Sarah Martin

Professor Lance Levenson

Northwestern University, School of Professional Studies

Foundations of Data Engineering

February 16, 2019

Project 1: Business Processes and Requirements Definition

for a Movie Recommender Engine

**Section 1: Management Overview and Recommendations**

The task at hand is to define requirements for the general technical system and related business processes for a new movie recommender engine based on the functions that need to be performed both pre and post launch of the new product. Initial ideas for technical solutions that could potentially fulfill these requirements are also sometimes suggested. However, additional specification of the product’s needs would be required to align on any final technical solutions.

To build this recommender engine, we would need to extract data from three sources, and build and/or buy the technical solutions to properly store and analyze this data. In addition, we would need to build the components that would allow users to interact with our product by submitting their own movie preferences, applying our model to generate predictions for which movies the user is expected to like upon the user requesting recommendations from our engine, and returning the results.

The next two sections go into more detail about the data that could be extracted from each of the data sources, our needs and initial ideas for storing the data, and our needs and initial suggestions of tools to use in analyzing the data, fitting models to predict movie preferences, and producing recommendations from our model based on user input and requests. The last section touches on our relative computing needs to support the above function and potential solutions for those computing needs.

**Section 2: Data and Database Software**

The movie recommender engine currently requires input data from three sources:

1. The Internet Movie Database (IMDb) owned by Amazon
2. MovieLens.org, a research site hosted by GroupLens Research, a group of undergraduate students, graduate students, staff, and visitors, engaged in social computing research at the University of Minnesota, and headed by faculty in the Department of Computer Science and Engineering.
3. Users of the recommender engine themselves

Section 2.1: IMDb Data

While IMDb does not provide an API for public queries, it offers seven daily updated downloadable compressed tab-separated UTF-8 encoded plain text files that contain much of the data found on IMDb via this link: <https://datasets.imdbws.com/>. This data is listed as available for personal and non-commercial uses only with their Non-Commercial Licensing compliance specifics available in here. Research will be required immediately to determine if we can publish modeling results based on the data, and if we ultimately intend to monetize our recommender engine product, we should first determine the viability of Commercial licensing with IMDb. If Commercial licensing is an option, technical and legal resources will be required to communicate and coordinate with IMDb to develop and maintain a functioning relationship ensuring continued use of their data.

IMDb lists three main options for accessing the information contained in these files: command-line interface tools, a Java-based GUI that allows searches and displays of the information provided, and a Python package, “IMDbPY”, that can process the files into a number of different SQL-based databases.

Our system will require loading the IMDb data into the analytics tooling we choose for Neural Network encoding. We will also need to extensively investigate, explore, manipulate, and potentially alter the structure or encoding of the data. In addition, ideally, the system would allow us to refer back to any previous version of the data at any time. As such we should probably store the data, and all or most previous versions of it, in house. Thus, an initial assessment suggests the IMDbPY package may be appropriate for accessing the data and feeding it into a SQL-based database and/or data warehouse.

Our system should also allow for regularly scheduled pulls of new versions of the files on IMDb and batch processing to load them into the database and/or data warehousing solutions we choose. The appropriate cadence of these data pulls will be highly dependent on the performance of the recommender results, both in terms of model accuracy and in terms of user affinity. As such, the precise cadence of IMDb data acquisition may change over time and should be reassessed periodically leveraging input from the modelers as well as business decision makers.

A technical resource or a team of technical resources should be put in charge of developing, executing, and monitoring data acquisition, basic pre-processing, and batch loading into our database and/or data warehouse systems. This resource or team must have open lines of communication with those coordinating any contract relationship with IMDb *and* with those tasked with developing, monitoring, and updating the neural network modeling in case of deviations in the source data.

Section 2.2: MovieLens.org Data

GroupLens offers downloads of 7 UTF-8 encoded csv structured zipped dataset files sampling various MovieLens.org data. Some of these datasets are static and others seems to be periodically, though infrequently, updated. Each dataset comes with a README file outlining appropriate use cases of the file. Specifically, “the user may not use this information for any commercial or revenue-bearing purposes without first obtaining permission from a faculty member of the GroupLens Research Project at the University of Minnesota.”[cite] As such, if we ultimately intend to monetize our recommender engine product, research will be required immediately to assess the viability of continued use of the MovieLens.org data. If viable, resources will again be required to communicate and coordinate with GroupLens to develop and maintain a functioning relationship ensuring continued use of their data.

Similar to the requirements necessitated by incorporation of the IMDb data, our data-flow system must be able to unpack the MovieLens.org files, load them into our own databases and/or data warehouses, and load them into any and all analytics tooling we choose for data exploration, Neural Network encoding and prediction generation.

Again, we likely want our system to allow for periodic pulls of new versions of the some of the files provided by GroupLens and batch processing to load them into the database and/or data warehousing solutions we choose. However, in this case, we can likely perform these data acquisition tasks at more infrequent intervals as compared with the IMDb data acquisition schedule. A technical resource or a team of technical resources should be in charge of developing, executing, and monitoring MovieLens.org data acquisition, basic pre-processing, and batch loading into our database and/or data warehouse systems. This resource or team could easily be the same resource/team handling the IMDb data acquisition and also, should probably have direct lines of communication with technical resources at GroupLens. Given GroupLens status as a small research organization versus IMDb as a subsidiary to a company founded by someone understood to be the richest man in the world, it is likely the relationship with GroupLens will be far less demanding to cultivate and monitor than that with IMDb.

Section 2.3: User Input Data

The last data input to create the personalized movie recommender engine is input from users. Any API can serve this function. However, movie recommender engines have broad potential appeal among anyone who watches movies. Since most people watch movies, and most have developed strong preferences for modern well-designed visually enticing websites, depending on our desired use case of the recommender engine, it is likely we would develop a website to handle user requests, as well a recommendation output.

Regardless of the front-end that users will interact with, ultimately, information should be sent to a server resource that we manage. Until we more fully develop the business use case for the recommender engine, we should assume that inputs could arrive in the form of single movie preferences or a bulk set of movie preferences. We probably would not want to require users to re-input all their movie preferences every time a recommendation is requested, but even if we did, previous user movie preferences would still prove useful in model validation and periodic re-training. As such, we would need to store previous movie preference inputs from each user in a database and/or data warehouse.

A technical resource of team of resources will be in charge of monitoring the functionality of whatever front-end we develop for the user input API. In addition, a (potentially) different resource or set of resources should be responsible for monitoring the data arriving at our server and its successful delivery into our databases and/or data warehouses in the proper format. Again, this resource should have direct lines of communication with the owners of the model itself. A well-designed API that automatically checks input data and maybe produces automated error messages back to the user would also be helpful in relieving the burden on our technical resources and the rest of our back-end system.

Section 2.4: Storage Systems

In terms of data storage systems, a relational (SQL) data model would probably be best as most of our input data would already be structured as relational-friendly tables. The precise relational data model to choose depends on existing infrastructure, technical expertise required and available, and cost. One option that would keep costs low would be SQLite, a free serverless (cloud-based) relational database solution, assuming it can handle our load appropriately. Another option is MySQL which uses MyRocks to offer a log-structured (LSM) solution in addition to the relational default B-tree solution. Since our volume of writes is likely to be higher than our volume of reads, MySQL would give us the freedom to test, potentially adopt, or potentially drop an LSM indexing system while still working within the confines of a relational database and without having to change providers if we decide to switch from one indexing system to another. As both of these relational systems are popular, many engineers are already familiar with them and it will be easier to locate and leverage any resources that would help foster maintainability.

A potential exception to the single relational data model would be the user input data. While storing user preference alongside preferences from our other data sources in a relational database, additionally storing user data in a document data model with each document representing an individual user would enable optimizations from locality when producing recommendations for a user. However, we would need to conduct a cost-benefit analysis once more architecture decisions have been made to determine if the added benefit of locality is worth costs including any required licensing as well as resource costs for monitoring and maintenance. If we chose to supplement the relational data model with document-based storage, we may also choose to store the parameters of all previous model formulas in the document storage system as well, in the rarer cases reversions or investigations are required.

Regardless of the systems used, a database administrator or team of database administrators will need to keep all database software updated, track issues, instantiate automated processing and data checks, ensure that any changes to the structure of incoming data can be handled appropriately, and continually monitor read/write volume and performance to ensure database tuning parameters are set properly.

**Section 3: Analytics and Modeling Software**

Normally, data warehousing in addition to data storage can provide valuable optimizations for an analytics product. However, in this case, further evaluation is needed. Much of the analytics necessary to *produce* the recommender engine (data quality investigations, exploratory analysis, feature engineering, model fitting) would require infrequently querying large portions of the sourced data, an ideal use case for data warehousing. In addition, the IMDb data in particularly seems to already be in a star-schema friendly structure, lending itself to data warehousing’s optimization of star schema structures.

However, the most frequently performed analytics for the recommender engine post-launch would involve model predictions from a pre-fitted model, requiring only the pre-fitted model parameters, and a single user’s input data to produce the model output. Since standard data warehousing solutions are not cheap, this use case would not warrant data warehousing on its own. The data being queried for the recommender engine would have no place in any consistent dashboard reporting for frequently used BI tools, a use case that data warehousing is built to support. The only case for consistent BI reporting post-launch would be on model performance and maybe load volume to assess growing interest in the new produce.

As such, a more formal assessment of the costs and benefits of data warehousing solutions as we learn more specifics about the load and expected future load of incoming data, the size of queries for analytics purposes, their frequency, our database read performance, and the impact that slow database query performance has on the progress of the recommender engine development, launch, and adoption should be conducted before deciding to purchase a data warehousing solution.

To execute the more complicated aspect of the analysis, including model fitting and prediction generation, more advanced analytical tooling would be necessary. Python may be a good solution for this as it is free, well known in both engineering and analytical circles, and may appear in other areas of the input-output data pipeline. To leverage Python, or any scripting language that has support for neural network modeling and predictions, whenever a user submits a recommendation request, the back-end server infrastructure would want to process any new movie preferences from the incoming request, consolidate that with the user’s existing encoded movie preferences (sourced from the chosen database or data warehouse), and run a model prediction Python script on leveraging the fitted model parameters (also from the chosen database or warehouse). There are also a number of options for Python development environments that can be leveraged for the model fitting portions of product development and maintenance.

Regardless of data warehouses used or what scripting language we leverage to fit our models, a resource, or more appropriately, a *small* team of resources, should be in charge of periodically monitoring recent incoming data for statistical consistency and conducting frequent model validation and tuning to ensure our recommendation engines continually maintains or increases its efficacy. In addition, this team should have direct lines of communication to those in charge of business decisions surrounding the recommender engine in case of model failure or required updates.

**Section 4: Computing and Communications Systems**

The computing infrastructure required for this project will have to support data storage (and maybe warehousing), model development computation, web hosting (most likely), ETL process flows, and production model predictions.

With over 5 million movie titles on IMDb, though only 600,000 to 700,000 of which are movies or TV movies, plus upwards of 20 million movie ratings applied to 27,000 movies in a single dataset from MovieLens, exploration of the data and neural network modeling in particular will take a great deal of computing power. Neural networks are particularly slow models to fit to begin with.

Storing the data for this product and fitting the models are such large datasets will no doubt require some sort of distributed architecture and/or cloud computing. Cloud computing may be particularly valuable to allow for elasticity. This would reduce costs when not fitting new neural network models whereas a distributed system, but no cloud architecture will require more c… In addition, as this is a new product for us, load estimation will not be exact and load may be highly unpredictable upon product launch. An elastic cloud computing infrastructure will allow us to respond to various load more quickly and efficiently.

Requesting additional information on the resources required for our model fitting and estimating load (and the growth of load) in request per minute from users, as well as the ratio of reads to writes of a production system will be key in determining the precise specifications of any cloud computing or distributed system. In addition, frequently monitoring throughput will allow us to prepare for growth in the product and anticipate and remedy scaling challenges before they impact the system.

To begin, our response time for producing recommendations upon user request in a production system should be less than 1 second for p99 (the 99th percentile). Reporting on past response times at various percentiles and analyzing the growth in user requests will help determine further appropriate performance metrics for response time.

Works Cited

A works cited page beginning on a separate page at the end of the paper. Ensure that all resource materials as properly cited, with APA or Chicago style references.

*Designing Data-Intensive Applications:*

Chapter 1: Foundations of Data Systems (pages 3–25),

Chapter 2: Data Models and Query Languages (pages 27–67)

Chapter 3: Storage and Retrieval (pages 69–107).

<https://reqexperts.com/resources/requirements-articles/articles-what-is-the-difference/>

<https://grouplens.org/>

[https://help.imdb.com/article/imdb/general-information/can-i-use-imdb-data-in-my-software/G5JTRESSHJBBHTGX?pf\_rd\_m=A2FGELUUNOQJNL&pf\_rd\_p=3aefe545-f8d3-4562-976a-e5eb47d1bb18&pf\_rd\_r=A4RHXCC91B96D7392C05&pf\_rd\_s=center-1&pf\_rd\_t=60601&pf\_rd\_i=interfaces&ref\_=fea\_mn\_lk1#](https://help.imdb.com/article/imdb/general-information/can-i-use-imdb-data-in-my-software/G5JTRESSHJBBHTGX?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=3aefe545-f8d3-4562-976a-e5eb47d1bb18&pf_rd_r=A4RHXCC91B96D7392C05&pf_rd_s=center-1&pf_rd_t=60601&pf_rd_i=interfaces&ref_=fea_mn_lk1)

<https://www.imdb.com/interfaces/>

<https://www.imdb.com/licensing/?ref_=helpms_ih_gi_license>

<https://grouplens.org/datasets/movielens/>

<http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html>

<https://blog.yugabyte.com/a-busy-developers-guide-to-database-storage-engines-the-basics/>