Comparison of Recommendation Systems

Team members

1. Chirag Jamadagni (cjamadagni3)
2. Nidhi Menon (nmenon34)
3. Sneha Venkatachalam (sneha30)
4. Vaibhav Tendulkar (vtendulkar6)

1. Introduction

Our group consists of four Master's in Computer Science students, three of whom are specializing in Machine Learning, and the other specializing in Computing Systems. The applicability of machine learning is a common debate. In order to conclude this debate, we have built and compared a four different classes of recommendation systems, and these classes have been enumerated below.

1. Graph Theory
2. Matrix Factorization
3. Machine Learning
4. Deep Learning

We begin this report by summarizing the existing literature in the field of recommendation systems in Section 2. In Sections 3 we describe the dataset on which these systems were evaluated. In Sections 4, 5, and 6 we discuss the theory and implementation details involved in building graph, matrix factorization, machine learning and deep learning-based recommendation systems. In Section 7 we discuss the experimental results, run-time statistics and the differences between these systems, and we conclude this report in Section 8.

Motivation & Objectives

While extensive literature elucidating the benefits of machine learning based recommendation systems have been published (elaborated in the Related Work Section), such systems do have disadvantages:

- Models can be quite complex and not necessarily be based on useful features, i.e. good models require good feature engineering, which is a hard problem to tackle.
- Efficient classification/prediction requires large amounts of data. If the recommendation service is nascent, and the amount of data collected is small, machine learning would not work well.
- Depending on inputs, training can be a time consuming process.
- Modifications to the dataset requires re-training.

We believe that by using the inherent properties of graphs, and concepts from graph theory, the above issues can be tackled, and that it is possible to build efficient recommendation systems using graphs as well. While there is existing work in this field (elaborated in the Related Work Section), an application-level comparison would help quantify the advantages and disadvantages of one system over another.

2. Related Work

The papers in [1] and [2] provide us with the necessary background about building recommender systems (RS) using various Machine Learning methods, and the different approaches that can be used to do so. Collaborative approaches refer to a RS which works with user information to identify users with similar tastes and makes recommendations based on items bought by like-minded users. The collaborative filtering approach is further classified into memory-based recommendations and model-based recommendations in [3] which is a survey paper. Content-based filtering methods use the data on the items that it can access to return recommendations to the user. The hybrid filtering approach combines the above two approaches and uses both user and item information to generate recommendations. Approaches based on k-means clustering and naive bayes are also described in [2] which concludes that Naive Bayes gives the best precision.

[1] also provides a detailed review of the types of Machine Learning algorithms used. It was found that Bayesian approaches were used by 7 out of the 26 publications studied by the authors in [1]. This was attributed to the reduced complexity of Bayesian calculations by the authors. Decision Trees were found to be the next most widely used algorithm, with Neural Networks too becoming a promising choice, with more scope for further research. Bandit and Ensemble algorithms are also gaining popularity. Bandit algorithms work to maximize a given value by deciding the order and number of times decisions have to be made to do so, but are less widely used may be because of their high complexity, according to the authors in [1]. Ensemble methods are a recent addition that combine results from several different algorithms to create a single knowledge base, and draw conclusions from it. However, these methods also vary in complexity, which could be a factor that prevents their widespread adoption, according to [1].

The article in [4] suggests a graphical method of building recommendation systems. The authors propose using weighted graphs where products are nodes and product interactions are represented by edges. They experimented with the multiplicative model and additive model to return the k-nearest products to users. The article in [5] proposes the usage of a bipartite graph where the nodes are traversed by means of a biased random walk. A graph-based clustering approach with an undirected and unweighted graph is proposed in [6] to exploit the geometry of user space to build a recommendation system.

Deep learning methods are also a promising approach to the task of obtaining recommendations. [7] is a recent paper published last year which explores how deep learning can be applied to the task of collaborative filtering in recommendation systems. The paper explains the concept of deep autoencoders and how their inherent property of dimensionality reduction can be useful in designing recommendation systems. This paper also introduces the concept of iterative output re-feeding and how it improves the generalization performance of the model.

3. Data

In order to compare the four different types of recommendation systems, we needed to select a dataset suitable for each approach. After considering many popular datasets, we decided on the MovieLens [8] dataset. This dataset was chosen for the below reasons.

1. It is a fairly clean and curated dataset which is considered an industry standard.
2. Subsets and extracts from this dataset have been used in many data science competitions and this dataset has been cited in numerous published work.
3. The structure of the data is suitable to be modelled as a graph, i.e. the data is well structures. Since a large part of our evaluation is around graphs and using a graph databases, having data which do not contain relationships between its attributes would be counterproductive.
4. The movies domain was the domain of choice in two-third of publications/proposals looked at for recommendation systems. In [1], the authors also mention the reason why the movie or IMDB dataset is commonly used.

This dataset is available in three different sizes: 100,000 ratings, 1 Million ratings, and 20 Million ratings. Due to hardware and resource constraints of using our personal machines, we decided to use the set with 100,000 ratings (100K). Having consulted numerous papers and articles, we concluded that this amount of data would be sufficient for all four approaches.

The properties of the MovieLens 100K dataset have been listed below.

1. 100,000 ratings
2. 1000 users
3. 1700 movies

4. Graph-based Approach

While extensive literature elucidating the benefits of machine learning based recommendation systems has been published, such systems do have disadvantages:

- Models can be quite complex and not necessarily be based on useful features, i.e. good models require good feature engineering, which is a hard problem to tackle.
- Optimization of hyperparameters requires sufficient amount of data. If the service is nascent and the amount of data collected is small, machine learning would not work well.
- Depending on the inputs, training can be a time consuming process.
- Modifications to the dataset (Addition of new movies or users) requires re-training.
- Machine Learning based systems predict ratings and then use these predicted ratings to recommend movies, but the former isn't necessary to obtain the latter.
- Scaling the model in production is challenging.

We believe that by using the inherent properties of graphs, and concepts from graph theory, the above issues can be tackled, and that it is possible to build efficient recommendation systems using graphs. In this section we first describe how the dataset was converted into a graph and efficiently stored. We then proceed to describe two approaches to providing recommendations using graphs, and then discuss how these approaches handle Experiments 2 and 3 (from Section 4). We conclude this section by describing how this system uses a combination of both approaches to provide personalized recommendations and by mentioning the pros and cons of a graph-based approach.

4.1 Graph Generation

In order to build a recommendation system based on concepts from graph theory, we first need to transform the given dataset into a graph and efficiently store it. This is achieved by using the Neo4j [9] graph database.

Neo4j is open source graph database management system. It is an ACID compliant transactional database which provides native graph storage and processing, and is the most popular graph database according to DB Engines-Ranking [10]. In Neo4j, data is represented as a node, edge or attribute. Each node and edge can have any number of attributes, and similar to a graph, nodes are connected to one another via edges. These edges are called relationships in the graph.

Neo4j can be queries using the Cypher Query Language [11]. Cyper is a declarative graph query language that allowed for expressive and efficient querying and updating of a property graph. Although it is relatively simple, it is a very powerful language for graph databases. Extremely complex database queries on dense and complex graphs can be expressed easily through Cypher, thus allowing users to focus on the problem at hand, and not having to spend too much time worrying about database access.

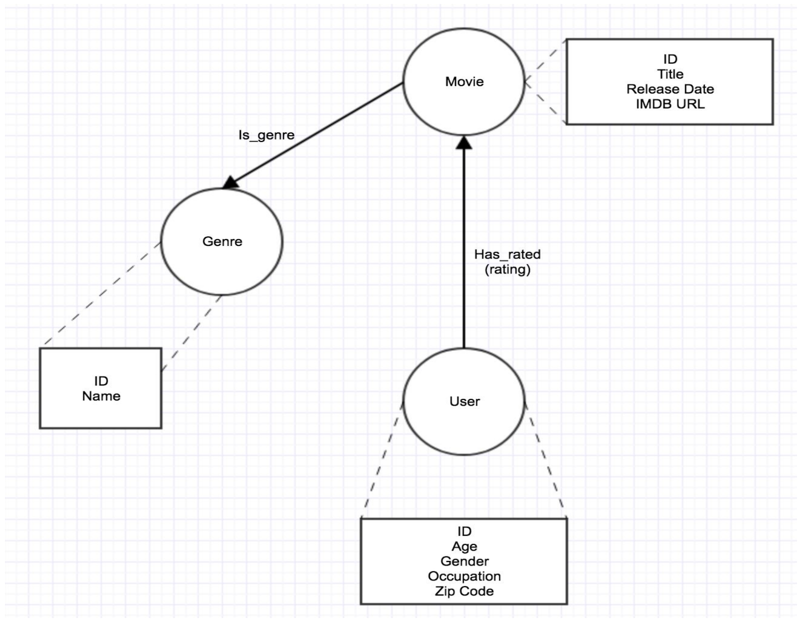


Figure 1

From Figure 1 we can see how the MovieLens dataset is modeled as a graph. The graph properties have also been mentioned below.

- Nodes
    - User
    - Movie
    - Genre
- Edges
    - Has_Rated
    - Is_Genre

All the nodes have associated attributes/properties and these have been mentioned in the rectangles associated with a node in Figure 1. The *Has_Rated* edge has one attribute, rating, which is the rating given to the movie by a user. The *Is_Genre* edge has no attribute since the presence of this edge implies that the movie belongs to a particular genre.

A script which leverages the Cypher Query Language was written to read data from the respective files, model the information as a graph, and appropriately write and store all information in Neo4j. Figures 2 and 3 shows the in-built Neo4j visualization for a subset of this dataset.

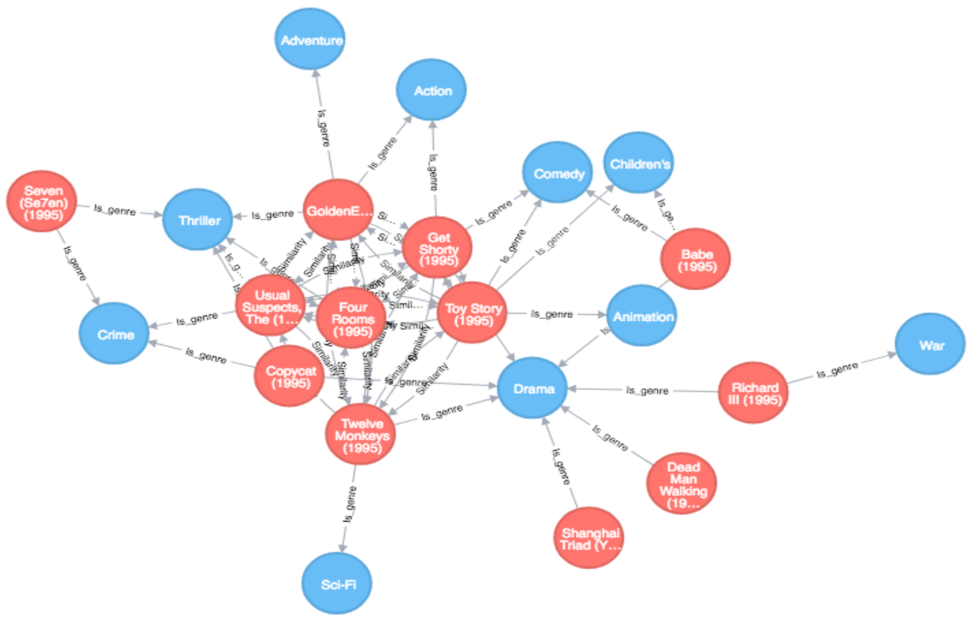


Figure 2

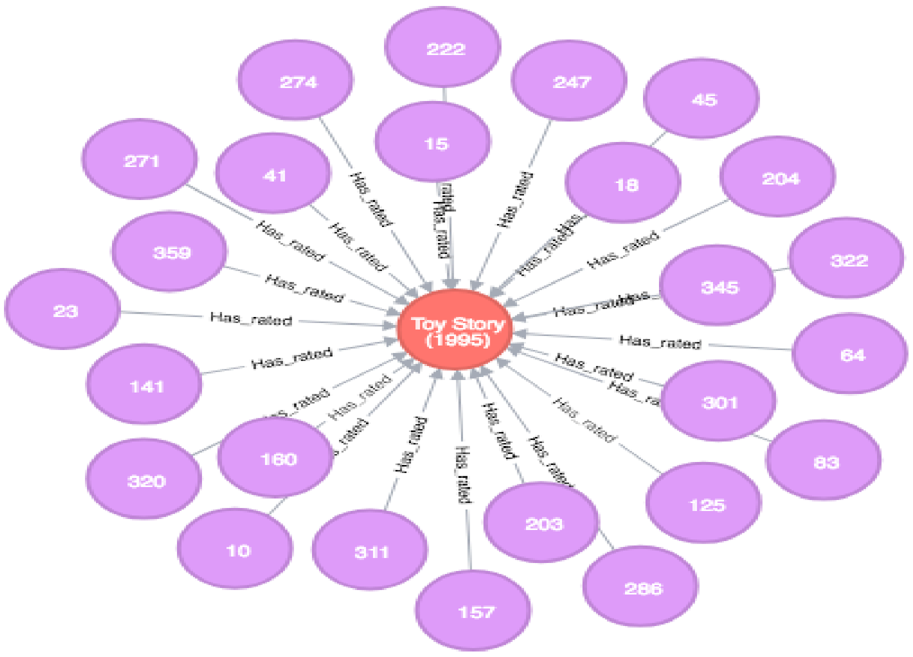


Figure 3

In Figure 2, one can notice an additional edge named “similarity” between movies. The significance of this edge has been explained in Section 5.3. In Figure 3, the purple nodes correspond to unique user IDs.

4.2 Collaborative Filtering Recommendation Strategy

Collaborative filtering is based on the similarity of users. The idea is to find users in a community who share/like similar things. If two users have given the same or very similar ratings to a set of items, they they have similar taste, and these users with similar taste can be grouped together. A user then is recommended items that he hasn’t rated but was rated positively by the members with similar taste. Figure 4 [12] depicts this idea.

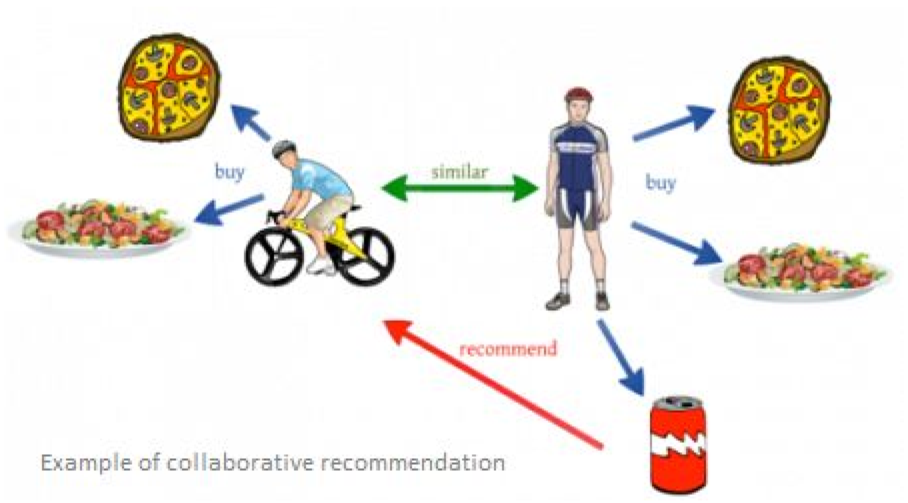
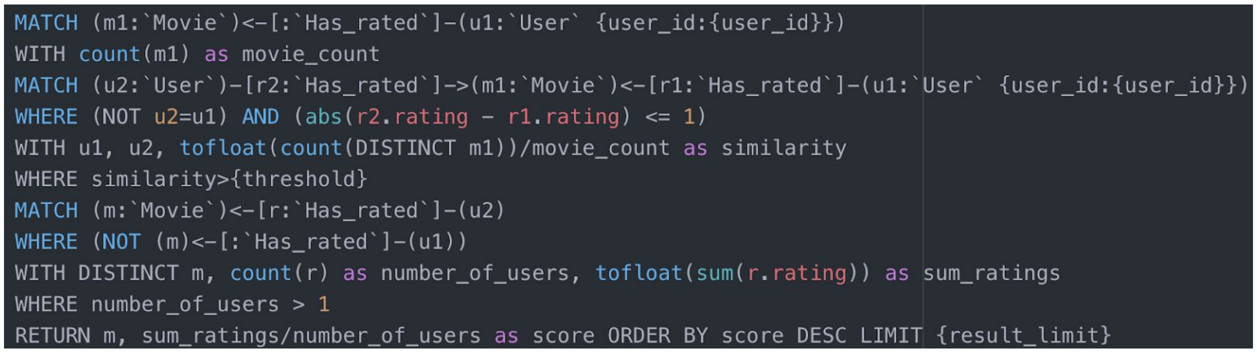


Figure 4

For the MovieLens dataset, users are considered similar if they have rated a sufficient amount of movies in common, and if they have given similar ratings to respective movies. Movies are then ranked according to the mean of the rating given to that movie by similar users. This procedure has been enumerated below.

- Similarity between users i and j is the ratio of the number of movies which have been rated by both with an absolute difference of less than or equal to one, to the number of movies seen by user i.
- Filter all users whose similarity with user in question is greater than a predefined threshold.
- Obtain the set of movies which have been seen by similar users but not the user in question.
- Rank these movies according to the mean rating given by the set of similar users.

The benefit of using Neo4j is that the above complex approach is carried out by a single Cypher query. By performing recommendations through queries, it enhances flexibility and allows for easy modifications to the recommendation strategy without any additional processing of the dataset. Figure 5 shows the query used for this approach. Here, the 'user_id' is the unique identifier for the person for whom recommendations are being determined, 'threshold' is the minimum similarity threshold and 'result_limit' is the number of movies to be recommended.



```
MATCH (m1:`Movie`)<-[:`Has_rated`]-(u1:`User` {user_id:{user_id}})
WITH count(m1) as movie_count
MATCH (u2:`User`)-[r2:`Has_rated`]->(m1:`Movie`)<-[r1:`Has_rated`]-(u1:`User` {user_id:{user_id}})
WHERE (NOT u2=u1) AND (abs(r2.rating - r1.rating) <= 1)
WITH u1, u2, tofloat(count(DISTINCT m1))/movie_count as similarity
WHERE similarity>{threshold}
MATCH (m:`Movie`)<-[r:`Has_rated`]-(u2)
WHERE (NOT (m)<-[:`Has_rated`]-(u1))
WITH DISTINCT m, count(r) as number_of_users, tofloat(sum(r.rating)) as sum_ratings
WHERE number_of_users > 1
RETURN m, sum_ratings/number_of_users as score ORDER BY score DESC LIMIT {result_limit}
```

Figure 5

4.3 Content Based Filtering Recommendation Strategy

Content based filtering is based on the similarity of the objects being recommended. The idea is that if a person likes a particular product X, and there exists a product Y which is similar to X, then Y should be recommended. Figure 6 [12] depicts this idea.

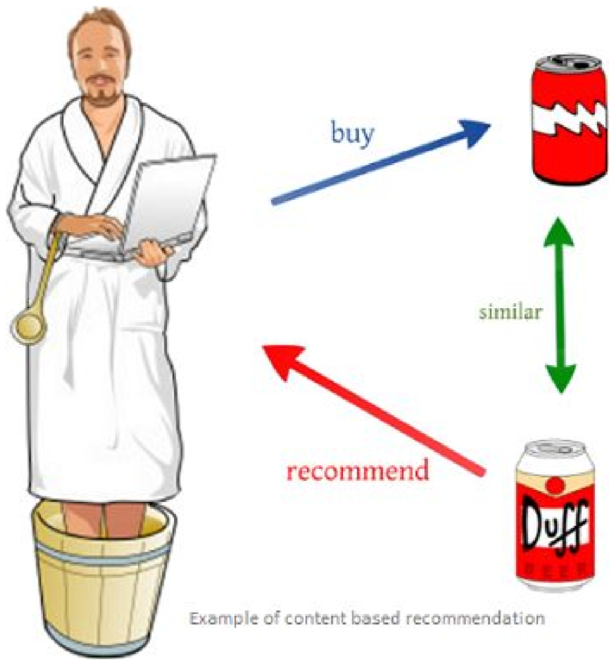


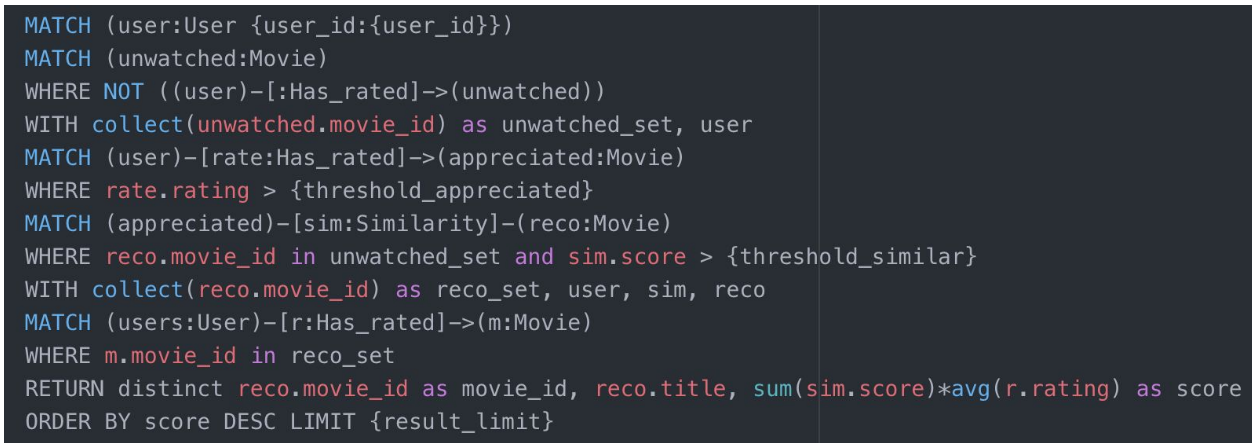
Figure 6

For the MovieLens dataset, the similarity between movie synopsis is determined by using Latent Semantic Analysis (LSA) [13]. A new edge representing this semantic similarity metric is then added to the graph. This new 'similarity' edge is between movies and can be seen in Figure 2. The similarity edge is then used to perform recommendations using the basic idea depicted in Figure 6. In order to add this edge to the graph the below steps had to be performed.

1. The IMDb database was queried using the movie title to obtain the corresponding movie synopsis. Some (old) movies didn't possess a synopsis and hence do not contain a similarity edge in the graph.
2. Each of these synopsis were then preprocessed.
    a. Removal of punctuations, names, stop words and other irrelevant words.
    b. Stemming and Lemmatization.
    c. Creating a dictionary using Gensim [14].
    d. Creating the corpus.
    e. Performing a tf-idf transformation
    f. Performing a single value decomposition to reduce dimensionality and identify relevant features.
    g. Creating the similarity model.
    h. Computing the pairwise similarity between all movies.
3. Using the computed similarity values, a new similarity edge, between movies, with a numeric similarity attribute was added to the graph.

Once this new information has been added to the graph, the Cypher query shown in Figure 7 was used to make recommendations. This recommendation strategy has been described in a stepwise fashion below.

- All the movies which have been rated highly by the target user (user for whom the recommendations are being determined) are first identified. A user defined threshold (threshold_appreciated) is used to determine the highly rated movies. This has been done to enhance flexibility and to understand the impact of changing this value.
- Using the similarity edge/relationship, the set of movies which the target user has not watched but are semantically similar to the movies he/she has highly rated are obtained. To filter out movies with a low semantic similarity, a user defined threshold (threshold_similar), similar to the approach in Section 5.2 is used.
- This set of unwatched movies are then ranked according to a weighted sum metric. This metric is the sum of the similarity scores/values weighted by the global average rating given to that movie.



```
MATCH (user:User {user_id:{user_id}})
MATCH (unwatched:Movie)
WHERE NOT ((user)-[:Has_rated]->(unwatched))
WITH collect(unwatched.movie_id) as unwatched_set, user
MATCH (user)-[rate:Has_rated]->(appreciated:Movie)
WHERE rate.rating > {threshold_appreciated}
MATCH (appreciated)-[sim:Similarity]-(reco:Movie)
WHERE reco.movie_id in unwatched_set and sim.score > {threshold_similar}
WITH collect(reco.movie_id) as reco_set, user, sim, reco
MATCH (users:User)-[r:Has_rated]->(m:Movie)
WHERE m.movie_id in reco_set
RETURN distinct reco.movie_id as movie_id, reco.title, sum(sim.score)*avg(r.rating) as score
ORDER BY score DESC LIMIT {result_limit}
```

Figure 7

4.4 Technical Implementation Specifications

This recommendation system was developed as a REST service implemented in Python using the Flask [15] framework. The Py2Neo [16] package was used to connect and interface with the Neo4j database and the the IMDbPY [17] package was used to query the IMDb database to obtain movie synopsis. Additional python scripts were used for graph creation and ingestion into the database, LSA model generation and for adding a 'similarity' edge to the graph. LSA model generation involved the use the Numpy [18], Pandas [19], Gensim [14] and NLTK [20] Python libraries.

The REST service provides the below APIs. *s represent mandatory input fields.

1. *~/recommendations_user*
   a. API for the approach detailed in Section 5.2.
   b. Accepts a JSON input with following fields:
      i. User ID *
      ii. Similarity threshold *
      iii. Number of movies to be recommended *
   c. Successful execution returns a JSON array, sorted in descending order according to similarity score, where each object has the following fields:
      i. Movie ID
      ii. Movie Title
      iii. Movie URL
      iv. Similarity score
2. *~/recommendations_movie*
   a. API for the approach detailed in Section 5.3.
   b. Accepts a JSON input with following fields:
      i. User ID *
      ii. Similarity threshold *

        iii. Minimum threshold for highly rated movies *
        iv. Number of movies to be recommended
    c. Successful execution returns a JSON array, sorted in descending order according to similarity score, where each object has the following fields:
        i. Movie ID
        ii. Movie Title
        iii. Movie URL
        iv. Similarity score

3. *~/add_user*
    a. API to add a new user to the dataset/graph.
    b. Accepts a JSON input with the following fields:
        i. Name *
        ii. Age
        iii. Gender
        iv. Occupation
        v. Zip code
        vi. JSON object containing a list of movies watched and their associated ratings.
    c. Successful execution returns the unique user ID for the new user.
4. *~/add_movie*
    a. API to add a new movie to the dataset/graph.
    b. Accepts a JSON input with the following fields:
        i. Title *
        ii. Genre *
        iii. Synopsis
        iv. Release Date
        v. IMDb URL

4.5 Experimentation Details

As we can see from the REST APIs mentioned in Section 5.4, obtaining the recommendations for a user already present in the dataset is straightforward. In this subsection we focus on the cases where a new user or movie is added to the graph database.

When a new user is added:

- If a set of movies and their corresponding ratings are provided, then either of the two approaches could be used to obtain recommendations.
- If this input is absent, the set of movies with the highest average global rating is returned.

When a new movie is added, a similarity edge is added to the graph for this movie and using this similarity edge, this movie will be recommended in the content based filtering approach. If the synopsis is not provided by the user, the IMDb database is queried to obtain this information.

4.6 Personalized Recommendations

The ideal application of a graph based recommendation system would involve the combination of both of these approaches which is tuned via a feedback mechanism. Since these approaches are very different, for the same user, they tend to recommend different sets of movies with little or no intersection. A possible ideal approach would be to initially return 50% of movies using either approach. Based on the movie picked from the recommended list and the rating given to it, this proportion should be adjusted on the fly. To formally describe, for both approaches, titled A and B

$$\textit{Recommended\_movies} = \alpha * \{\textit{set of movies from A}\} + (1-\alpha) * \{\textit{set of movies from B}\}$$

Here α is the proportion of movies from either set and is initially set to 0.5. Thus if 10 movies are being recommended to the user, the top 5 recommended movies from either approach will be returned. If the user picks a movie from the first approach and rates it highly, α will be changed such that from the next iteration a higher proportion of movies returned by the first approach will be returned. Conversely, if the user rates the movie poorly, α will be changed such that a higher proportion of movies from the second approach will be returned. The α value will change for every rated movie and will be tuned to each user's tastes, thus providing personalized recommendations and improving the overall quality of the recommendation system.

4.7 Advantages and Disadvantages

The advantages and disadvantages of graph based recommendations systems have been mentioned below.

1. Advantages
    a. Flexibility
        i. Recommendation are made via Cypher queries, thus enabling easy modification and the development of new techniques with low overhead.
    b. Performance
        i. Neo4j is an efficient and scalable in-memory graph database.
        ii. Recommendation latency and system utilization can be reduced with efficient indexing and appropriate graph partitioning.
    c. Cost
        i. Can be efficiently operated on low cost commodity hardware, i.e. does not require GPU support.
2. Disadvantages
    a. Space
        i. While machine learning and deep learning approaches can operate directly on csv or text files to build models, storing this data in a database does increase the memory/storage footprint.

5. Machine Learning-based Approaches

5.1 Collaborative Filtering Strategy

Collaborative Filtering can be further decomposed into two techniques:

(1) Memory-based approach
(2) Model-based approach

The key difference between these two techniques is that we do not learn any parameter using gradient descent (or any other optimization algorithm) in case of memory-based approach. The closest users or items are calculated only by using metrics like Cosine similarity or Pearson correlation coefficients, which are based on simple arithmetic operations. On the other hand model-based approach uses Machine Learning to build models for prediction tasks.

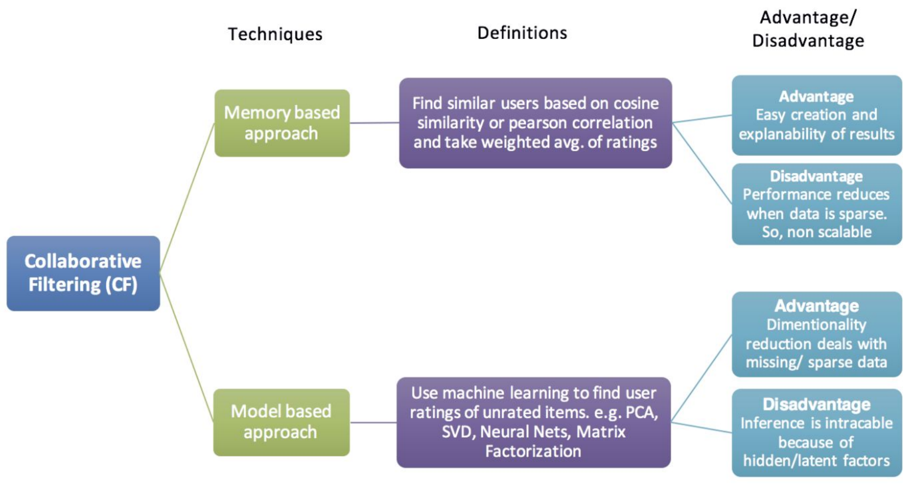

Techniques
Definitions
Advantage/ Disadvantage
Collaborative Filtering (CF)
Memory based approach
Find similar users based on cosine similarity or pearson correlation and take weighted avg. of ratings
Advantage Easy creation and explanability of results
Disadvantage Performance reduces when data is sparse. So, non scalable
Model based approach
Use machine learning to find user ratings of unrated items. e.g. PCA, SVD, Neural Nets, Matrix Factorization
Advantage Dimentionality reduction deals with missing/ sparse data
Disadvantage Inference is intracable because of hidden/latent factors

Figure 8: Types of collaborative filtering approaches

5.1.1 Memory-based approach

Memory-based approaches can be further divided into two main sections:

A. *USER-ITEM Filtering*
In this approach we take a particular user, and find similar users in the database on the basis of similarity of ratings. We then look at the movies these users liked based on the ratings they had given. A recommendation engine that uses this approach is likely to display recommendations with a message similar to “Users who are similar to you also liked these movies”.

B. *ITEM-ITEM Filtering*
In this approach we take a particular user, and look at the movies that they liked based on the ratings. We then find other users who have liked these movies, and display their highly rated movies as recommendations to the user in consideration. A recommendation engine that uses

this approach is likely to display recommendations with a message similar to "Users who liked this movie also liked these movies".

The advantages of this approach are:

- Explainability of the results
- Easy creation and use
- Content-independence of the items being recommended

The disadvantages of this approach are:

- Performance decreases when data gets sparse
- Scalability hindered in case of large datasets
- Easy to handle new users, but not new movies

Technical Implementation Specifications for Memory-Based Collaborative Filtering approach

1. Memory-Based User-Based Collaborative Filtering (UBCF):

This approach was implemented in Python using sklearn [21] and numpy [18].
The steps to implement a system that performs UBCF are as follows:

1. In the MovieLens dataset, each movie is said to belong to one or more of the 18 genres. So we create a 1D vector of size 18 to represent movies.
2. We then define *user profile* which will also be 1D vector of size 18. The difference here is that the values in the vector would be a weighted average of movies rated by the user.
    a. The movies rated by users will have the weights of (2.75 - rating)^2. The intuition here is to promote movies which user rated highly (above 2.75) and to penalize movies that was rated poorly (below 2.75).
    b. Values in *user profile* are between -1 and 1 and will now tell us if a user does or doesn't like a given genre.
3. Predict ratings for all movies that user hasn't watched.
    a. For a given movie and user, recommender system finds few other users who have watched the movie and have most similar profiles using cosine similarity metric.
    b. Then the predicted rating will be the average of ratings of those users for this film.
4. Remove movies that the user has already watched from the new list generated.
5. The system will then recommend top *n* movies with the highest ratings.

A modification made to this system was to group users from training set using k-Means algorithm, and then when predicting ratings for given movie and user, we would consider only users from the nearest group to the *user profile*.

2. Memory-Based Item-Based Collaborative Filtering (IBCF):

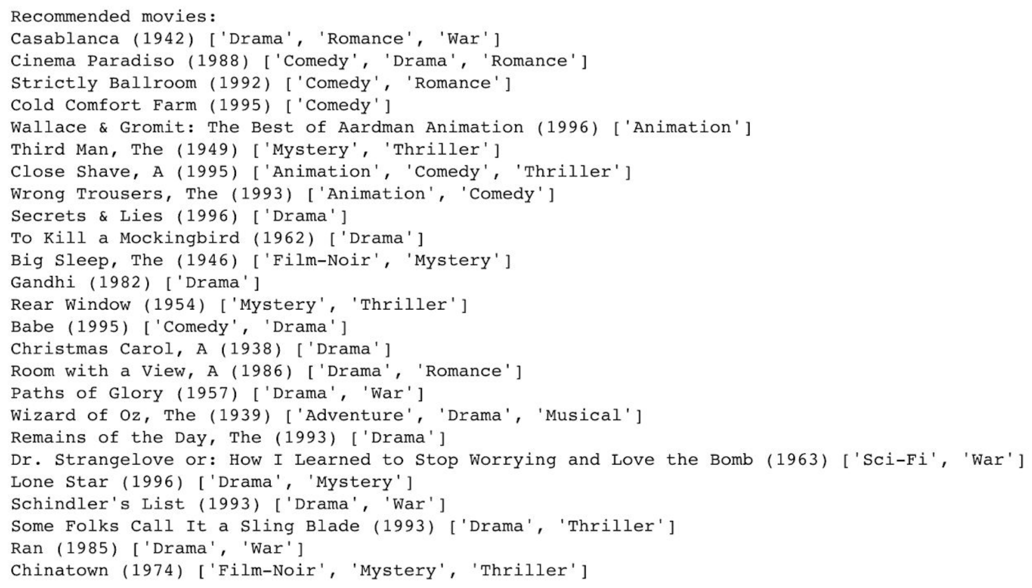
This approach was implemented in Python using pandas [19].
The steps to implement a system that performs IBCF are as follows:

1. Generate a user-ratings matrix for all users and movies.
2. Generate a correlation matrix of all movies using Pearson's correlation coefficient.

3. From the user-ratings matrix, find movies that a user has watched to create a *myRatings* list.
4. Using correlation matrix, find similar movies for each movie in *myRatings* in decreasing order of similarity scores.
5. Add up the similarity scores for duplicate movie entries.
6. Filter the resulting movie list to remove already rated movies to generate recommendations for the user.

A few sample screenshots of the system output are included below:

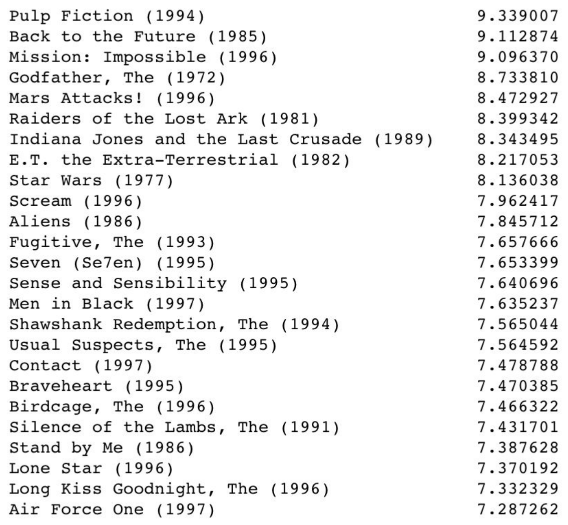
1. **Memory-Based User-Based Collaborative Filtering (UBCF):** *User ID = 17, count = 25*



```
Recommended movies:
Casablanca (1942) ['Drama', 'Romance', 'War']
Cinema Paradiso (1988) ['Comedy', 'Drama', 'Romance']
Strictly Ballroom (1992) ['Comedy', 'Romance']
Cold Comfort Farm (1995) ['Comedy']
Wallace & Gromit: The Best of Aardman Animation (1996) ['Animation']
Third Man, The (1949) ['Mystery', 'Thriller']
Close Shave, A (1995) ['Animation', 'Comedy', 'Thriller']
Wrong Trousers, The (1993) ['Animation', 'Comedy']
Secrets & Lies (1996) ['Drama']
To Kill a Mockingbird (1962) ['Drama']
Big Sleep, The (1946) ['Film-Noir', 'Mystery']
Gandhi (1982) ['Drama']
Rear Window (1954) ['Mystery', 'Thriller']
Babe (1995) ['Comedy', 'Drama']
Christmas Carol, A (1938) ['Drama']
Room with a View, A (1986) ['Drama', 'Romance']
Paths of Glory (1957) ['Drama', 'War']
Wizard of Oz, The (1939) ['Adventure', 'Drama', 'Musical']
Remains of the Day, The (1993) ['Drama']
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963) ['Sci-Fi', 'War']
Lone Star (1996) ['Drama', 'Mystery']
Schindler's List (1993) ['Drama', 'War']
Some Folks Call It a Sling Blade (1993) ['Drama', 'Thriller']
Ran (1985) ['Drama', 'War']
Chinatown (1974) ['Film-Noir', 'Mystery', 'Thriller']
```

Figure 9

2. **Memory-Based Item-Based Collaborative Filtering (IBCF):** *User ID = 17, count = 25*



```
Pulp Fiction (1994)                          9.339007
Back to the Future (1985)                    9.112874
Mission: Impossible (1996)                   9.096370
Godfather, The (1972)                        8.733810
Mars Attacks! (1996)                         8.472927
Raiders of the Lost Ark (1981)               8.399342
Indiana Jones and the Last Crusade (1989)    8.343495
E.T. the Extra-Terrestrial (1982)            8.217053
Star Wars (1977)                             8.136038
Scream (1996)                                7.962417
Aliens (1986)                                7.845712
Fugitive, The (1993)                         7.657666
Seven (Se7en) (1995)                         7.653399
Sense and Sensibility (1995)                 7.640696
Men in Black (1997)                          7.635237
Shawshank Redemption, The (1994)             7.565044
Usual Suspects, The (1995)                   7.564592
Contact (1997)                               7.478788
Braveheart (1995)                            7.470385
Birdcage, The (1996)                         7.466322
Silence of the Lambs, The (1991)             7.431701
Stand by Me (1986)                           7.387628
Lone Star (1996)                             7.370192
Long Kiss Goodnight, The (1996)              7.332329
Air Force One (1997)                         7.287262
```

Figure 10

5.1.2 Model-based approach

This approach involves developing collaborative models using Machine Learning algorithms to predict user's ratings of unrated items. Though there are multiple possible approaches, for the purpose of this project we only consider the method of Matrix Factorization.

Matrix Factorization works by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. The idea is that the attitudes or preferences of users can be determined by a few low-dimensional hidden factors called **embeddings**. Embeddings can be understood as latent features learned from the data to describe users and their preference for items (movies in this case). These embeddings aren't necessarily interpretable.

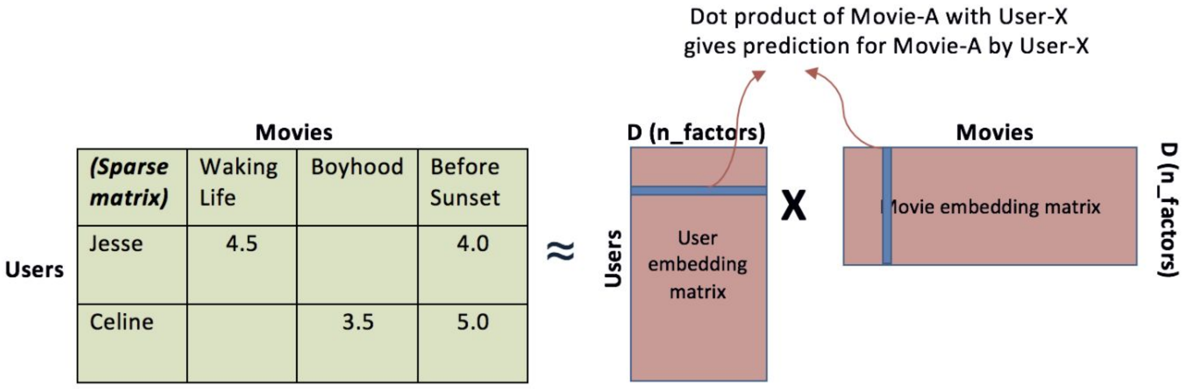


Figure 11: Visualization of Matrix Factorization

As shown in the above figure, when the users-movies matrix is factorized into two lower-dimensional matrices namely the user embedding matrix and the movie embedding matrix, an additional parameter of *D* gets introduced. The parameter *D* here denotes the dimensions of the embeddings for both, users and movies. Then for user-A and movie-X we can say that the *D* numbers might represent *D* different characteristics about the user and the movie.

The advantages of this approach are:

- User-movie matrix compressed into latent features
    - Low-dimensional representation that requires less storage
    - Handles sparsity better than other approaches
    - Much more scalable compared to other approaches
- Movie Representation
    - Described by analogous set of *k* attributes or features
    - Captures similarities beyond obvious attributes like genre
    - Handles new users (cold start) well

The disadvantages of this approach are:

- Requires complete matrices to work with

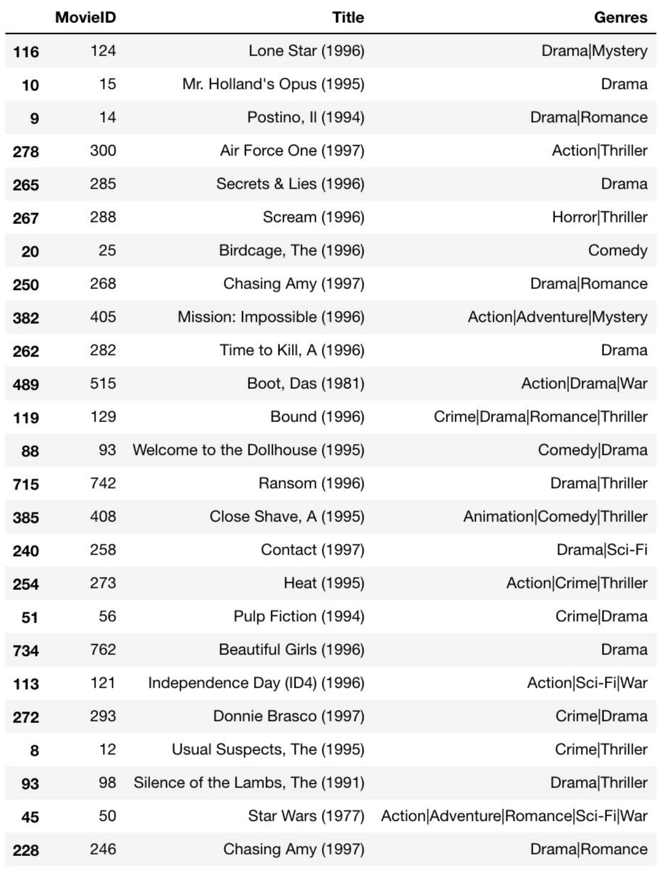
- Model building might be expensive depending on data size

Technical Implementation Specifications for Model-Based Collaborative Filtering approach

Matrix Factorization cad be done by various methods:
- Orthogonal Factorization (SVD)
- Probabilistic Factorization (PMF)
- Non-negative Factorization (NMF)

For this project we implemented orthogonal matrix factorization using SVD technique. The code makes use of the SciPy [22] package for the implementation. Since the users-movies matrix would generally be sparse, there might be missing values. However, this approach requires a complete matrix. To deal with the missing values, we assume them to be 0 instead of imputing (by taking a mean of the remaining values), or ignoring the instances.

A sample screenshot of the system output is included below:
1. Model-Based Collaborative Filtering (Matrix-Factorization): *User ID = 17, count = 25*



	MovieID	Title	Genres
116	124	Lone Star (1996)	Drama|Mystery
10	15	Mr. Holland's Opus (1995)	Drama
9	14	Postino, Il (1994)	Drama|Romance
278	300	Air Force One (1997)	Action|Thriller
265	285	Secrets & Lies (1996)	Drama
267	288	Scream (1996)	Horror|Thriller
20	25	Birdcage, The (1996)	Comedy
250	268	Chasing Amy (1997)	Drama|Romance
382	405	Mission: Impossible (1996)	Action|Adventure|Mystery
262	282	Time to Kill, A (1996)	Drama
489	515	Boot, Das (1981)	Action|Drama|War
119	129	Bound (1996)	Crime|Drama|Romance|Thriller
88	93	Welcome to the Dollhouse (1995)	Comedy|Drama
715	742	Ransom (1996)	Drama|Thriller
385	408	Close Shave, A (1995)	Animation|Comedy|Thriller
240	258	Contact (1997)	Drama|Sci-Fi
254	273	Heat (1995)	Action|Crime|Thriller
51	56	Pulp Fiction (1994)	Crime|Drama
734	762	Beautiful Girls (1996)	Drama
113	121	Independence Day (ID4) (1996)	Action|Sci-Fi|War
272	293	Donnie Brasco (1997)	Crime|Drama
8	12	Usual Suspects, The (1995)	Crime|Thriller
93	98	Silence of the Lambs, The (1991)	Drama|Thriller
45	50	Star Wars (1977)	Action|Adventure|Romance|Sci-Fi|War
228	246	Chasing Amy (1997)	Drama|Romance

Figure 12

5.1.3 K-Means Clustering Approach

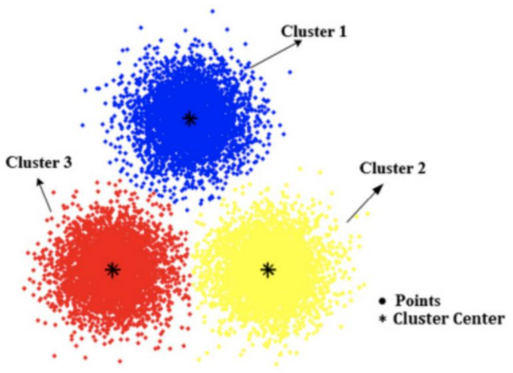


Figure 13: K-Means clustering

K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster [Wikipedia]

Approach:

This approach makes use of K-means clustering on the MovieLens dataset in order to predict movies to users.

- The approach involves clustering users into groups, depending on similarity in movie ratings.
- The cosine similarity metric was used to find these groups, based on Euclidean distance similarity measure.
- For each cluster, the 'n' most rated movies are identified along with their genre, which is then predicted for each cluster.

Choosing K:

The value of K for K-means is chosen in the following ways:

- **Elbow method:** The elbow method involves plotting a graph of K values alongside metrics such as distortion to find the appropriate K value. The number of clusters should be chosen in a way that adding another cluster doesn't give much better modeling of the data. The first few clusters will add much information but the marginal gain drops and gives an angle in the graph, which gives us the K value or 'elbow criterion'. [Wikipedia]

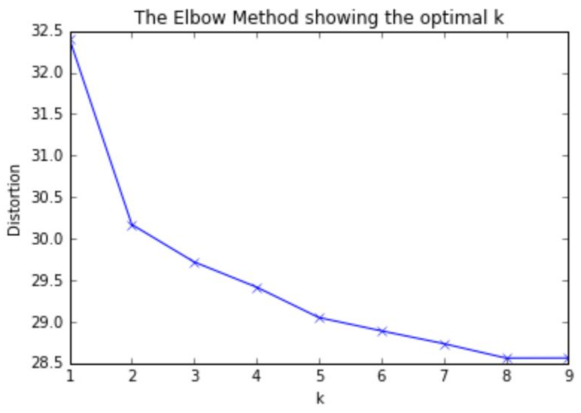


Figure 14: Elbow graph (k v/s Distortion)

The above figure depicts a weak elbow point at k=2. Therefore, we explore another metric, namely the Silhouette score.

- **Silhouette score:** This technique provides a representation of how well each object lies within its cluster. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. [Wikipedia]

```
('Sil score: ', 0.23389422149430766)
&quot;For number of clusters: &quot; 2 &quot; average sil score:&quot; 0.233894221494
('Sil score: ', 0.12940146000344746)
&quot;For number of clusters: &quot; 3 &quot; average sil score:&quot; 0.129401460003
('Sil score: ', 0.13680472445797762)
&quot;For number of clusters: &quot; 4 &quot; average sil score:&quot; 0.136804724458
('Sil score: ', 0.028963897502352901)
&quot;For number of clusters: &quot; 5 &quot; average sil score:&quot; 0.0289638975024
('Sil score: ', 0.018426879956807116)
&quot;For number of clusters: &quot; 6 &quot; average sil score:&quot; 0.0184268799568
('Sil score: ', -0.013050973891557649)
&quot;For number of clusters: &quot; 7 &quot; average sil score:&quot; -0.0130509738916
('Sil score: ', -0.014885665972013242)
&quot;For number of clusters: &quot; 8 &quot; average sil score:&quot; -0.014885665972
('Sil score: ', -0.01584753482386066)
&quot;For number of clusters: &quot; 9 &quot; average sil score:&quot; -0.0158475348239
('Sil score: ', -0.015816501704100806)
&quot;For number of clusters: &quot; 10 &quot; average sil score:&quot; -0.0158165017041
```

Figure 15: Silhouette scores for k=2 to k = 10

It is observed that silhouette scores for k=2 and k=4 depict the top two highest values, hence experiments are conducted using both k values.

Advantages of this Approach:

- Can assign new user to a cluster
- Can identify groups of users if present
- Finds similarity in user ratings, considers similarity in judgements of other users

Disadvantages of this Approach:

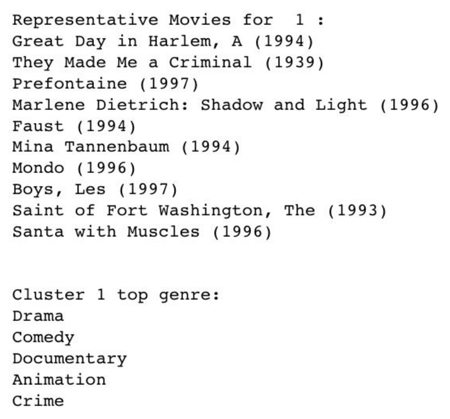
- Assumes data can be grouped easily into groups of users

- Too much relies on an accurate k value
- Does not take user's individual preferences into account
- Does not take in information such as genre into account

Screenshots of the sample output:

For User ID = 944

K=2

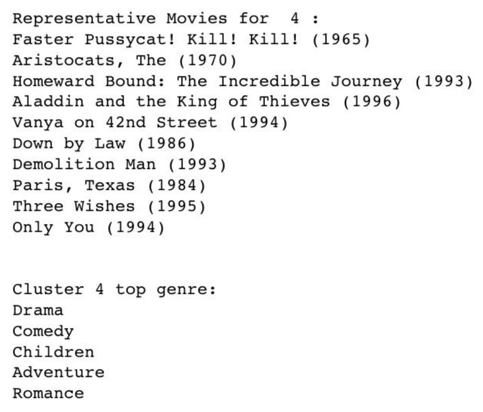


```
Representative Movies for  1 :
Great Day in Harlem, A (1994)
They Made Me a Criminal (1939)
Prefontaine (1997)
Marlene Dietrich: Shadow and Light (1996)
Faust (1994)
Mina Tannenbaum (1994)
Mondo (1996)
Boys, Les (1997)
Saint of Fort Washington, The (1993)
Santa with Muscles (1996)


Cluster 1 top genre:
Drama
Comedy
Documentary
Animation
Crime
```

Figure 16: Genre and movies predicted for user's cluster (k=2)

K=4



```
Representative Movies for  4 :
Faster Pussycat! Kill! Kill! (1965)
Aristocats, The (1970)
Homeward Bound: The Incredible Journey (1993)
Aladdin and the King of Thieves (1996)
Vanya on 42nd Street (1994)
Down by Law (1986)
Demolition Man (1993)
Paris, Texas (1984)
Three Wishes (1995)
Only You (1994)


Cluster 4 top genre:
Drama
Comedy
Children
Adventure
Romance
```

Figure 17: Genre and movies predicted for user's cluster (k=4)

5.2 Content-based Filtering Strategy

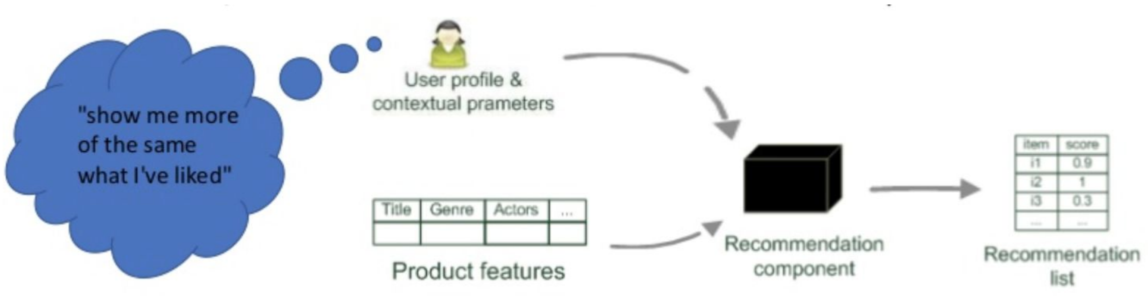

"show me more of the same what I've liked"
User profile & contextual prameters
Title | Genre | Actors | ...
Product features
Recommendation component
item | score
i1 | 0.9
i2 | 1
i3 | 0.3
Recommendation list

Figure 18: Diagram depicting Content-based Filtering approach

Content-based filtering is a technique that focuses on learning user preferences and recommending items similar to these preferences. This similarity is computed from item attributes such as: finding similarity between movies by genre, year or finding similarity in text by looking at words or topics.

5.2.1 Identifying similar users using TF-IDF

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.[1] It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. [Wikipedia]

The calculation of TF-IDF for a word i in document j is given by the formula below:

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \log\left(\frac{\mathrm{N}}{\mathrm{df}_i}\right)$$

$\mathrm{tf}_{i,j}$ = total number of occurences of i in j

df_i = total number of documents (speeches) containing i

N = total number of documents (speeches)

Figure 19: Calculation of TF-IDF scores

Approach

This approach computes similarity between movies based on list of genres associated with user-rated movies

- The highest rated movie for each user is chosen as the starting point
- TF-IDF transforms the text (genres) into feature vectors which can now be input to the estimator
- The pairwise cosine similarity is obtained via a 'dot product' of the TF-IDF matrix
- The N highest scores are then returned as recommendations to the user

Advantages of this Approach

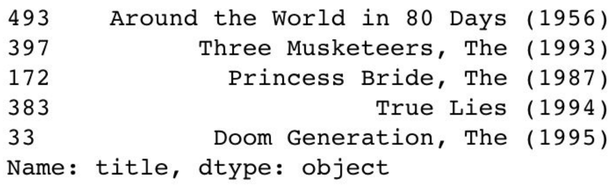
- Can identify user's preferred genre
- Relies on movie genre data; data on other users is not needed
- Takes into account multiple genre tags while recommendation
- Can recommend movies of unique genre

Disadvantages of this Approach

- Does not take into account user judgement
- Does not take into account user preferences apart from genre

Screenshots of the sample output

For User ID = 944



```
493     Around the World in 80 Days (1956)
397            Three Musketeers, The (1993)
172              Princess Bride, The (1987)
383                        True Lies (1994)
33              Doom Generation, The (1995)
Name: title, dtype: object
```

Figure 20: Top 5 movie recommendations for User ID = 944

5.2.2 Preference-based approach using KNN

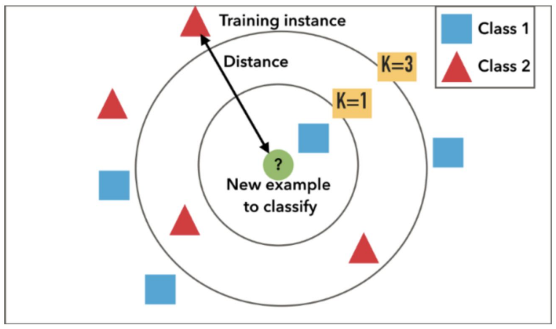


Figure 21: Diagram depicting KNN classification

The *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric method used for classification and regression. The input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression. In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.[Wikipedia]

Approach

This approach computes user profile based on a weighted average of types of movies watched

- User profile is created by computing weighted average of movies watched using genre and rating information
- K-NN algorithm is used along with a cosine similarity metric to find the closest user's profile
- Top-rated movies are then output based on similar user profile

Advantages of this Approach

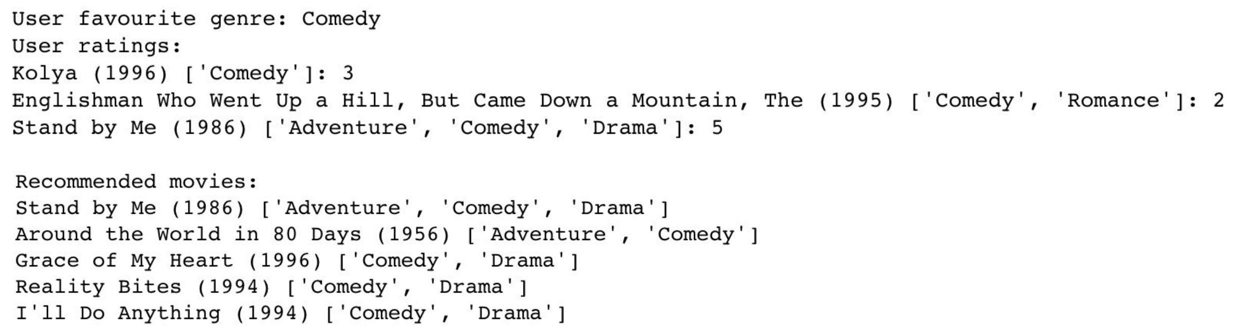
- This approach requires building user profile using genre information and corresponding rating
- It takes into account genre information as well as ratings for movies to identify recommended movies, thus modeling user preference

Disadvantages of this Approach

- It does not rely on other information such as age group for building user profile
- User profile is not influenced by peers or friends (which is not the case in real-world scenarios)

Screenshots of the sample output

For User ID = 944



```
User favourite genre: Comedy
User ratings:
Kolya (1996) ['Comedy']: 3
Englishman Who Went Up a Hill, But Came Down a Mountain, The (1995) ['Comedy', 'Romance']: 2
Stand by Me (1986) ['Adventure', 'Comedy', 'Drama']: 5

Recommended movies:
Stand by Me (1986) ['Adventure', 'Comedy', 'Drama']
Around the World in 80 Days (1956) ['Adventure', 'Comedy']
Grace of My Heart (1996) ['Comedy', 'Drama']
Reality Bites (1994) ['Comedy', 'Drama']
I'll Do Anything (1994) ['Comedy', 'Drama']
```

Figure 22: Identifying user's favorite genre and recommending movies accordingly

5.2.3 PROs of Content-based Filtering

- There is no need for data on other users, thus no sparsity problems
- We can recommend items to users with unique tastes
- New and even unpopular items can be recommended due to preference-based similarity
- We can provide explanations for recommended items by listing content-features that caused an item to be recommended (such as genre)

5.2.4 CONs of Content-based Filtering

- Finding the appropriate features is hard
- This method does not recommend items outside a user's content profile
- We are unable to exploit quality judgments of other users

6. Autoencoder

6.1 Introduction

Autoencoders are unsupervised machine learning algorithms which can be used to learn a low dimensional encoding for input data. Autoencoders typically consist of 2 parts: an encoder and a decoder. The encoder takes in a high dimensional input and maps it to a lower dimensional output called latent representation. This latent representation is then fed to the decoder which generates a high dimensional output. The loss function tries to minimize the difference between the input to the encoder and the output of the decoder. Once this loss function is minimized, the latent representation at the output of the encoder can act as a low dimensional equivalent of the high dimensional input. Hence by limiting the number of dimensions of the latent representation, we are able to learn complex representations of the input. Hence, autoencoder is able to achieve dimensionality reduction. So, in terms of its objective, an autoencoder is similar to Principal Component Analysis. However, its advantage lies in the fact that, while PCA is restricted to a linear map, autoencoders can have non-linear encoders/decoders.

6.2 Network Architecture

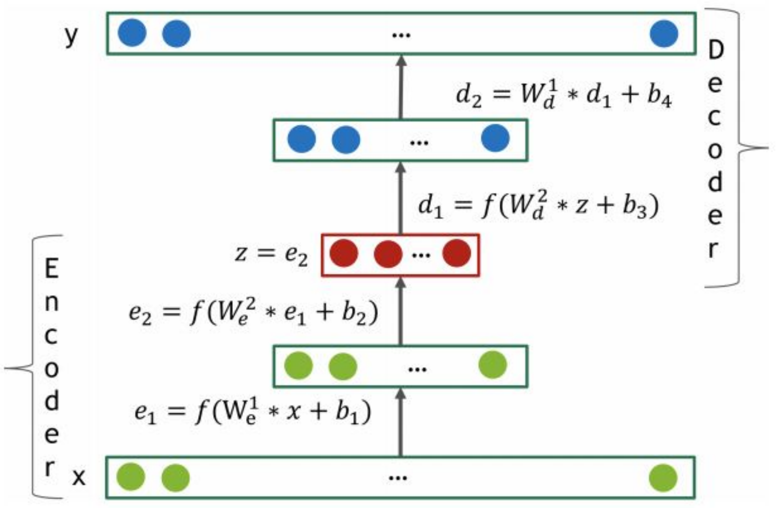

y
$d_2 = W_d^1 * d_1 + b_4$
$d_1 = f(W_d^2 * z + b_3)$
$z = e_2$
$e_2 = f(W_e^2 * e_1 + b_2)$
$e_1 = f(W_e^1 * x + b_1)$
x
Decoder
Encoder

Figure 23: Architecture for autoencoder used for recommendation systems

The first layer (from the bottom) marked as x is a single row in the user-movie matrix i.e. it contains the ratings a single user has given to all the movies in the database. If the user has not rated a movie, the entry for the corresponding position will be zero. This vector is then passed through a fully connected layer with a nonlinearity denoted by the function f. This nonlinearity can be sigmoid, tanh or ReLU. After another fully connected layer, we get z which is the latent representation. This comprises the encoder part of the network. The output of the encoder is then fed to the decoder which again consists of 2 fully connected layers. While the dimension of the input consistently decreases for the encoder, for the decoder, it consistently increases. So at the end of the decoder, we have a vector which has the same number of dimensions as the input.

Even though the dimensions of the input to the encoder and the output of the decoder are same, we cannot use Root Mean Square Error (RMSE) as a loss function for gradient descent. This is because, as stated in the beginning, the user has not rated all the movies in the dataset, in fact, each user has rated only a few movies, so the input vector will be highly sparse. We will have to penalize the network only in the cases where the user has rated the movie and the network predicts an incorrect rating. For this purpose, we design a custom loss function called Masked Mean Square Error (MMSE) as given below.

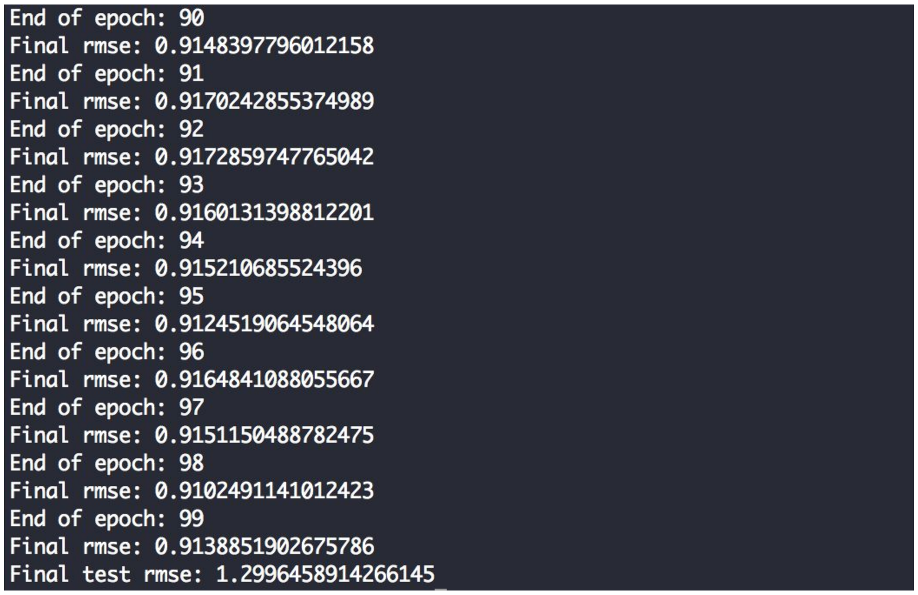
$$MMSE = \frac{m_i * (r_i - y_i)^2}{\sum_{i=0}^{i=n} m_i}$$

Figure 24: MMSE- Where r_i = actual rating
y_i =reconstructed rating
m_i = masking function such that $m_i = 0$ is $r_i = 0$ and $m_i = 1$ otherwise

We use gradient descent to minimize MMSE and once gradient descent converges, the autoencoder has been trained to reduce the dimensionality of the sparse input and produce a dense latent representation. During the testing phase, we feed the tuple of ratings to the encoder for the user in concern. The output of the decoder will be of the same dimensions as the input but will not be as sparse. We can use this decoder output to figure out which movies the user is likely to enjoy by finding the indices where the output of the decoder is high.

6.3 Results

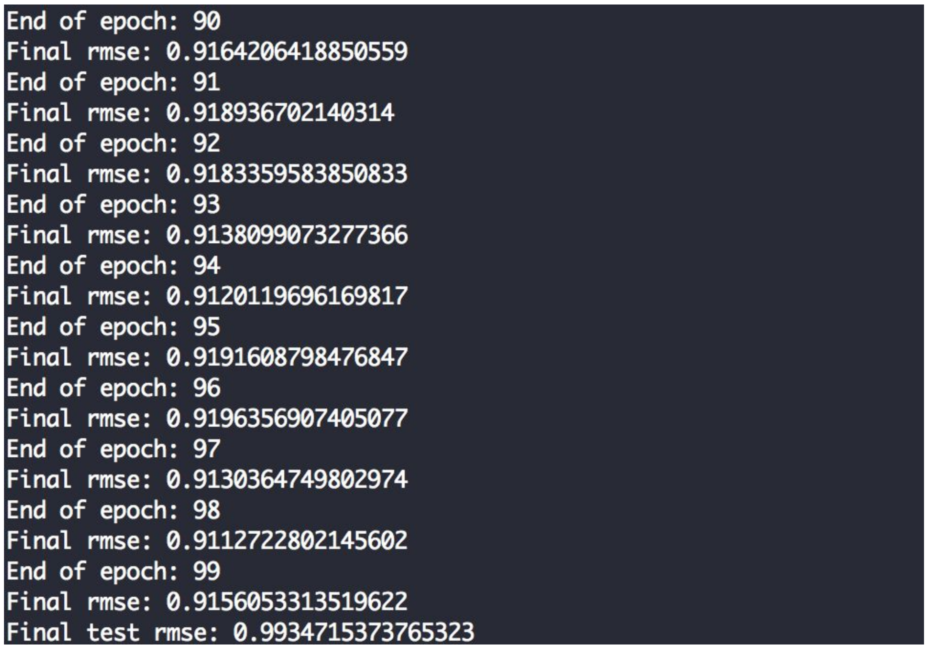
To measure the accuracy of the autoencoder, we test the performance of the trained model on the test set. We use Root Mean Square Error (RMSE) as a measure of performance. RMSE is calculated as the square root of the sum of squares of differences between the predicted rating and the actual rating given by the user. If we train the model on MovieLens 100k dataset, we see the following results:



```
End of epoch: 90
Final rmse: 0.9148397796012158
End of epoch: 91
Final rmse: 0.9170242855374989
End of epoch: 92
Final rmse: 0.9172859747765042
End of epoch: 93
Final rmse: 0.9160131398812201
End of epoch: 94
Final rmse: 0.915210685524396
End of epoch: 95
Final rmse: 0.9124519064548064
End of epoch: 96
Final rmse: 0.9164841088055667
End of epoch: 97
Final rmse: 0.9151150488782475
End of epoch: 98
Final rmse: 0.9102491141012423
End of epoch: 99
Final rmse: 0.9138851902675786
Final test rmse: 1.2996458914266145
```

Figure 25: Train and Test RMSE for MovieLens 100k dataset

From this figure, we see that the train RMSE is 0.9138 and the test RMSE is 1.2996. This might be an indication of the fact that the network was overfitting the training set. In order to verify our intuition, we tested the performance of our algorithm on MovieLens 1M dataset that has 1 million rows rather than 100,000.



```
End of epoch: 90
Final rmse: 0.9164206418850559
End of epoch: 91
Final rmse: 0.918936702140314
End of epoch: 92
Final rmse: 0.9183359583850833
End of epoch: 93
Final rmse: 0.9138099073277366
End of epoch: 94
Final rmse: 0.9120119696169817
End of epoch: 95
Final rmse: 0.9191608798476847
End of epoch: 96
Final rmse: 0.9196356907405077
End of epoch: 97
Final rmse: 0.9130364749802974
End of epoch: 98
Final rmse: 0.9112722802145602
End of epoch: 99
Final rmse: 0.9156053313519622
Final test rmse: 0.9934715373765323
```

Figure 26: Train and Test RMSE for MovieLens 1M dataset

From this figure, it is evident that the train RMSE has dropped to 0.91560 and the test RMSE has been reduced to 0.9934. This proves that our intuition was correct and the network was indeed overfitting the smaller training set. This does not come as a surprise as we know that deep neural networks are indeed data hungry. As a large number of weights are to be learnt, they can easily overfit smaller training sets. The greater the size of the training set, the better will be the generalization capability of the deep neural network.

7. Experimental Results

7.1 Training / Preprocessing Time

In this experiment, we investigate how much time is needed for training the machine learning and deep learning models for different training set sizes. For the graph, we find how much time is needed to build a graph for different dataset sizes. In the figure below, we increased the size of the training set in increments of 10,000 and noted the training / pre-processing time for various algorithms.

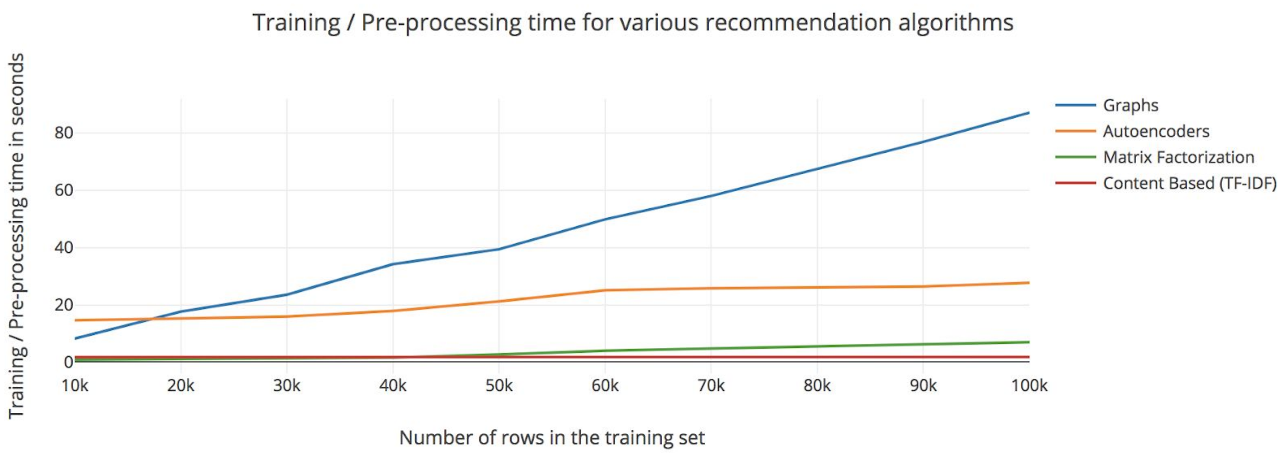


Figure 27: Training time analysis

From the above graph, it is evident that the time needed to build a graph increases almost linearly with respect to the number of rows in the dataset. Even though there is increase in the training time with respect to size for the machine learning and deep learning models, the increase is not as significant as that for the graph. Between the machine learning and deep learning algorithms, the deep learning one consumes more training time as due to more layers, there is significantly more time needed for forward and backward propagation. The training time for matrix factorization and content based filtering algorithms also increases with the size of the training dataset but this increase is miniscule with respect what is observed in graphs and autoencoders.

7.2 Size of the model / graph

In this experiment, we investigate how much space is needed to save the trained model / graph for machine learning and deep learning models for different training set sizes. As an equivalent, for the graph, we find size of the graph built for different dataset sizes. In the figure below, we increased the size of the training set in increments of 10,000 and noted the size of the models formed.

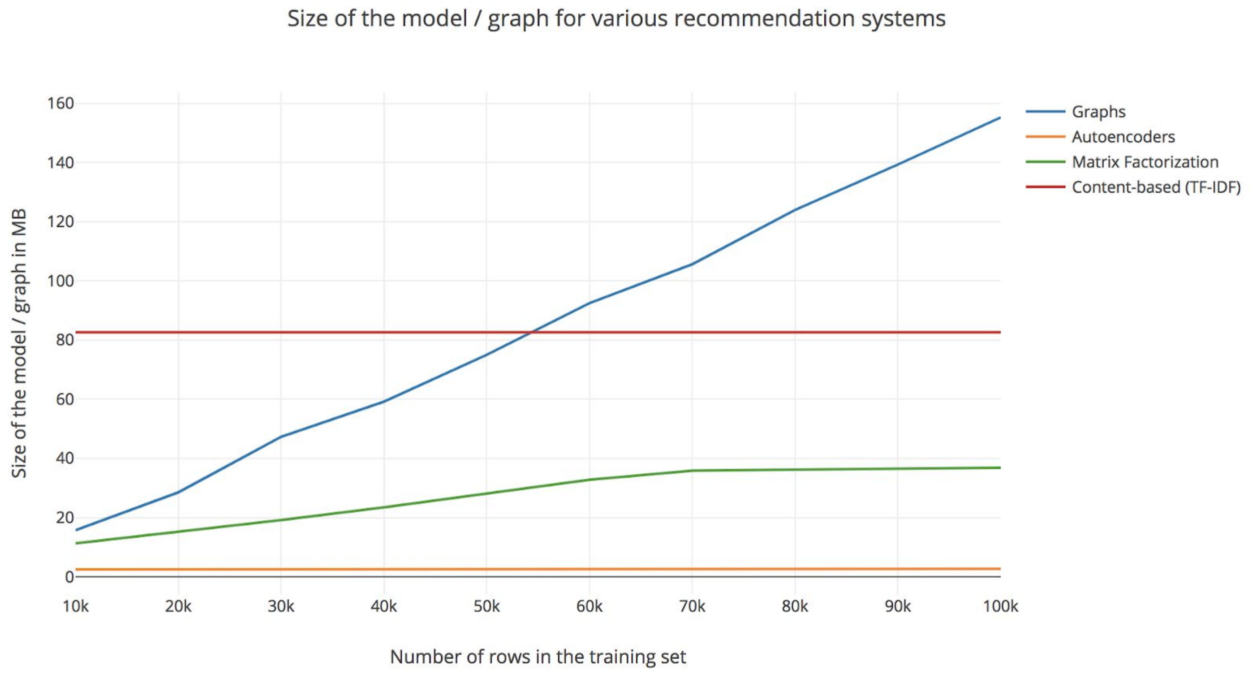


Figure 28: Model size analysis

From the above graph, we observe that model size for the graph-based approach increases almost linearly with the increase in the number of ratings. This is justified as with each new user and with each new movie, a new node will be added to the graph. This will also be accompanied by new edges which will contribute to an increase in size of the graph model. However, in case of deep learning, the size of the model increases by a very small margin (2.5MB to 2.7MB) when the number of rows in the training set increase from 10,000 to 100,000. We feel this is justified, as even with an increase in the number of rows in the training set, the model architecture is not changing. There are no nodes or edges being added to the computation graph. So as the same weights are being updated, there is no significant increase in the size of the model. For the Matrix-Factorization based collaborative filtering approach, the size of the intermediate matrix model increases with the increase in the number of ratings initially, but after a point the difference in size is not easily visible. The Content-based approach involved the construction of a TF-IDF matrix using genre information for each movie in the dataset, and takes up considerable space, which, however remains constant and does not change with the number of ratings.

7.3 Recommendations for a Guinea Pig User

In order to determine the success/failure ratio of our recommendation engines, we decided to ask a guinea pig user to rate his experience. We asked him to rate 15 classic movies that he had seen previously, from a variety of genres, and then proceeded to provide 10 recommendations. For each of these recommended movies, he provided his rating if he had seen the movie. If the movie was unseen, the rating was left blank and that movie wasn't considered. The ratings given to these recommended movies have been mentioned in the table below. The 'Rating' column refers to the rating given by the guinea pig user. Here, for the graph based method, only the first run results were considered, that is 50% from collaborative and 50% from content-based approaches.

Graph (Personalized Recommendations)		**ML Collaborative filtering**		**ML Content-based Filtering**		**Deep Learning**	
Movie	**Rating**	**Movie**	**Rating**	**Movie**	**Rating**	**Movie**	**Rating**
Godfather, The (1972)	5	Apt Pupil (1998)	3	Diva (1981)	5	Man Who Knew Too Little, The (1997)	2
Fargo (1996)	4	Full Monty, The (1997)	4	City Hall (1996)	3	Mrs. Doubtfire (1993)	5
Raiders of the Lost Ark (1981)	3	Seven Years in Tibet (1997)	5	Silence of the Lambs, The (1991)	5	Jungle Book, The (1994)	5
Pulp Fiction (1994)	5	Starship Troopers (1997)	4	Extreme Measures (1996)	3	Blue Angel, The (1930)	1
Toy Story (1995)	2	Tomorrow Never Dies (1997)	3	Murder in the First (1995)	4	Big Bully (1996)	2
Twelve Monkeys (1995)	3	Amistad (1997)	4	Eye for an Eye (1996)	4	Speed (1994)	5
Fugitive, The (1993)	4	As Good As It Gets (1997)	5	Volcano (1997)	4	Secret Garden, The (1993)	3
Back to the Future (1985)	3	Cop Land (1997)	2	Trigger Effect, The (1996)	3	Somebody to Love (1994)	3
Shawshank Redemption, The (1994)	5	G.I. Jane (1997)	2	Apt Pupil (1998)	3	Mortal Kombat (1995)	5
Blade Runner (1982)	4	Scream 2 (1997)	3	Last Supper, The (1995)	2	Shadows (Cienie) (1988)	2

Based on the data present in the table, we came up with a weighted metric to rank each of these factors. We used such a weighted rank since we believe that highest weight should be given to movies which the user absolutely loved, and lowest to the ones he disliked.

Weighted_rank = 5(# of 5 ratings given) + 4*(# of 4 ratings given) + 3*(# of 3 ratings given) + 2*(# of 2 ratings given) + 1*(# of 1 ratings given)*

The table below shows the weighted ranks for each of the approaches. For this evaluation, there are a few important points to note.

- These rating values are human dependent. Each users tastes are subjective and different people would provide different ratings. These weighted scores show the approach which is favored by a a guinea pig user, and can not be generalized. I.e., these scores provide the ranking of approaches **specifically for this user**.
- For the graph method, only the first run was considered. The graph-based approach has been built with a feedback mechanism such that as the number of ratings given by the user increases, the perform of the recommendation engine improves as it is tuned to the likings/behaviour of the user. This behaviour has not been modeled in this experiment to keep all approaches on a level playing field.

Graph	38
ML Collaborating Filtering	35
ML Content-based Filtering	36
Deep Learning	33

From the weighted score results we can see that all of these approaches performed almost similarly for this user. These values may change based on user behaviour, and hence it is important to take the users feedback into consideration. In a real world scenario, since the graph approach tailors recommendations towards user behaviour, after a few runs, the graph approach would provide better recommendations. These results were obtained for MovieLens 100k dataset, and as we have seen before, the RMSE on test set decreases for the neural network when we train it on MovieLens 1Million dataset. So we believe that the low score of the autoencoder in this case can be attributed to overfitting the small training set.

7.4 Recommendations for a New User

In this experiment, we added a new user to the system. The following describes how each approach handles the case of the new user.

1. **Graphs :** If the user doesn't have any ratings, most popular movies (movie which is rated highly by a majority of people in the data set) are returned. On the other hand, even if a few, or just

one user, has rated this new movie, both approaches will work well. The collaborative filtering approach will work since, even with one rating, that's one edge in the graph which can be used to determine other similar users.

2. **ML Collaborative Filtering:** If the user doesn't have any ratings, most popular movies (movie which is rated highly by a majority of people in the data set) are returned by all collaborative filtering approaches. If a few ratings can be gathered initially, matrix factorization outperforms other approaches since it can generate the user-movie vector. We already have the movie-feature vector, and thus we can easily generate the user-feature vector.
3. **ML Content-based Filtering:** For a new user, both content-based filtering approaches return most popular movies. However, even if there is a single rating, we can model both approaches based on the genre of the watched movie and the rating.
4. **Deep Learning (Autoencoder):** As is the case with most deep learning algorithms, the results of a autoencoders are difficult to interpret. This was the case in this experiment as well. We tried creating multiple new users in order to see whether we find some pattern (eg. predicting classics or highest rated movies) in the output of the autoencoder but we were unable to find any specific pattern. Hence, the autoencoder will need some more ratings from the user before it can provide suggestions that align with his/her interests.

All the above approaches rely on the popularity model, and 'FAIL' in the following two ways:

1. If the user has unique tastes, the popularity model fails to take his individuality into account
2. The popularity model does not incorporate a language preference
   For example: Most popular movies in the dataset that were observed to be recommended were in English, and if user is not familiar with English, this strategy fails thoroughly.

Hence, for experimentation purposes, we decided to simulate the case of a cold start by adding a new user with only 3 ratings as shown below:

Movie ID	**Rating**	**Movie Title**	**Genre**
242	3	Kolya (1996)	Comedy
580	2	Englishman Who Went Up a Hill, But Came Down a Mountain, The (1995)	Comedy|Romance
655	5	Stand by Me (1986)	Adventure|Comedy|Drama

We then tested all four of our main approaches to see what movies might be recommended to the new user, and recorded our observations.

Graph (Personalized Recommendations)	**ML Collaborative filtering**	**ML Content-based Filtering**	**Deep Learning**
Prefontaine	Field of Dreams	Around the World in 80	Above the Rim

		Days	
Star Kid	Philadelphia	Three Musketeers, The	Angela
Godfather, The	Dead Man Walking	Princess Bride, The	Big Bang Theory, The
Everest	Back to the Future	True Lies	Johns
Usual Suspects, The	When Harry Met Sally	Doom Generation, The	Chairman of the Board
Star Wars	To Kill a Mockingbird	Eat Drink Man Woman	Nightwatch
Shawshank Redemption, The	Dances With Wolves	Ed Wood	Great Day in Harlem
Roman Holiday	The Shawshank Redemption	What's Eating Gilbert Grape	All Things Fair
Rear Window	Indiana Jones and the Last Crusade	Welcome to the Dollhouse	Godfather, The
Schindler’s List	2 Days in the Valley	Swingers	Target

From the above table, we observe that very few movies (highlighted) were found to be similar amongst the implemented approaches. We try to explain this variation by citing the differences in each approach. The graph model has a few popular movies amongst its recommendations using the popularity model. The Collaborative Filtering approach used matrix factorization technique to return movies with genres similar to the 3 rated movies while also considering popular movies. The Content-based approach, however only takes into account user-preferences, without including user popularity. It was found that the Content-based approach was successful in identifying and recommending movies based on the most commonly observed genres associated with the user’s previously watched movies. It is difficult to interpret the recommendations given by the autoencoder as only one movie in its output seems to be a popular classic. This is one of the drawbacks of neural networks as they are less interpretable as compared to graph algorithms or traditional machine learning models.

7.5 Recommendations for a New Movie

In this experiment, we added a new movie in in the database. We then checked which of our methods recommended this new movie despite it not having any ratings.

Graphs:
When a new movie is added to the database, and it hasn’t been rated by anyone, the collaborative filtering approach will fail since there is no edge between this movie and users. On the other hand, the

content-based filtering approach will work since it is only dependent on the semantic similarity between movie synopsis.

ML Collaborative Filtering:
User-based collaborative filtering approach will fail for a new movie. In case of item-based collaborative filtering if the approach can find a similarity between the new movie and some existing movie in the user's profile, it can recommend new movies. If not, then the approach might fail. As for matrix factorization, if a new movie is added we can generate the movie-feature vector and we already have the user-feature vector, making it easy to generate user-movie vector for recommendations.

ML Content-based Filtering:
Both content-based approaches are genre-dependent, and equipped to handle a case where a new movie is added. The algorithms are able to recommend the new movie if the user has seen movies belonging to the same genre as the new movie.

Autoencoder:
When a new movie is added to the dataset, for the changes to reflect in the output, the autoencoder must be retrained. Even after retraining, the movie will not cause a significant change in the latent representation of the encoder unless it has significant number of ratings. Hence, a new movie added to the dataset is not immediately recommended.

7.6 Recommendations for a New Genre

In this experiment, we added a new movie in in the database which had a previously unseen genre. We then checked which of our methods recommended this new movie despite never having seen its genre and not having any ratings.

Graphs:
The movie will still be recommended in the content-based filtering method. For example, if a new movie X is added to the database with genre 'action-adventure', and the target user has watched and highly rated movie Y whose genre is 'Action', then movie X will be recommended to the user due to the similarity in the movie synopsis between X and Y, thus, allowing for recommendations even when no rating (edge between users and movies) is present in the graph.

ML Collaborative Filtering:
User-based collaborative filtering technique will fail in this case since it works on existing genres in the dataset. Item-based collaborative filtering technique can give recommendations if the movie with the new genre has a high correlation with other movies in the dataset. Matrix Factorization takes in only the ratings of the movie, but the way it functions is that it captures the underlying hidden features that are similar to a group of movies in the dataset. Hence, even for a movie with the new genre Matrix Factorization can still recommend similar movies.

Autoencoders:
The autoencoder does not take genre of the movie into account during training. Hence, the autoencoder is completely oblivious to the new genre added. It does not use the metadata in the form of genre or movie synopsis while recommending movies. This is big drawback of the autoencoder when compared with content based approaches which utilize the metadata.

8. Conclusion and Lessons Learnt

This project has helped us understand how Graph Theory, Machine Learning (ML) and Deep Learning (DL) can be applied to the problem of movie recommendations, and how each of the implemented approaches demonstrated their own advantages and disadvantages for each of the use cases that we explored.

In this section we enumerate the important points we have learnt while doing this project.

1. The 'No Free Lunch Theorem' applies to recommendation systems as well. There is no single algorithm that works in all use cases. The choice of algorithm depends on the requirements.
2. If memory size is a concern, machine learning and deep learning methods should be preferred as, according to figure no. 28, we learnt that the size of the graph increases linearly with respect to the size of the training dataset, but the size of the ML and DL model does not increase much as more data will just update the same number of weights in the model.
3. Graph approach takes a lot of time to build the initial graph as per figure no. 27. This is greater than the time taken to train the machine learning or deep learning models.
4. If there is a requirement for always providing recommendations using the most recent data, the graph algorithm should be preferred as the graph always reflects all the data we have in our dataset. On the other hand, the machine learning and deep learning models require periodic retraining if we need them to reflect the most recent data. This retraining overhead can prove to be costly. Hence graph models are more scalable.
5. If there are new items (in this case movies) being added to your dataset, it helps to use some metadata about them. For example, approaches like Content based filtering using TF-IDF which look at the movie description work better than graph or deep learning models which do not make use of such metadata.
6. In case of graph models, it is important to perform efficient indexing of data and transaction processing to prevent single large commits.
7. Graph models have better interpretability as compared to ML and DL techniques.

As future scope, we intend to explore techniques that combine graph-based and deep-learning based approaches by creating a system that would use the TensorFlow machine learning framework and the Neo4j graph database to generate recommendations for the user [14].

References

[1] Portugal, Ivens, Paulo Alencar, and Donald Cowan. "The use of machine learning algorithms in recommender systems: a systematic review." *Expert Systems with Applications* (2017).

[2] Sahu, S.P., Nautiyal, A. and Prasad, M., “Machine Learning Algorithms for Recommender System-a comparative analysis”

[3] Bhatt, Bhumika G. et al. “A Review Paper on Machine Learning Based Recommendation System.” (2014).

[4] Graph based recommendation engine for Amazon products
https://towardsdatascience.com/graph-based-recommendation-engine-for-amazon-products-1a373e639263

[5] Introducing Pixie, an advanced graph-based recommendation system
https://medium.com/@Pinterest_Engineering/introducing-pixie-an-advanced-graph-based-recommendation-system-e7b4229b664b

[6] Yang, Kaige, and Laura Toni. "Graph-Based Recommendation System." *arXiv preprint arXiv:1808.00004* (2018).

[7] Kuchaiev, Oleksii, and Boris Ginsburg. "Training deep autoencoders for collaborative filtering." *arXiv preprint arXiv:1708.01715* (2017).

[8] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[9] Neo4j Graph Database
https://neo4j.com

[10] DB-Engines Ranking
https://en.wikipedia.org/wiki/DB-Engines_ranking

[11] Cypher Query Language
https://en.wikipedia.org/wiki/Cypher_Query_Language

[12] Difference between content-based-filtering and collaborative-filtering
https://www.quora.com/What-is-the-difference-between-content-based-filtering-and-collaborative-filtering

[13] Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, *25*(2-3), 259-284.

[14] Review prediction with Neo4j and TensorFlow
https://medium.com/octavian-ai/review-prediction-with-neo4j-and-tensorflow-1cd33996632a