Dimensionality Reduction, Unsupervised learning and query processing over time series data.

**Group members:**

Aneesh Shastry [ 1207047955]

George Thomas [1207073656]

Akshay Chitneni [1207050490]

Anil Kuncham [1206342510]

Karthik Chandrasekar [1206293825]

**Abstract:**

Dimensionality  curse is significant problem while dealing with high dimensional data. At very high dimensions data become very sparse and hence its loses its statistical significance. Indexing and performing search operations with such high dimension data is not very effective and many of information retrieval, data mining, machine learning algorithms do not perform well with high dimensional data. Hence there is a need for dimensional reduction techniques to find out the inherent dimensions of the data and to cut down less significant ones.

In this project we employ several dimensionality reduction techniques like SVD, LDA, FastMap to find out the significant top k dimensions and perform our indexing and search operations over the reduced space. We also use several similarity measures like Dynamic Time Warping, Dot product similarity, word-word similarity to order the top k relevant documents for the given query document.

Heat maps are used to visualize the query file and its similar files when using different similarity measures.

**A set of keywords:**

Dimensionality reduction, SVD(Singular Value Decomposition), LDA(Latent Dirichlet Allocation), Query processing, Time series, Fast Map, DTW(Dynamic Time Warping), Similarity, Dot product, Distances, Time series similarity, Time series search, Latent epidemic analysis, Latent epidemic search, Heatmap, Unsupervised learning.

**Introduction:**

**Dynamic Time Warping:**

Dynamic Time Warping is used to test cases where data could be out of phase. Here the distance are computed in a matrix and the minimum distance path from 0,0 to n,n of the matrix is used for computation of similarity. Here n is the number of entries in the file given as input.

Heat map gives a visualization of the simulation file. The x-axis of the heat map is the time and the y axis is the magnitude of the simulation vector in different cases.

**Singular Value Decomposition**

* It is a factorization of real or complex matrix which has many applications in signal processing and statistics[3].
* Let us consider a matrix of dimension m \* n. It may be factorizable of the form M=UEV\*.
* U is a m \* m real or complex unitary matrix.
* E is a m \* n is a rectangular diagonal matrix with non-negative real numbers.
* V\* is a  n \* n real or complex unitary matrix.
* The diagonal entry of Ei,i  are called singular values  of M
* The m columns of U and n columns of V are called the left singular vectors and right singular vectors of M respectively.
* This process can be compared to eigen decomposition process.
* The left-singular vectors of M are eigenvectors of MM\*.
* The right-singular vectors of M are eigenvectors of M\*M
* The non-zero singular values of M are the square roots of the non-zero eigenvalues of both M\*M and MM\*

**Applications of Singular Value of Decomposition:**

* Image compression
* Empirical Orthogonal Functions
* Web searching
* Pseudoinverse
* Total least squares minimization
* Low-rank matrix approximation
* Separable models

**Latent Dirichlet Allocation (LDA):**

* Latent Dirichlet Allocation(LDA)[2] , a generative probabilistic model for collections of discrete data such as text documents.
* LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as finite mixture over an underlying set of topics.
* In the context of text modeling, the topic probabilities provide an explicit representation of a document.
* Generally this method uses efficient approximate inference techniques based on variation methods and an EM algorithm for empirical Bayes parameter estimation.
* This is similar to probabilistic latent semantic analysis(pLSA) except that in LDA the topic distribution is assumed to have a Dirichlet prior.
* In case of uniform Dirichlet prior distribution pLSA model is equivalent to the LDA model.
* Here when we say a topic, it is not strongly defined semantically.
* It is basically from supervised labeling and pruning on the basis of their likelihood co-occurrence. Because a lexical word may occur in several topics with a different probability with different set of neighboring words in each topic.
* The LDA model is highly modular and hence it can be easily extended. The correlated topic model follows this approach, inducing a correlation structure between topics by using logistic normal distribution instead of the Dirichlet.
* We can also use it to for hierarchical LDA(hLDA), where topics can be joined together in a hierarchy by using the nested Chinese restaurant process.
* Non parametric extensions of LDA include the Hierarchical Dirichlet process mixture model, which allows the number of topics to be unbounded and learnt from data. Nested Chinese Restaurant Process which allows topics to be arranged in a hierarchy whose structure is learnt from data.

**Applications of LDA:**

* Document Modeling
* Information Retrieval
* Classification
* Novelty detection
* Summarization
* Similarity and relevance judgments
* Collaborative filtering.

**FAST MAP:**

* For fast searching[1] in traditional and multimedia databases, it is necessary to map objects into points in k-d space, using k feature-extraction functions provided by a domain expert.
* Fast map is a fast algorithm to map objects into points in k-dimensional space such that the dis-similarities are preserved.
* Thus we can use highly fine-tuned spatial access methods to answer several types of queries including
* Query By Example
* All pairs query
* Nearest-neighbor query
* Best-match query
* Also we can use it for visualization and data-mining. The objects can now be plotted in 2-d, 3-d space, revealing potential clusters, correlations among attributes.
* This method is compared with Multidimensional Scaling although it is unsuitable for indexing whereas Fast Map is suitable for indexing.
* It is significantly faster than MDS. Fast map is a linear time algorithm where as MDS is a quadratic time algorithm.
* Fast map is suitable for indexing as it preserves distances and the overall structure of the data-se
* Multidimensional scaling is used to discover the underlying spatial structures of a set of data items from the dissimilarity information among them.
* The algorithm expects set of N items, the pair-wise similarities and the desirable dimensionality k.
* Then algorithm will map each object to a point in a k dimensional space, to minimize the stress function.
* The stress function given the relative error that the distances in k-d space suffer from, on the average.
* But basically MDS suffers from two drawbacks,
* It requires O(N^2) time, where N is the number of items. Hence it becomes impractical for large datasets.
* Its use for fast retrieval. In the query-by-example setting, the query item has to be mapped to a point in a k-d space. MDS does not provide operations for this and hence complexity of answering a query would be as bad as sequential scanning.
* Hence fast-map comes to rescue to handle the above listed drawbacks of MDS.

**Terminology:**

**Task 1a, 1b, 2**

DTW - Dynamic TIme Warping

simfiles - The time series files that are given as input in csv.

word files - The simulation files(word, average and difference files) generated as the output of task 1.

**Task 1f, 1g, 1h**

“A” Matrix - Word word similarity matrix  for the words taken from given two files

Window similarity - Similarity measure between the windows in the given pair of words

Time similarity - Similarity measure between time in the given pair of words

State similarity - Similarity measure between states in the given pair of words

Word similarity - Similarity measure between the given two words which includes the sum of all other similarities.

**Task 1c,1d,1e**

Binary Vector – The vector to represent the presence of common words in the file used for comparison

Word similarity – Similarity measure between the two files based on the common words

**Task 3:**

SVD - Singular Value Decomposition

LDA - Latent Dirichlet Allocation

r - Top k latent semantics

**Task 4:**

**FM -** FastMap

**Goal Description:**

**Task 1a**

Write a program which gives similarity value between two simfiles when two simulation files are given as input by computing the euclidean distance between them.

**Task 1b**

Write a program which gives similarity value between two simfiles when two simulation files are given as input by computing the dynamic time warping value between them.

**Task 1c, 1d, 1e**

The goal is to find out the similarity between the two files based on the number of common words between the two file. A word can be from a window of any time/iteration for the given two files.

**Task 1f,1g,1h**

To find out the similarity of the two given files based on

How similar the states of the two given words

How similar the time of the two given words

How similar the two given windows of the two words

**Task 3a,3b,3c,3d,3e,3f**

Overall Task 3 is aimed at identifying the latent semantics for the given data and do querying on the data in latent semantic space to get top k results. Task 3a and Task 3c use SVD (eigen decomposition ) for dimensionality reduction and Task 3b uses a generative model based approach which is parametric Latent Dirichlet Allocation.

**Task 4**

Map the given set of objects to reduced space for efficient retrieval.

For Query by example, Map query object into the reduced space and retrieve similar objects.

**Assumptions:**

**Task 1c,1d,1e**

* From each file, distinct window vectors are extracted. Iteration/time of the window vector is not considered. These vectors are considered as a word.
* A set of window vectors is formed by combining the words extracted from both the files. This set contains the distinct words from both the files
* A binary vector is formed for each file . The length n of this binary vector is equal to the size of the window vector set ( number of words in the set ).
* The features of the file’s binary vector gets a value of 1 if the corresponding word from the set is present in the file. If not found, the value is 0.

**Task 1f,1g,1h**

* w1 binary vector is essentially formed from words taken from file 1 which will be all 1’s, and it is a  row vector with the length equal to  the number of unique words in the file.
* w2  binary vector is essentially formed from words taken from file 2 which will be all 1’s, and it is a  column vector with the length of the number of unique words in the file.
* A matrix is of dimension m \* n where m corresponds to the unique words taken from file f1and n corresponds to the unique words taken from file f2.
* Entries in the A matrix is filled by the similarity value computed between words A[i,j].
* Similarity function gives a weightage of 1 when the states are same between given two words and given a weightage of 0.5 when the states present in the two words are neighbours.
* Similarity function gives a weightage of 0.2 when the time between two words are same.
* Similarity function gives a weightage of (match\_count/total\_len) \* 3  where match\_count is the number of corresponding elements matched between  windows of the two given words. total\_len is the maximum of the length of windows of the two given words.
* Since performing multiplication between w1, A and w2 transpose is essentially a summation function of the values present in the matrix A, we do not store any of these matrices in memory. We just keep adding each value of the matrix as and when it is computed and hence it works fine every for very large matrices without hitting any out of space error.

**Task 3a,3b,3c,3d,3e,3f,3g**

* All the words in the simulation files are treated as bag of words which assume conditional independence and identically distributed(positional independence).
* Every document or simulation file is a mixture of unigrams.
* Number of iterations to converge Expectation Maximization algorithm is 100
* Initializing BETA=0.01; ALPHA=50/T;
* DP and WP matrices of the LDA toolkit gives word counts as values. Similarity measure between query and document object is calculated over these term frequencies.
* For SVD, feature space has windows and window\_iteration\_number.

**Task 4a, 4b**

* Distance between the objects is taken as the inverse of the similarity measure between those objects.
* Distance between the objects in the reduced space is taken as the euclidean distance.
* Error in mapping objects to the reduced space is defined as the difference of the initial distances and the Euclidean distances between objects in the reduced space.
* If the similarity between objects is zero, then distance is assumed to be a Maximum value which is Double. MAXVALUE in java implementation.
* In finding the pivot objects in every iteration, always first object in the collection is selected as random object.

**Description of the solution/proposed implementation:**

**Task 1c, 1d, 1e**

Given two files, a similarity measure between 0.0 and 1.0 is returned

https://lh4.googleusercontent.com/0fn7PdAcgx18tOaqqsrEF3V8lqJHbDoufpR-qffHAzfY3k9rdqnntPAgujQ7T-vCMUZLBER7XdUREGCenzMClTpjdSZD6fnYC4JTZdR4R_eqEOC0cYDUZXC6J94grFiZng

where Wi is a binary vector consisting of the words extracted from fi

https://lh6.googleusercontent.com/qNFAeuz8_5CiOMeBS_qzGBen0Q29DUDGqxdBmkU7jFPMfSVdjD8T2LkHH25J6Fwom__1U1p5Mygxdw5JK-Qg3cEOsobOtF8bxA4OSdgTZa8hECpiAqWP_TWbYNQK1Nwl-w

where Wavg,i is a binary vector consisting of the average words extracted from fi

https://lh4.googleusercontent.com/Tvb732Rxau5GV4MaLvq9XMrusFQfgnElogiTiinHWieVmj3ujLlehPdzaSo6XPrTh006Awfs1NflANB-Bt-n9EUkoNIEMM6gzVMFN5g3R7IrnnqL7zJZS4QerOXWYA-0gw

where Wdif f,i  is a binary vector consisting of the difference words extracted from fi

**Task 1c,1d,1e - Algorithmic Flow**

1. Read words from the files
2. Store these words in array list
3. Reduce the words for each file by considering only the distinct words. This is done be adding all the words from the array list into a hash set.
4. Once the distinct words are extracted from both the files, we combine these words to get a set of words which contains the words fom boh the files
5. Now, we create a binary vector for each file
6. We compare the binary vector of each file. When we have the value of 1 at the corresponding value of each binary vector, we increment the similarity score count by 1. Once we iterate over the whole binary vector, we divide this count by the size of the binary vector. This gives us a double value. This is the similarity measure between the two files

https://lh6.googleusercontent.com/jDi6IFNGlk6zw801dCG4r9GlB_SyBtVA92gs2OwxYEme-jAf_W2fj38Uh3z3ea3JdUQr6zyk3rkH68JZTWgimujmxrzrh33HvF540r1mL4jVw9iH7oM5wVbXZ3h0MLd5Vw

where w1 is a binary vector consisting of the words extracted from fi

https://lh3.googleusercontent.com/CIJjd01jf3LaGuhNLiwYlI9NvoIxGRtMVu3JrBOsKHDuiHRN6DzrDo2BSpAjSTeThQtxgKF9aGP9fCAwFwTHJG6LBBhC070RTIckUHsEWbongiF_LsYluo4t7-Vk3xNvSg

where avg,i ~w is a binary vector consisting of the average words extracted from fi

https://lh3.googleusercontent.com/iP6EeRckgKbfTw1zqhxXP1oblgCyHzS28qk8ZEj0bA9uSddpyBfuOYfahoFxv32lOLRySN7RdUvfxYqto6WkSgrmWZl-6BT1J8Q_nwjYRNBiZmm4cveC20JbUEV-19jlhA

where dif f,i ~w is a binary vector consisting of the difference words extracted from fi

*public Double compareFiles(String file1, String file2)*

1. Accepts two files names as arguments
2. Calls the readFile(String f, ArrayList<String> bag) method for File 1 to read the csv file and collect the words into the arraylist bag1
3. Adds all elements from bag1 to the Hashset **hs** to remove the duplicates from bag1
4. Clear bag1 and add all elements from **hs** to bag1. Now bag1 will contain only distinct words. Clear the hash set **hs**
5. Now, call the method readFile using bag2 for File 2 and repeats the steps 3 and 4
6. Now bag1 and bag 2 will have distinct elements from File 1 and File 2
7. We need the distinct words from both the files, so we add all the elements from bag1 and bag2 into the hash set hs
8. Create a binary vector for each corresponding to each element in the hash set.  If the element in the hash set exists in the file, the vector feature is given the value 1, else 0.
9. The similarity score is calculated by ratio of number of features with the value 1 to the total number of features in the binary vector.
10. A double values similarity measure ( between 0.0 and 1.0 ) is returned by the method

**private** **static** **void** readFile(String f, ArrayList<String> bag)

Accepts the file name as a string and an arraylist to collect the words

Scan through the file and extract the words Window vector) line by line and add them into the array list bag.

**Task 2**

Given a query file and similarity function, generate the heat maps of the query and the k most similar images to the file.

**Algorithmic flow - Task 1f,1g,1h**

Load unique words from each given file and form two lists where each list corresponds to words taken from individual file.

For every word in list 1 iterate it through all the words present in list2.

Find the similarity between each pair of words which comprises of similarity value from window match, state match and time match.

Keep adding each of this value with the final result.

Finally normalize the final result value by dividing it by (m \* n)

where m and n is the number of words present in file1 and file2 respectively

**Task 1f:**

1f-formula.png

w1 vector is the binary vector of the unique words from epidemic word file 1.

w2 vector is the binary vector of the unique words from epidemic word file 2.

A - Matrix.png

Sh is the state of the word from file 1

Sj is the state of the word from file2

th is the time period of the word from file1

tj is the time period of the word from file2

**Task 1g:**

1g - formula.png

W avg,1 vector is the binary vector of the unique words from epidemic word average file 1.

W avg, 2 vector is the binary vector of the unique words from epidemic word average file 2.

A avg matrix is same as above except it uses averaged windows

**Task 1h:**

**1h-formula.png**

W avg,1 vector is the binary vector of the unique words from epidemic word difference file 1.

W avg, 2 vector is the binary vector of the unique words from epidemic word difference file 2.

A avg matrix is same as above except it uses differenced  windows

**double getFileSimilarity(String fileNameOne, String fileNameTwo)**

* Function take in two files as input.

**public HashMap<String, List<String>> formAdjacencyHashMap(String inputFilePath)**

* Function forms the adjacency hash map for states.
* This is required to identify whether two given states are neighbours.
* We give boost of 0.5 to the similarity value computed between words if the states in the words are of neighbors.

**public void collectWords(String fileName, List<List<String>> wordList)**

It collects all the unique words in the two given files in  a list.

**public double constructAMatrix(List<List<String>> fileOneWordList, List<List<String>> fileTwoWordList)**

* It actually does not stores a matrix in memory.
* It computes the values of the matrix and keeps adding it to a result value. As the binary vectors are all 1’s multiplying the matrix is essentially equal to the sum of the values present in the matrix

**public double getWordSimilarity(List<String> rowWord, List<String> colWord)**

* Computes the similarity between two given words.
* This similarity is contributed by three different similarity function getStateMatch, getTimeMatch, getWordMatch.

**public double getStateMatch(List<String> listOne, List<String> listTwo)**

* This function checks if the states between two given words are same and returns the similarity value of 1 if they are same.
* Returns a value of 0.5 if they are neighbors states.
* Returns a value of 0 if they are neither same nor neighbors

**double getTimeMatch(List<String> listOne, List<String> listTwo)**

* This function checks if time between two given words are same and returns the similarity value of 0.2 if they are same.
* Returns a value of 0 if they are not same.

**public double getWindowMatch(List<String> rowWord, List<String> colWord)**

* This function checks for number of corresponding  matching elements in the given two windows of the words.
* If only there are more than ½ total length of the max of the two window matches, similarity value is computed as match\_count \*  1/total length of the max of the two window \* windowMatchBoost which is the boost value given for the window match as it should be weighted more compared to state and time matches.
* If match count is less than ½ total length of the max of the two window matches, similarity value is 0.

**Task 3a,3b,3c,3d,3e,3f,3g**

**Task 3a**

*Aim:* SVD on the given epidemic simulation files and return top 'r' latent semantics.

*Input*: A set of epidemic word files where each row is determined by <filename, state, timestamp, window>.

Feature space: <windowij> ,<windowI\_time> where i belongs to all unique windows in the simulation files given as input.

*Algorithm:*

1.    Construct the feature space for the input data matrix. The feature space is constructed by selecting the unique windows from all the files followed by all unique windows indexed by iteration value.

2.    Construct input data matrix consisting of objects X features where objects are the files and features as described above.

3.    SVD is performed on the this data matrix.

4.    SVD output is the set of UƩV matrices where U is the object X new features ,  Ʃ is the diagonal matrices consisting of the square root of the eigen values of the UUT and VVT matrices. Each cell value along the diagonal represents the strength of the eigen vector which is the principal component in the new dimensional space.

5.    Select top 'r' latent semantics from the U matrix and sort the values in descending order to understand the contribution of each of the object(data file) in each of the 'r' latent dimensions.

SVD(M) = UƩVT

**Task 3b**

*Aim:* LDA should be used to generate set of topics 'k' which determine the epidemic simulation files formation where each feature contributes to the topic in a multinomial distribution generated by estimating Dirchilet distribution.

*Input:* A files consisting of <file name, word/feature(window), number of times word occurs in the given file>

*Algorithm:*

1.    A matlab toolkit is used to calculate the LDA of the given data.

2.    Given input file, number of topics, initial alpha, beta, number of iterations to converge; the toolkit will return three matrices.

3.    DP matrix is a object X 'k' topics matrix where objects are the files. DP(i,j) will determine, in the given document or file 'i', number of words that contribute to the topic represented by column 'j'. This is the final output which can be interpreted as the document X topic matrix.

4.    For each of the 'k' topics, contribution of each file is returned in the descending order.

**Task 3c**

*Aim:* The aim of this task is to perform SVD on the file X file similarity matrix and identify the top 'r' latent semantics and contributions of each file in the latent semantic space.

*Input*: A set of epidemic simulation or epidemic word files and the type of similarity measure to be used.

Feature space: Set of input files represented as dimensions.

*Algorithm:*

1.    Construct the input data matrix 'D', object X features where each object is the file represented as a vector in files.size() dimensions and each feature is also a file in the given set of input files. The value of D(i,j) is the file-file similarity calculated from either of the tasks 1a to 1h.

2.    SVD is performed on the this data matrix 'D'.

3.    SVD output is the set of UƩV matrices where U is the object X new features ,  Ʃ is the diagonal matrices consisting of the square root of the eigen values of the UUT and VVT matrices. Each cell value along the diagonal represents the strength of the eigen vector which is the principal component in the new dimensional space.

Select top 'r' latent semantics from the U matrix and sort the values in descending order to understand the contribution of each of the object(data file) in each of the 'r' latent dimensions.

**Task 3d**

*Aim:* The aim of this task is perform a search operation in the top 'r' latent semantic space of the given input files and return the 'k' most similar files to the given query file

*Input*: A set of epidemic simulation or epidemic word files and the type of similarity measure to be used.

Feature space:  Feature space same as the input files in the database.<windowij> ,<windowI\_time> where i belongs to all unique windows in the database of simulation files.

*Algorithm:*

1.    A query(Q) is also a file object represented as a column vector defined in the original feature space dimensions as that of the input files in the database.

2.    Since it is difficult to query on huge number of dimensions, the files in the database are represented in top 'r' latent semantic space.

3.    To query on 'r' latent semantic dimensional space, the query vector should also be mapped to that space.

4.    Construct Vr matrix representing the V matrix in 'r' latent semantics by selecting top 'r' rows.

5.    Multiply Q which is 1 X features with features X 'r' latent semantics to get a Qr matrix with 1 X 'r' dimensions.

6.    Now both query Qr and Dr are both in the 'r' dimensional space. Find dot product of Qr with each of the document in the Dr matrix.

7.    The result is a column vector of similarity values for each of the document objects.

8.    Sort the values and return the 'k' most similar files(document objects) to the given query thus performing the search operation.

**Task 3e:**

*Aim:* The aim of this task is perform a search operation in the top 'r' latent semantic space of the given input files and return the 'k' most similar files to the given query file

*Input*: A set of epidemic simulation or epidemic word files and the type of similarity measure to be used.

Feature space:  The search process has to be done in the 'k' topic space that are generated using the LDA.

*Algorithm:*

1.    A query(Q) is represented as a term frequency vector of the features in the original document space.

2.    Since it is difficult to query on huge number of dimensions, the files in the database are modeled as a distribution of 'k' topics.

3.    To query on 'k' topic space, the query vector should also be mapped to that space.

4.    Construct QTmatrix representing the Q matrix in 'k' topic space by multiplying with document topic matrix(DT) obtained by doing LDA.

5.    Now both query QTand DT are both in the 'k' dimensional space. Find dot product of QTwith each of the document in the DT matrix.

6.    The result is a column vector of similarity values for each of the document objects.

7.    Sort the values and return the 'k' most similar files(document objects) to the given query thus performing the search operation.

**Task 3f**

*Aim:* The aim of this task is perform a search operation in the top 'r' latent semantic space of the given input files and return the 'k' most similar files to the give n query file

*Input*: A set of epidemic simulation or epidemic word files and the type of similarity measure to be used.

Feature space:  Set of input files represented as dimensions.

*Algorithm:*

1.    A query(Q) is also a file object represented as a column vector defined in the original feature space.

2.    Since it is difficult to query on huge number of dimensions, the files in the database are represented in top 'r' latent semantic space.

3.    To query on 'r' latent semantic dimensional space, the query vector should also be mapped to that space.

4.    Construct Vr matrix representing the V matrix in 'r' latent semantics by selecting top 'r' rows.

5.    Multiply Q which is 1 X features with features X 'r' latent semantics to get a Qr matrix with 1 X 'r' dimensions.

6.    Now both query Qr and Dr are both in the 'r' dimensional space. Find dot product of Qr with each of the document in the Dr matrix.

7.    The result is a column vector of similarity values for each of the document objects.

8.    Sort the values and return the 'k' most similar files(document objects) to the given query thus performing the search operation.

**Task 4:**

Given a set of simulation files, coordinates of each file is computed in the reduced space. Given a simulation file as a query object, top k similar simulations are returned. Distance between files is considered to be the inverse of the similarity measure of the files.

input: set of simulation files, similarity measure, integer r

output: set of pivot objects in every iteration, set of new coordinates in the reduced space.

compute distance matrix. Distance between two objects is the inverse of the similarity of the two files.

choose pivot objects using ChooseDistantObjects algorithm using the ProjectedDistanceFunction

project every object on the pivot line by calculating the coordinate of that object on the line using the below formula.

https://lh3.googleusercontent.com/yU84muFU6fYFt2Yfoq2odEnYxrx76rMQxCoHNN7onqZ8SyK7PaId76z6HZ5ZmC7eyHdWpkRTV9wNq8h2Jm8IXg80mrtCvw2Hes4dGYYB8FU8emMi-BQ0YoMlDB18fyK4-A

        where a and b are pivot objects in the current iteration and d is the projected distance between

        the objects.

Project every object on the hyperplane perpendicular to the pivot line and calculate the projected distance function using ProjectedDistanceFunction recursive algorithm.

reduce r to r-1.

if r is greater than 0. proceed to step -2 with new distance function.

After mapping all the objects into the reduced space, **Mapping Error** is computed as the difference of the initial distances and the Euclidean distances between objects in the reduced space.

**ChooseDistantObjects**

input: set of simulation files, distance function

output: array of two pivot objects.

choose one object as the pivot object a.

using the distance function calculate the farthest object form the a which is b.

using the same distance function calculate the farthest object from b which is c.

return b and c.

**ProjectedDistanceFunction**:

input: two objects, iteration number, coordinates in the reduced space

output: projected distance

if iteration==-1, calculate distance between objects from the computed distance matrix

else,  calculate the projected distance of the two objects on the hyperplane perpendicular to the pivot line recursively using,

https://lh4.googleusercontent.com/wzu0cy5R7bLWu-8y3CueTCa_Nir2O49-feXETBBn32OMRGfqwYkRXKOGS9XHQZLZMrRdNYhI0wuCjlQMxC3uBk8zTMfGfpai90zqOKq6uYsTiOv7Asm5ol56r4ghhhfL_g

where D’ is the projected distance in the current iteration and D is the projected distance between objects Oi and Oj in the previous iteration. Xi and Xj are coordinates of Oi and Oj in the on the pivot line in the previous iteration

**FindSimilarSimulations**

input: Coordinates of every object in the reduced space, reduced space dimension r, pivot objects in every iteration, query object, distance function, integer k

ouput: top k similar objects

* For every iteration, using the pivot objects of the iteration, project query object on the pivot line and compute the coordinate on that line using the formula, where distances are the projected distances on the hyperplane in the iteration.
* After mapping the query object in the reduced space, top k nearest neighbours are found using the euclidean distance measure in the reduced dimension

**Complexity**

Computation of distance matrix is O(N^2) where N is the number of objects. Given the distance matrix, finding the coordinates in the reduced space is O(rN) as number of iterations is r and in each iteration finding the distant objects is O(N).

For mapping the query object, as the pivot objects and projected distances between the pivot objects are stored in the reduction phase, time complexity is O(r) as for every iteration only the coordinate(projection) of the query object on the pivot line is to be calculated.

**Interface Specifications:**

**Task 1c,1d,1e**

**public Double compareFiles(String file1, String file2)**

Input: File paths of the two given files

**private static void readFile(String f, ArrayList<String> bag)**

Input:

1. File path of the file to be read,
2. Array list to return the bag of words

**Task 1f,1g,1h**

**double getFileSimilarity(String fileNameOne, String fileNameTwo)**

Input: File path of two given files

Output: Similarity value of the two given files**.**

**SimilarityGenerator simObj = new SimilarityGenerator(locationFile)**

File path of the LocationMatrix.csv is given during the SimilarityGenerator object creation.

**Task 1f:**

**double getFileSimilarity(String EpidemicWordFileOne, String EpidemicWordFileTwo)**

Two EpidemicWordFiles are passed as input arguments to the getFileSimilarity function.

**Task 1g:**

**double getFileSimilarity(String EpidemicWordFileDiffOne, String EpidemicWordFileDiffTwo)**

Two EpidemicWordFileDiff files are passed as input arguments to the getFileSimilarity function.

**Task 1h:**

**double getFileSimilarity(String EpidemicWordFileAvgOne, String EpidemicWordFileAvgTwo)**

Two EpidemicWordFileAvg files  are passed as input arguments to the getFileSimilarity function.

**Task 3a,3b,3c,3d,3e,3f**

**Task 3**

·         Task3.java is a wrapper class which will instantiate appropriate class objects for sub tasks of Task 3 based on the input from the user.

·         All the path variables and constants are initialized and set in this wrapper class.

·         Matlab proxy and factory settings are public static variables initialized so that all sub tasks can use the same settings. This implementation will initialize only one matlab instance and execute all functions in it.

·         Input relevant to each of the sub tasks is taken and assigned to corresponding static variables of sub classes.

**Task 3a**

·         constructFeatureSpace();

Input to this method is folder path to the epidemic word/average/difference files. It then computes the feature space. All the files from the folder are read and for each row, windows and time iteration values are extracted. A featureIndexMap is constructed which will have the unique windows from all the files as keys and the number of occurrences of the windows(term frequency) as the value.  A fileIndex map is also constructed which will map the filename to an index value.  This feaureIndexMap will only store the non zero occurrences of unique words

·         constructFeatureVectorsSVD();

This method is used to construct the feature vectors required for the input to SVD. It will read every word in the file and construct a term frequency vector of type string and write to a file so that SVD function in matlab will be able to consume it.

·         calculateSVD(r);

This method will call the corresponding SVD calculation function in Matlab. The input given to matlab function is number of latent semantics 'r'. Output received from Matlab will be 2D double array having strengths and file indexes of all the 'r' latent semantics.

·         [final\_result] = function svdcalc(r)

This function will read the feature vectors created from the constructFeatureVectorsSVD() function and performs SVD operation. The results of the SVD are persisted for later search purposes(sub task 3d). In the obtained U matrix, for every latent semantic dimension 'r', the contributions of each of the document object are given in the descending order and returned. The results are then displayed using java and persisted in a file.

**Task 3b**

·         constructFeatureSpace();

Input to this method is folder path to the epidemic word/average/difference files. It then computes the feature space. All the files from the folder are read and for each row, windows and time iteration values are extracted. A featureIndexMap is constructed which will have the unique windows from all the files as keys and the number of occurrences of the windows(term frequency) as the value.  A fileIndex map is also constructed which will map

·         constructLDAInput()

Input to the LDA matlab toolkit used takes a file with each row having the following format. <file, word, number of times the word occurs in that file>. This particular vector is constructed and stored in a file so that matlab function can consume it.

·         calculateLDA(k)

This function will invoke the matlab function to calculate LDA. Input to the matlab function is number of topics to be generated.  This function gets the output from the matlab and displays the 'k' topics and the documents having that topic.

·         function [final\_result] = calcldanew(k)

This function will calculate LDA.  importworddoccounts() function is called to read the word document counts generated from java and generate WS, DS matrices. Iteration count for the EM algorithm to halt, seed, initial alpha, beta values are given prior. WP, DP, Z matrices are generated. These matrices are persisted for further use by sub tasks 3e.

**Task 3c**

·         constructSimilaritySimilarityMatrix(option)

This function will generate file-file similarity matrix for the SVD function. Given path to a folder which has collection of files, file-file similarity matrix is constructed. This function will call getSimilarityForFiles() function of SimilarityWrapper class which to get the similarity measure.

·         getSimilarityForFiles()

This function will take two files and type of user similarity measure as input and appropriately invokes sub tasks of Task 1 to get a similarity measure.

·         doFileFileSVD(r)

This function will invoke SVD function in matlab and compute UƩV matrices and persists them for searching purposes.

**Task 3d**

·         constructFeatureVectorsQuerySVD()

This function will take the query file as input and reads the featureIndexMap created using subtask of 3a and constructs the input for the matlab function.

·         doSVDSearch(r,k)

This function will invoke the matlab function which will perform the actual search operation. The input to the matlab function is number of latent semantics and top 'k' results that are to be returned.

·         [final\_output] = SVDSearch(r,k)

This function will construct the 1 X features matrix for the query. It will U,Ʃ,V matrices computed and convert the query Q to the 'r' latent semantic space. Then a dot product is computed between query vector and the U matrix in 'r' dimensional space. The resultant column vector is returned to the java function.

**Task 3e**

·         constructLDAInputQuery()

This function will take the input query file and construct a feature vector in terms of the features from the original input files. This is saved as a file so that matlab function will read and do further query processing.

·         doLDASearch(k)

This function will invoke the matlab function to do the query processing. The input to the matlab function is the number of top 'k' results that have to be returned.

·         [final\_output] = LDASearch(k)

Given search query file is converted to 'k' topic space by multiplying with the WP matrix output of the LDA function. Then dot product similarity is calculated between the Qk and DT matrices. The resultant column vector values are sorted and top 'k' results are returned to the java function.

**Task 3f**

·         constructFileFileSimilarityQuery()

For the given input query file, and for a given similarity measure, file-file similarity is calculated between this file and all other files. This represented as a feature vector is passed as input to the matlab function.

·         doFileFileSVDSearch()

This function will invoke the matlab function for further query processing.

·         [final\_output] = SVDSearch\_file\_file(r,k)

·         This function will construct the 1 X features(number of input files) matrix for the query. It will read U,Ʃ,V matrices computed and convert the query Q to the 'r' latent semantic space. Then a dot product is computed between query vector and the U matrix in 'r' dimensional space. The resultant column vector is sorted and top

'k ' results are returned to the java function.

**SimilarityWrapper**

**getSimilarityForFiles**(o1,o2,option)

function which  provides similarity measure between objects given a userinput on how to compute the similarity.

**FastMap**

**computeDistanceMatrix()**

given set of simlutaion files, taking each file as an object, this function computes distance matrix. Distance between each object is the inverse of the similarity measure between the objects.

**getReducedSpace**()

implementation of the FastMap algorithm.

Function takes distance matrix with dimension N\*N , reduced space dimention r.

Function outputs pivot objects PV[2][r] which has indices of the pivot objects in every iteration. coordinates[N][r] which has all the coordinates of N objects in r dimensional space. pivotDistances[r] which has distances between pivot objects in every iteration.

**double distancebtwObjects(int i, int i2, int iteration)**

implementation of ProjectedDistanceFunction algorithm.

i and i2 are the indices of the loaded file objects. iteration is the iteration in which projected distance between the objects needs to be computed.

int[] **chooseDistantObjects**(int iteration)

implementation of the ChooseDistantObjects algorithm

iteration is to use the projected distance in that iteration to compute the distant objects.

**double calculateMappingError()**

calculates the mapping error of mapping the given objects to the reduced space.

difference of the initial distances and the euclidean distances between objects in the reduced space.

**getSimilarSimulations(String filePath,int r, int k)**

implementation of FindSimilarSimulations algorithm.

input filepath is the query simulation file, r is the reduced space dimension and k is the top k similar simulations files to be returned.

Euclidean distances are used to calculate the distances between objects in the reduces space.

**double getEuclideanDistances(double[] newCoordiantes, double[] coordinates2)**

calculates the euclidean distance between the two coordinates in the euclidean space.

**Task 1a**

The number of entries in both simulation files should be the same.

The input is csv format for the python program to read it.

**Task 1b**

The number of entries in both simulation files should be the same.

The inputs should be in csv format for the program to read it.

**System requirements/installation and execution instructions**

java version - 1.7.0\_45

Matlab version - MATLAB\_R2013b

Java - Matlab connectivity based on phase1 of the given connectivity instruction is used.

python 2.7.5, Matplotlib

**Related Work:**

Multidimensional scaling is the means of visualizing the level of similarity. An MDS algorithm  aims to place each object in N-dimensional space such that object distances are preserved as much as possible. Each object will have its own representation in N dimension. The different variations of multidimensional scaling are classical multidimensional scaling, metric multidimensional scaling, non-metric multidimensional scaling, generalized multidimensional scaling.

k-Nearest Neighbors algorithm is a non-parametric method used for classification and regression. It basically selects the k-nearest neighbors of the given data point or query. It is based on the intuition more similar vectors will be present in nearer to a given vector than least similar one.

Levenshtein distance is a string metric for finding out the edit distances between pair of strings. As the distance increases it means the similarity decreases. This can be employed while performing partial strings matches.

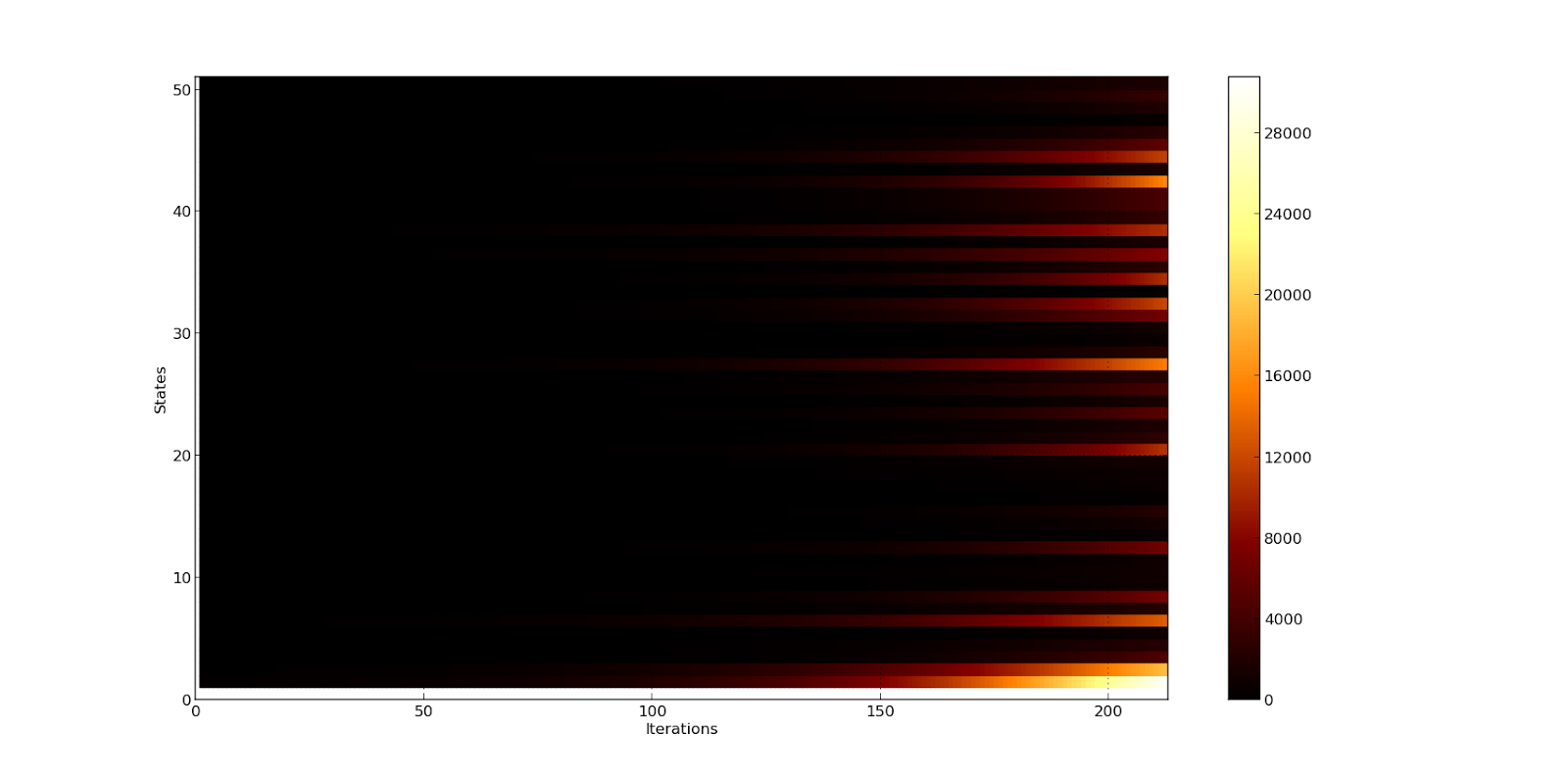
**Results:**

**Task 1a, 1b**

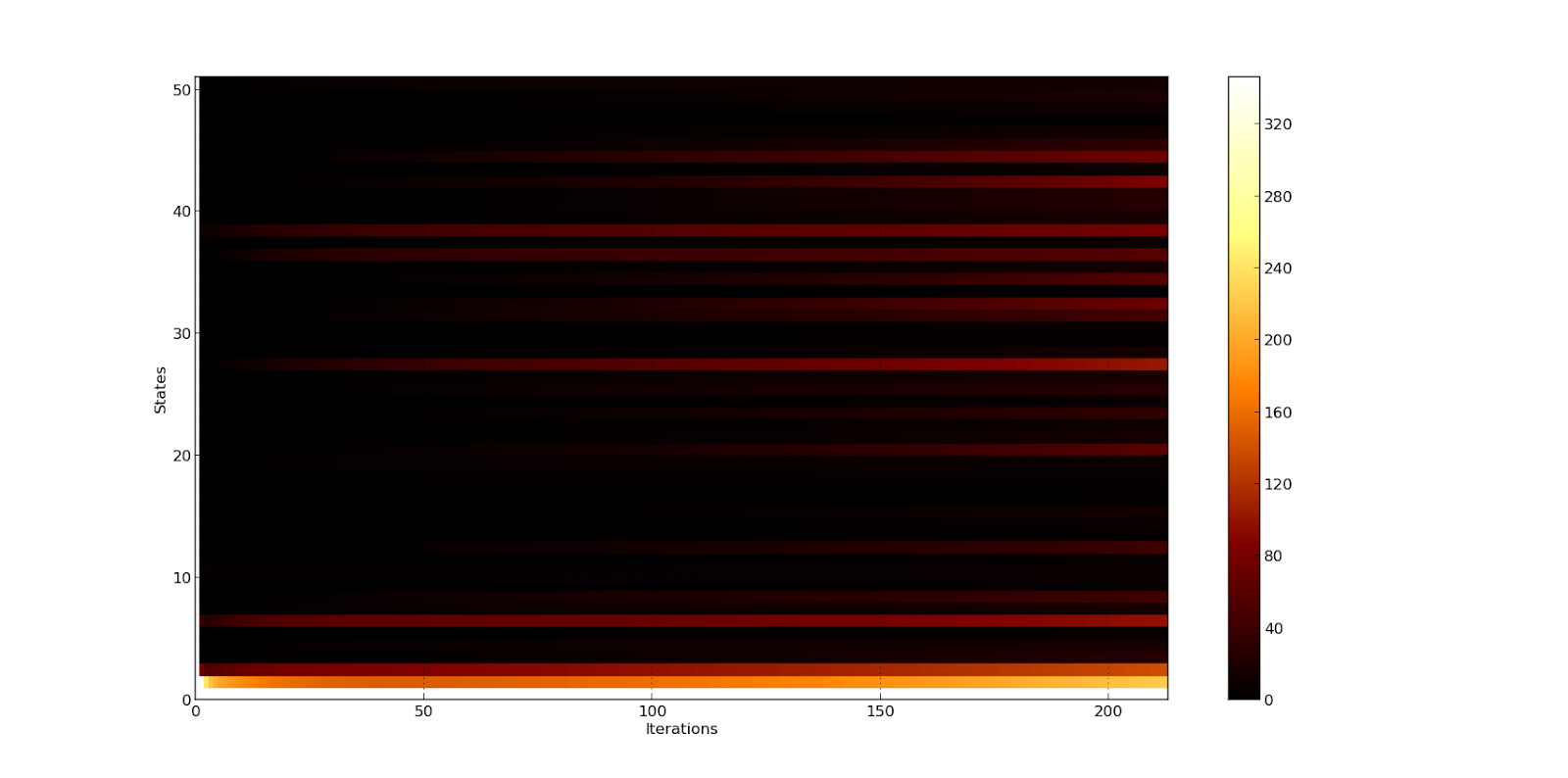
Task 1a - The program successfully returned the euclidean similarity value when simfiles are passed to it.

|  |  |  |
| --- | --- | --- |
| Object Files | Eucledian | DTW |
| 1.csv | 1.17815216543e-06 | 1.56563285013e-06 |
| 2.csv | 1.07756569863e-06 | 1.51055029055e-06 |
| 3.csv | 1.01887936712e-06 | 1.46634133324e-06 |
| 4.csv | 1.14813233283e-06 | 1.55171272631e-06 |
| 5.csv | 2.08231881452e-06 | 2.99587731312e-06 |
| 6.csv | 2.16112239515e-07 | 2.17870263583e-07 |
| 7.csv | 2.97192798884e-07 | 3.19936545508e-07 |
| 8.csv | 2.97619279632e-07 | 3.20127893702e-07 |
| 9.csv | 2.98870412462e-07 | 3.22727254538e-07 |
| 10.csv | 3.00159726654e-07 | 3.26931320792e-07 |
| 11.csv | 5.12377260483e-06 | 5.12446709468e-06 |
| 12.csv | 5.12533775291e-06 | 5.1264618572e-06 |
| 13.csv | 5.13545759044e-06 | 5.13566128868e-06 |
| 14.csv | 5.12332370625e-06 | 5.12419619659e-06 |
| 15.csv | 5.12242497096e-06 | 5.12304331178e-06 |
| 16.csv | 5.18087684231e-06 | 5.22156859684e-06 |
| 17.csv | 5.18597560476e-06 | 5.21956284547e-06 |
| 18.csv | 5.22692786834e-06 | 5.28087447747e-06 |
| 19.csv | 5.17867061273e-06 | 5.21820947224e-06 |
| 20.csv | 5.1444024815e-06 | 5.16283109315e-06 |

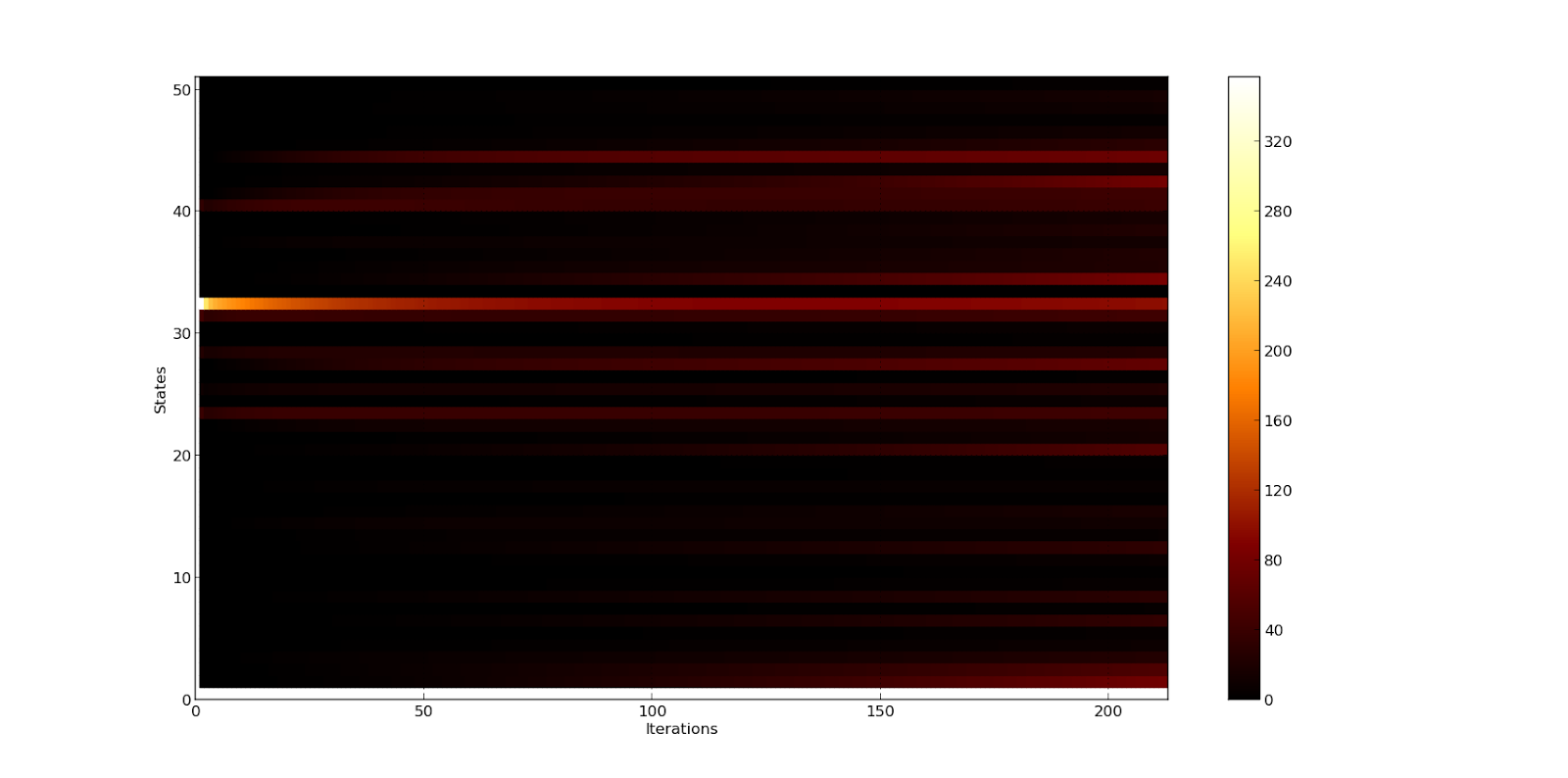
The heat map images for Euclidean are as follows.

****

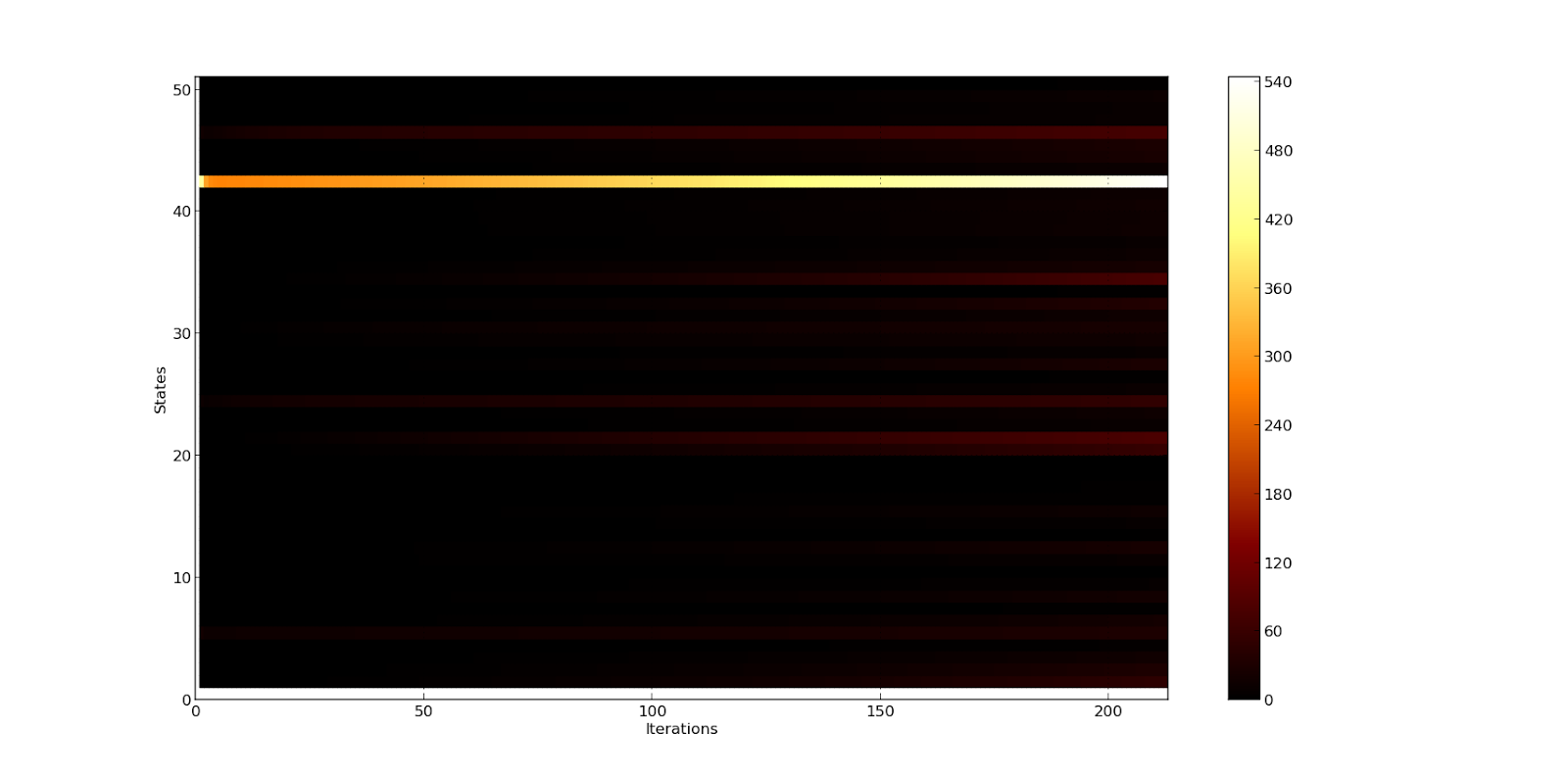
Query.csv



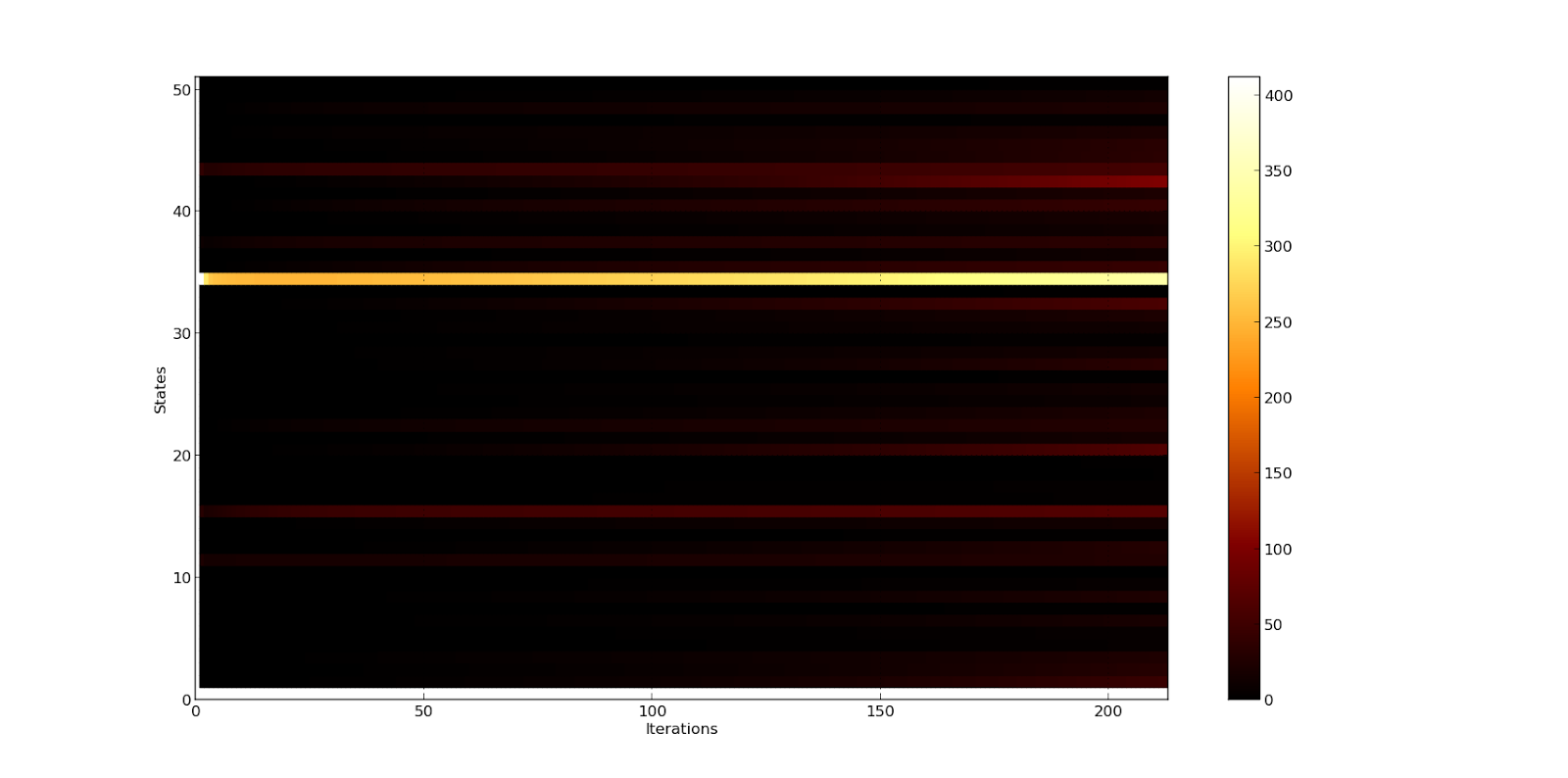
18.csv Rank1



17.csv Rank 2



16.csv Rank 3



19.csv Rank 4

The rest of the output images can be found in the output folder.

Task 4

Results: r=4, k=4

Results of Task 4a and 4b on the set of test simulation files.

Dimension of the reduced space ( r) = 4

Number of top similar simulation files (k) = 4

|  |  |  |
| --- | --- | --- |
| Similarity Measure | Mapping Error (4a) | Top 4 simulations for the given query file. (4b) |
| 1a | -1.0755737970261097E7 | 18.csv  19.csv  16.csv  17.csv |
| 1c | -88.9992913389633 | 4.csv  9.csv  2.csv  14.csv |
| 1d | -40.3049765403307 | 9.csv  10.csv  5.csv  4.csv |
| 1e | -40.3049765403307 | 2.csv  9.csv  4.csv  10.csv |
| 1f | 59.94100784908601 | 14.csv  15.csv  12.csv  13.csv |
| 1g | 65.12751161109463 | 4.csv  2.csv  3.csv  6.csv |
| 1h | 65.12751161109463 | 4.csv  2.csv  3.csv  6.csv |

**Task 1c, 1d, 1e**

**File SImilarity Results**

|  |  |  |
| --- | --- | --- |
| **File 1** | **File 2** | **Similarity** |
| n1 | n2 | 0.153846154 |
| n1 | n4 | 0.230769231 |
| avgn11 | avgn14 | 0.571428571 |
| Diffn3 | Diffn19 | 0.35 |
| n11 | n14 | 1.0 |
| Diffn9 | n9 | 0.0 |
| Q-n1 | n2 | 0.285714286 |
| Q-n1 | n17 | 0.166666667 |
| Avgn16 | n5 | 0.0 |
| Q-Avg | Avgn16 | 0.105263158 |

**Task 1f,1g,1h** - **File Similarity Results:**

|  |  |
| --- | --- |
| Files | Similarity |
| avgn1 vs avgn2 | 1.8131689495092942 |
| avgn2 vs avgn1 | 1.8131689495092942 |
| avgn2 vs avgn2 | 1.8434064257636078 |
| avgn1 vs n1 | 0.022697314219676717 |
| n1 vs avgn1 | 0.022697314219676717 |
| n1 vs n1 | 1.9994486901660429 |
| diffn1 vs n1 | 0.022697314219676717 |
| n1 vs diffn1 | 0.022697314219676717 |
| diffn1 vs diffn1 | 1.6426208827856956 |
| diffn1 vs avgn5 | 0.022697314219676717 |
| avgn5 vs diffn1 | 0.022697314219676717 |

**Results**

**Task 3a:**

**Input:  20 word files [1.csv, 2.csv ... 20.csv]**

**Number of Latent Semantics: 4**

**Output:**

**Latent semantic - 0**

**n16.csv --> -0.2217867059607522**

**n19.csv --> -0.2219961454597444**

**n1.csv --> -0.22272703675206185**

**n10.csv --> -0.22356363377661848**

**n8.csv --> -0.2235636551636351**

**n9.csv --> -0.22356376221063703**

**n5.csv --> -0.22356392870275532**

**n7.csv --> -0.22366831421269237**

**n4.csv --> -0.22366858062913073**

**n6.csv --> -0.2237729111941798**

**n2.csv --> -0.22377323014662648**

**n3.csv --> -0.2237732372883119**

**n17.csv --> -0.2240869383968987**

**n18.csv --> -0.2240869383968987**

**n20.csv --> -0.2240869383968987**

**n11.csv --> -0.22408694323707387**

**n12.csv --> -0.22408694323707387**

**n13.csv --> -0.22408694323707387**

**n14.csv --> -0.22408694323707387**

**n15.csv --> -0.22408694323707387**

**Latent semantic - 1**

**n19.csv --> 0.6674993301118941**

**n16.csv --> 0.63473462028367**

**n1.csv --> 0.11555454542729363**

**n5.csv --> -0.01568894064268282**

**n4.csv --> -0.019932488302996893**

**n3.csv --> -0.020527783656252012**

**n2.csv --> -0.022374338246929068**

**n9.csv --> -0.06987278153626812**

**n20.csv --> -0.1023350783156285**

**n18.csv --> -0.10233507831562852**

**n17.csv --> -0.10233507831562853**

**n8.csv --> -0.10265875371313185**

**n7.csv --> -0.10347588791556449**

**n13.csv --> -0.10503909375801675**

**n14.csv --> -0.10503909375801675**

**n12.csv --> -0.10503909375801676**

**n15.csv --> -0.10503909375801676**

**n11.csv --> -0.1050390937580169**

**n6.csv --> -0.10888042808809482**

**n10.csv --> -0.10987088840594061**

**Latent semantic - 2**

**n1.csv --> 0.596798442898666**

**n5.csv --> 0.31364766923105813**

**n4.csv --> 0.2778826784101943**

**n2.csv --> 0.21351315187913653**

**n3.csv --> 0.19624370264492594**

**n6.csv --> -0.001677999019281973**

**n11.csv --> -0.01886979471559791**

**n15.csv --> -0.01886979471559816**

**n14.csv --> -0.018869794715598173**

**n13.csv --> -0.018869794715598187**

**n12.csv --> -0.018869794715598354**

**n20.csv --> -0.022191483806399202**

**n18.csv --> -0.02219148380639921**

**n17.csv --> -0.02219148380639925**

**n19.csv --> -0.08464620785301483**

**n16.csv --> -0.19114650713627462**

**n7.csv --> -0.26606756556904065**

**n10.csv --> -0.28537885268928886**

**n8.csv --> -0.3003310386534944**

**n9.csv --> -0.30780550927275846**

**Latent semantic - 3**

**n1.csv --> 0.44829962372500526**

**n8.csv --> 0.3459985593718416**

**n9.csv --> 0.32470962696677924**

**n10.csv --> 0.2944326690685478**

**n7.csv --> 0.289046201741225**

**n5.csv --> 0.09789767458998491**

**n4.csv --> 0.08786092260690086**

**n16.csv --> 0.05456456865096843**

**n3.csv --> 0.009812522557957688**

**n2.csv --> 0.002244459966690839**

**n6.csv --> -0.15530680191385068**

**n17.csv --> -0.18518953979803734**

**n20.csv --> -0.18518953979803734**

**n18.csv --> -0.18518953979803737**

**n12.csv --> -0.20486416482310182**

**n15.csv --> -0.2048641648231021**

**n11.csv --> -0.20486416482310216**

**n13.csv --> -0.2048641648231022**

**n14.csv --> -0.2048641648231022**

**n19.csv --> -0.21541438554789347**

**Task 3d:**

**Input: 20 word files**

**Number of latent semantics: 4**

**Search query: query\_word.csv**

**Output:**

**n11.csv->4.4840535175331085E10**

**n12.csv->4.4840535175331085E10**

**n13.csv->4.4840535175331085E10**

**n14.csv->4.4840535175331085E10**

**n15.csv->4.4840535175331085E10**

**Task 3a:**

**Input:  20 word files + 20 average files + 20 difference files generated for given sample files**

**Number of Latent Semantics: 4**

**Output:**

**Latent semantic - 0**

**n11.csv --> 0.21008121662003612**

**n12.csv --> 0.21008121662003612**

**n13.csv --> 0.21008121662003612**

**n14.csv --> 0.21008121662003612**

**n15.csv --> 0.21008121662003612**

**n17.csv --> 0.21007963167262553**

**n18.csv --> 0.21007963167262553**

**n20.csv --> 0.21007963167262553**

**n3.csv --> 0.20978873563613082**

**n2.csv --> 0.2097887347425238**

**n6.csv --> 0.20978523748268046**

**n4.csv --> 0.2097248438105863**

**n7.csv --> 0.2096887733760257**

**n9.csv --> 0.20962492805886226**

**n10.csv --> 0.2095985594815448**

**n5.csv --> 0.209593328623299**

**n8.csv --> 0.20959145397303247**

**n1.csv --> 0.20885967571610164**

**n19.csv --> 0.20813968137569613**

**n16.csv --> 0.20794723757351807**

**avgn15.csv --> 0.07849400613551268**

**avgn20.csv --> 0.07849400088306582**

**avgn11.csv --> 0.0784598602395233**

**avgn14.csv --> 0.07842571764738886**

**avgn12.csv --> 0.07839157009587168**

**avgn17.csv --> 0.07839156146180182**

**avgn13.csv --> 0.07839155851365225**

**avgn18.csv --> 0.07839155484400856**

**avgn3.csv --> 0.07805025650009262**

**avgn2.csv --> 0.07805024442274885**

**avgn6.csv --> 0.07804947166533831**

**avgn4.csv --> 0.07791359255617687**

**avgn7.csv --> 0.07791302291332244**

**avgn5.csv --> 0.07777686987181393**

**avgn9.csv --> 0.07777644484427426**

**avgn8.csv --> 0.07777631772872827**

**avgn10.csv --> 0.07777614260807197**

**avgn1.csv --> 0.0766840583527944**

**avgn16.csv --> 0.0754563395428429**

**avgn19.csv --> 0.075011717876825**

**diffn7.csv --> 2.3679825180861574E-17**

**diffn4.csv --> 2.3581688655826377E-17**

**diffn8.csv --> 2.3259740976938702E-17**

**diffn5.csv --> 2.3081469813506377E-17**

**diffn6.csv --> 2.2224997876328243E-17**

**diffn20.csv --> 2.215374938574561E-17**

**diffn9.csv --> 2.1991215019732857E-17**

**diffn3.csv --> 2.1758334730505402E-17**

**diffn2.csv --> 2.1222334249150273E-17**

**diffn16.csv --> 2.1216972990274445E-17**

**diffn17.csv --> 2.0812320968522936E-17**

**diffn12.csv --> 2.0808949745693043E-17**

**diffn15.csv --> 2.0622301389989664E-17**

**diffn11.csv --> 2.0605337259314007E-17**

**diffn10.csv --> 2.058187585260103E-17**

**diffn13.csv --> 2.0435447908167084E-17**

**diffn18.csv --> 2.0427590847790105E-17**

**diffn14.csv --> 1.9932073024903317E-17**

**diffn19.csv --> 1.7279677085444883E-17**

**diffn1.csv --> 1.348304084512892E-17**

**Latent semantic - 1**

**avgn15.csv --> 0.2117923971291409**

**avgn20.csv --> 0.21179238092199007**

**avgn11.csv --> 0.21168795216922384**

**avgn14.csv --> 0.21158351732724748**

**avgn12.csv --> 0.21147906729585692**

**avgn17.csv --> 0.21147904070069667**

**avgn13.csv --> 0.21147903181802963**

**avgn18.csv --> 0.21147902042979813**

**avgn3.csv --> 0.21043503313397155**

**avgn2.csv --> 0.21043499627564077**

**avgn6.csv --> 0.21043269139056864**

**avgn4.csv --> 0.2100169300520729**

**avgn7.csv --> 0.21001536915728025**

**avgn5.csv --> 0.2095988474305603**

**avgn9.csv --> 0.20959771210563902**

**avgn8.csv --> 0.2095972012761663**

**avgn10.csv --> 0.20959697332068694**

**avgn1.csv --> 0.20625531884743897**

**avgn16.csv --> 0.202500210208806**

**avgn19.csv --> 0.20114082619977314**

**diffn1.csv --> -7.300749690740476E-18**

**diffn19.csv --> -8.859015197936651E-18**

**diffn14.csv --> -1.0060490930160133E-17**

**diffn18.csv --> -1.0268401049293899E-17**

**diffn13.csv --> -1.0271711223430807E-17**

**diffn10.csv --> -1.0318246585597065E-17**

**diffn11.csv --> -1.0344964358045185E-17**

**diffn15.csv --> -1.0352951804254066E-17**

**diffn12.csv --> -1.0430670207878005E-17**

**diffn17.csv --> -1.0432090087122124E-17**

**diffn16.csv --> -1.052866546361169E-17**

**diffn2.csv --> -1.0594805352751644E-17**

**diffn3.csv --> -1.0820609624884216E-17**

**diffn9.csv --> -1.0911976377333547E-17**

**diffn20.csv --> -1.0998119991778622E-17**

**diffn6.csv --> -1.1017179005343833E-17**

**diffn5.csv --> -1.1371298358880525E-17**

**diffn8.csv --> -1.1446366910581042E-17**

**diffn4.csv --> -1.15853881612056E-17**

**diffn7.csv --> -1.1626719902030362E-17**

**n16.csv --> -0.07711350087428526**

**n19.csv --> -0.0771983360536038**

**n1.csv --> -0.07734625681310543**

**n9.csv --> -0.07769611260508277**

**n4.csv --> -0.07772754610253346**

**n10.csv --> -0.07777637845006545**

**n5.csv --> -0.07779333289977784**

**n8.csv --> -0.07779818889508233**

**n7.csv --> -0.07783705285422389**

**n2.csv --> -0.0778686598385996**

**n3.csv --> -0.07786868005979761**

**n6.csv --> -0.0778783343246913**

**n11.csv --> -0.07798280056784485**

**n12.csv --> -0.07798280056784485**

**n13.csv --> -0.07798280056784485**

**n14.csv --> -0.07798280056784485**

**n15.csv --> -0.07798280056784485**

**n17.csv --> -0.07798763459399752**

**n18.csv --> -0.07798763459399752**

**n20.csv --> -0.07798763459399752**

**Latent semantic - 2**

**diffn15.csv --> 0.22593565355558126**

**diffn20.csv --> 0.2259356350616312**

**diffn11.csv --> 0.2258215539748804**

**diffn12.csv --> 0.22581180489237174**

**diffn17.csv --> 0.22581177455535628**

**diffn14.csv --> 0.22570746591656338**

**diffn13.csv --> 0.22559332011115713**

**diffn18.csv --> 0.22559330710112394**

**diffn3.csv --> 0.22445302395000488**

**diffn2.csv --> 0.2244529819847413**

**diffn6.csv --> 0.22445035032023591**

**diffn4.csv --> 0.22399631388342767**

**diffn7.csv --> 0.2239945125550733**

**diffn5.csv --> 0.22353959946125776**

**diffn9.csv --> 0.2235382848584139**

**diffn8.csv --> 0.2235377202366573**

**diffn10.csv --> 0.22353741851933576**

**diffn1.csv --> 0.21988742527064664**

**diffn16.csv --> 0.2157857608131951**

**diffn19.csv --> 0.21430069329036644**

**avgn11.csv --> 9.077522736734746E-17**

**avgn19.csv --> 7.533137270479511E-17**

**avgn12.csv --> 6.143040606385054E-17**

**avgn16.csv --> 5.973271877262018E-17**

**avgn13.csv --> 4.2609145378680324E-17**

**avgn9.csv --> 3.170042908233467E-17**

**avgn8.csv --> 2.918169523325123E-17**

**avgn7.csv --> 2.828697720288476E-17**

**avgn17.csv --> 1.9802889427055372E-17**

**avgn18.csv --> 1.9528252872533918E-17**

**avgn10.csv --> 1.570842099456239E-17**

**avgn15.csv --> 1.5612511283791264E-17**

**avgn20.csv --> 1.518230420046021E-17**

**avgn6.csv --> 1.4965316458195768E-17**

**avgn3.csv --> 7.852142418174323E-18**

**avgn2.csv --> 7.71384848457889E-18**

**avgn4.csv --> -6.983530731754951E-19**

**avgn5.csv --> -1.920861223472318E-18**

**n17.csv --> -8.264542567679436E-18**

**n18.csv --> -8.264542567679436E-18**

**n20.csv --> -8.264542567679436E-18**

**n11.csv --> -1.0676480053545335E-17**

**n12.csv --> -1.0676480053545335E-17**

**n13.csv --> -1.0676480053545335E-17**

**n14.csv --> -1.0676480053545335E-17**

**n15.csv --> -1.0676480053545335E-17**

**n7.csv --> -1.3078562887331573E-17**

**n9.csv --> -1.3877823969446994E-17**

**n3.csv --> -1.777167747221967E-17**

**n6.csv --> -1.9554941155858217E-17**

**avgn1.csv --> -2.1602059005923655E-17**

**n10.csv --> -2.261083788235102E-17**

**n16.csv --> -2.2853238474863488E-17**

**n2.csv --> -2.6525132286803855E-17**

**n1.csv --> -2.6979147948370742E-17**

**n4.csv --> -2.849796003824365E-17**

**n19.csv --> -3.0892719249372784E-17**

**n5.csv --> -3.116754399073666E-17**

**n8.csv --> -3.160115028172949E-17**

**avgn14.csv --> -5.551115123125783E-16**

**Latent semantic - 3**

**diffn10.csv --> 0.11281757620409122**

**diffn6.csv --> 0.1120053032916703**

**diffn12.csv --> 0.10309594098609179**

**diffn17.csv --> 0.1004256253924577**

**diffn7.csv --> 0.10031392984133525**

**diffn14.csv --> 0.09952599279475698**

**diffn11.csv --> 0.09930642016121137**

**diffn15.csv --> 0.09919669772408009**

**diffn13.csv --> 0.09918043356679157**

**diffn20.csv --> 0.09712193949273912**

**diffn18.csv --> 0.09668349283140619**

**diffn8.csv --> 0.09665587449953211**

**diffn9.csv --> 0.07663275901780826**

**diffn2.csv --> 0.021354774082194574**

**diffn3.csv --> 0.019940209008472886**

**diffn4.csv --> 0.015441190241888855**

**diffn5.csv --> 0.007788805313248694**

**avgn14.csv --> 5.208338902635392E-15**

**n5.csv --> 2.057707303160239E-15**

**n19.csv --> 1.3788883229670645E-15**

**avgn2.csv --> 1.1774895108566464E-15**

**avgn7.csv --> 8.106579643674117E-16**

**n4.csv --> 6.093216209368535E-16**

**n1.csv --> 5.251875323519783E-16**

**avgn16.csv --> 4.75139307615956E-16**

**n7.csv --> 3.608224830031759E-16**

**avgn10.csv --> 3.476449009073124E-16**

**n17.csv --> 2.820009850634797E-16**

**n18.csv --> 2.820009850634797E-16**

**n20.csv --> 2.820009850634797E-16**

**avgn20.csv --> 1.810346577507671E-16**

**n11.csv --> 1.2324668195728972E-16**

**n12.csv --> 1.2321957690297758E-16**

**n13.csv --> 1.2321957690297758E-16**

**n14.csv --> 1.2321957690297758E-16**

**n15.csv --> 1.2321957690297758E-16**

**avgn4.csv --> 1.1102230246251565E-16**

**avgn1.csv --> -7.113106377440163E-17**

**n2.csv --> -1.0581813203458523E-16**

**avgn15.csv --> -1.4079360646276565E-16**

**avgn5.csv --> -1.6653345369377348E-16**

**avgn19.csv --> -1.8462989405847166E-16**

**avgn6.csv --> -2.662800535624399E-16**

**avgn13.csv --> -2.7149397234075346E-16**

**avgn18.csv --> -2.7401124428273104E-16**

**n9.csv --> -4.456341979458545E-16**

**avgn17.csv --> -6.592873465811315E-16**

**avgn9.csv --> -6.731811288962497E-16**

**avgn8.csv --> -6.925883477837402E-16**

**n3.csv --> -8.789355961796863E-16**

**n16.csv --> -9.21436321341118E-16**

**n6.csv --> -1.1272992088418032E-15**

**avgn12.csv --> -1.3352375705162955E-15**

**n10.csv --> -1.4653534462227835E-15**

**avgn11.csv --> -1.7508796413876902E-15**

**n8.csv --> -1.7535885937780549E-15**

**avgn3.csv --> -1.8214596497756474E-15**

**diffn1.csv --> -0.12624707170224875**

**diffn16.csv --> -0.5485454733768544**

**diffn19.csv --> -0.7433152835664172**

**Task 3d**

**Input**

**Input Data:  20 word files + 20 average files + 20 difference files generated for given sample files**

**Query Data: query\_word\_file.csv**

**Number of Latent Semantics: 4**

**Output**

**Enter number of latent semantics : 4**

**Enter number of top documents to be retrieved : 5**

**task 3a fileindex size60**

**n11.csv->3.903343118930666E10**

**n12.csv->3.903343118930666E10**

**n13.csv->3.903343118930666E10**

**n14.csv->3.903343118930666E10**

**n15.csv->3.903343118930666E10**

**Task 3b**

**Input**

**Input data: 20 word files**

**Number of topics to be generated: 4**

**Output**

**Topic - 0**

**n19.csv --> 1150.0**

**n1.csv --> 1089.0**

**n16.csv --> 1076.0**

**n20.csv --> 1008.0**

**n9.csv --> 984.0**

**n11.csv --> 974.0**

**n5.csv --> 971.0**

**n14.csv --> 967.0**

**n17.csv --> 962.0**

**n15.csv --> 931.0**

**n12.csv --> 928.0**

**n18.csv --> 925.0**

**n2.csv --> 909.0**

**n10.csv --> 900.0**

**n13.csv --> 898.0**

**n6.csv --> 894.0**

**n8.csv --> 877.0**

**n4.csv --> 875.0**

**n3.csv --> 873.0**

**n7.csv --> 869.0**

**Topic - 1**

**n8.csv --> 1156.0**

**n6.csv --> 1130.0**

**n4.csv --> 1114.0**

**n2.csv --> 1111.0**

**n18.csv --> 1099.0**

**n11.csv --> 1087.0**

**n17.csv --> 1074.0**

**n5.csv --> 1066.0**

**n7.csv --> 1031.0**

**n3.csv --> 1023.0**

**n12.csv --> 1017.0**

**n1.csv --> 999.0**

**n14.csv --> 967.0**

**n10.csv --> 946.0**

**n9.csv --> 945.0**

**n15.csv --> 937.0**

**n20.csv --> 920.0**

**n13.csv --> 919.0**

**n16.csv --> 900.0**

**n19.csv --> 890.0**

**Topic - 2**

**n10.csv --> 1423.0**

**n13.csv --> 1271.0**

**n14.csv --> 1264.0**

**n3.csv --> 1263.0**

**n5.csv --> 1256.0**

**n9.csv --> 1255.0**

**n6.csv --> 1232.0**

**n20.csv --> 1230.0**

**n17.csv --> 1216.0**

**n4.csv --> 1203.0**

**n15.csv --> 1197.0**

**n12.csv --> 1189.0**

**n19.csv --> 1186.0**

**n7.csv --> 1152.0**

**n11.csv --> 1141.0**

**n16.csv --> 1138.0**

**n18.csv --> 1120.0**

**n2.csv --> 1104.0**

**n8.csv --> 1097.0**

**n1.csv --> 1063.0**

**Topic - 3**

**n7.csv --> 1232.0**

**n15.csv --> 1219.0**

**n13.csv --> 1196.0**

**n16.csv --> 1170.0**

**n2.csv --> 1160.0**

**n8.csv --> 1154.0**

**n12.csv --> 1150.0**

**n18.csv --> 1140.0**

**n1.csv --> 1133.0**

**n20.csv --> 1126.0**

**n3.csv --> 1125.0**

**n9.csv --> 1100.0**

**n4.csv --> 1092.0**

**n14.csv --> 1086.0**

**n11.csv --> 1082.0**

**n19.csv --> 1058.0**

**n17.csv --> 1032.0**

**n6.csv --> 1028.0**

**n10.csv --> 1015.0**

**n5.csv --> 991.0**

**Task 3e**

**Input**

**Input Data: 20 word files**

**Query: query\_word\_file.csv**

**Output:**

**Enter number of top documents to be retrieved : 5**

**n10.csv->1.01839054755E11**

**n3.csv->1.01572682151E11**

**n13.csv->1.01494128053E11**

**n4.csv->1.01396845345E11**

**Task 3c**

**Input**

**Input Data: 4 word files**

**Number of latent semantics: 4**

**Similarity Measure selected : Task 1f**

**Output:**

**Latent semantic - 0**

**n1.csv --> -0.44587404562217514**

**n5.csv --> -0.4473734477735393**

**n4.csv --> -0.4475543428112737**

**n2.csv --> -0.4476317105187262**

**n3.csv --> -0.4476318751346684**

**Latent semantic - 1**

**n2.csv --> 0.3532971362992029**

**n3.csv --> 0.26014471204587214**

**n4.csv --> 0.19998591943569466**

**n5.csv --> 0.05740199351962619**

**n1.csv --> -0.874194775461845**

**Latent semantic - 2**

**n3.csv --> 0.5557182586669432**

**n2.csv --> 0.2033677275138601**

**n1.csv --> 0.06256544368569335**

**n5.csv --> -0.018076842830037035**

**n4.csv --> -0.8034784237300584**

**Latent semantic - 3**

**n5.csv --> 0.7564675294949008**

**n3.csv --> 0.12784298892168056**

**n4.csv --> -0.09651184302252683**

**n1.csv --> -0.18008195033658034**

**n2.csv --> -0.6080041137384026**

**Task 3c**

**Input:**

**Input Data : 20 word files**

**Number of latent semantics: 4**

**Similarity Measure: Task1c**

**Output:**

**Latent semantic - 0**

**n1.csv --> -0.08557777708977227**

**n10.csv --> -0.09062369677531679**

**n8.csv --> -0.0977282259418928**

**n6.csv --> -0.10405206198606877**

**n9.csv --> -0.10808584919995641**

**n7.csv --> -0.11060188413301801**

**n5.csv --> -0.12210985670319456**

**n16.csv --> -0.12828135602369634**

**n4.csv --> -0.13385149994138046**

**n2.csv --> -0.15844170674102653**

**n3.csv --> -0.1734175674461707**

**n19.csv --> -0.20454368135834977**

**n17.csv --> -0.2556495060155664**

**n18.csv --> -0.25564950601556646**

**n20.csv --> -0.25564950601556646**

**n11.csv --> -0.34577510679379286**

**n12.csv --> -0.34577510679379286**

**n13.csv --> -0.34577510679379286**

**n14.csv --> -0.34577510679379286**

**n15.csv --> -0.34577510679379286**

**Latent semantic - 1**

**n19.csv --> 0.32880436340100205**

**n3.csv --> 0.2714568571034192**

**n16.csv --> 0.2550241855460837**

**n20.csv --> 0.22404655849059388**

**n17.csv --> 0.22404655849059385**

**n18.csv --> 0.22404655849059385**

**n2.csv --> 0.21147675968751348**

**n5.csv --> 0.1948000102139859**

**n4.csv --> 0.18330902928660792**

**n1.csv --> 0.17377073208494984**

**n9.csv --> 0.16445421218894554**

**n7.csv --> 0.14514564112538042**

**n8.csv --> 0.13174656867999748**

**n10.csv --> 0.08990850009242662**

**n6.csv --> 0.05596327340324643**

**n12.csv --> -0.2754697761407826**

**n13.csv --> -0.2754697761407826**

**n14.csv --> -0.2754697761407826**

**n15.csv --> -0.27546977614078266**

**n11.csv --> -0.27546977614078294**

**Latent semantic - 2**

**n20.csv --> 0.4391433778256057**

**n17.csv --> 0.43914337782560564**

**n18.csv --> 0.43914337782560564**

**n19.csv --> 0.016034692744455514**

**n6.csv --> -0.015051521323866413**

**n11.csv --> -0.056810290659564444**

**n12.csv --> -0.056810290659564444**

**n13.csv --> -0.056810290659564444**

**n15.csv --> -0.05681029065956457**

**n14.csv --> -0.05681029065956458**

**n10.csv --> -0.08941294280682344**

**n8.csv --> -0.13054574201876906**

**n7.csv --> -0.13977770838881723**

**n9.csv --> -0.1829632052435359**

**n4.csv --> -0.18835031470433036**

**n2.csv --> -0.2016453469568236**

**n5.csv --> -0.22630808203196479**

**n1.csv --> -0.2344193774678594**

**n3.csv --> -0.23872792632845505**

**n16.csv --> -0.2957919305474483**

**Latent semantic - 3**

**n10.csv --> 0.48553428808428667**

**n9.csv --> 0.4346341728304895**

**n4.csv --> 0.34246821569515085**

**n2.csv --> 0.2701097294717476**

**n6.csv --> 0.053129656247452056**

**n3.csv --> 0.025439189730581874**

**n18.csv --> -0.011167061846767662**

**n17.csv --> -0.011167061846767676**

**n20.csv --> -0.011167061846767676**

**n14.csv --> -0.013856567339911252**

**n11.csv --> -0.013856567339911391**

**n12.csv --> -0.013856567339911391**

**n13.csv --> -0.013856567339911419**

**n15.csv --> -0.013856567339911474**

**n19.csv --> -0.0680585826581206**

**n5.csv --> -0.14018923076707407**

**n1.csv --> -0.16630836050959252**

**n8.csv --> -0.3163492494248559**

**n7.csv --> -0.3278157752984548**

**n16.csv --> -0.34758928251230853**

**Task 3f**

**Input: 20 word files**

**Input Query: query\_word\_file.csv**

**Similarity Measure: Task1c**

**Number of latent semantics: 4**

**Output:**

**n16.csv->0.40787904203234865**

**n1.csv->0.2766782872822218**

**n5.csv->0.27341602613033394**

**n7.csv->0.2640713745117726**

**n3.csv->0.2508864739552982**

**Task 3c**

**Input:  20 word files**

**Number of latent semantics: 4**

**Similarity Measure: Task1f**

**Output:**

**Latent semantic - 0**

**n16.csv --> -0.22182136550530626**

**n19.csv --> -0.2220153539264395**

**n1.csv --> -0.2228151495610734**

**n10.csv --> -0.22356427324716766**

**n5.csv --> -0.22356453325071937**

**n8.csv --> -0.22357733788512138**

**n7.csv --> -0.22359023824083493**

**n9.csv --> -0.2236289284483651**

**n4.csv --> -0.22365493651468313**

**n2.csv --> -0.2236936193303869**

**n3.csv --> -0.22369370055785828**

**n6.csv --> -0.22378383938389262**

**n17.csv --> -0.22408119546567326**

**n18.csv --> -0.22408119546567348**

**n20.csv --> -0.22408119546567418**

**n11.csv --> -0.2240941228955807**

**n12.csv --> -0.2240941228955807**

**n13.csv --> -0.22409412289558098**

**n14.csv --> -0.22409412289558098**

**n15.csv --> -0.22409412289558098**

**Latent semantic - 1**

**n19.csv --> 0.6408449053107083**

**n16.csv --> 0.631506312929941**

**n1.csv --> 0.16136583127267365**

**n5.csv --> 0.00989898749154481**

**n4.csv --> -0.015560080003976774**

**n8.csv --> -0.02193005558411556**

**n3.csv --> -0.02563539719560754**

**n7.csv --> -0.029990064881366973**

**n2.csv --> -0.048999363397973984**

**n9.csv --> -0.05520387231358997**

**n10.csv --> -0.0602355570940164**

**n6.csv --> -0.12719072252371488**

**n14.csv --> -0.13078633563526718**

**n13.csv --> -0.1307863356352672**

**n15.csv --> -0.13078633563794634**

**n11.csv --> -0.1307863356419482**

**n12.csv --> -0.1307863356431971**

**n20.csv --> -0.13079437567389468**

**n18.csv --> -0.13079437598082078**

**n17.csv --> -0.13079437599197088**

**Latent semantic - 2**

**n1.csv --> 0.5283908013002472**

**n10.csv --> 0.2948650831538085**

**n6.csv --> 0.2853939192185794**

**n9.csv --> 0.23349978683822212**

**n7.csv --> 0.13393478854607954**

**n4.csv --> 0.13359093923359341**

**n5.csv --> 0.13206104788873674**

**n2.csv --> 0.09539226516732058**

**n16.csv --> 0.0587774020002729**

**n8.csv --> 0.033706983693792165**

**n3.csv --> -0.03705380267529305**

**n17.csv --> -0.1828194920537165**

**n18.csv --> -0.1828194924756689**

**n20.csv --> -0.1828195041958274**

**n15.csv --> -0.19224789552462054**

**n14.csv --> -0.19224789574947992**

**n13.csv --> -0.19224789574947998**

**n11.csv --> -0.19224789575811752**

**n12.csv --> -0.19224789593908487**

**n19.csv --> -0.3801735567727711**

**Latent semantic - 3**

**n6.csv --> 0.45103485166248986**

**n10.csv --> 0.3574019853095831**

**n8.csv --> 0.2527936941181703**

**n19.csv --> 0.23083715226090057**

**n20.csv --> 0.10344587225631574**

**n17.csv --> 0.1034458585150656**

**n18.csv --> 0.10344585770330861**

**n9.csv --> 0.013397413191013802**

**n1.csv --> -9.74068829618836E-5**

**n7.csv --> -0.007991346354923781**

**n13.csv --> -0.01709093269169777**

**n14.csv --> -0.017090932691697782**

**n12.csv --> -0.017090932711180437**

**n11.csv --> -0.017090933219614336**

**n15.csv --> -0.01709093367886317**

**n16.csv --> -0.10191048367777983**

**n5.csv --> -0.30704224349777154**

**n4.csv --> -0.356483762631431**

**n3.csv --> -0.37560359028056867**

**n2.csv --> -0.38080772505318006**

**Matlab execution done**

**Task 3f**

**Input:**

**Input Data: 20 word files**

**Query : query\_word.csv**

**Number of latent semantics: 4**

**Similarity Measure: Task 1f**

**Output:**

**n2.csv->0.7218116934177407**

**n4.csv->0.688818868320357**

**n3.csv->0.62773480256632**

**n5.csv->0.5366396279791015**

**n16.csv->0.22359473784410122**

**Results Comparison Chart:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sno** | **Task** | **Input** | **Latent Semantics**  **/Topics - r** | **Query** | **Dimensionality Reduction Technique** | **Results(Top 5)** |
| **1** | **3a, 3d** | **20 Word files for sample data** | **4** | **Query\_word\_file** | **SVD** | **n11.csv->4.4840535175331085E10**  **n12.csv->4.4840535175331085E10**  **n13.csv->4.4840535175331085E10**  **n14.csv->4.4840535175331085E10**  **n15.csv->4.4840535175331085E10** |
| **2** | **3b, 3e** | **20 Word files for sample data** | **4** | **Query\_word\_file** | **LDA** | **n10.csv->1.01969346581E11**  **n13.csv->1.01749580828E11**  **n3.csv->1.01546074601E11**  **n12.csv->1.01423057594E11**  **n6.csv->1.01336972602E11** |
| **3** | **3c, 3f** | **20 Word files for sample data** | **4** | **Query\_word\_file** | **SVD on file-file similarity Matrix (Task1c)** | **n16.csv->0.40787904203234865**  **n1.csv->0.2766782872822218**  **n5.csv->0.27341602613033394**  **n7.csv->0.2640713745117726**  **n3.csv->0.2508864739552982** |
| **4** | **3c, 3f** | **20 Word files for sample data** | **4** | **Query\_word\_file** | **SVD on file-file similarity Matrix (Task1f)** | **n2.csv->0.7218116934177407**  **n4.csv->0.688818868320357**  **n3.csv->0.62773480256632**  **n5.csv->0.5366396279791015**  **n16.csv->0.22359473784410122** |

**Results Comparison Chart:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sno** | **Task** | **Input** | **Latent Semantics**  **/Topics - r** | **Query** | **Dimensionality Reduction Technique** | **Results(Top 5)** |
| **1** | **3a, 3d** | **20 Averagefiles for sample data** | **4** | **Query\_avg\_file** | **SVD** | **avgn15.csv->3.4143736234527622E10**  **avgn20.csv->3.4143734992725754E10**  **avgn11.csv->3.412715055561787E10**  **avgn14.csv->3.4110565757860916E10**  **avgn12.csv->3.4093979636159664E10** |
| **2** | **3b, 3e** | **20 Averagefiles for sample data** | **4** | **Query\_avg\_file** | **LDA** | **avgn12.csv->9.7365850362E10**  **avgn6.csv->9.7360233206E10**  **avgn10.csv->9.7185901042E10**  **avgn15.csv->9.6878977182E10**  **avgn7.csv->9.6460818908E10** |
| **3** | **3c, 3f** | **20 Averagefiles for sample data** | **4** | **Query\_word\_file** | **SVD on file-file similarity Matrix (Task1c)** | **avgn20.csv->0.3404368853167555**  **avgn17.csv->0.334079066641776**  **avgn18.csv->0.26392206187946193**  **avgn19.csv->0.1846326333073577**  **avgn10.csv->0.09365445166286304** |

**Conclusion:**

* Several dimensionality reduction techniques were used to identify the top k-significant dimensions and used them as search space for processing the input query documents.
* Dimensionality reduction techniques used were SVD, LDA, Fast Map.
* Various file similarity techniques like Dynamic Time Warping, Dot product similarity and word word similarity were to used to find the file similarities.
* For the given query document, the top k most relevant results were found out by converting the original query document into reduced space and then finding out the matching documents by using any of the above similarity measures.

**Bibliography**

1. C. Faloutsos, K. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of traditional and Multimedia Datasets," in Proceedings of 1995 ACM SIGMOD, SIGMOD RECORD (June 1995), vol.24, no.2, p 163-174.
2. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3 (March 2003), 993-1022.
3. [G. W. Furna](http://dl.acm.org/author_page.cfm?id=81100649937&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [S. Deerwester](http://dl.acm.org/author_page.cfm?id=81100019458&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [S. T. Dumais](http://dl.acm.org/author_page.cfm?id=81100134501&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [T. K. Landauer](http://dl.acm.org/author_page.cfm?id=81100552367&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [R. A. Harshman](http://dl.acm.org/author_page.cfm?id=81541660056&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [L. A. Streeter](http://dl.acm.org/author_page.cfm?id=81100464482&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218), [K. E. Lochbau](http://dl.acm.org/author_page.cfm?id=81545590756&coll=DL&dl=ACM&trk=0&cfid=451008156&cftoken=18007218)m .“Information retrieval using a singular value decomposition model of latent semantic structure”, in Proceedings of SIGIR '88 Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval, Pages 465 - 480.
4. Structure Learning in Random Fields for Heart Motion Abnormality Detection. M. Schmidt, K. Murphy, G. Fung, R. Rosales. CVPR'08.
5. Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm. M. Schmidt, E. van den Berg, M. Friedlander, K. Murphy. AISTATS'09.
6. Increased Discrimination in Level Set Methods with Embedded Conditional Random Fields. D. Cobzas, M. Schmidt. CVPR'09.
7. Modeling Discrete Interventional Data using Directed Cyclic Graphical Models. M. Schmidt, K. Murphy. UAI'09.
8. Causal Learning without DAGs. D. Duvenaud, D. Eaton, K. Murphy, M. Schmidt. JMLR W&CP'10
9. Generalized Fast Approximate Energy Minimization via Graph Cuts: Alpha-Expansion Beta-Shrink Moves. M. Schmidt, K. Alahari. UAI'11.
10. Hybrid Deterministic-Stochastic Methods for Data Fitting. M. Friedlander, M. Schmidt, SISC'12.
11. E. Keogh, C. A. Ratanamahatana, Exact indexing of dynamic time warping, Knowledge and Information Systems 7 (3) (2005) 358-386

**Appendix:**

George Thomas - Task 1a, 1b, 2

Aneesh Shastry - Task 1c, 1d, 1e

Anil Kuncham - Task 3a,3b,3c,3d-f

Akshay Chitneni - Task 4a, 4b

Karthik Chandrasekar - Task 1f,1g,1h