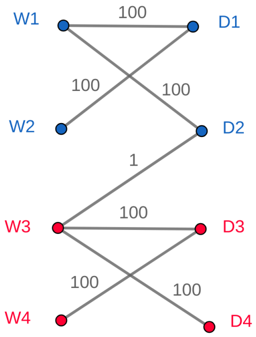


Fig. 8. Input figure

Very small weight was given to edge w3-d2 as compare to others in the above graph/dataset to make two clusters - one having w1,w2,d1,d2 of the upper half and another having w3,d3,w4,d4 of the lower half. For code walkthrough , the following four intermediate files generated from above dataset are used along with explanation of how the code generates and uses their contents.

10.1 Files

10.2 Input files

The `input.in` file contains the dimensions (4×4), number of clusters(2) and the word by document matrix of our graph. And `label.in` (1×4) contains the true labels for documents.After the starting of the main function, the lines 175-183 read these matrix in coordinate format and lines 195-199 read the labels.in file.

10.3 Intermediate Matrices File

After reading ,at line 186 the spectral co clustering object x is created by passing in dimensions and matrix itself as parameters. The fit function of this object called on line 188 basically applies SVD and k means , for which several intermediate matrices are produced.

Inside this fit function, as we can see on lines 96-102 three matrices are produced d1(diagonal matrix of A's row sums),d2(diagonal matrix of A's column sums) by calling scale normalize method and further a_n(normalized matrix) is calculated from them.

The scale normalize method(line 28) makes each diagonal entry of D1 equal to inverse of square root of the row sums and for D2 its just takes column sums instead.

In the intermediate_matrices.txt file we can see that the $(1, 1)$ entry of D1 is 0.0707. This can be explained as the row sum of A which is $100 + 100 + 0 + 0 = 200$. hence $\frac{1}{\sqrt{200}} = \frac{1}{10\sqrt{2}} = \frac{0.707}{10} = 0.0707$. Similarly the $(1, 1)$ entry of D2 is also 0.0707 which can be deducted from the column sum of A which is also $100 + 100 + 0 + 0 = 200$.

As for An , it is clearly apparent that the edges that exist in A are the only ones that appear in An. The normalization is carried out so as to factor in the degree sums division of cut included in a normalized cut.For example entry $(1, 1)$ in A is 0.500 which is also non zero i.e 100 in A while $(1, 3)$ is zero in both A and An.

10.4 SVD output file

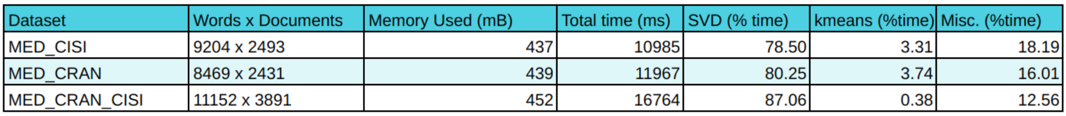
Now further $l = \log 2 = 1$ singular value,left singular vector and right singular vector are calculated in lines 115- 121. The SVD method here(line 76-81) of the base clustering class calls svds library function to perform truncated SVD.

The singular values matrix S in `SVD_output.txt` file contains the singular values clearly in decreasing order {1.0000,0.9967,0.5007,0.4972}. As the first singular value corresponds to the full graph and not clusters it is discarded , and hence second column vectors from U and V corresponding to value 0.9967 are taken for z calculation as reflected in u1 and u2 calculation on lines 126 and 129(they are only taking vector number 1(0 indexed) from U and V). Before k means is carried out on this 1 dimensional data z, we can see in the file that entries corresponding to w1,w2 $(0.0407, 0.0412)$ and d1,d2 $(0.0411, 0.0400)$ are nearly same hence they will be in one cluster. And also w3,w4 $(-0.0405, -0.0411)$ and d3,d4 $(-0.0409, -0.0407)$, all four of which are having close values will clustered into other one. On line 139 , k-means is performed on this z and results are stored in assignments vector . We return these assignments from our class on line 189 by the x.get_assignments() call .

10.5 Confusion Matrix Output file

Lines 202-204 create the confusion matrix from the true labels and the labels obtained by algorithm. Matrix in output.in has $(1, 1)$ entry 2 as both documents d1,d2 belong to one cluster and are also predicted be in one by the code. Similarly documents d3,d4 assignments are reflected in $(2, 2)$ entry which is also 2. No false positive or false negative make $(1, 2)$ and $(2, 1)$ zero.

10.6 Memory and Time Used Across Datasets



Dataset	Words x Documents	Memory Used (mB)	Total time (ms)	SVD (% time)	kmeans (%time)	Misc. (%time)
MED_CISI	9204 x 2493	437	10985	78.50	3.31	18.19
MED_CRAN	8469 x 2431	439	11967	80.25	3.74	16.01
MED_CRAN_CISI	11152 x 3891	452	16764	87.06	0.38	12.56

Fig. 9. Results for various datasets.

The first implementation of the algorithm has been modified to improve its space and time utilization. Specifically, the following major improvements have been made:

- The input has been changed from a 2D matrix to a sparse matrix in the co-ordinate format.
- Sparse matrices have been used throughout the entire algorithm.
- For the singular value decomposition only the Truncated SVD method is used since we require only the first view singular vectors for the most part.

As can be seen from figure 11, the slowest step of the algorithm is performing the SVD. The time taken for the k-means step is significantly smaller. A large fraction of the time categorized as miscellaneous goes into creating the input sparse matrices.

10.7 Incremental Version Plan

Performing SVD takes almost all of the execution time of the algorithm(as high as 87 percent MED_CRAN_CISI). Hence for minor changes in the word by document matrix A, focusing on calculating new singular vectors (U and V) and singular values(σ) from the initial ones can make the algorithm incremental. For this purpose LWI-SVD method [1] can be used. It reduces the SVD time by performing the decomposition of a $k \times k$ matrix instead of the $m \times n$ dimension matrix A (the new dataset). This problem reduction is achieved by the QR decomposition of some intermediate matrices which removes redundant calculations. It can save upto 50 percent of execution time as compared to a full SVD along with having a 2% inaccuracy on 300×300 matrices. .

11 PROPOSED INCREMENTAL ALGORITHM

11.1 Static and Incremental versions of Algorithm

```
Algorithm 1: Static Algorithm
  Result: Bipartitioning/Multipartitoning the Bipartite Graph
  Input: The word by document matrix $X$ and k(the number of clusters)
  Output: Assignment vector for the vertices of the graph
  /* Input data preparation                                            */
  $D1{=}(rowSum(X))^{-\frac{1}{2}}$;
  $D2{=}(colSum(X))^{-\frac{1}{2}}$;
  $X_n{=}D1XD2$;
  /* Computation                                                       */
  Compute $U$ and $V$ through Truncated SVD of $X_n$
  Using the left and right singular matrices generated $U = [u_1 u_2 u_3 \ldots u_k]$, $V =$
    $[v_1, v_2, v_3, \ldots, v_k]$ where k is the number of clusters
  $Z$ is the matrix of Laplacian $L$'s eigenvectors calculated as below
      $$Z = \begin{bmatrix} D_1 U \\ D_2 V \end{bmatrix}$$
  /* Post Processing                                                   */
  K-means clustering on $Z$ for bipartitioning to get cluster assignments for the vertices(words,
    documents) of the graph;
```

Algorithm 2: Incremental Algorithm 1

```
Result: Bipartitioning/Multipartitoning the Bipartite Graph
Input: The original word by document matrix $X$, the $\Delta$ matrix containing updates in data
       relative to $X$, Rank r which specifies the degree of approximation, higher the r value
       better the results
Output: Assignment vector for the vertices of the graph
/* Input data preparation                                                          */
Set $\Delta$ =AB$^T$ where $A = I$ and $\Delta = B^T$
Calculate $D_1 = (rowSum(X))^{-\frac{1}{2}}$ and $D_2 = (colSum(X))^{-\frac{1}{2}}$;
Compute Normalized matrix $X_n$=$D1XD2$;
/* Notation Original SVD components as inputs $U$,$V$ and $S$ Let new SVD components to be
   calculated be $U'$,$V'$ and $S'$                                                */
/* Computation                                                                     */

QR decomposition of $(I - V_x V_x^T)$B to get $Q_b$ and $R_b$;
Set $R_a$= $[0 \ I_{n'-n}]$ for the QR decomposition $(I - U_x U_x^T)A$ and
    $$Q_a = \begin{bmatrix} 0 \\ I_{n'-n} \end{bmatrix}$$
Quasi Gram-Schmidt method for pivoted column QR Decomposition
Compute matrix $K$
Low-rank(Rank $k$) decomposition of $K$ to get $U_k$ $V_k$ and $S_K^T$ through normal SVD of the
  matrix $W(k \times k)$
Combining the terms obtained above to get $U'x$, $V'x$ and $S'x$ to get k rank decomposition of
  $X'$
/* Post processing                                                                 */
Using the left and right singular matrices calculated above $U'x$ = $[u_1 u_2 u_3 \dots u_k]$, $V'x$=
  $[v_1, v_2, v_3, \dots, v_k]$ where k is the number of clusters
    $$Z = \begin{bmatrix} D_1 U \\ D_2 V \end{bmatrix}$$
K-means clustering on $Z$ for bipartitioning to get cluster assignments for the vertices(words,
  documents) of the graph;
```

11.2 Workflow Diagram

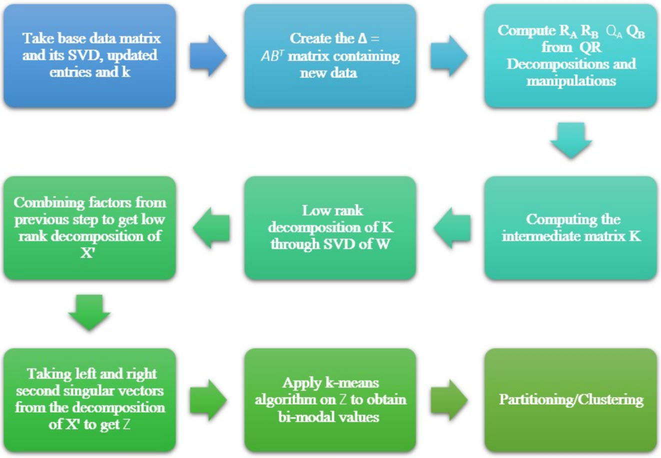

Take base data matrix and its SVD, updated entries and k
Create the $\Delta = AB^T$ matrix containing new data
Compute R_A R_B Q_A Q_B from QR Decompositions and manipulations
Computing the intermediate matrix K
Low rank decomposition of K through SVD of W
Combining factors from previous step to get low rank decomposition of X'
Taking left and right second singular vectors from the decomposition of X' to get Z
Apply k-means algorithm on Z to obtain bi-modal values
Partitioning/Clustering

Fig. 10. Overview of the bipartitioning algorithm

11.3 Intuition

The intuition being the changes suggested is as below

(1) By taking more number of eigenvectors for co clustering (in multiples of k) instead of $\log(k)$, we can ensure that we are providing k means to have high dimensional data for grouping leading to increase in accuracy. The corresponding decrease in time gain will be less due to restarts being required rarely in incremental case. This increase in accuracy can offset the error occurring due to incremental SVD approximations

(2) The LWI-SVD method [1] used for incremental update reduces the size of the matrix to be decomposed by two layered QR decompositions. Instead of $m' \times n'$ matrix SVD, this allows us to just do SVD of much smaller $k \times k$ matrix. After the first layer QR decomposition, the K matrix we are left with has an arrow like structure with density concentrated at new rows/columns. For the second layer pivoted QR decomposition, this allows drawing samples of size K both from initial first columns, which tells us about cluster structure, as well as last few columns which is the new data.The approximation for this matrix allows us to tell the affect of the changes/insertions on our initial K clusters.

(3) The centroids for k-means can be initialized to the previous centroids to which k-means converged earlier. As the change in structure of the data is not significant mostly, these serve as good starting points for a faster convergence

12 INCREMENTAL IDEA 2

12.1 Algorithm Description

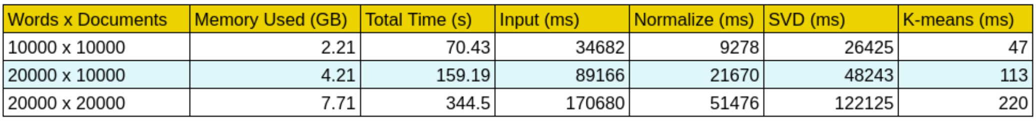
Intuition:The main intuition is that the insertion or deletion of node in a large graph, generally speaking, causes only local changes to the structure of clusters. Therefore, just checking a neighbourhood of the inserted or deleted vertex is enough to approximate the new cluster structure of the graph.

Algorithm 3: Incremental Bipartitioning Algorithm

```
Input: The original word by document matrix $X$, sequence of updates (insertion/deletion of
       documents and words) in batches $B_1, B_2 \cdots B_m$
Output: Two Clusters for each update $(C_1^1, C_2^1)$, $(C_1^2, C_2^2)$ ... $(C_1^m, C_2^m)$
Run the spectral co-clustering algorithm on $X$ to get the initial clusters $C_1^0$ and $C_2^0$
For $i = 1$ to $m$:
/* Here $B_i$ is a set of vertices(docs or words) that will be added or deleted in ith update */
    • Let $G'$ be the induced graph on the $d$-neighbourhood, $D_i$, of $B_i$ .
    • For each vertex $u$ in $D$:
          $G'$.add(BFS(G,u,d))
      /* G' is obtained by performing a BFS from all vertices in $B_i$ upto depth d.            */
    • Calculate connected components $(G^1, G^2, G^3 \cdots G^l)$ of the subgraph D.
    • Let X[1 2 . . . l] be an array of word document matrices
    • For $j = 1$ to $l$:
          $X[j]$=compute_word_document_matrix($G^j$).
          $(C_1[j], C_2[j])$=spectral_co_clustering($X[j]$,2)
          merge( $(C_1[j]$,$C_2[j])$ , $(C_1^{i-1}$,$C_2^{i-1})$ )
          /* Merge in a way that minimizes the normalized cut of the graph.                    */
    • For every vertex $u$ in $D_i$:
          try_switching(u,p,q)
          /* switch only if changing label of $u$ from p to q decreases normalized cut         */
    • $C_1^i$ and $C_2^i$ are the two new clusters.
```

12.2 Memory and Time usage for Original Algorithm

The memory and time taken by the algorithm on synthetic datasets of various sizes are documented as below for the static version of the algorithm



Words x Documents	Memory Used (GB)	Total Time (s)	Input (ms)	Normalize (ms)	SVD (ms)	K-means (ms)
10000 x 10000	2.21	70.43	34682	9278	26425	47
20000 x 10000	4.21	159.19	89166	21670	48243	113
20000 x 20000	7.71	344.5	170680	51476	122125	220

Fig. 11. Results for synthetic datasets

The memory used and SVD time are almost linearly proportional to the matrix size. Its clear that larger matrices than $20k \times 20k$, say for example $25k \times 25k$, are going to take more than 8GB which is larger that RAM size and can't be given as input to the algorithm.
On the other hand, the running time for the algorithm isn't a bottleneck, as for graphs with as many as $40k$ nodes and 6 million edges the running time is just 344 seconds i.e almost 6 minutes.

12.3 WorkFlow

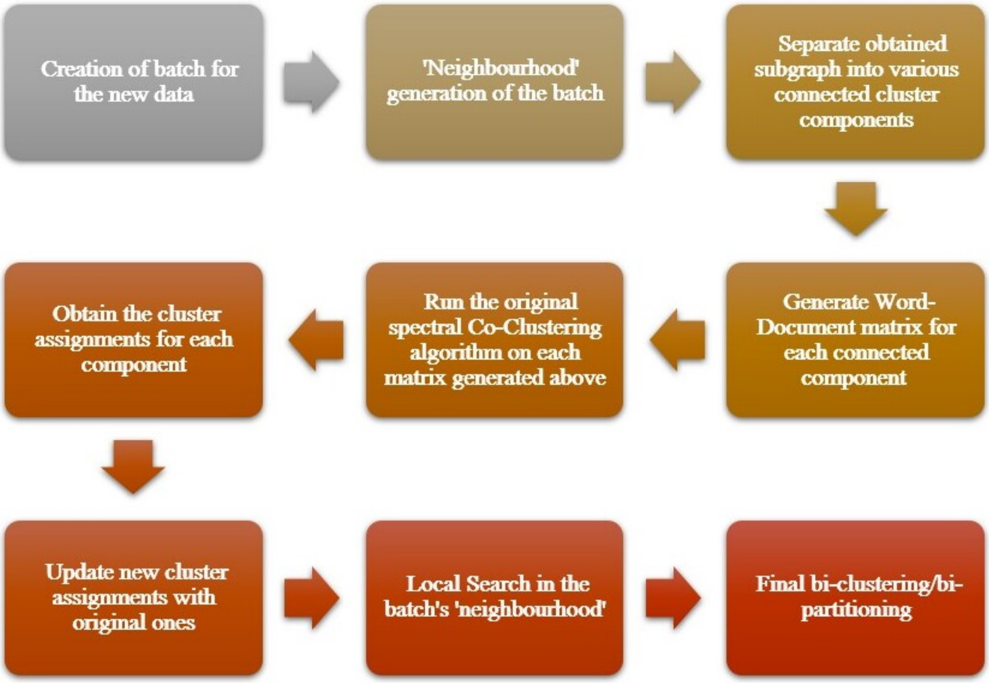

Creation of batch for the new data
'Neighbourhood' generation of the batch
Separate obtained subgraph into various connected cluster components
Generate Word-Document matrix for each connected component
Run the original spectral Co-Clustering algorithm on each matrix generated above
Obtain the cluster assignments for each component
Update new cluster assignments with original ones
Local Search in the batch's 'neighbourhood'
Final bi-clustering/bi-partitioning

Fig. 12. Overview of the Bipartitioning algorithm

12.4 Code Architecture

12.4.1 Functions. :

(1) **normalizedCut**
- **Input:** G, Assigned cluster labels
- **Output:** The value of cut, the weight of the two clusters and the value of the normalized cut.
- **Description:** The function calculates the value of the normalized cut by finding the weights of the clusters and size of the cut.

(2) **switchClusters**
- **Input:** A vertex v, the cut value, the weights of the clusters
- **Output:** True if switching its label decreases the objective function and false otherwise.

(3) **getNeighbourhood**
- **Input:** G, the new batch B_i, the distance parameter d and the edge-weight threshold t
- **Output:** a d-neighbourhood of the set batch B_i with threshold t,i.e, edges have weights larger than t they are not included in the neighbourhood.

(4) **findWordByDocumentMatrix**
- **Input:** Set of words and documents
- **Output:** Word by document matrix corresponding to the input words and documents

(5) **getConnectedComponents**
- **Input:** A graph G'

- **Output:** The connected components of G'.

(6) **clusterMerge**
- **Input:** The two original clusters C_1 and C_2, the two new clusters D_1 and D_2
- **Output:** One of the clusters out of $(C_1 \cup D_1, C_2 \cup D_2)$ and $(C_1 D_2, C_2 \cup D_1)$ that minimizes the normalized cut value.

12.4.2 Insertion. The insertions happens in batches of size B. The input is taken in the co-ordinate format i.e the new edges created and corresponding weights. We then get the d-neighbourhood of the this batch using getNeighbourhood function. Next we separate this subgraph into connected components using getConnectedComponents. Now we convert these components to word-document matrix form using findWordByDocumentMatrix function. Following this, we use the spectral co-clustering algorithm to bipartition every connected component . The clusters found are then merged with the existing clusters using clusterMerge. A local search is performed to further improve the objective function by iterating over all words and documents in the subgraph(i.e the points which were newly assigned or whose assignments were changed) , and changing their cluster label using switchClusters. This step is necessary to handle misclassifications which happen to maintain balanced clusters during bipartitioning of components of the subgraph. In the special case when one of the graphs has edges with only one original cluster , there is no need to partition and the points in that graph are merged in that cluster only.

12.4.3 Deletion. Deletion is done in batches as well. The approach will be analogous to the one used for insertion. The neighbourhood (which does not contain the vertex itself) of each of the vertices to be deleted is found using same getNeighbourhood function but the vertex itself is then deleted from the output.Rest of the implementation will be same as insertion case.

13 EXPERIMENTAL RESULTS

13.1 Datasets Used

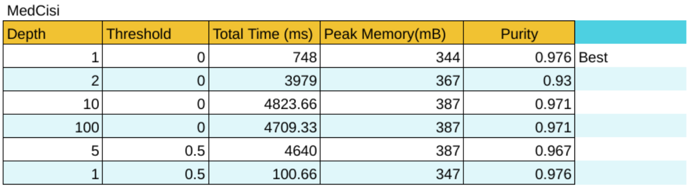
The MedCran and MedCisi datasets were used for experiments. All words and most documents(say $d-300$ where d is number of documents and can be about 2000) were taken as the initial dataset and then three batches of documents of size x (say 100 each) were inserted further. Synthetic dataset of 200×200 size and 3 batches was also used for testing

13.2 Metrics used

To evaluate the results and performance of the incremental version of the algorithm with the static version, we have considered the following metrics and tabulated the results by varying the hyperparameters as below in fig 13, 14 and 15.

(1) Time
(2) Memory
(3) Purity/Accuracy

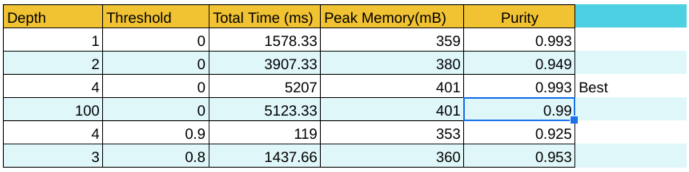
Comparison between incremental and non-incremental clustering: As can be clearly seen from Fig.16, the execution time used of incremental algorithm is more than 70 percent less than that of the static algorithm with comparable accuracy results, specially for larger datasets. The memory consumed by the incremental algorithm is however 25 percent greater than used by the static one.



MedCisi

Depth	Threshold	Total Time (ms)	Peak Memory(mB)	Purity	
1	0	748	344	0.976	Best
2	0	3979	367	0.93	
10	0	4823.66	387	0.971	
100	0	4709.33	387	0.971	
5	0.5	4640	387	0.967	
1	0.5	100.66	347	0.976	

Fig. 13. MedCisi Results

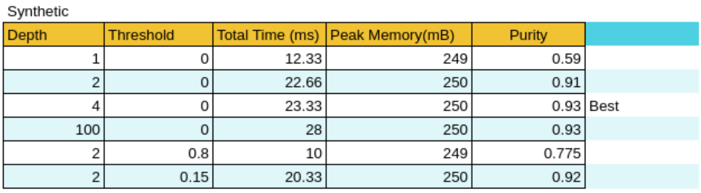


Depth	Threshold	Total Time (ms)	Peak Memory(mB)	Purity	
1	0	1578.33	359	0.993	
2	0	3907.33	380	0.949	
4	0	5207	401	0.993	Best
100	0	5123.33	401	0.99	
4	0.9	119	353	0.925	
3	0.8	1437.66	360	0.953	

Fig. 14. MedCran Results

13.3 Contribution of various steps to the result

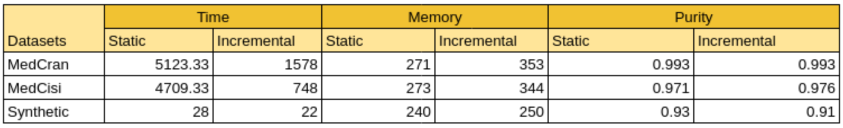
(1) **Effect of depth** The memory as well as running time is grows with increasing depth as size of the neighbourhood subgraph size becomes larger. Applying the original algorithm on a larger neighbourhood requires more time due to higher number of vertices and edges. The relation of accuracy with depth can't be reliably observed as the graph is quite dense and accuracy seems to be fluctuating with depth instead of behaving with a pattern.



Synthetic

Depth	Threshold	Total Time (ms)	Peak Memory(mB)	Purity	
1	0	12.33	249	0.59	
2	0	22.66	250	0.91	
4	0	23.33	250	0.93	Best
100	0	28	250	0.93	
2	0.8	10	249	0.775	
2	0.15	20.33	250	0.92	

Fig. 15. Synthetic Dataset Results



	Time		Memory		Purity	
Datasets	Static	Incremental	Static	Incremental	Static	Incremental
MedCran	5123.33	1578	271	353	0.993	0.993
MedCisi	4709.33	748	273	344	0.971	0.976
Synthetic	28	22	240	250	0.93	0.91

Fig. 16. Comparsion betweeen Static and Incremental versions of algorithm

(2) **Effect of threshold** From the results of our experiments, the threshold seems to control the size of the neighbourhood graph significantly. Larger thresholds result in smaller neighbourhood graphs and hence quicker execution of the algorithm and less memory . But this results in dip of accuracy which can be seen from the tables above.

(3) **Use of connected components function:** Due to lower diameters of the graphs, the neighbourhoods are connected, and as a result only one component is obtained in the neighbourhood subgraph.

13.4 Further plans

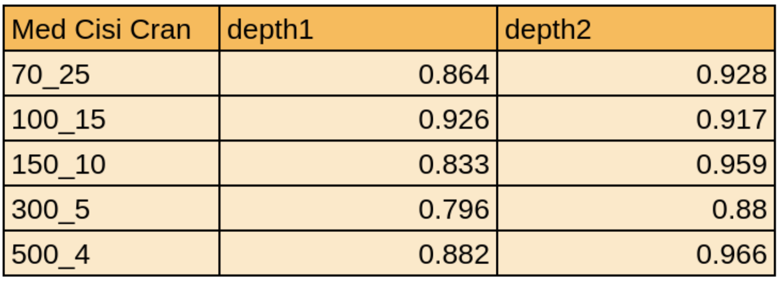
- **Testing with more datasets** The datasets with which we have tested have diameter value of less than 5. For observing the behaviour of incremental algorithm with the depth parameter, datasets of larger diameters, or ones with large number of documents are required. Also the performance of algorithm depends on batch size as well. We plan to use the following two:
  (1) Synthetic datasets with higher depth values and varying batch sizes.
  (2) Reuters-21578
- **Local Search:** The current version of the algorithm does not perform the local search to improve the value of the normalized cut. We plan to include some heuristics that will hopefully improve the clustering.
- **Optimizing the code:** There is a lot of redundancy in the storage of data structures in the code and re-computations of subgraphs as well as the normalized cut values, which if removed, can save a lot of time and memory.
- **Allowing deletion of documents/words:** The code for deletions is yet to be added.
- **Allowing more than 2 clusters** The algorithm can be extended for the multiparitioning case. The number of clusters maintained in the clustering class of the code can be made equal to an arbitrary k by maintaining an array, instead of taking $k = 2$ and having only two sets. Similarly for merging, the clusters can be merged with one of the k clusters which minimizes the normalized cut value.
- **Modifying the neighbourhood** The number of vertices in the neighbourhood increases exponentially with the depth. Therefore, it might help to take the x nearest vertices instead of taking all the neighbours at a particular depth.

14 EXPERIMENTAL RESULTS 2

We have performed three tasks during this week.
1.Generalizing code for multipartitioning case
2.Added code for deletion
3.Performed experiments with varying batch sizes and also with k larger than 2

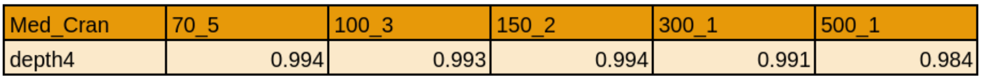
14.1 Multipartitioning of the Graph/Multiple Clusters

As discussed before, we have generalized the concept of running static algorithm on the neighbourhood and merging of algorithm for the multipartitioning case. The purity results obtained for the Med_Cisi_Cran, Synthetic datasets(with k>2) are tabulated below



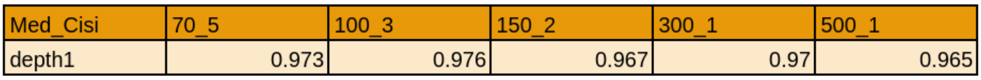
Med Cisi Cran	depth1	depth2
70_25	0.864	0.928
100_15	0.926	0.917
150_10	0.833	0.959
300_5	0.796	0.88
500_4	0.882	0.966

Fig. 17. MedCisiCran Multipartitioning Results



Med_Cran	70_5	100_3	150_2	300_1	500_1
depth4	0.994	0.993	0.994	0.991	0.984

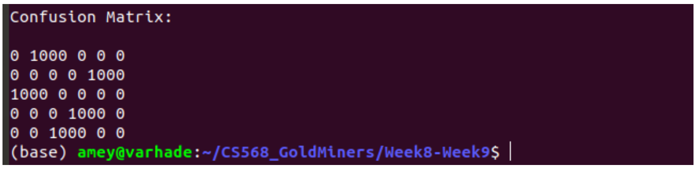
Fig. 18. MedCran Multipartitioning Results



Med_Cisi	70_5	100_3	150_2	300_1	500_1
depth1	0.973	0.976	0.967	0.97	0.965

Fig. 19. MedCisi Multipartitioning Results

The confusion matrices for a synthetic dataset are shown below. Here we have 5000 words and 500 documents, where 400 documents are fed initially and 1000 are added incrementally in batches of size 250 and 500. The depth parameter is set to unity.

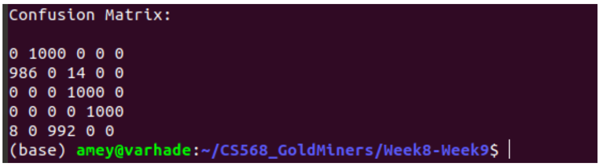


```
Confusion Matrix:

0 1000 0 0 0
0 0 0 0 1000
1000 0 0 0 0
0 0 0 1000 0
0 0 1000 0 0
(base) amey@varhade:~/CS568_GoldMiners/Week8-Week9$ 
```

Fig. 20. Synthetic(Batch size 250) Results

The purity values are 1 and 0.9956 respectively



```
Confusion Matrix:

0 1000 0 0 0
986 0 14 0 0
0 0 0 1000 0
0 0 0 0 1000
8 0 992 0 0
(base) amey@varhade:~/CS568_GoldMiners/Week8-Week9$ |
```

Fig. 21. Synthetic(Batch size 500) Results

14.2 Handling Deletions

The code for deletions has been added. The algorithm used is similar to insertion with the vertices to be deleted taken in batches, they are removed from the graph, their neighbourhood is obtained and then clustering is performed on the sub-graph. One obstacle we encountered in the testing of deletion was that for word-document co-clustering, deletion of words doesn't change the labels of other documents . Thus existing text datasets are not suitable for testing this. One way to overcome this is to use customer segmentation or similar datasets. Because of analogy of customer-items with document-words, co-clustering can be performed on these datasets. Further the deletion of some customers does change the structure of the segments and thus may change the labels too. The other way is to ignore the deletion case, reason being that the static algorithm itself mentions only application to document-word co clustering , and it is natural for the incremental one to go with it too.

14.3 Batch Size Variation

Varying the batch sizes to include 70, 100,150, 300, 500 documents were taken for MED_CISI, MED_CRAN and MED_CISI_CRAN datasets. What was seen was that lower batch sizes lead to more of the documents being clustered during the initial static run but also result in smaller neighbourhoods and thus misclassification of newly inserted points. Thus a pattern of increase and decrease is seen across increasing batch sizes.

14.4 Relation of Depth and Accuracy

As pointed out in our last discussion, the accuracy should increase with increasing depth but that was not observed .For example, in MED_CISI , the accuracy for depth 1 was 0.976 while for depth 2 was 0.93 . One of the reasons for this is that for odd depths we take some subset of documents out of the whole dataset and their corresponding words thus forming a sub-problem , but for even depths all words for the documents located at the maximum depth are not taken leading to their misclassification .

14.5 Further plans

We plan to go ahead with finalizing our report in the next week.

REFERENCES

[1] X. Chen and K. S. Candan. Lwi-svd: Low-rank, windowed, incremental singular value decompositions on time-evolving data sets. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 987–996, New York, NY, USA, 2014. Association for Computing Machinery.

[2] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274, 2001.

[3] N. Green, M. Rege, X. Liu, and R. Bailey. Evolutionary spectral co-clustering. In *The 2011 international joint conference on neural networks*, pages 1074–1081. IEEE, 2011.

[4] F. Nie, X. Wang, C. Deng, and H. Huang. Learning a structured optimal bipartite graph for co-clustering. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4132–4141, 2017.

[5] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127, 2010.

[6] M. Rege, M. Dong, and F. Fotouhi. Co-clustering documents and words using bipartite isoperimetric graph partitioning. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 532–541. IEEE, 2006.

[7] X. Shi, W. Fan, and P. S. Yu. Efficient semi-supervised spectral co-clustering with constraints. In *2010 IEEE International Conference on Data Mining*, pages 1043–1048, 2010.

[8] X. Zhou, J. He, G. Huang, and Y. Zhang. Svd-based incremental approaches for recommender systems. *Journal of Computer and System Sciences*, 81(4):717–733, 2015.