ECE 356 Project - Movie Dataset

Table of Contents

Introduction

For this report, we have decided to move forward with the dataset for movies. The dataset is a complex one, holding multivariate values, movies from several different CSV with different columns and data. Our plan was to use Python for the CLI and SQL for the table manipulation and file loading. Now moving to the client application and our thought processes involving our decisions.

Client Application

Before loading the client application, we have to parse the credits data, given its unconventional format in the kaggle CSV. This is not possible to do in SQL, and so we resorted to a Python Script. The script itself takes around 5-7 minutes to load on a fast local machine. On the eceubuntu servers, it is highly dependent on the server load at the time. One of our group members had some trouble with mysql.connector, but it worked for everyone else. For the client application, we chose Python, because of its simplicity and our familiarity with it. In the CLI, you can add a User or login. Once logged in, and depending on permissions, you can create, read, update, and delete the user's reviews and movies. Adding and deleting movies is restricted to admin users.

Entity-Relationship Design

Movies:

It felt appropriate to add the relevant id's into the table Movies, as a way to reference the movies to their respective weak entity sets as well as CSV that require a certain id which may not be available otherwise. We decided to keep important information a movie may have which a user may query for, such as title, overview, status, etc. These important singular pieces of data were found to be more relevant to users as well as defining features for describing a 'Movie'.

Metadata:

For the Metadata table, it is a weak entity set dependent on the Movies table. The alternative was to add everything into the Movies table, but that would be counter intuitive to organizing the data appropriately with the according relationships. The Metadata attributes are independent values that may give further insight to the 'Movie' if they wish to.

Movie Votes and Movie Ratings:

Initially, we were planning on merging the two into one table as they all displayed how users perceive a certain movie. We opted to separate them into two different tables as the attributes ‘popularity’, ‘vote_average’, and ‘vote_count’ as they are singular values based on a multitude of voters. In contrast, the ‘rating’ attribute is the ratings given to a specific movie by a specific user with users being able to vote for multiple movies yet not the same one. Thus, the two tables were separated with a weak entity relationship with the table, Movies.

Movie Genre:

The table is a weak entity set based off of Movies. As there are multiple genres for movies, we decided to add a discriminator called, ‘genre_id’. In the csv, each genre is attributed with an id with no duplicates for the same movie, thus being a great discriminator.

Movie Production Company and Country:

The group initially decided to combine the two tables together, but due to the fact the two multivariate values may have separate amounts for each and wanting to avoid losing data we opted to separate them into two different entity sets.

Crew, Cast & People:

Within the ER Model, we have three entity sets. Crew and Cast are in a weak entity set relationship with movies while People is a common specialization for Crew and Cast members where the two entity sets are disjoint. At first, we wanted to keep the Crew and Cast members separated but, as there were many common attributes and it felt organized to have an entity set with people involved, the People entity set was made.

HSX and Box Office Info:

The two entity sets have the same cardinality and are both weak entity sets to Movies. The only difference between the two is that the primary key for the Hollywood Stock Exchange is the ‘title’ attribute as the identifier provided in the CSV could not be related to the other CSVs, more specifically the ‘movies_metadata.csv’

Average Ticket Prices:

The entity set does not have any direct relationship with the Movies entity set or any other entity sets. The data it contains is based on the average movie ticket price (USD) for a specified year starting from 1910 and going till 2019.

Relational Schema

For our relational schema we created a table for every entity set in the ER model. In our ER model we had the Movies entity as our main entity and the rest were mostly weak entities based off of Movies. Due to this reason, we decided to make the primary key for most tables to be the movie_id. The movie_id originated from the Movies table and was common in many of the csv files in the data set. Besides the movie_id we also used the imdb_id to relate many tables together. The imdb_id and movie_id are unique identifiers in all the tables that allow for us to be able to reference the same movie across multiple tables. In order to create the relationships between the Movies table and all the other tables, we had to create a foreign key from the primary key of the table that references the movie_id or imdb_id of the Movies table. For the tables MovieGenre, MovieProductionCompanies and MovieProductionCountries, we made the primary key to be a combination of the two columns in the table. This decision was based on the fact that we needed to individually identify the movie and genre together since there can be multiple genres to a movie. The same follows for the combination of movie_id and company and the combination of movie_id and country. For the CastMembers and CrewMembers tables, we also put a foreign key to the People table since these two tables are a specialization of the People table. Having this foreign key relates the People information to the cast and crew information while the foreign key referencing movie_id in Movies creates the relationship between the cast and crew and the movie.

In order to populate the tables, we first loaded the csv files into temporary tables and cleaned up the columns and rows based on our needs. For columns which we did not need in any of our tables, we did not load into the temporary tables. For the columns that had different names, we renamed them as they were loaded into the temporary tables to make all the column names consistent and understandable. The main columns which we had to rename were id columns which were mostly either movie_id or imdb_id. We used the Nullif function to make sure that the values that were inserted into the rows were the correct. For the rows that had the constraint Not Null, we had to ensure that the rows we wanted to insert had the column present and if it was not present we would not insert that row since it would be invalid. We also used a lot of type casting, for example on strings that were numbers or formatting dates correctly.

After loading the csv files into the temporary tables, we populated the main tables by inserting the appropriate columns. There were many tables that had information spread across multiple temporary files so we had to use joins on the temporary tables to get all the columns that the main table required. This is where having consistent names and ids was useful. In order to join the temporary tables, we needed the data to have columns in common. Most tables had either imdb_id or movie_id so we were able to join on those columns and insert the joined tables into the main table.

Data-Mining Investigation

For our data mining investigation of the Movies dataset, we decided to analyze the movie ratings by users in order to answer the question: ‘How likely is a user to like a movie based on their previous ratings of other movies?’.

In order to conduct this analysis, we decided to implement the technique of Association Discovery, specifically using the Apriori algorithm. This technique allows us to identify relationships and patterns which can be used to group movies together based on the criteria of ratings given by a user. So we can see that if a user liked some set of movies, then how likely are they to like any other movie.

The code below implements the Apriori algorithm for the Association Discovery. It connects to a MySQL database, retrieves movie rating data, namely the three columns, ‘UserId’, ‘movie_id’, and ‘rating’.The ratings are then converted to binary values which are then used to create a binary user-item matrix. Then, applying the Apriori algorithm we generate the association rules and we have an efficient data model.

```
# How likely is a user to like a movie based on their previous ratings of
other movies?
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import pymysql

hostname = 'riku.shoshin.uwaterloo.ca'
username = 'username'
password = 'password'
database = 'db356_team31'

# Connect to MySQL database
connection = pymysql.connect(
          host=hostname, user=username, password=password,
database=database)

# Load the data from the MySQL database into a DataFrame
sql_query = "SELECT userId, movie_id, rating FROM Ratings"
data = pd.read_sql(sql_query, connection)
connection.close()

# Convert ratings to binary values (liked or not liked)
```

```
data["liked"] = data["rating"] >= 4.0

# Create a binary user-item matrix using pivot_table
basket = data.pivot_table(
    index="userId", columns="movie_id", values="liked", aggfunc="max",
fill_value=False
)

# Apply Apriori algorithm
frequent_itemsets = apriori(basket, min_support=0.05, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7)

print(rules[["antecedents", "consequents", "support", "confidence"]])
```

Figure 1: Data Model for Association Discovery

The output of the code provides a list of association rules, showing antecedent and consequent movie combinations, support, and confidence values. For example, one rule states that if a user has rated the movie “Dancer in the Dark” (id: 16) positively (liked), there's a 80.33% confidence that they will also like the movie “Terminator 3: Rise of the Machines” (296). In comparison, another rule states that if a user has rated the movie “Dancer in the Dark” (id: 16) positively (liked), there's a 70.49% confidence that they will also like the movie “Solaris” (593).

We can determine the validity of our model by observing the created association rules. These association rules include the support metrics which measure how frequently the antecedent and consequent items co-occur, and confidence metrics which measure the likelihood of the consequent item being rated positively when given the antecedent. We can also look at the real-world context to ascertain the validity of the model. Based on the two rules above, the similarity is that it is based on whether the user positively (liked) the movie “Dancer in the Dark”. The user is more likely to like the Terminator movie over Solaris. Based on knowledge of the three movies, I can ascertain some truth as Terminator 3 is a popular movie with a bit of a crime aspect similar to “Dancer in the Dark”.

Similar insights can be extracted from other rules, which will further assist us in understanding user preferences and predicting future likes based on past ratings.

	antecedents	consequents	support	confidence
0	(2355)	(1)	0.061103	0.759259
1	(3114)	(1)	0.090909	0.772152
2	(16)	(296)	0.073025	0.803279
3	(16)	(593)	0.064083	0.704918
4	(47)	(296)	0.166915	0.767123
...	...	...	...	...
38668	(7153, 1210, 260, 1198)	(5952, 4993, 2571, 1196)	0.053651	0.705882
38669	(7153, 2571, 1196, 1198)	(5952, 4993, 1210, 260)	0.053651	0.837209
38670	(7153, 1210, 2571, 1196)	(5952, 4993, 260, 1198)	0.053651	0.720000
38671	(7153, 1210, 2571, 1198)	(5952, 4993, 1196, 260)	0.053651	0.923077
38672	(7153, 1210, 1196, 1198)	(5952, 4993, 2571, 260)	0.053651	0.720000

[38673 rows x 4 columns]

Figure 2: Data model output

Conclusion

In conclusion, through our project, we have addressed the complexities of the movie dataset. We designed our entity relationship model as well as schema with efficiency while ensuring effective organization and representation of the dataset. The client application focused on information that users would normally have access to in similar applications while keeping usability in mind. Lastly, the data mining focused on predicting user preferences utilizing the Apriori algorithm to generate useful association rules. This project has enhanced our skills in database design and data mining. It has offered valuable insights into database design, its implementation as well as user behavior. This project has significantly enhanced our skills in database design and data mining, providing insights applicable to future applications requiring database knowledge.