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

**Management Overview and Recommendations**

Introduce the management problem and use cases.

The task at hand is to define the business processes and high-level system requirement for a new movie recommender engine. (vs. Systems Design and Specification). This is: Functions, performance, qualities, and constraints – the “what” to guide the design. Not, Architecture and the technical “how” to meet the requirements and to guide the builders or coders

A Requirement is a statement of one thing a product must do or a quality it must have. A Requirement Specification is a collection of the set of all requirements that are to be imposed on the design and verification of the product. The specification also contains other related information necessary for the design, verification, and maintenance of the product.

Functional refers to business needs such as business rules, process flows

A data scientist has a new business/product opportunity. Inspired by neural network encoding techniques, the data scientist has devised a way to represent feature length movies by vectors of numbers. The method draws data from the entire Internet Movie Database (IMDb) from Amazon, as well as open-source movie ratings from MovieLens.org. With movie embeddings and preference rankings of twenty or more movies from an individual reviewer, the data scientist can generate personalized movie recommendations that are more accurate than those from other recommendation engines.

* What should the system do?
* How can it best be defined to serve movie consumers?

Provide a general description of the recommended solution in terms that management can understand. Functional requirements definition for a working prototype:

* How shall data be stored, searched, and processed?
* business processes
* system requirements

Review alternatives that were considered in arriving at the recommended data engineering solution.

Justify system and application design decisions in terms of benefits and costs.

Content Notes:

* reliability - must be able to tolerate faults (hardware, software, human)
  + not a finance thing, enjoyment vs necessity so the world will still turn if the system fails, but at the same time, the more it fails, the less users will like, use, and recommend our new product
* scalability - handling load, performant
* maintainability - should be easy/simple to evolve
* batch processing of imdb updates
* stream processing of new user movie preferences from MovieLens.org?
* api + queues for new recommendation requests
* break down that application into what types of tools to use for each step/part, must be used in combination with each other
* handle expected issues with incoming data
  + movie db
  + rankings db
  + users input
* well designed api that checks/limits input
* automated & comprehensive testing
* phased rollout for launch of product
* save old files in case we need to roll-back newer modeling version
* load
  + requests per second from client to a web server
  + ratio of reads to writes in a database
* what is the throughput—the number of records we can process per second, or the total time it takes to run a job on a dataset of a certain size
  + load parameters – which of the following will be more common, vs. more rare
    - volume of reads, (increasing faster)
    - volume of writes, (increasing slower)
    - volume of data to store (increasing slower – slowest for movie db)
    - complexity of the data, (data is simple, relational)
    - response time requirements (below)
    - access patterns, (web server)
  + use netflix as a guide for load growth, but it won’t be that fast
  + start with good machines, easy to add more (scale out)
  + elastic to add machines when load gets bigger. New product, not existing with known load, highly unpredictable
* + response time - the time between a client sending a request and receiving a response.
  + Median (p50) vs highest 99th percentile (p99)? (less than 1 second?)
  + Amazon has also observed that a 100 ms increase in response time reduces sales by 1% [[20](https://learning.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ch01.html#MakeDataUseful2006td)], and others report that a 1-second slowdown reduces a customer satisfaction metric by 16% [[21](https://learning.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ch01.html#Everts2014vm),[22](https://learning.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ch01.html#Brutlag2009ut)].
  + Reporting of response times
* Use systems and languages that are known by many to foster maintainability.
  + Operations team to monitor health, track down issues, keep software/platforms updated, anticipating changes to movie/preference db sources, setting up processes and preserving knowledge
  + Want data systems that will provide visibility into load, runtime, support automation – this use case is pretty perfect for automation,
  + Keep documentation up to date and clean + consistent terminology and naming
  + Limit special casing, abstraction, sql, python for modeling
* Fewer movies than online transactions, stock market changes, etc.

**Data and Database Software**

The data required for this new personalized movie recommender engine must be acquired 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. Interested users of the recommender engine themselves.

While IMDb does not provide an API for public queries, it offers downloadable compressed plain text files containing most of the data found on IMDb. There are three options for accessing the information contained in these files: command-line interface tools provided, and Java-based GUI that can process the files and 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.

MovielensSubsets of IMDb data are available for access to customers for personal and non-commercial use. You can hold local copies of this data, and it is subject to our terms and conditions. Please refer to the [Non-Commercial Licensing](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) and [copyright/license](http://www.imdb.com/Copyright?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_lk2) and verify compliance.  
  
**Data Location**   
  
The dataset files can be accessed and downloaded from [https://datasets.imdbws.com/](https://www.imdb.com/offsite/?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&page-action=offsite-imdbws&token=BCYnusbHWZdwO9J0lFHhZ2eeFMwTDfHL0Nl7N6ApQBKsFwhmHUlsT_z8URZO84YmsyVSywUMl1zq%0D%0Ak69mBIAXrXiiC3p_amCAzgrqyhZuJL2pYOKjqztPiJ2QFzSky-YKjL6_v26zu5zdZkPCIDSAA5Bk%0D%0A9SQ6kyrk6XRAGxZ2ulRrYIxoEJ4KINmnkcCCJAuIGOAG2uYlI3g4mIodSVDVTHK2xg%0D%0A&ref_=fea_mn_lk3). The data is refreshed daily.  
  
**IMDb Dataset Details**   
  
Each dataset is contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. A *‘\N’* is used to denote that a particular field is missing or null for that title/name. The available datasets are as follows:   
  
**title.akas.tsv.gz** - Contains the following information for titles:

* titleId (string) - a tconst, an alphanumeric unique identifier of the title
* ordering (integer) – a number to uniquely identify rows for a given titleId
* title (string) – the localized title
* region (string) - the region for this version of the title
* language (string) - the language of the title
* types (array) - Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay". New values may be added in the future without warning
* attributes (array) - Additional terms to describe this alternative title, not enumerated
* isOriginalTitle (boolean) – 0: not original title; 1: original title

**title.basics.tsv.gz** - Contains the following information for titles:

* tconst (string) - alphanumeric unique identifier of the title
* titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
* primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
* originalTitle (string) - original title, in the original language
* isAdult (boolean) - 0: non-adult title; 1: adult title
* startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
* endYear (YYYY) – TV Series end year. ‘\N’ for all other title types
* runtimeMinutes – primary runtime of the title, in minutes
* genres (string array) – includes up to three genres associated with the title

**title.crew.tsv.gz** – Contains the director and writer information for all the titles in IMDb. Fields include:

* tconst (string) - alphanumeric unique identifier of the title
* directors (array of nconsts) - director(s) of the given title
* writers (array of nconsts) – writer(s) of the given title

**title.episode.tsv.gz** – Contains the tv episode information. Fields include:

* tconst (string) - alphanumeric identifier of episode
* parentTconst (string) - alphanumeric identifier of the parent TV Series
* seasonNumber (integer) – season number the episode belongs to
* episodeNumber (integer) – episode number of the tconst in the TV series

**title.principals.tsv.gz** – Contains the principal cast/crew for titles

* tconst (string) - alphanumeric unique identifier of the title
* ordering (integer) – a number to uniquely identify rows for a given titleId
* nconst (string) - alphanumeric unique identifier of the name/person
* category (string) - the category of job that person was in
* job (string) - the specific job title if applicable, else '\N'
* characters (string) - the name of the character played if applicable, else '\N'

**title.ratings.tsv.gz** – Contains the IMDb rating and votes information for titles

* tconst (string) - alphanumeric unique identifier of the title
* averageRating – weighted average of all the individual user ratings
* numVotes - number of votes the title has received

**name.basics.tsv.gz** – Contains the following information for names:

* nconst (string) - alphanumeric unique identifier of the name/person
* primaryName (string)– name by which the person is most often credited
* birthYear – in YYYY format
* deathYear – in YYYY format if applicable, else '\N'
* primaryProfession (array of strings)– the top-3 professions of the person
* knownForTitles (array of tconsts) – titles the person is known for

users

To do this, we would require a unique identifier for the user ideally based on an email account but potentially based on an IP address if we do not want to require users to create accounts with us. In addition, users should input user-friendly movie identification information, such as title and release year, as well as some sort of preference rating, the nature of which will depend on how we choose to encode final inputs into our modeling.

databases

s, but the movie related data are in English. A [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) package called IMDbPY can also be used to process the compressed plain text files into a number of different [SQL](https://en.wikipedia.org/wiki/SQL) databases, enabling easier access to the entire dataset for searching or data mining.[[26]](https://en.wikipedia.org/wiki/IMDb#cite_note-26)

from Amazon, as well as open-source movie ratings from MovieLens.org. With movie embeddings and preference rankings of twenty or more movies from an individual reviewer, the data scientist can generate personalized movie recommendations

Specify data sources and how data will be acquired and maintained.

Note database systems that will likely be used for implementing the application or system.

Justify data and database design decisions.

Notes:

* store movie encodings in database
  + leverage search indexes - hashing
  + anything to cache?
* store user preferences…
* duplicate imp data on multiple hardware to account for crashes
  + especially previous user inputs
  + can always re-pull from imdb, etc.
  + + software fault-tolerance techniques
* Sql (relational) data model
  + This part would be commercial
    - Can we use existing software? No if it’s not appropriate, no force-fitting
  + *Maybe* document model for user inputs
* Diff data
  + 1 table with movie id & movie info (title + year?)
  + 1 table with training preference data??: user type??? move ids? Ratings?
  + 1 table with user preferences to predict?: user id, movie Id, rating
    - Else document for each user with their preferences – updated via web-api. LOCALITY.
      * Relational dbs that support: postgresql, mysql
  + Sql lends itself to parallism
* Indexing = yes since reads more and more imp than writes
  + Bloom filters for movie lookup prior to the imdb update every 10 days?
* Cassandra? (nosql)
* Mysql can switch to lsm data storage if need be (One such embedded RocksDB implementation is MyRocks that replaces the default InnoDB engine in MySQL: <https://blog.yugabyte.com/a-busy-developers-guide-to-database-storage-engines-the-basics/> )
* Sqllite? If possible
* Or….. etl into data warehouse? Maybe not.
  + Sap hana (optimized for both storage and warehousing) for new incoming preferences?
  + More recently, a plethora of open source SQL-on-Hadoop projects have emerged; they are young but aiming to compete with commercial data warehouse systems. These include Apache Hive, Spark SQL, Cloudera Impala, Facebook Presto, Apache Tajo, and Apache Drill
  + Not broad use case of querying full co. data and performing analytics. Limited data, so star/snowflakes/columnar data storage may not be necessary
  + No simple aggregation. Complex modeling. So pull from data *storage* instead
  + Maybe if the models are only re-trained once per 10-days? But nahhhhh. Then trained mode parameters stored….. in a document? In data *storage*?

**Analytics and Modeling Software**

Specify analytics and modeling software that will be needed for development, testing, and implementation.

Justify analytics and modeling design decisions.

Notes:

* Should be able it identify when user asks for recommendation with a LOT of previous preference inputs. May take up more resources so potential alternate processes/resources should be leveraged upon large model identification
* Prepare for faulty input data and produce appropriate ‘error’ messages when this happens
* Some way of confirming that each request is dealt with and output is sent
* Separate (copied) sandbox environment for periodic (and frequent) model testing & evolution
* Phased roll-outs of new models
* Monitoring of inputs/output
  + Business processes for maintenance checks, quick escalation, easy methods for roll-backs/fixes

**Computing and Communications Systems**

Specify the computing infrastructure, resource requirements (in terms of processing power and memory?)

information systems and connections between systems.

Justify information infrastructure design decisions.

**Notes:**

* clarity of expression
* correctness of exposition
* The way you write for management affects the way management views your work.
* five-page text maximum. Appendices, tables, figures, and reference lists not included
* The report should be submitted as an Adobe Acrobat pdf file.

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/>