CSE 515 Phase 1 Report

Greggory Scherer

Xiangyu Guo

Alfred Gonsalves

Chenchu Gowtham Yerrapothu

Siddhant Tanpure

# Abstract

This paper describes four queries for retrieving data related to three different entities in a subset of the Internet Movie Database. These four queries provide the ability to determine keyword relationships between actors and movies, genres and movies, and users and movies. Term frequency, inverse-document-frequency, and a relational model are some of the tools explored. Three methods of differentiating two different genres are also explored, using inverse-document-frequency and probabilistic feedback.

# Keywords

IMDb, document, term frequency, inverse document frequency, dimension, measure, weight, vector, timestamp, rank, tag, differentiation function.

# Introduction

The Internet Movie Database (IMDb) began as a searchable list of credits aggregated from a Usenet group and now contains over 4,517,776 titles and 8,136,918 people [1][2]. IMDb allows users join the site and vote on titles using a 1-10 scale, as well as provide feedback or comments on the titles [3]. For this project, a set of data was provided from IMDb containing relationships between titles (movies), actors, genres, and users. Movies ranked based on votes using an undisclosed formula [3]. IMDb movies are also tagged with a set of keywords.

## Terminology

* **Document** refers to a collection of words, but can also be used to refer to an entity or object such as actor or genre.
* **Term Frequency (TF)** refers to a proportion of how often a term is appears in a document and the total terms in a document [4].
* **Term Frequency - Inverse Document Frequency (TF-IDF)** refers to the discriminative power a term has in a given document. In other words, how well a particular term describes a document compared to other terms.
* **Dimension** refers to any arbitrary aspect of an entity that is distinguishable from any other.
* **Measure** refers to a numerical value determined by the intensity of some dimension of an entity. For example, length is the measure of how long (intensity) something is.
* **Measurement Unit** (unit of measure) refers to a mapping of a measure discrete proportions of a predetermined referential measurement. For example, a meter is a predetermined referential measurement of length.
* **Weight** refers to the importance of particular measure. Higher weight is analogous to higher importance.
* **Vector** refers to an ordered series of entities, and in this project the entities are measures.
* **Timestamp** refers to a date and time which is logged for any arbitrary entity. A timestamp usually grants an entity the dimension of time, and in this document, a timestamp manifests itself in POSIX time, or seconds since January 1, 1970 at midnight UTC.
* **Rank** refers to the rank computed by IMDb.
* **Tag** refers to a keyword assigned to a particular movie by IMDb.
* **Differentiation Function** refers to a function that computes a single numerical measurement of difference between two vectors.
* **Cardinality** refers to the numerical relationship between two entities, in which three outcomes are possible: one to one, one to many, and many to many. It may also refer to the size of a set.

## Goal Description

This project contained four goals, separated into phases. Henceforth, these will be referred to as Phase 1, Phase 2, Phase 3, and Phase 4 respectively. In all cases, a tag vector is output that shows a measure for each tag. The goal for Phase 1 is to find the tags that best describe a given actor, based on the age of the tag and the rank of the movie. Tags applied to movies with higher rank should be weighted higher than tags applied to movies of lower rank. Additionally, older tags should be given less weight than newer tags. The ratio of movies to tags is one to many, so the number of times a tag appears should also play a role in the weight of a tag. The importance of these tags should be computed using either TF or TF-IDF models.

The goals for Phase 2 and Phase 3 are similar. The goal for Phase 2 is to find the tags that best describe a genre and the goal for Phase 3 is to find the tags that best describe a user. In both cases, older tags should be given less weight than newer tags and the frequency of a tag should directly affect the weight. Also in both cases, the importance of these tags should also be computed using either TF or TF-IDF models.

The Phase 4 has three subtasks, hereon referred to as TF-IDF-DIFF, PDIFF1, and PDIFF2 respectively. In all cases, the goal is to determine how different a given genre is to another genre. For TF-IDF-DIFF, the goal is to compare two vectors using the same TF-weighting mechanism in Phase 2 and applying an IDF computed using the movies from both genres instead of the entire database of movies. For PDIFF1 and PDIFF2, a probabilistic feedback mechanism is provided to determine the weights for the tag-vectors.

## Assumptions

1. The database is unchanging. Some or all measurements can be computed in advance, and stored in memory for output or be hard-coded in the program.
2. A movie can only have one rank.
3. The cardinality of movies to tags is many to many.
4. The cardinality of users to movies is many to many.
5. The cardinality of genres to movies is many to many.
6. The cardinality of actors to movies is many to many.
7. When counting total actors, only actors in movies that have tags are important to consider.
8. When counting total genres, only genres for movies that have tags are important to consider.
9. When counting total users, only users that have tagged or voted on movies with tags are important to consider.
10. Higher ranks have lower numbers.
11. Invalid input will print error.
12. Queries with no results will return empty.

# Proposed Solution

For Phases 1-3, two equations are proposed to generate the results of the respective queries. is an equation that generates a TF vector for a defined input, and is an equation that generates a TF-IDF vector for a defined input. For Phase 4, three equations are proposed to generate the comparison of two genres, namely , , and that correspond to TF-IDF-DIFF, PDIFF1, and PDIFF2.

## Phase 1

The team determined that the best way to generate a weighted TF by tag-age, tag-frequency, and movie-rank was to first calculate the weight of the tag by both frequency and age and then multiply that weight by the weight of the rank. The weight of tag by age and frequency is computed by taking the distance of the timestamp from the minimum timestamp and dividing it by the total timespan offered in the database. In this way, the newest timestamps would be closer to 1 and the oldest timestamps would be closer to 0. The total timespan in the database was chosen instead of a measure of the distance from the current day in order to make the algorithm consistent regardless of the current date and to prevent the calculations from getting too small. By taking the sum of all these calculations the frequency of the tag is implicitly factored in. By adding 1 to the numerator and denominator, division by zero or other undefined cases. In order to compute rank, since lower numbers are considered higher rank, the inverse of the rank is used so that rank 1 is closer to 1 and the lowest rank (highest number) is closer to 0. The timestamps are based on the tags on movies in which an actor appears. These decisions are described in the equations below.

## Phase 2

Since movie rank is not taken into account in Phase 2, a simplified weighting function is used. The weighting function follows a similar procedure as in Phase 1, however there is no multiplication with the inverse rank. Additionally, the timestamps are based on tags for movies in a given genre instead of movies in which an actor appears.

## Phase 3

Similar to Phase 2, the weighting function does not take into account movie rank. Instead of determining timestamps based on tags for movies in a given genre, Phase 3 determines timestamps based on tags for movies either tagged or rated by a given user. Since there are movies that are not rated by users, it was necessary to determine all movies either rated or tagged by a user first, then cross-reference that to movies tagged by any user. In this way, a user that has not tagged a movie can still be described by a tag.

## Phase 4

TF-IDF-DIFF reuses the TF-weighting function from Phase 2, and uses a specialized IDF function to compute TF-IDF vectors for both functions. It then computes the difference between the two functions.

P-DIFF-1 and P-DIFF-2 both use the same probabilistic feedback mechanism, however with different parameters. Both methods produce a single tag vector that describes the difference (by tag) of both genres.

### TF-IDF-DIFF

#### Differentiation Functions

### P-DIFF Weighting Function

This function computes the weight for tags for either P-DIFF-1 or P-DIFF-2. This is parameterized by the following two sections. The first case is the exceptional case, where the numerator of any fraction may be zero, or divide-by-zero cases, where 0.5 and 1.0 is added to the respective numerators and denominators to prevent this.

## P-DIFF-1

## P-DIFF-2

# Interface Specifications

The system-level interface is by command line. The following commands should produce tag vectors for each Phase.

java -cp ./bin:./lib/postgresql-42.1.4.jar cli/print\_actor\_vector actor1 model

Phase : Command-Line Interface

* *actor1* refers to an integer associated with an actor’s id
* *model* refers to a string in { “tf”, “tfidf” }

java -cp ./bin:./lib/postgresql-42.1.4.jar cli/print\_genre\_vector genre1 model

Phase : Command-Line Interface

* *genre1* refers to a string associated with a genre name
* *model* refers to a string in { “tf”, “tfidf” }

java -cp ./bin:./lib/postgresql-42.1.4.jar cli/print\_user\_vector user1 model

Phase : Command-Line Interface

* *user1* refers to an integer associated with a user’s id
* *model* refers to a string in { “tf”, “tfidf” }

java -cp ./bin:./lib/postgresql-42.1.4.jar cli/differentiate\_genre genre1 genre2 model

Phase : Command-Line Interface

* *genre1* refers to a string associated with a genre name
* *genre2* refers to a string associated with a different genre name
* *model* refers to a string in { “tf-idf-diff”, “pdiff1”, “pdiff2” }

The API for each Phase is realized by a class called VectorFactory. This class provides five methods to produce tag vectors in the form of Java ArrayLists. The values within the vectors are instances of TaskVector class, which is a container of the tag string, tag id, and weight. The API is described below.

Table : API by Task

|  |  |
| --- | --- |
| Task | Interface |
| Phase 1 | buildTask1Vector(Integer, String) : ArrayList<TaskVector> |
| Phase 2 | buildTask2Vector(String, String) : ArrayList<TaskVector> |
| Phase 3 | buildTask3Vector(Integer, String) : ArrayList<TaskVector> |
| TF-IDF-DIFF | buildTask4TF\_IDF\_DIFFVector(String, String, Object[]) : ArrayList<TaskVector> |
| P-DIFF-1/P-DIFF-2 | buildTask4PDIFFVector(String, String, Object[]) : ArrayList<TaskVector> |

buildTask1Vector takes an actor ID and a model string to produce a tag vector.

buildTask2Vector takes a genre string and a model string to produce a tag vector.

buildTask3Vector takes an user ID and a model string to produce a tag vector.

buildTask4TF\_IDF\_DIFFVector takes two different genre strings and an optional array of objects reserved for debug parameters.

buildTask4PDIFFVector takes two different strings and an array of objects where the first index is reserved for debug parameters and the second index must be a Boolean which indicates whether to use P-DIFF-2 model.

# System Requirements and Installation Instructions

* Operating System
  + Linux
  + Mac
  + Windows
* PostgreSQL database
  + Install PostgreSQL
  + Create a user named greggoryscherer with no password (see SQL.txt)
  + Create a database named greggoryscherer owned by user greggoryscherer (see SQL.txt)
  + Run SQL Commands in SQL.txt replacing /path/to with the path to the files on your machine
* Eclipse
  + Import the archived project into eclipse
  + Right-click package **test** and choose **Run As** -> **JUnit Test**
  + All tests will pass if the database is set up correctly
* Running the program
  + Build project in eclipse
  + Navigate to project directory in workspace
  + Navigate to **bin** directory within project
  + Run one of the commands described in the previous section

# Related Work

Juan Ramos. 2003. Using TF-IDF to Determine Word Relevance in Document Queries. (January 2003) DOI: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf>

K. Selçuk Candan and Maria Luisa Sapino. 2010. Data Management For Multimedia Retrieval. 147.

# Conclusions

Term frequency and inverse-document-frequency measurements help determine relevance of a term to an entity and the discrimination power of a term respectively. It is important to use the correct superset when computing the IDF otherwise the weight will have less effect. When comparing genres, using a specialized IDF with union of both documents can help determine how well one term describes one document as opposed to the other. Additionally, differentiation functions such as p-norm, dot-product, cosine, and angle help further describe how far apart one document is from another in vector space. When comparing any two documents, p-norm calculations were relatively consistent while dot product, cosine, and angle varied greatly between any two genres.

# Bibliography

[1] Col Needham. 2010. IMDb History. (October 17, 2010). DOI: <http://www.imdb.com/help/show_leaf?history>

[2] IMDb Stats. Retrieved September 17, 2017 from <http://www.imdb.com/stats>

[3] IMDb Votes/Ratings Top Frequently Asked Questions Retrieved September 17, 2017 from <http://www.imdb.com/help/show_leaf?votestopfaq>

[4] How to compute. Retrieved September 17, 2017 from <http://www.tfidf.com>

# Appendix

## Specific Roles of Group Members

1. Xiangyu Guo - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing
2. Siddhant Tanpure - Group discussion, Independent implementation
3. Chenchu Gowtham Yerrapothu - Group discussion, Independent implementation
4. Alfred Gonsalyes - Group discussion, Independent implementation
5. Greggory Scherer - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing