CSE 515: Multimedia and Web Databases Phase 3 Report

GROUP 6

Shreekrishna Prasad Sethuraman
ASU ID:1213126887

Soundarya Sankarasubramanian
ASU ID:1213119620

Vishnunarayanan Ramasubramanian
ASU ID:1213169553

Dipika Ramaswamy
ASU ID:1213237517

Krithika Narayanan
ASU ID:1213134817

Eswaran Rathinam
ASU ID:1211343299

ABSTRACT:

The goal of this phase is to apply different techniques to implement a Multi-Dimensional Index Structures for recommendation and retrieval, along with classification. For recommendation, Locality Sensitive Hashing and a ranking algorithm like PageRank is used and for retrieval, dimensionality reduction techniques like SVD, PCA and LDA are used. For movie classification, techniques like k-NN, SVM and Decision Trees are used. The dataset given is an aggregation of IMDB and Movie-Lens data, containing user's ratings and tags for movies played by various actors. The aforementioned techniques are accordingly applied on different spaces of varying dimensions based on the application.

Keywords: Locality Sensitive Hashing, Nearest Neighbors, Decision Tree, SVM, Dimensionality reduction, PageRank, Tensor, Vector Spaces, Recommendation, Decomposition, Similarity, Random Walk with Restarts, Personalized PageRank.

1. INTRODUCTION

1.1 Terminology:

1. Singular Value Decomposition (SVD) - The singular value decomposition of a matrix A is the factorization of A into the product of three matrices A = UDV T where the columns of U and V are orthonormal, and the matrix D is diagonal with positive real entries.
2. Principal Component Analysis (PCA) - A variation of the SVD where the covariance matrix of A is used as input so as to get a latent semantic space that preserves the variance of the data.
3. Latent Dirichlet Allocation (LDA) - A probabilistic topic modelling technique that finds hidden topics in a generative statistical model to explain observed parameters. This model considers each document as a mixture of topics where the topic distribution is a Dirichlet distribution.
4. Locality Sensitive Hashing (LSH) - It is an algorithm for solving Nearest Neighbor Search problems in high dimensional spaces.
5. Support Vector Machines (SVM) - A classifier that maximizes the margin between the classes with the assumption that the classes are separable.

6. Classification – A supervised learning process which tries to label objects into a predetermined set of classes.
7. Supervised Learning – A learning process in which the model is given what kind of results are expected given a set of known result values for a set of data objects.
8. Multi Class and Two class classification – If the classification process has just two labels, then it is a two-class classification problem. Otherwise, it is a multi-class classification problem.
9. Support Vectors – Boundaries that effectively describe a given data space.
10. Convex optimization – A class of optimization problems that try to optimize a mathematical function with one minima.
11. Quadratic Programming – A technique to solve optimization problems where the objective function is quadratic in nature, with the constraints being linear.
12. PageRank (PR) - PageRank is an algorithm used to rank website pages according to their “importance” on the web. The “importance” of any web page can be described in terms of the pages that are linked to it and the importance of each of these linkages. “Good” pages point to and are pointed to by other good pages and have a higher PageRank score.
13. Personalized PageRank (PPR) - Personalized PageRank is a version of the PageRank algorithm where webpages are ranked with respect to a given seed set of web pages. PageRank scores are assigned to pages in the network based on their importance relative to the webpages in the seed set. This gives us the most relevant pages for a specific term search or user query operation.
14. Tensor: Tensors are multi-dimensional arrays. An r-way tensor can be visualized as having feature values across r dimensions in space. They are useful for embedding inter-dimensional relationships for arrays having more than two dimensions in vector spaces.
15. Tensor Decomposition- A n-way tensor can be decomposed to form ‘n’ factor matrices, with each factor matrix describing values across a single dimension in terms of latent semantics and a core tensor, which describes the relationships between the factor matrices.

1.2 Goal Description:

This phase has five tasks. Each task has several sub-tasks which apply various techniques on different data spaces.

1.2.1 Task 1

Given information about the movies watched by a user, five more movies should be recommended to the user using the following approaches, considering the order in which they have been watched:

- Using SVD or PCA
- Using Latent Dirichlet Allocation(LDA)
- Using tensor decomposition
- Using Personalized PageRank
- A combination of the methods described above

1.2.2 Task 2

After recommending a list of movies for the user, the user's feedback must be considered to improve the accuracy of matches from Task1. Then the list of suggestions after feedback must be displayed.

1.2.3 Task 3

Task 3A:

The goal of Task 3 sub task 'a' is to map each movie into a 500-dimensional space.

Task 3B:

The goal of Task 3 sub task 'b' is to implement a locality sensitive hashing tool with L layers and k Hash Functions.

Task 3C:

The goal of Task 3 sub task 'c' is to find R similar movies given a movie and R.

1.2.4 Task 4:

The goal is to implement an r-nearest neighbor based relevance feedback algorithm to improve the nearest neighbor matches.

1.2.5 Task 5:

The goal of this task is to do movie classification. Given a set of labeled movies, the objective of the task is to predict the label of the rest of the movies in the database using the three methods given in the specification: r-Nearest neighbors, Support Vector Machines or Decision trees.

1.3 Assumptions:

Task 1:

- Item to item similarity is used to recommend movies in the space of tags and hence we are assuming that the movies with close tag scores represent high similarity and be a good recommendation for the users.
- Also, the rating of a movie by the given user is also considered along with similarity, so that the recommendation will consider the movies liked by the user.
- While computing the recommendation score for each movie as mentioned in section 1.4, we are taking only the top 500 most similar movies for each movie due to memory and performance constraints. We assume that this will still give us the best recommendations.
- A subset of {users, movies, tags} was used to build a user-movie-tag tensor in Task 1b.
- CP decomposition was used to factorize the {user, movie, tag} tensor.
- The timestamps of the tags to the movies are taken while assigning tensor values. Instead of binary values, each non-zero entry in the tensor is assigned a max-min

normalized weight for the timestamp. The max-min normalized value of an entity is calculated as

$$Normalized\ timestamp\ value = \frac{currentTime - minTime}{maxTime - minTime}$$

where

minTime- represents the oldest timestamp
maxTime- represents the most recent timestamp
currentTime- represents the current timestamp

- A movie-movie similarity matrix was used to construct the transition matrix for personalized page-rank computation.
- The set of seed actors is not null.
- The number of iterations is taken as 1000 by which the steady state probabilities of all the nodes are considered to stabilize.
- The value of restart probability 'r' is taken as 0.15.
- The order in which the user watched the seed movies was considered while defining teleport vector values for those movies. The entry for each seed movie in the teleport matrix is (Rank of the movie as per the user/sum-of-ranks-of-seed-movies).
- The following libraries were used for the task: numpy, sktensor, pandas, sklearn.

Task 2:

- The relevant movies are considered as the movies labelled relevant by the user.
- The non-relevant movies are all the movies other than the relevant ones.

Task 3:

- TF-IDF Tag vector for each movie was calculated and was given as input to all the three subtasks.
- Singular Value Decomposition was used to map each movie into a 500-dimensional latent space.
- For hashing purposes, Euclidean distance measure was used.
- The following libraries were used for the task: Numpy, Scipy.

Task 4:

- For hashing purposes, Euclidean distance measure was used.
- The following libraries were used for the task: Numpy, Scipy.
- After receiving the feedback for the initially suggested list of movies, movies with both neutral and positive feedback were considered while displaying the revised list.

Task 5:

- A movie can only belong to one class/label, i.e, all labels are assumed to be mutually exclusive and non-overlapping.
- There are at least two or more labels provided. The system will not work with just one label provided as it can't train with one label.

1. 4 Description of the proposed solution/Implementation:

Task 1:

For all the tasks under task1, TF-IDF movie tag space is used. The movie to tag space is computed as mentioned below:

$$TFi,j = \frac{No\ of\ times\ tag\ ti\ occurs\ in\ movie\ mj}{Total\ no\ of\ tags\ in\ movie\ mj}$$

$$IDFi = \log\left[\frac{\text{Total no of movies}}{\text{No of movies with tag ti}}\right]$$

$$TF\_IDFi,j = TFi,j * IDF$$

where 'i 'corresponds to position of one tag and 'j ' corresponds to position of one movie.
The recommendation for tasks a, b, c and e are performed as below but for task d, the page-rank scores are used:

$$P_{u,i} = \frac{\sum_{k=1}^{m} sim(k,i) * r_{u,k} * 2^{-\lambda(t_k - t_i)}}{\sum_{k=1}^{m} sim(k,i)}$$

Where,

- $P_{u,I}$ is preference for the user u for an movie i.
- sim(k,i) is the similarity between movie k and movie i.
- $r_{u,k}$ is the rating for the user u for movie k.
- t_k is the timestamp for the movie by the user u.
- t_i is the recent timestamp by that user u.

While computing the above formula, due to performance and time constraints, only the top 500 most similar movies are taken into consideration. We assume that this will still give us the best recommendations.

The movie-movie similarity matrix is constructed by taking the matrix multiplication of the movie-tag matrix and its transpose.

In task 1a, SVD is applied on TFIDF tag space which is time weighted. The U matrix which contains the left singular vectors are multiplied with the eigen values and is used as the reduced space. From this, the item to item similarity is computed and the movies are recommended as per the formula mentioned above.
For principal component analysis, PCA is applied on TFIDF tag space after which is time weighted after mean subtraction. The U matrix which contains the left singular vectors are multiplied with the eigen values and is used as the reduced space. From this, the item to item similarity is computed and the movies are recommended as per the formula mentioned above.

In task 1b, a movie to tag matrix is constructed with each entry denoting the number of times a tag has been used for a movie. This is the equivalent of a document-term matrix which is then fitted into a LDA model that would generate 4 latent topics in the document and term-spaces, namely the movie and tag spaces. The movies are now represented in the reduced space of latent topics. From this, the similarity matrix is computed between movies and the movies are recommended as per the formula mentioned above.

In task 1c, a {user, movie, tag} tensor is created with 33 users,765 movies and 25 tags. This tensor is then decomposed through CP decomposition, with the number of latent topics k=4, to generate three factor matrices and a diagonal singular core matrix of order k*k*k. Now, when reconstruct the original tensor from the factor matrices and the core tensor values will

be updated with the influence of latent topics that were discovered. Given a user, we sort the tensor values across all movies and tags to find top five values corresponding to top five movie recommendations.

In task 1d, a movie-movie similarity matrix is created using the tag vectors for each of the movies. The tag vector for each of the movies are calculated using the TF-IDF method.
Constructing the transition matrix: To transform the movie-movie matrix into a transition matrix that is used in the PageRank algorithm, we need to column-normalize the entire matrix. This creates our transition matrix which describes the edges and their weights between pairs of nodes, in this case, the movies.

Computing the steady-state probability vector: In a random walk, the walker can either move to his neighbor or restart the walk teleporting to a random node in the graph. This can be represented as a mathematical equation.
Let

A - Transition matrix of size (n*n)
c– Restart probability factor, c £ (0, 1)
Vector $\bar{u}$ - steady-state probability vector of size (n*1)
Vector $\bar{v}$ - restart probability vector of size (n*1)

Then, the PPR scores are computed iteratively as

$$\bar{u} = [1\text{-}c]\, A\bar{u} + c\bar{v}$$

Since we need to personalize the PR scores with respect to the seed movies in a way that accounts for the order of user watching the movies, the entry for each seed movie in $\bar{v}$ will be

$$Entry\ for\ seed\ movie\ in\ teleport = \frac{Rank\ of\ seed\ movie\ as\ per\ user's\ viewing}{Sum\ of\ ranks\ of\ all\ the\ seed\ movies}$$

Nodes that are seeds will start with initial PR values of 1 in $\bar{u}$. When $\bar{u}$ converges and stabilizes, the list of values in $\bar{u}$ gives the PPR scores for each node with respect to the given seed set. The list of five most related movies to the given seed set can be obtained by sorting the PPR scores in the decreasing order and fetching the movies for the first five values.

In Task 1e, the combination of measures is fed from the user's input. Each of this measure is multiplied with the movie scores from each of the methods and the scores are added to get the final scores. The movies are recommended as per the maximum scores.

Task 2:
Results of recommendation are shown to the user upon which the user gives a list of relevant movies and irrelevant from the results. From this, a new query is calculated using the below formula:

$$q_j = \log\left(\frac{\frac{r_j+0.5}{R-r_j+0.5}}{\frac{m_{1,j}-r_{1,j}+0.5}{M-m_{1,j}-R+r_{1,j}+0.5}}\right) * \left|\frac{r_{1,j}}{R} - \frac{m_{1,j}-r_{1,j}}{M-R}\right|$$

Where,

- r_j – number of relevant movies containing the tag t_j.
- $m_{1,j}$ – number of movies containing the tag tj.
- R – number of relevant movies.
- M− number of movies.

This new query is used to compute the similarity between all the other movies in the database and the most similar 5 movies which are not suggested to the user before are suggested. Cosine similarity is used to find the similarity.
This is as shown below.

$$sim(\vec{u}, \vec{v})\vec{u} * \vec{v}/(\|\vec{u}\| * \|\vec{v}\|)$$

Where,

- u and v are movie vectors

Task 3:

For task 3a, a movie-tag matrix is constructed by using the time-weighted TF-IDF space of tags. Those tags that are not present for a movie are given a score of zero for uniformity. This movie tag matrix is then reduced to a 500-dimensional latent space using SVD. This 500-dimensional latent space is then fed as input for task 3b and task 3c.

In task 3b, Locality Sensitive Hashing is implemented using L2 distance. For each hash function, a random vector is created and each of the input movie tag vectors are projected on the random vector and divided by a constant W=4. Depending on the number of hash functions and layers provided as input, it is possible that the projected values may get skewed to very small decimal values (in the range of 10^{-9}). In order to avoid that, we normalize the values so that they fall in the range of 0 to 1 and then these values are scaled by 100 since the random vector is split based on the value of W. We finally bin the projected values into their respective bins.

In task 3c, the movie given as input is searched across bins and the bin containing the movie is used to retrieve the R similar movies. Since all the movies in the bin in which the input movie is present are similar, initially we retrieve R movies in a random manner and display them. This will work if the number of movies (lenBin) in the bin is greater than R. Otherwise, for (R-lenBin) movies, we proceed to the next bin. For the suggested list of movies, user feedback is received and displayed which is used as input for Task 4.

Task 4:

In this task, the movie given as input is searched across bins and the bin containing the movie is used to retrieve the R similar movies. Since all the movies in the bin in which the input movie is present are similar, initially we retrieve R movies in a random manner and display them. This will work if the number of movies (lenBin) in the bin is greater than R. Otherwise, for (R-lenBin) movies, we proceed to the next bin. User feedback for initially suggested list of movies is received. Based on the feedback received, we move more towards movies which were provided with positive feedback and suggest a revised list of movies for the user.

Task 5:

The objective of this task is to perform movie classification using three approaches: k-Nearest Neighbors, n-ary Support Vector Machine and Decision Tree.
The k-Nearest neighbors approach works as follows. The training phase of the k-NN algorithm is simply loading the given input dataset, along with the labels into the local memory space of the program. Then, each movie in the database that is not in the training set is processed to get a prediction. For a particular movie, the distance between this and every movie in the training set is computed and the nearest k movies are calculated. Each of these movies already have a label, since they are from the training set. Out of these k labels, the movie in the test set is assigned the label which is the most common. This process is called majority voting. This voted results is then assigned to the test movie as the final label. This process is repeated for all the movies in the test set and the results are stored. To compute the distance, the Euclidean distance measure is used. The Euclidean distance measure between two data points x and y with m features is described by the following equation:

$$Dist(x, y) = \sqrt{\sum_{i=0}^{m}(x_i - y_i)^2}$$

All the movies in the database are thus classified using this approach. The most important point to note here is that the k-NN approach works on the assumption that all objects that are labelled together tend to cluster in the feature space. Depending on the degree to which the dataset follows this property, the classification error is thus impacted.

The second approach used in this task for classification is the Support Vector Machine (SVM). The SVM is a Max-margin classifier that assumes that the given dataset is separable, i.e, all objects that belong to a label can essentially be separated from the rest of the dataset with a boundary. The SVM algorithm strives to mathematically determine this boundary. For this task, a linear SVM is constructed, which assumes that the boundary to be constructed is described by a straight line given by the equation:

$$y = wx + b$$

where w is a (m*1) vector that describes the weights vector along each of the m features. The variable ‘b’ is called the bias and it determines the position of the boundary line with respect to the origin of the feature space. X is the input data point’s feature vector and y would thus be the prediction of the data point as described by the SVM. Given a set of input movies and labels, and considering the tag vector space as the feature space for the movies, this algorithm can thus make predictions for the rest of the movies in the dataset. A hard-margin SVM is one that makes a prediction based on the exact position of the test object with respect to the boundary. This does come with a low error, but at the cost of extra computation. Moreover, this assumes that the given dataset is perfectly linearly separable, which may not be the case always. Instead, a soft-margin SVM is used which strives to find a boundary with some level of tolerable error. In this task, a soft-margin n-ary SVM is implemented. This concept of additional error is described mathematically by a set of “slack” variables, which tell the SVM the amount of error it can afford to make. These variables give a limit on how far a point can be misrepresented by the margin. These slack variables are always greater than or equal to zero. To avoid overfitting, the concept of a regularization parameter, C, is also used. By giving this leeway to the SVM, it is ensured that the SVM finds a good solution overall rather than trying to find one that represents the training set to a

higher extent and end up overfitting. All of the above conditions hold on the parameters of the SVM with the additional objective of minimizing the error to train better. This constraint on the classification error can be demonstrated using the following equation:

$$y_i(w * x_i + b) \geq (1 - \varepsilon_i),\ i = 1..n$$

where $\varepsilon_i \geq 0$ are the slack variables and n being the number of data points. The cost function in the SVM is described as follows:

$$Cost = \frac{1}{2} * (w^T * w) + C * \sum_i \quad \varepsilon_i$$

The SVM can thus be described as an optimization problem with the above constraints with the goal being to find w, b and ε that satisfy these constraints.

In this particular task, this has been modelled as a quadratic programming optimization problem, which is described by the following formulation:

$$Min: \frac{1}{2} * (z^T * Q * z) + (C * z),$$

Subject to the constraints:

$$Gz \leq h \ and \ Az = b$$

Applying the above general quadratic form to the SVM equation gives the following:

$$Min: \frac{1}{2} * (w^T * w) + C * \sum_i \quad \varepsilon_i,$$

Subject to the constraints:

$$y_i(w * x_i + b) \geq (1 - \varepsilon_i) \ and \ \varepsilon_i \geq 0, i = 1..n$$

Applying these equations to the above general quadratic form gives the following:

$$z = \begin{bmatrix} w \\ \epsilon \\ b \end{bmatrix}$$

$$Q = \begin{bmatrix} I_{(m*m)} & 0_{(m*(n+1))} \\ 0_{((n+1)*m)} & 0_{((n+1)*(n+1))} \end{bmatrix}$$

$$C = \begin{bmatrix} 0_{((m+1)*1)} \\ C_{(n*1)} \end{bmatrix}$$

$$G = (-1) * \begin{bmatrix} y_i * x_{i1} & y_i * x_{i2} & y_i * x_{i3} & \dots & y_i * x_{im} & 0_{1*i} & 1 & 0_{1*(n-i)} & y_i \end{bmatrix}$$

Here, since the SVM has no equality constraint, A and b are zeros and are thus not included. Plugging the rest of the values into the general quadratic form will yield the original SVM primal equation, along with the appropriate constraints. The goal thus reduces to finding the variable 'z' that satisfy these constraints. Once this is done, predictions can be done easily by computing $y = (w * x + b)$ for all points in the test data set. The '-' sign in the G matrix is

to denote that the constraint in the SVM is a ≥, while the general quadratic form has a ≤ in that place. This follows from the fact that if $A \leq B$, then $-A \geq -B$. The rest of the derivation is self-explanatory. To implement this task, the Quadratic Programming solver from the 'cvxopt' package in python was used. Once the solver converged to a solution based on the above constraints, predictions were made. This describes the implementation of a single two-class SVM.

However, in the task, since it is required to classify into n labels, an n-ary SVM is constructed using the one-vs-all approach. Here, for n given labels, n two-class SVMs are run based on each label versus the rest. Then, a simple majority voting of all the predictions of the SVMs are performed and if the SVM is still undecided about the prediction in the worst case, a random label is assigned to the object. Thus, each movie in the test set is evaluated and labelled using an n-ary SVM implemented using the above method and the resultant labels are stored as output.

The third approach to movie classification in this task is decision trees. Here a hierarchy of features is assumed to exist and predictions are made based on this taxonomy. A tree is constructed based on this hierarchy using the given set of input objects and this is the process of training. While constructing this tree, at each split point, a decision needs to be made on which feature to choose. To make this decision, the feature with the maximum information gain is chosen. Information gain is a measure of the degree to which a particular split will produce homogenous children in the tree. It is calculated by computing the difference in entropy of the parent dataset and the entropy of the children sub-splits. The entropy of a particular dataset containing J labels is described by the following equation:

$$H = -\sum_{i=1}^{J} \quad (p_i * log\ p_i\ ),$$

where p_i represents the fraction of the i^{th} label in the training set. The information gain is described as the difference between the entropy of the parent and the sum of entropies of the respective children generated after the split. In the decision tree, the split that produces the maximum information gain across all features is chosen at each level. Once this is chosen, the node is split and the above process is applied iteratively on the children produced due to the split. A node is made a terminal node if it is homogenous, i.e, if all the points in the node have a single label. The tree is recursively built until this condition is reached.

For this implementation, since it is possible that the leaves of the resultant tree may be underutilized, leading to a very long tree, a maximum depth of the tree is also taken as a hyper-parameter to the model, along with the minimum node size. If a node has equal to or fewer data points than the minimum size, it is made a terminal node even if it is not homogenous. A node is also made a terminal node if the current depth of the tree is more than the maximum depth allowed. This is a summary of how the training process in a decision tree works. For testing, the tree is simply traversed and predictions in a particular leaf are assigned as the final label to the test data point. This is done for all the data objects in the test dataset and the labels produced are stored as output.

2. System requirements and Execution Instructions:

2.1 System requirements:

- A Linux-based Operating System
- Python version 2.7 or greater with suitable packages installed
- MySQL server
- Python Packages required:
    - Numpy
    - SciPy
    - Sklearn
    - Pandas
    - Lda
    - Sktensor
    - cvxopt

2.2 Execution Instructions:

Task 1:

>python task1.py userid model

Model

0- SVD
1- PCA
2- LDA
3- Tensor Decomposition
4- PageRank
5- Combination of measures for each of the above methods.

Task 2:

Task 2 is followed by Task1 and hence 1 is entered for all the relevant movies and 0 for irrelevant/unlabeled movies in the order of recommendation.

Task 3a:

>python task3a.py

Task 3b and Task 3c:

>python task3bc.py <no of layers> <no of hashes> <input movie name> <no of neighbors>

Task 4:

>python task3bc.py <no of layers> <no of hashes> <input movie name> <no of neighbors>

Task 5:

The Task5.py implements three classifiers based on an input training set. The training set is provided by typing in the 'input_task5.txt' file using the following format:

> Movie_id Label

There is no limit on the number of entries in the text file. Once the training set is provided, the user must save the file before executing the program.

The program takes in a command-line argument that can take values 1,2 or 3 corresponding to the following methods used internally for classification.

Values for the argument:

1: Denotes the r-nearest neighbours algorithm
2: Denotes the Support Vector Machine algorithm
3: Denotes the decision tree algorithm

Once the classifier finishes execution, the output is grouped by label and written into either 'outputknn.txt', 'outputsvm.txt' or 'outputdtree.txt' corresponding to whether the algorithm used was r-nearest neighbours, support vector machines or the decision tree, respectively.

Example:

> python Task5.py 1

This runs the r-NN algorithm on the training dataset provided in 'input_task5.txt' and writes the output to 'outputknn.txt'.

4. Conclusion

Each of the aforementioned tasks are implemented according to the methods described in the implementation section and the outputs for each of the tasks are generated and attached in the folder Outputs.

5. Bibliography

[1] "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions" (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.
[2] Pan,Yang,Duygulu and Faloustos. Algorithm CCD: Cross-modal Correlation Discovery. In
[3] Automatic Multimedia Cross-modal Correlation Discovery(2004).
[4] David M. Blei,Andrew Y. Ng, Michael I. Jordan. Latent Dirichlet Allocation. Journal of Machine Learning Research.
[5] Virginia C. Klema and Alan J. Laub. The singular value decomposition: Its computation and some applications. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. AC-25, NO. 2, APFSL 1980
[6] Tomonari Masada, Senya Kiyasu, and Sueharu Miyahara. Latent Dirichlet Allocation In Comparing LDA with pLSI as a Dimensionality Reduction Method in Document Clustering.
[7] Steven P. Crain, Ke Zhou, Shuang-Hong Yang. Dimensionality reduction and topic modeling: from latent semantic indexing to latent dirichlet allocation and beyond.
[8] Karatzoglou,Amatriain,Baltrunas,Oliver.Multiverse Recommendation:N-dimensional Tensor Factorization for Context-aware Collaborative Filtering.
[9] Panagiotis Symeondis. Matrix and Tensor Decomposition in Recommender Systems.ACM Digital Library(2016).
[10] Vojislav Kecman, Ivana Hadzic. Support Vector Selection by Linear Programming (2000)
[11] Sebastian Maldonado, Juan Perez, Martine Labbe et.al. Feature Selection for Support Vector Machines via Mixed Integer Linear Programming (2013)
[12] Colin Campbell, Nello Cristianini. Simple Learning Algorithms for Training Support Vector Machines.

[13] F. Perez Cruz, P.L. Alarcon Diana, A. Navia-Vazquez, A. Artes-Rodriguez. Fast training of Support Vector Classifiers.
[14] Stephane Canu. SVM and Kernel Machine.
[15] Thorsten Joachims. Making Large-Scale SVM Learning Practical. (1998)
[16] Ginny Mak. The implementation of support vector machines using the sequential minimal optimization algorithm. (2000)
[17] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM.
[18] John.C.Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization.
[19] Tatjana Eitrich, Bruno Lang. Efficient optimization of support vector machine learning parameters for unbalanced datasets. (2005)
[20] Sebastian Nowozin. Improved Information Gain Estimates for Decision Tree Induction.
[21] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. Support Vector Machines: The linearly separable case. Introduction to Information Retrieval.
[22] Vojtech Franc. Support Vector Machine Classification: Application of Quadratic Programming and Lagrange duality.
[23] Jason Brownlee. How to Implement the Decision Tree from Scratch in Python.
Saed Sayad. Decision Tree- Classification.

[24] Collaborative Filtering Recommender Systems By Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan

[25] Thomas Hofmann. 2004. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*22, 1 (January 2004), 89-115. DOI=http://dx.doi.org/10.1145/963770.963774

6. Appendix

Task 1 and Task 2: Eswaran Rathinam, Dipika Ramaswamy
Task 3 and Task 4: Soundarya Sankarasubramanian, Krithika Narayanan, Shreekrishna Prasad Sethuraman
Task 5: Vishnunarayanan Ramasubramanian