MOVIE RECOMMENDATION SYSTEM

Divija Palleti - 33612743[1]
Megha Shishodia - 33629102[1]
Tanaya Atmaram Kambli - 33534024 [1]

ABSTRACT

Movie recommendation systems are instruments that can be leveraged by media streaming platforms to suggest movies to a user depending on their previously noted interests, behaviours, search history, etc. From the perspective of the organizations owning these platforms, it is essential to ensure that their customers are satisfied with the product in order to avoid losing subscribers and in turn, profits. In view of that, it is favourable to increase customer engagement and one way to do that is by recommending relevant content. Of course, there are higher chances that customers return to a platform if they see content that interests them and, a movie recommendation system achieves exactly that! On the other hand, this is also desirable by end users as they can select a movie in a short time without going through the trouble of browsing through millions of movies generally hosted on such platforms, thus, enhancing user experience. In this project, we aim to build a movie recommender system with the help of Spark GraphX and KNN algorithm.

1 INTRODUCTION

Recommendation systems have proven highly effective for commercial applications to predict what their customers would like and provide them with the most relevant information. In today's age, this has been useful in alleviating the information overload on internet users. The objective of an entity recommendation system over big data is to develop a system that is scalable, efficient and offers the best answers for a wide variety of queries due to the vast number of data that these entity recommendation systems analyze.

In this project, we focus on the movie recommendation system that can be used seamlessly by streaming media platforms to improve user experience. To do this, we will be making use of the MovieLens dataset that consists of over 24 million ratings given by over 250,000 users across more than 40,000 movies. The dataset will be stored on a database instance. Based on the ratings given by each user (identified by userId) for the movies (identified by movieId, we can determine the types of movies that are generally preferred by the user by focusing on the specifics (example, actors, genre, etc) of movies highly rated by the user. Then, we will use content-based filtering. Further, Spark GraphX will be used to build knowledge graphs based on these relationships. Lastly, we can run an KNN algorithm to list the top recommendations to the user.

Existing systems have approached the problem in different manner, example, using Neo4j graphs and PageRank in [2] or using Spark with Machine Learning models [3]. We plan to implement the system using GraphX knowledge graphs and KNN to efficiently provide recommendations to the users.

	Content Filtering	Collaborative Filtering
Memory Based	• TF-IDF & cosine similarity	• KNN & cosine similarity
Model Based	• Neural networks • Bayesian classifiers	• Latent factor model

Figure 1. Algorithms for Recommender System

2 BACKGROUND

The two types of approaches that recommender systems often use are collaborative filtering and content-based filtering. Some of these systems even adopt a hybrid strategy that combines these two strategies. In collaborative filtering, recommendations are based on the collaboration of different users whose choices are similar. It is not based on the features of users/movies. But in the case of collaborative filtering, recommendations are based on users' behavior or their features. Hybrid methods [4], as the name suggests combine both the above approaches. It starts with content-based filtering and shifts to collaborative filtering.

2.1 Basic Algorithms for Recommender Systems

This can be classified into two main categories:

- Memory-based algorithms

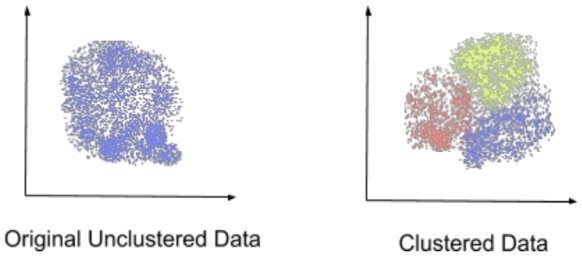

Figure 2. K Means Clustering

- Model-based algorithms

Memory-based algorithms use the previously seen data to associate the users and their choices where as model-based approaches use models i.e, ML models to predict the recommendations. Collaborative filtering is suitable because our dataset contains ratings provided by users and this will help us identify similarity in users. It can predict the movie rating without knowing the attributes of the movies and users

Clustering methods can detect organization in data that initially appears to be random. It is a form of Unsupervised learning. These techniques rely on discovering similarity across a feature space/latent space in the dataset. One or more qualities may be present in the feature space. People who enjoy comedy movies, for instance, can be a feature, and everyone who meets this requirement may be a member of the same cluster. The dimensionality of a feature space is determined by the quantity of features in the feature space.

K-means clustering is a common clustering algorithm. The things in this are separated into k clusters. Each element is initially arranged in random groups. The centroid is then determined for each cluster. Then, the distance between each element and the centriod is calculated to determine to which cluster each of these elements belong to. Each iteration changed the centriod to attain the minimum error thus giving the clusters[5].

NoSQL databases differ from relational databases in how they store data, i.e, they store the data in a non-tabular format. There are many different formats in which data is stored in the NoSQL. The four major categories are : columnar, documental, key-value, and graph databases. This division is mainly done on how the Create, Read, Update, and Delete operations are performed on the data.

Neo4j is a graph database that uses NoSQL technology and is based on trees. The nodes and the connections between them, according to Justin Miller [6], form the foundation of the concept of a graph. According to the author, nodes, relationships, and arista all have qualities that allow people

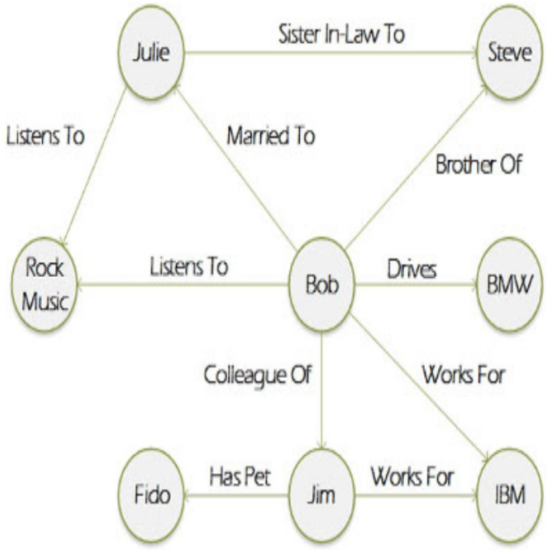

Figure 3. Graph Database

or information systems to store data using a pattern known as keyvalue. In light of this, Neo4j uses a structure in which the nodes are represented as vertices and the connections or arista are represented as edges. For effective data retrieval, this organization uses complex algorithms and mathematical calculations. Additionally, it maintains the usage of a dynamic structure, the assignment of values just as needed, and a more exact design in line with the business principles.

One of the main characteristics of Neo4j is that it is highly available. The master-slave cluster divides Neo4j into two parts: the database and the cluster management component. In the traditional relational database paradigm, data is inserted into and changed in a collection of tables whose schema is already defined. It is linked together through relationships, indexed, and recognized by main and foreign keys using relational algebra and the logical underpinning of mathematics. Neo4j, on the other hand, uses a non-structured repository, which has the benefit of not requiring a prior database design or a rigid and challenging manipulation of a set of nodes. This is well illustrated in the paper "Facebook Graph Search with Neo4j" [7] (a Facebook-based social network search). Several graph processing systems have been created in the previous ten years. Based on the workloads they handle, these are often separated into graph analytics systems (like Giraph) and graph database systems (like Neo4j). GraphX is developed using Scala and run on top of Apache Spark.

Spark stores dataset partitions in memory rather than on the local disk for later usage, similar to how Hadoop does. Resilient distributed datasets, Spark's fault-tolerant in-memory abstraction, is a key component (RDD). A graph framework

called GraphX [8] enhances Spark abstractions to carry out graph computations. Vertex-cut partitioning is utilized, just like in GraphLab. Each Spark job is part of an iteration. What data chunks should be stored for later usage can be decided by a developer. However, because cached data is used as immutable RDDs, it cannot be altered.

3 APPROACH

Following is an illustration of our pipeline:

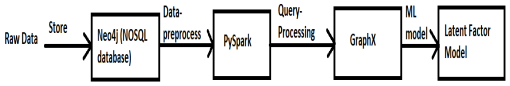

Figure 4. Methodology Pipeline

1. *Dataset* - In this project, we have worked with the "Movielens 1M Dataset" which contains 1 million ratings from 6000 users on 4000 movies. This dataset was released in 2/2003.

   The dataset consists of three files which capture the ratings given by users to the movies - movies.csv, users.csv, ratings.csv. Following is a visual representation of a part of the data contained in these files.

   - **movies.csv**

Movie_Id	Movie_Title	Genres
1	Toy Story (1995)	Animation | Children's | Comedy
2	Jumanji (1995)	Adventure |Children's |Fantasy

   - **users.csv**

User_Id	Gender	Age	Occup - ation	Zip_ Code
1	F	1	10	48067
2	M	56	16	70072

   - **ratings.csv**

User_Id	Movie_Id	Rating	Time Stamp
1	1193	5	978300760
1	661	3	978302109

2. *Data Storage: Neo4j* -

We have used Neo4j, which is an open-source, NoSQL, native graph database, for storing our data.

The reason we went ahead with a Graph Database is as follows:

(a) Often times, we are more concerned about the relationships between the data. Connections existing between the data can be more insightful in some scenarios as compared to the data itself. Graph databases help us achieve exactly that! We can perform complex and intricate data analysis with graph databases and they aid in bringing out significant associations between the data points.

(b) Hence graph databases are essential in storing large amount of connected data like our movie-user-rating dataset. They make visualization pretty easy.

(c) Secondly, in our case particularly, we are also dealing with sparse data. This is since not all users rate all the movies. In such a case, if we were to choose a relational database, there would be multiple null values which in turn would increase memory usage unnecessarily. Thus, we went ahead with a Graph Database as there are edges only for non-null values which in turn solves the memory crisis.

The reason we chose Neo4j as our graph database is due to the following advantages:

(a) **Highly Available** - Neo4j is highly suitable to use for graphs that contain data used in production scenarios.

(b) **High Performance** - Neo4j if highly efficient when it comes to performance. Following is a table of execution time comparison between Neo4j vs MySQL. As we can see, for a simple friends of friends query, Neo4j is 60 % faster than MySQL. For the depth four query, Neo4j is 1,135 times faster.

Depth	Execution Time - MySQL	Execution Time - Neo4j
2	0.016	0.010
3	30.267	0.168
4	1,543.505	1.359
5	Not Finished in 1 Hour	2.132

Figure 5. Neo4j vs MySQL comparison table

(c) Neo4j also provides other database characteristics, like ACID transaction compliance, cluster support, and runtime failover.

3. *Query Processing: GraphX* -

   We have used GraphX for query processing. It is a graph processing layer built on top of Spark.One question may arise - Why do we use GraphX and not Neo4j for Query Processing?

   This is because Neo4j is a graph database and although it provides query language support, it is not as efficient as compared to GRaphX whixh is an in-memory graph processing system.

4. *ML model: KNN* -

   KNN is a memory based recommendation system with collaborative filtering. In our project, we use collaborative filtering since:

   (a) Our movielens dataset contains ratings given by users and we can use this information to find similarity between users.

   (b) Collaborative filtering can also predict the movie rating without knowing the attributes of the movies and users

4 EXPERIMENT

1. *Graph Visualization in Neo4j*:

   Database: Neo4j

   Nodes: There are two types of nodes : Users and Movies

   Relationships: A directed edge from User to Movie if the user rates it.

   Properties: Nodes: User Node: User_Id, Age, Gender Movie Node: Movie_Id, Genres, Movie_Title

   Relationships: Rating

   Following is the pictorial representation of the nodes and the complete graph.

Figure 6. Movie Nodes

Figure 7. User Nodes

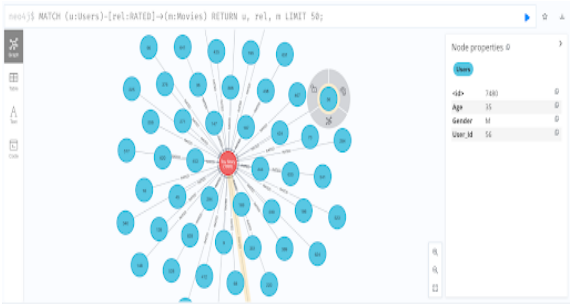

Figure 8. Complete Graph with Edges

2. *Neo4j Performance*:

   (a) We can see that Neo4j takes very less time even if the data size is huge. This would have taken even more time in other databases as a lot of joins of tables need to be done to get the related data (my SQL).

   (b) We can also observe that the only overhead is loading the data, but as it is just done once per data this shouldn't be a problem.

   (c) As a part of future work, we can explore how to optimize the data loading.

Query	X (2000)	10X(20000)	100X (200000)
Loading data	4845 ms	37751 ms	386964 ms.
MATCH (m:Movie)<-[r:RATED]-(u:User) RETURN m,r,u	5 ms.	10 ms	103 ms

Figure 9. Neo4j Performance

3. *Data Preprocessing using PySpark*:

   Preprocessing performed on the data is as follows:

(a) Dropping unwanted columns (the TimeStamp column in ratings.csv, the ZipCode column in users.csv)

(b) Typecasting column data types (Example: Ratings column)

(c) Merging the three tables i.e., movies, ratings, users

Final table looks as follows:

User_Id	Movie_Id	Movie_Title	Genres	Rating	Gender	Age	Occupation
1	3105	Awakenings (1990)	Drama	5	F	1	10
1	2799	Big (1988)	Comedy|Fantasy	4	F	1	10

Figure 10. Final data

4. *Query Processing using GraphX:*

The data in GraphX is stored as Graphframes which is alayer over Dataframes. It aims to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames.

GraphFrame is created from the combination of Vertex DataFrame and Edge DataFrame Also, since GraphFrames represent graphs as pairs of vertex and edge DataFrames, it is easy to make powerful queries directly on the vertex and edge DataFrames. Those DataFrames are made available as vertices and edges fields in the GraphFrame.

```
graph.vertices.show()
```

```
+----+-------+--------------------+--------+--------------------+------+---+----------+--------------------+
|  id|User_Id|         Movie_Title|Movie_Id|              Genres|Gender|Age|Occupation|count_movieRatedTotal|
+----+-------+--------------------+--------+--------------------+------+---+----------+--------------------+
| 296|    296|Snow White and th...|     594|Animation|Childre...|     M| 50|         5|                 763|
| 296|    296|       Psycho (1960)|    1219|      Horror|Thriller|     M| 50|         5|                1263|
| 829|    829|Snow White and th...|     594|Animation|Childre...|     M|  1|        19|                 763|
| 829|    829|In the Heat of th...|    1950|       Drama|Mystery|     M|  1|        19|                 348|
| 829|    829|       Psycho (1960)|    1219|      Horror|Thriller|     M|  1|        19|                1263|
| 829|    829|  Annie Hall (1977)|    1230|      Comedy|Romance|     M|  1|        19|                1334|
|1436|   1436|  Elizabeth (1998)|    2336|               Drama|     F| 50|         6|                 930|
|2069|   2069|       Psycho (1960)|    1219|      Horror|Thriller|     M| 56|        15|                1263|
```

Figure 11. Query Processing using GraphX

5. *Insights from Data using GraphX:*

(a) Most popular movies, where popularity is defined by the inDegree of a movie (Figure 12):
Execution time : 0.97910976401ms

(b) Least popular movies, where popularity is defined by the inDegree of a movie (Figure 13):
Execution time : 0.97910976401ms

(c) Top Genres for young adults (Figure 14):
Execution time : 0.476099016561ms

6. *Recommendation using KNN:*

```
+-----------------------------------------------+--------+
|id                                             |inDegree|
+-----------------------------------------------+--------+
|American Beauty (1999)                         |464     |
|Star Wars: Episode V - The Empire Strikes Back (1980)|445     |
|Star Wars: Episode IV - A New Hope (1977)      |424     |
|Star Wars: Episode VI - Return of the Jedi (1983)|421     |
|Terminator 2: Judgment Day (1991)              |395     |
+-----------------------------------------------+--------+
```

Figure 12. Most Popular Movies

```
+---------------------------------+--------+
|id                               |inDegree|
+---------------------------------+--------+
|Bitter Sugar (Azucar Amargo) (1996)|1       |
|X: The Unknown (1956)            |1       |
|In the Bleak Midwinter (1995)    |1       |
|American Dream (1990)            |1       |
|Secret Agent (1936)              |1       |
+---------------------------------+--------+
```

Figure 13. Least Popular Movies

The Neo4j Graph Data Science (GDS) library contains several algorithms for machine learning on graphs.We used the K-Nearest Neighbors algorithm (kNN) to identify similar customers and base our movie recommendations on that.

In order to be able to leverage topological information about the graph in kNN, we will first create node embeddings using FastRP. Node embeddings allow for aggressive dimensionality reduction while preserving most distance information. The FastRP algorithm operates on graphs, in which case we care about preserving similarity between nodes and their neighbors.

This means that two nodes with similar neighborhoods should be assigned similar embedding vectors. These embeddings will then be the input to the kNN algorithm.

We compute the similarity matrix using KNN. Further, we used these similarities to recommend movies to users making sure to exclude movies they have already watched.

Following is a part of the similarity matrix for users (Figure 15):

We take a user as input and provide recommendation using KNN and similarity matrix (Figure 16)

```
+--------------------+-----------+
|Genres              |Watch_Count|
+--------------------+-----------+
|Drama               |502        |
|Comedy              |399        |
|Horror              |135        |
|Comedy|Romance      |122        |
|Comedy|Drama        |121        |
|Drama|Romance       |111        |
|Thriller            |80         |
|Documentary         |54         |
|Action              |53         |
|Drama|Thriller      |52         |
|Children's|Comedy   |44         |
|Action|Thriller     |43         |
|Drama|War           |35         |
|Comedy|Drama|Romance|31         |
|Animation|Children's|31         |
|Crime|Drama         |30         |
|Action|Drama        |30         |
|Horror|Thriller     |30         |
|Comedy|Horror       |30         |
|Action|Sci-Fi       |27         |
+--------------------+-----------+
only showing top 20 rows
```

Figure 14. Top Genres for Young adults

Person1	Person2	Similarity
148	13	0.997060998769907
13	148	0.997060998769907
115	113	0.989005699676789
113	115	0.989005699676789
84	139	0.980676788960934

Figure 15. Similarity Matrix for Users

5 Conclusion

Building the data pipeline for the movie recommendation system involved assessing various technologies for data persistence, data preprocessing, data querying, and machine learning. On comparing the available options, we found that for data persistence, Neo4j works best for our use case of sparse data, which can be stored as a graph. We used PySpark for data preprocessing because it optimizes for memory and time by running a distributed system. We used GraphX for graph processing because it provides an in-memory distributed graph processing system like Spark but for graphs. KNN was employed for computing the similarity matrix to output the list of recommendations given to a user.

The future work entails using other ML models for recommendations, like the latent factor model, wherein the objective is to obtain a vector for each user and movie, which represents their latent features. The latent Factor Model works very well with sparse data and will suit our objective and data very well.

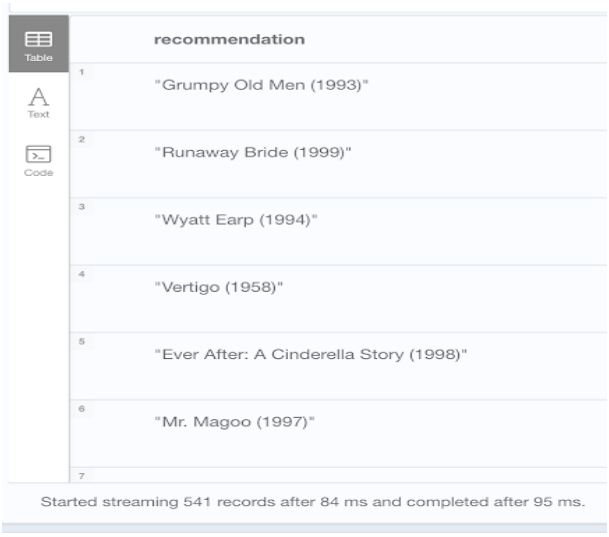

Figure 16. Recommendation provide to a user

6 Team Contribution

All members of the team contributed equally to the execution of this project.

References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

[2] Anders Brams. (2020). Movie Recommendations powered by Knowledge Graphs and Neo4j https://towardsdatascience.com/movie-recommendations-powered-by-knowledge-graphs-and-neo4j-33603a212ad0

[3] Jose A Dianes. (2015). Building a Movie Recommendation Service with Apache Spark Flask - Part 1 https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw

[4] Hussien, M.M.; Helmy, K.M.; Hasan, I.M. Recommender Systems Challenges and Solutions Survey. In Proceedings of the 2019 International Conference on Innovative Trends in Computer Engineering (ITCE), Aswan, Egypt, 2–4 February 2019; pp. 149–155

[5] Wikipedia article. Internet. “K means Clustering” en.wikipedia.org/wiki/K means_clustering

[6] M. Justin, “Graph Database Applications and Concepts with Neo4j,” in Proceedings of the Southern Association for Information Systems Conference, Atlanta, USA, 2013.

[7] J. Zhang, Z. Li and S. Liu, "Facebook Graph Search with Neo4j," Department of Computer Science, Georgia State University, Atlanta, USA, 2013.

[8] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In Proc. 11th USENIX Symp. on Operating System Design and Implementation, pages 599–613, 2014.

[9] FastRP on Neo4j : https://neo4j.com/docs/graph-data-science/current/end-to-end-examples/fastrp-knn-example/

[10] Graph Data Science Lib for Neo4j : https://neo4j.com/docs/graph-data-science/current/

[11] Spark GraphX : http://amplab.github.io/graphx/

[12] ML on Spark GraphX : https://livebook.manning.com/book/spark-graphx-in-action/chapter-7/

[13] Spark GraphX Official Documentation : https://spark.apache.org/graphx/