CSE 515 Phase 2 Report

Greggory Scherer
Xiangyu Guo
Alfred Gonsalves
Chenchu Gowtham Yerrapothu
Siddhant Tanpure

Abstract

This paper describes several queries for retrieving data related to three different entities in a subset of the Internet Movie Database. These queries provide the ability to determine keyword relationships between actors and movies, genres and movies, and users and movies. Term frequency, inverse-document-frequency, and a relational model are some of the tools explored. In addition, several strategies for extracting latent semantics are used to determine hidden features and relationships between objects.

Keywords

IMDb, document, term frequency, inverse document frequency, dimension, measure, weight, vector, timestamp, rank, tag, differentiation function, dot-product, cosine similarity, random walk with restarts, personalized page rank, tensor, matrix, eigen decomposition, singular value decomposition, principle component analysis, latent Dirichlet allocation

Introduction

The Internet Movie Database (IMDb) began as a searchable list of credits aggregated from a Usenet group and now contains over 4,517,776 titles and 8,136,918 people [1][2]. IMDb allows users join the site and vote on titles using a 1-10 scale, as well as provide feedback or comments on the titles [3]. For this project, a set of data was provided from IMDb containing relationships between titles (movies), actors, genres, and users. Movies ranked based on votes using an undisclosed formula [3]. IMDb movies are also tagged with a set of keywords.

Terminology

- **Document** refers to a collection of words, but can also be used to refer to an entity or object such as actor or genre.
- **Term Frequency (TF)** refers to a proportion of how often a term is appears in a document and the total terms in a document [4].
- **Term Frequency - Inverse Document Frequency (TF-IDF)** refers to the discriminative power a term has in a given document. In other words, how well a particular term describes a document compared to other terms.
- **Dimension** refers to any arbitrary aspect of an entity that is distinguishable from any other.
- **Measure** refers to a numerical value determined by the intensity of some dimension of an entity. For example, length is the measure of how long (intensity) something is.
- **Measurement Unit** (unit of measure) refers to a mapping of a measure discrete proportions of a predetermined referential measurement. For example, a meter is a predetermined referential measurement of length.
- **Weight** refers to the importance of particular measure. Higher weight is analogous to higher importance.
- **Vector** refers to an ordered series of entities, and in this project the entities are measures.
- **Timestamp** refers to a date and time which is logged for any arbitrary entity. A timestamp usually grants an entity the dimension of time, and in this document, a timestamp manifests itself in POSIX time, or seconds since January 1, 1970 at midnight UTC.
- **Rank** refers to the rank computed by IMDb.
- **Tag** refers to a keyword assigned to a particular movie by IMDb.
- **Differentiation Function** refers to a function that computes a single numerical measurement of difference between two vectors.
- **Cardinality** refers to the numerical relationship between two entities, in which three outcomes are possible: one to one, one to many, and many to many. It may also refer to the size of a set.
- **Array** refers to an ordered list of entities or objects with a fixed size
- **Matrix** refers to a two-dimensional array

- **Tensor** refers to a n-dimensional array
- **Latent Semantic** refers to some relationship or meaning that is not visible in data
- **PCA** refers to **Principle Component Analysis**, an algorithm for Latent Semantic Analysis
- **SVD** refers to **Singular Value Decomposition**, an algorithm for eigen decomposition and Latent Semantic Analysis
- **LDA** refers to **Latent Dirichlet Allocation**, a generative probabilistic model for topic modeling
- **Personalized Page Rank** refers to a specific version of the **Page Rank** algorithm that allows the specification of seed nodes

Goal Description

This project contained four goals, separated into subtasks. Henceforth, these will be referred to as Task 1, Task 2, Task 3, and Task 4 respectively.

Assumptions

1. A movie can only have one rank.
2. The cardinality of movies to tags is many to many.
3. The cardinality of users to movies is many to many.
4. The cardinality of genres to movies is many to many.
5. The cardinality of actors to movies is many to many.
6. When counting total actors, only actors in movies that have tags are important to consider.
7. When counting total genres, only genres for movies that have tags are important to consider.
8. When counting total users, only users that have tagged or voted on movies with tags are important to consider.
9. Higher ranks have lower numbers.
10. Invalid input will print error.
11. Queries with no results will return empty.

Proposed Solution

Use information retrieval techniques to determine similarities using TF-IDF and vector similarities for visible features, and PCA, SVD, LDA for latent semantic analysis. CP and PPR is also considered for cluster analysis.

Genre tag vector determination

Since movie rank is not taken into account in Task 1a, a comparatively simplified weighting function is used. The weighting function follows a similar procedure as in other tasks, however there is no multiplication with the inverse rank. Additionally, the timestamps are based on tags for movies in a given genre instead of movies in which an actor appears.

Given tag t_i, and $s \in timestamp(g, t_i)$ and $|\text{timestamp}(g, t_i)| > 0$,
s_{min} is the smallest timestamp in the entire database,
s_{max} is the largest timestamp in the entire database,
and $timestamp(g, t_i)$ is all timestamps with tag t_i for movies in genre g and tag t_i,

$$TF_{weighted}(g, t_i) = \frac{\left(\sum_{s \in timestamp(g,t_i)} \frac{s - s_{min} + 1}{s_{max} - s_{min} + 1}\right)}{\sum_{t_j \in T}\left(\sum_{s \in timestamp(g,t_j)} \frac{s_j - s_{min} + 1}{s_{max} - s_{min} + 1}\right)}$$

Given T_g is the set of tags t_i for movies in genre g,
$vector_{TF}(g) = < TF_{weighted}(g, t_1), TF_{weighted}(g, t_2), \ldots >$ for all t_i in T_g

Given a weight $w_i = weight(g, t_i)$,
the set of all genres G,
and the set G_{t_i} of all genres with movies with tag t_i
There is a function gpt of genres per tag:
$gpt(t_i) = \left|\{g | g \in G_{t_i}\}\right|$,

$$IDF(w_i, t_i) = \log\left(\frac{|G|}{gpt(t_i)}\right) * w_i$$

$$vector_{TF\_IDF}(g) = < IDF\left(TF_{weighted}(g, t_1), t_1\right), IDF\left(TF_{weighted}(g, t_2), t_2\right), \ldots >$$

Actor Tag Vector Determination

The team determined that the best way to generate a weighted TF by tag-age, tag-frequency, and movie rank was to first calculate the weight of the tag by both frequency and age and then multiply that weight by the weight of the rank. The weight of tag by age and frequency is computed by taking the distance of the timestamp from the minimum timestamp and dividing it by the total timespan offered in the database. In this way, the newest timestamps would be closer to 1 and the oldest timestamps would be closer to 0. The total timespan in the database was chosen instead of a measure of the distance from the current day in order to make the algorithm consistent regardless of the current date and to prevent the calculations from getting too small. By taking the sum of all these calculations the frequency of the tag is implicitly factored in. By adding 1 to the numerator and denominator, division by zero or other undefined cases. In order to compute rank, since lower numbers are considered higher rank, the inverse of the rank is used so that rank 1 is closer to 1 and the lowest rank (highest number) is closer to 0. The timestamps are based on the tags on movies in which an actor appears. These decisions are described in the equations below.

Given tag t_i, and $s \in timestamp(a, t_i)$ and $|\text{timestamp}(a, t_i)| > 0$,
s_{min} is the smallest timestamp in the entire database,

$s_{max}$is the largest timestamp in the entire database,
and $timestamp(a, t_i)$ is all timestamps with tag t_i for movies with actor a,
and $rank(a, t_i)$ is the rank of the movie with actor a and tag t_i

$$TF_{weighted}(a, t_i) = \frac{\left(\sum_{s \in timestamp(a,t_i)} \frac{s - s_{min} + 1}{s_{max} - s_{min} + 1}\right)\left(\frac{1}{rank(a, t_i)}\right)}{\sum_{t_j \in T}\left(\sum_{s \in timestamp(a,t_j)}\left(\frac{s_j - s_{min} + 1}{s_{max} - s_{min} + 1}\right)\left(\frac{1}{rank(a, t_j)}\right)\right)}$$

Given T_a is the set of tags t_i for an actor a,
$vector_{TF}(a) =< TF_{weighted}(a, t_1), TF_{weighted}(a, t_2), \ldots >$ for all t_i in T_a

Given a weight $w_i = weight(a, t_i)$,
the set of all actors A,
and the set $A_{t_i}$of all actors in movies with tag t_i
There is a function apt of actors per tag:
$apt(t_i) = \left|\{a \middle| a \in A_{t_i}\}\right|$,

$$IDF(w_i, t_i) = \log\left(\frac{|A|}{apt(t_i)}\right) * w_i$$

$$vector_{TF\_IDF}(a) =< IDF\left(TF_{weighted}(a, t_1), t_1\right), IDF\left(TF_{weighted}(a, t_2), t_2\right), \ldots >$$

Task 1a and 1b

PCA: Principal components Analysis

Approach for PCA: When the data matrix is passed in to the PCA function with the number of components as a parameter, it forms the covariance matrix (obtained from get_covarience()) and gives the decomposition of that covariance matrix. The decomposition consists of three matrices, left factor matrix (U), right factor matrix(U Transpose) and S the diagonals matrix with decreasing magnitude of the Eigen values.

The fit-transform function will give the factor matrix with n_component latent semantics as the features.

Explained variance will give the diagonal matrix with Eigen values related to the n- latent features in the decreasing order of the magnitude.

Choosing the top latent semantics: The top three latent features are selected using these values in the diagonal matrix

Libraries used: decompositions from sklearn

From sklearn import decompositions

PCA(n_components)

N_components : no of components to keep (no of latent features reduced from actual no of dimensions)

Parameters used for PCA:

components : array, shape (n_components, n_features)

Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained varience

explained_variance_: array, shape (n_components,)

The amount of variance explained by each of the selected components.Equal to n_components largest eigenvalues of the covariance matrix of Data.

SVD: Singular valued Decomposition

Approach for SVD: When the data matrix is passed in to the SVD function, it gives the decomposition of that data matrix. The decomposition consists of three matrices, left factor matrix (U), right factor matrix(V Transpose) and S the diagonals matrix with decreasing magnitude of the square root of the Eigen values(singular values).
Choosing the top latent semantics: The top three latent features are selected using these values in the diagonal matrix

Libraries used : linalg from numpy libraries

From numpy import linalg.svd :- Linalg.svd(data, full_matrices=1,compute_uv=1)

Parameters that svd function will take:

Data: data : (..., M, N) array_like
A real or complex matrix of shape (M, N)

full_matrices : bool, optional
If True (default), u and v have the shapes (M, M) and (N, N), respectively. Otherwise, the shapes are (M, K) and (K, N), respectively, where K = min(M, N).

SVD function Returns:

u : { (..., M, M), (..., M, K) } array
Unitary matrices. The actual shape depends on the value of full_matrices Only returned when compute_uv is True.
s : (..., K) array
The singular values for every matrix, sorted in descending order.
v : { (..., N, N), (..., K, N) } array
Unitary matrices. The actual shape depends on the value of full_matrices Only returned when compute_uv is True.

LDA: Latent Dirichlet Allocation:
LDA is a generative probabilistic model for collections of discrete data such as text corpora. There are various libraries that implement LDA.
The class genism.models.ldamodels.LdaModel provides various parameters that can be assigned at the time of declaration.
gensim.models.ldamodel.LdaModel(corpus=None, num_topics=100, id2word=None, distributed=False, chunksize=2000, passes=1, update_every=1, alpha='symmetric', eta=None, decay=0.5, offset=1.0, eval_every=10, iterations=50, gamma_threshold=0.001, minimum_probability=0.01, random_state=None, ns_conf=None, minimum_phi_value=0.01, per_word_topics=False, callbacks=None)

Parameters:
1. corpus – documents that need to be considered
2. num_topics – K number of random topics that are to be considered for assigning the words in the documents. This parameter takes the number of latent semantics required
3. id2word – this parameter takes a mapping from id (integers) to words (strings). It is used to determine the size of the vocabulary

LDA provides a probabilistic distribution upon its data. Based on this distribution feature is assigned to one of the hidden slots. This process is governed by multinomial distribution.
<doubtful about this statement>
The top_topics() will output topics based on the probability distribution.

Task-1c Top-10 Most Similar actors

The output for this task then used TF-IDF vectors to compute pure dot product similarities and ranked from greatest to least. Dot-product was used here and in many cases because it takes into account angle and length of vectors. A matrix was constructed of TF-IDF data vectors from all actors. The SVD of this matrix was then used to determine the 10 most related actors by using the object-to-latent semantic relational matrix. Similarly, the PCA was computed for the covariance matrix of the matrix used in SVD. This then underwent eigen-decomposition to determine the object-to-latent semantic relational matrix, which was then used to determine the 10 most related actors.

For LDA, a model was generated using the chosen actor's tag vector as a document of the corpus. Afterwards, other actors' tag vectors were analyzed with the model to determine the probabilities of tags being grouped into the top-5 latent topics compared to the model generated by the chosen actor. The generated grouping probabilities were then compared to the chosen actor's using dot product.

Libraries: numpy, gensim.

Task-1d: Top-10 Related Actors to A Movie

The output for this task then used TF-IDF vectors to compute pure dot product similarities and ranked from greatest to least. A matrix was constructed of TF-IDF data vectors from all actors. The SVD of this matrix was then used to determine the 10 most related actors by using the

object-to-latent semantic relational matrix. Similarly, the PCA was computed for the covariance matrix of the matrix used in SVD. This then underwent eigen-decomposition to determine the object-to-latent semantic relational matrix, which was then used to determine the 10 most related actors.

For LDA, a model was generated using the chosen movie's tag vector as a document of the corpus. Afterwards, other actors' tag vectors were analyzed with the model to determine the probabilities of tags being grouped into the top-5 latent topics compared to the model generated by the chosen movie. The generated grouping probabilities were then compared to the chosen movie's using dot product.

Libraries: sklearn, gensim, numpy

Task 2a and Task 2b

Task 2a was aimed at creating groupings among actors based on their similarity to each other. Actors were related to each other in terms of the movies they acted in, movie genres, tag description for the movies and so on. The problem statement mentioned to calculate the similarity matrix in terms of the tag vectors. We calculated the actor-tag vector based on the movies they worked in and the ratings they received.

To find the similarity among these vectors we made use of dot-product similarity. We chose dot-product similarity measure because cosine similarity only cares about angle difference, while dot product cares about angle and magnitude. We were dealing in the space of tag vectors and so it was important to use the weight magnitude of the tag to determine its similarity to other values. Also dot-product is cheaper in terms of complexity and implementation.

Task 2b was aimed at creating groupings among actors based on their co-acting similarity. Here, the co-acting relationship determined the number of movies that the two actors have worked together.

The coactor-coactor matrix was calculated by counting the number of movies that have actor i and actor j.

ASSUMPTION: The diagonal element of the matrix would give us the coacting relationship between an actor with himself. We kept a counter on this value too. This would then give the total movies that the actor has worked in.

We had to perform SVD on the actor-actor similarity matrix or thee cofactor-coactor matrix. We used the svd function from the numpy library. The linalg (Linear Algebra) class provided by numpy library calculates the three decomposition matrices for SVD.

U, S, Vt = np.linalg.svd(data) would provide the three matrices.

This function also took a parameter as 'full_matrices', which is True, would add elements to the matrix to make it a square matrix. We kept the parameter 'False' to be able to see the proper dimensions.

To create the groupings, we simply selected top N (here, 5) latent semantics from U, then selected the max value for an object from the 5 latent sematics and then assigned the object to the group index.

For 5 latent semantics, we made 5 sets of groups, lets say, gr0, gr1, gr2, gr3, gr4.

Then for object1 with latent semantics valued [1,5,2,6,2], we assigned object1 to gr3.

Task 2c

Approach: Once the three-mode tensor Actor-Movie-Year is created and passed in to decomposition function with a target rank as five, the tensor then decomposes to three factor matrices one for each mode of the tensor i.e. Actor, Movie and year respectively.
Choosing top five latent semantics for Movie actor and year modes: As the target rank is set to five, these factors that are decomposed has the five latent semantics as their features.
The initial factor matrices are chosen randomly and optimized using iterative approach till the converge is optimal, the library function uses ALS approach to optimize the decomposition.
Groupings: Based the latent semantics we obtain we form required number of non-overlapping groups for objects along each mode of the tensor (Actor, movie and year). The objects are mapped to the latent groups based on their maximum value in their respective latent groups

Libraries used: decompositions from tensorly, numpy
parafac : library for Alternative least square approach to compute [CANDECOMP/PARAllel FACtors(PARAFAC)] CP decomposition. **parafac(tensor,rank,init)**
Result we get has a decomposed tensor with three factor matrices because it is a three mode tensor. Factor matrices are returned with latent semantics , the number of latent semantics returned will be based on the target rank given (no of components)
The parametrs that the parafac function takes are:

Data: the input tensor to be decomposed
Rank: Tensor rank for the decomposition (no of latent features in to each factor matrices is decomposed in to).
init: {'random','nvecs'} – optional parameter for initialization of factor matrices
-random: Factor matrices are initialized randomly.
-nvecs: Factor matrices are initialized via SVD.
Default is :'svd'
max_iter: int, this is also a optional parameter
Maximum nuber of iterations of the ALS algorithm.
Default is 100 iterations
tol: float- this parameter is optional
Tolerance: the algorithm stops when the variation in the reconstruction error is less than the tolerance
Default value is 1e-06

This library function returns:
Factors-: ndarray list
List of factors of the CP decomposition element I is of shape (tensor.shape[i],rank)
It returns the factor matrices of the decomposed tensor along all the modes of the tensor.

Task 2d

Approach: Once the three-mode tensor Tag-Movie-Rating is created and passed in to decomposition function with a target rank as five, the tensor then decomposes to three factor matrices one for each mode of the tensor i.e. Tag, Movie and Rating respectively.
Choosing top five latent semantics for Movie actor and year modes: As the target rank is set to five, these factors that are decomposed has the five latent semantics as their features.
The initial factor matrices are chosen randomly and optimized using iterative approach till the converge is optimal, the library function uses ALS approach to optimize the decomposition.
Groupings: Based the latent semantics we obtain we form required number of non-overlapping groups for objects along each mode of the tensor (Tag, movie and Rating). The objects are mapped to the latent groups based on their maximum value in their respective latent groups

Libraries used : decompositions from tensorly, numpy
parafac : Library for Alternative least square approach to compute [CANDECOMP/PARAllel FACtors(PARAFAC)] CP decomposition. **parafac(tensor,rank,init)**
Result we get has a decomposed tensor with three factor matrices because it is a three-mode tensor. Factor matrices are returned with latent semantics, the number of latent semantics returned will be based on the target rank given (no of components)
The parametrs that the parafac function takes are:

Data: the input tensor to be decomposed
Rank: Tensor rank for the decomposition (no of latent features in to each factor matrices is decomposed in to).
Init: {'random','nvecs'} – optional parameter for initialization of factor matrices
-random: Factor matrices are initialized randomly.
-nvecs: Factor matrices are initialized via SVD.
Default is :'svd'
max_iter: int, this is also a optional parameter
Maximum nuber of iterations of the ALS algorithm.
Default is 100 iterations
tol: float- this parameter is optional
Tolerance: the algorithm stops when the variation in the reconstruction error is less than the tolerance
Default value is 1e-06

This library function returns:
Factors-: ndarraylist
List of factors of the CP decomposition element I is of shape (tensor.shape[i],rank)
It returns the factor matrices of the decomposed tensor along all the modes of the tensor.

Task 3

We implemented the Random Walk with Restart (RWR) algorithm in task3. The RWR algorithm will take a set of seeds as the teleport vector (v_q) to calculate the page ranking vector(u_q)

Transition matrix A: we construct the transition matrix by counting the outgoing degrees of each node 1/out_degree(node). And for nodes without outgoing edges, we set them to 1/number(nodes).
Initial vector u_q v_q: initially, we set each node in seed with 1/number(seeds) for both u_q and v_q.
Constant c, as we discussed in the class, the constant c is application related. And in Personalized Page Rank (PPR), we should increase the chances that our algorithm will back to the seed nodes. After few rounds experiment, we found out c = 0.2 works fine on our application.
When to stop(converged), when the max(Δu) < 1e-06, we consider our algorithm converged and stop.

Libraries: numpy

Task 4

First of all, we realized that different user may have different taste or preference on the movie watching actives. That is the subjectivity plays a very important role in our recommender system. However, we can't resolve it using one simple algorithm or one subsystem, it requires multiple subsystems to achieve a better result. So, based on the limited information related to the movies watched by the users. We made following assumptions, 1. the users will interest in the movies which played by the same actors in those movies he watched; 2. the users will interest in the movies which has higher ratings than the movies he watched; 3. the users will interest in the movies which belongs to some particular genre from the movies he watched; 4. the users will interest in the movies which from one year among all the movies he watched; 5. the user has no preference.
From above assumptions, we will run multiple Personalized Page Rank (PPR) algorithm on different sets of movies given the movies watched by user as seeds. Then we will use rank joint or fuzzing merge method to merge the ranking results from five outputs. We decided to use sum as our merging function.

Rank Joint

In order to resolve the subjectivity in our recommender system, we assume all ranking results are equality weighted. So, we add the rank of the same movie from different output, then sort this new value again, output the highest ten to the users.
Libraries: numpy

Interface Specifications

The system-level interface is by command line. The following commands should produce the corresponding outputs for each task.
task1a <genre> [PCA|SVD|LDA]
task1b <genre> [PCA|SVD|LDA]
genre, a valid genre name, without quote, e.g. Thriller
model, a valid model name in big letters without the quote

task1c <actorid> [PCA|SVD|LDA|TF-IDF]

actor_id, a valid actor id, without quote, e.g. 1582699 model, a valid model name in big letters without the quote

task1d <movieid> [PCA|SVD|LDA|TF-IDF]
movie_id, a valid movie id, without quote, e.g. 4252 model, a valid model name in big letters without the quote

task2a
task2b
task2c
Task2d

task3a <actorid> {<actorid>}
task3b <actorid> {<actorid>}
actor_id, a list of valid actors id, without quote, e.g. 1582699 606633

task4 <user_id>
user_id, a valid user id, without quote, e.g. 450

System requirements/installation and execution instructions

0. Running environment, any machine/OS with Python3.0 installed. (numpy, tensorly, sklearn, gensim are prerequisite Python libraries)
1. Put all csv data file under the “data” folder.
2. Go to the “Code” folder using your command line tool. (Shell or CMD)
3. Execute the program with “python3 main.py”.
4. Wait for around 20 seconds, you will see the following information during this progress.
“loading csv data into memory...
loading completed!
preprocessing data...
preprocessing completed!”
5. After you see a “query>”, you can running your query now.
6. How to run each query command, please refer the section “Interface Specifications” for more detail.

Related Work

Juan Ramos. 2003. Using TF-IDF to Determine Word Relevance in Document Queries. (January 2003) DOI:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf
Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, Pinar Duygulu. 2004. Automatic Multimedia Cross-modal Correlation Discovery. (August 2004) DOI:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.8179&rep=rep1&type=pdf

K. Selçuk Candan and Maria Luisa Sapino. 2010. Data Management For Multimedia Retrieval. 147.

Conclusions

Term frequency and inverse-document-frequency measurements help determine relevance of a term to an entity and the discrimination power of a term respectively. Using these measurements, it is possible to further determine similarities between objects using both observed and latent semantics. Singular Value Decomposition, Principle Component Analysis, Latent Dirichlet Allocation, and Personalized Page Rank are algorithms that can be used to analyze latent semantics. Observed features can be compared using various similarity measures, such as dot-product, cosine similarity, or even Euclidean distance. The more sophisticated algorithms help exploit hidden relationships and can be useful when trying to build recommendation systems.

Bibliography

[1] Col Needham. 2010. IMDb History. (October 17, 2010). DOI: http://www.imdb.com/help/show_leaf?history

[2] IMDb Stats. Retrieved September 17, 2017 from http://www.imdb.com/stats

[3] IMDb Votes/Ratings Top Frequently Asked Questions Retrieved September 17, 2017 from http://www.imdb.com/help/show_leaf?votestopfaq

[4] How to compute. Retrieved September 17, 2017 from http://www.tfidf.com

[5] K. Selçuk Candan and Maria Luisa Sapino. 2010. Data Management For Multimedia Retrieval.

[6] SVD PCA and CP libraries from http://scikitlearn.org, https://docs.scipy.org andhttps://tensorly.github.io

Appendix

Specific Roles of Group Members

1. Xiangyu Guo - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing
2. Siddhant Tanpure - Group discussion, Independent implementation
3. Chenchu Gowtham Yerrapothu - Group discussion, Independent implementation
4. Alfred Gonsalyes - Group discussion, Independent implementation
5. Greggory Scherer - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing