Professor Lance Levenson

Northwestern University

School of Professional Studies

Foundations of Data Engineering

Project 1: Business Processes and Requirements Definition

for a Movie Recommender Engine

by Sarah Martin

February 21, 2019

**Section 1: Management Overview and Recommendations**

The task at hand is to define general requirements for the technical system and related business processes for a new movie recommender engine. Ideas for technical solutions that might help fulfill these requirements are also occasionally suggested. However, additional clarity around the product’s needs would be required to align on any final technical solutions for adoption.

To build and support this recommender engine, data should be collected from three sources, and technical solutions to properly store and analyze this data would need to be built and/or bought. Components would also have to be built that would allow users to interact with our product by submitting their own movie preferences, requesting recommendations from our engine, having their data applied to our model to generate predictions, and receiving results.

The next section goes into more detail about the data that could be extracted from each of the data sources, and our needs and initial ideas for storing the data. The following section covers our needs and initial ideas for tools to use in analyzing the data, fitting models, and producing user-specific recommendations. The last section touches on our relative computing needs to support the above functions and potential solutions for those computing needs.

**Section 2: Data and Database Software**

The movie recommender engine currently requires input data from three sources: The Internet Movie Database (IMDb) owned by Amazon, MovieLens.org hosted by GroupLens Research, a group of students, staff, and visitors engaged in social computing research at the University of Minnesota, and 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/>. The data is listed as available for personal and non-commercial uses only per their Non-Commercial Licensing. Immediate research is required to determine if we can publish modeling results leveraging the data. If we eventually intend to monetize our recommender engine, we must first determine the viability of Commercial licensing with IMDb. If Commercial licensing is an option, technical and legal resources will be required to coordinate with IMDb to develop and maintain a relationship to ensure continued use of their data.

We will need to explore and manipulate the IMDb data at will, feed it into our analytics tooling of choice for Neural Network encoding, and, ideally, be able to refer back to previous versions at any time. As such the data should be stored in-house, alongside previous versions of it. Along with command-line interface tools and a Java-based GUI that allows searches and displays of the information provided, IMDb offers a Python package, “IMDbPY”, that can process and load the files into a number of different SQL-based databases. Thus, an initial assessment suggests the IMDbPY package may be well suited for accessing this data and feeding it into a SQL-based database.

Our system should manage regularly scheduled pulls of new versions of the IMDb files and batch processing to load them into data storage. The proper cadence of these data pulls will be dependent on its effects on the performance of the recommender results in terms of accuracy *and* user affinity. Modelers and business leaders should periodically reassess the cadence of IMDb data acquisition.

A technical resource[[1]](#footnote-1) should be put in charge of developing, executing, and monitoring data

acquisition, pre-processing, and batch loading into data storage. This resource must easily communicate with those managing an IMDb relationship *and* with the modelers in case of changes in the source data.

Section 2.2: MovieLens.org Data

GroupLens offers downloads of 7 UTF-8 encoded csv structured zipped files sampling the MovieLens data. Some files are static and others appear to be updated periodically, though infrequently. Each dataset comes with a README file outlining proper uses of the data. 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.”[cite] If we eventually intend to monetize our recommender engine, immediate research is required to assess the viability of use of the MovieLens.org data. If viable, resources will again be required to coordinate with GroupLens to develop and maintain a relationship to ensure continued use of their data.

Similar to the system requirements for use of the IMDb data, our data-flow system must be able to retrieve the MovieLens files and load the data into storage, repeating these tasks at periodic intervals for updated files. However, we can likely perform this data re-acquisition at more infrequent intervals compared with the IMDb data acquisition schedule. A technical resource will have to develop, execute, and monitor MovieLens data acquisition, any pre-processing, and batch loading into storage. This resource could easily be the same resource handling the IMDb data acquisition and also, should have direct lines of communication with technical resources at GroupLens. Since it is a small research organization, cultivating and maintaining the relationship with GroupLens will likely be far less demanding than doing so with IMDb, a subsidiary of one of the most powerful companies in the world.

Section 2.3: User Input Data

The last data source required for the recommender engine is input from users. Since movie recommender engines have broad potential appeal and most digital consumers have an affinity for web applications, depending on the desired use case for the engine, a website is likely required to handle user input, requests, and recommendation output, operating as the engine’s API. To make the product user friendly and to leverage previously submitted movie preferences in periodic model validation or re-training, we should update users’ movie preferences every time input is received from the website. Thus, website input should be delivered to a server that will load the data into storage.

A technical resource should monitor the functionality of the user input API. In addition, a (potentially) different resource should be responsible for monitoring the condition of the data arriving at our server and its successful delivery into data storage. This resource should have direct lines of communication with the modelers. The website API should have some data checks in place and be able to automatically abandon data ingestion and produce error messages back to the user to relieve burden on our technical resources and the rest of our back-end system.

Section 2.4: Storage Systems

Most of our input data is already structured in relational-friendly tables, so storage needs should be fulfilled by a relational (SQL) model. The exact database to use depends on existing infrastructure, technical expertise required and available, and cost. SQLite, a free serverless (cloud-based) database, would keep costs low assuming it can handle our load appropriately. Since the volume of writes will likely be higher than the volume of reads, another option is MySQL which offers a log-structured (LSM) alternative to its default B-tree solution. MySQL would give us the freedom to test and switch between LSM and B-tree indexing within the confines of a relational model and without having to change solution. Many engineers are familiar with SQLite and MySQL making maintainability easier to fulfill.

A potential exception to the relational data model would be the user input data. While storing user preferences alongside preferences from other data sources in a relational database, storing user data in a document data model as well would enable optimizations from locality when producing recommendations for a user. However, a cost-benefit analysis is needed 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.

Regardless of the products used, a database administrator will need to keep 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**

To *develop* the recommender engine, analysis would require large infrequent queries of the data, an ideal use case for data warehousing. The IMDb data is particularly predisposed to a star-schema structure. However, the most frequently performed analysis for the *production* engine requires only the pre-fitted model parameters and a single user’s input data. Applications for dashboards, which data warehousing is built to support, are limited to reporting on model performance and load volume. As most data warehousing solutions are not cheap, several unknowns -- including the load and expected future load of incoming data, the size of queries for analysis, their frequency, our database read performance, and the impact that slow database read times have on engine development, launch, and adoption -- need answering before deciding to purchase a data warehouse solution.

To execute analyses such as model fitting and prediction generation, other analytical tooling is required. Python is free, well known to engineers and analysists alike, and may be used in other areas of the data pipeline. To leverage Python, or any scripting language that supports neural networks, whenever a recommendation request is submitted, the back-end server would process any new data submitted with the request, consolidate that data with the user’s existing data sourced from storage, and run a Python script leveraging the fitted model (also from data storage) on that data. There are also several options for Python development environments for exploration and model fitting.

A resource should be in charge of periodically evaluating new data for statistical consistency and conducting frequent model validation and tuning to ensure the engine continually maintains or increases in efficacy. This resource should have direct communication with decision makers in case of model failure or required updates.

**Section 4: Computing and Communications Systems**

The computing infrastructure for this project will have to support data storage (and maybe warehousing), model development, web hosting, ETL process flows, and production model predictions. With over 600,000 to 700,000 movie titles on IMDb plus upwards of 20 million movie ratings applied to 27,000 movies in a single file from MovieLens, storing and exploring the data, and fitting neural network models to it will take a great deal of computing power, requiring distributed and/or cloud computing. Since bulk data uploads and neural network fitting – which is notoriously slow -- will be only periodic, and estimated user input load may be highly inaccurate, the elasticity of a cloud system would allow us to dynamically and efficiently scale computing to accommodate these variable jobs in close to real time.

Additional specifics on the estimated load (and growth of load) in request per minute from users and the ratio of reads to writes of a production system, plus the computing power required for model fitting, will be key in determining the precise specifications of any cloud computing or distributed system. To begin, our response time for delivering recommendations upon user request should be less than 1 second for p99 (the 99th percentile). Monitoring throughput and past response times at various percentiles will help us anticipate and prepare for scaling challenges before they impact the system.

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/>

1. Resource may always refer to a single individual or a team of people [↑](#footnote-ref-1)