LinguisticDB Final Project Report

By Maya Messinger, Alex Angel, John Benhart, Ryan Piersma

Description of the Application

LinguisticDB is a website that provides an extensive search interface for all 30,000+ books and 12,000+ authors of the Project Gutenberg database including a comparison algorithm that compares books based on linguistic features such as cosine similarity and word sequences. Our website also supports the ability for users to create accounts, rate, and comment on books.

The website has numerous features. The homepage (the “Search” page) provides a simple search as well as an advanced search feature via the drop-down button with 11 different fields the user can specify. The user can click the chosen book title in the results list to be redirected to the book page for that book. A Project Gutenberg link is also provided in the search results if the user wishes to read the book. (Note: Our website does not store the actual book content.)

The “About” page provides general information about our website, including the fact that we are not affiliated with the Project Gutenberg.

The “Statistics” page provides aggregate information for the user to better understand our dataset, like the numbers of books, users, and reviews on our website.

The “Sign In” page which lets the user create an account or sign in to an existing account. Being signed in is necessary to be able to comment or rate books on their respective book pages.

The “Profile” page lets the user change the account password, view the titles for the books that have been rated using that account and the corresponding rating, and view the titles for the books that have been commented on. The links in the “Profile” page will redirect the user to the book pages.

The individual book pages include summary information about each book, including the following:

- Title
- Book uid (Note: The uids on LinguisticDB and the uids on Project Gutenberg match, which makes it easy for the user to retrieve a book from the Project Gutenberg website by uid)

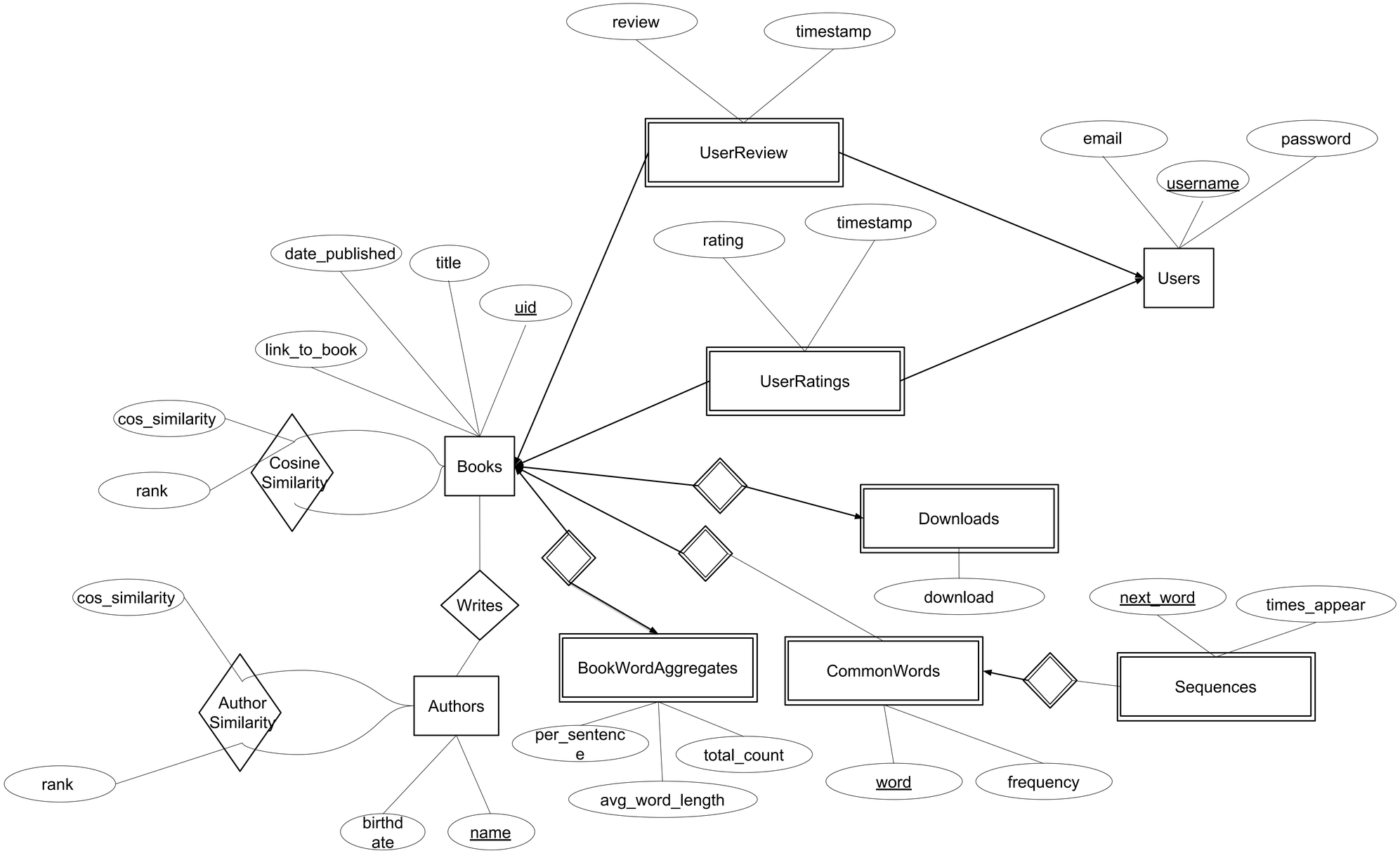
- Author name and birthyear
- “Date released” is the date the book was added to Project Gutenberg
- The word count for the book
- Average sentence length (in words)
- Average word length (in characters)
- Number of downloads on the Project Gutenberg website (rolling 30 days value)
- Average user rating on LinguisticDB
- Number of ratings on LinguisticDB
- “Rate this book” feature if the user is logged in
- Top 5 most common words in the book (words with length > 3)
- Top 5 most popular word sequences (words of all lengths)
- List of comments with username and timestamp
- Ability to comment if the user is logged in

On the right hand-side of a book page, the user will find the book titles for the five books that are most common to the book the user is viewing (using our cosine similarity algorithm).

Survey of Related Work

Other web services that provide resources in the same vein as LinguisticDB include Goodreads, Project Gutenberg itself, LibraryThing, and the World Atlas of Language Structures. These services can roughly be divided into two categories - those which cater to “ordinary” people by placing books and other reading materials into a social network (Goodreads, LibraryThing), and those which cater to linguists by offering linguistic analysis and study (Project Gutenberg, World Atlas of Language Structures). LinguisticDB seeks to fill the niche occupying both of these areas, by integrating a socially oriented book rating system with linguistic analysis such that a familiar platform can introduce true linguistic concepts to the masses while still encapsulating the functionality of a modern social book network.

E/R Diagram


review
timestamp
UserReview
email
password
username
Users
rating
timestamp
date_published
title
uid
link_to_book
UserRatings
cos_similarity
Cosine Similarity
rank
Books
Downloads
download
Writes
cos_similarity
next_word
times_appear
BookWordAggregates
CommonWords
Sequences
Author Similarity
Authors
per_sentenc e
total_count
avg_word_length
word
frequency
rank
birthd ate
name

Assumptions about the data

A big assumption was figuring out how to account for every book's Project Gutenberg licensing/introduction - instead of parsing every book until we found the start of the actual text, we assumed every Gutenberg intro was approximately equivalent in length, and just cut off 1200 words from the length of each book. So some words parsed and counted as actual book text might not be actual book text, and some words from books may have been cut if the Gutenberg intro was shorter than expected.

For sentence lengths, we assumed any period, question mark, or exclamation mark ended a sentence - stats/averages are thrown off is someone is quoting, or by names with Mr./Mrs. Prefixes but there wasn't a way to account for this without significantly compromising efficiency of parsing.

For Sequences and CommonWords relations, the underlying scripts relied on the inclusion of the strings "START OF THE PROJECT GUTENBERG EBOOK" and "FOOTNOTES" in each book text file in order to begin and terminate (respectively) the collection of data to be recorded. If some of the books did not contain these strings, then extra garbage data was included.

Outlier books with extreme data points barely affect overall statistics - we did not sanitize the database to remove books without punctuation (affects words per sentence stat).

Schema

The schema is as follows:

Authors (name VARCHAR(256) NOT NULL PRIMARY KEY,
birthdate INTEGER CHECK(birthdate < date_part('year', current_date)));
Books (uid INTEGER NOT NULL PRIMARY KEY,
title VARCHAR(512) NOT NULL,
date_published VARCHAR(256) NOT NULL,
link_to_book VARCHAR(256) NOT NULL);
Writes (uid INTEGER NOT NULL REFERENCES Books(uid) PRIMARY KEY,
name VARCHAR(256) NOT NULL REFERENCES Authors(name));
BookWordAggregates (uid INTEGER NOT NULL REFERENCES Books(uid) PRIMARY KEY,
per_sentence REAL NOT NULL,
total_count REAL NOT NULL,
avg_word_length REAL NOT NULL);
CommonWords (uid INTEGER NOT NULL REFERENCES Books(uid),
word VARCHAR(256) NOT NULL,
frequency INTEGER NOT NULL,
PRIMARY KEY(uid, word));
Downloads (uid INTEGER NOT NULL REFERENCES Books(uid) PRIMARY KEY,

download INTEGER NOT NULL);
Sequences (uid INTEGER NOT NULL REFERENCES Books(uid),
word VARCHAR(256) NOT NULL,
next_word VARCHAR(256) NOT NULL,
times_appear REAL NOT NULL,
PRIMARY KEY(uid, word, next_word));
Users (username VARCHAR(256) NOT NULL PRIMARY KEY,
email VARCHAR(256) NOT NULL,
password VARCHAR(256) NOT NULL);
UserRatings (username VARCHAR(256) NOT NULL REFERENCES Users(username),
book_id INTEGER NOT NULL REFERENCES Books(uid),
rating INTEGER NOT NULL,
timestamp TIMESTAMP NOT NULL,
PRIMARY KEY(username, book_id));
UserReview (username VARCHAR(256) NOT NULL REFERENCES Users(username),
book_id INTEGER NOT NULL REFERENCES Books(uid),
review VARCHAR(256) NOT NULL,
timestamp TIMESTAMP NOT NULL,
PRIMARY KEY(username, book_id));
CosineSimilarity (uid1 INTEGER NOT NULL REFERENCES Books(uid),
uid2 INTEGER NOT NULL REFERENCES Books(uid),
cos_similarity REAL NOT NULL,
rank INTEGER NOT NULL,
PRIMARY KEY(uid1, uid2));
AuthorSimilarity (author1 VARCHAR(256) NOT NULL REFERENCES Authors(name),
author2 VARCHAR(256) NOT NULL REFERENCES Authors(name),
cos_similarity REAL NOT NULL,
rank INTEGER NOT NULL,
PRIMARY KEY(author1, author2));

Indexes

Indexes were useful to improve the performance of our website, especially for the "Advanced Search" feature. We have implemented the following, as best agreed on by members of the team in order to best speed up the LinguisticDB website without cluttering up space with too many indexes:

- CREATE INDEX BookTitles ON Books(title);
- CREATE INDEX AuthorWrites ON Writes(name);
- CREATE INDEX Aggregates1 ON BookWordAggregates(per_sentence);
- CREATE INDEX Aggregates2 ON BookWordAggregates(total_count);

- CREATE INDEX Aggregates3 ON BookWordAggregates(avg_word_length);
- CREATE INDEX CommonWordsIndex ON CommonWords(word);
- CREATE INDEX SequencesIndex ON Sequences(word);
- CREATE INDEX Cos1 ON CosineSimilarity(uid1);
- CREATE INDEX Author1 ON AuthorSimilarity(author1);

Implementation

Google Cloud VMs (3 - one for database, one for back-end server, one for front-end)
HTML
Node.js
Vue.js
Python scripts to parse books

Algorithms Used - Cosine Similarity Algorithm

We approximate similarity between books and between authors to provide database users with information on how closely related two members of these entities are in the database. The algorithm used to compare books is Cosine Similarity, comparing each book as a vector over the vocabulary of words across all texts. Authors are compared using an average of the cosine similarities between their books.

The components for a given word in the book vector are defined by the tfidf scores for each word in that entity. TF refers to term frequency for the word within a book, and IDF refers to the inverse document frequency of that word across all books. These terms are multiplied together for each word in the book's text, generating tfidf values that give the relative importance of a word's appearance in the text. For example, a word such as "the" has a high number of occurrences (tf) in each text but a low idf (almost every document would have "the" in it); so, it's tfidf score would be relatively low compared to a word like "chariot," which presumably would not occur in very many texts.

After generating this vector of tfidf scores over the vocabulary for each text, two texts are compared by finding the dot product of the two vectors and normalizing by the product of their magnitudes. This produces an approximation of the angle between the two vectors. A cosine similarity score closer to 1 signifies highly similar texts, while a score closer to zero or even negative signifies dissimilar texts. For our purposes, because tf and idf are always zero or positive, cosine similarity always returns a positive value. We compared authors with other authors and books with other books to generate a list of similar authors/books for use by our website's user. Below, you can find the equation for cosine similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiH74Hgz5bfAhVvmeAKHfwECJMQjhx6BAgBEAM&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FCosine_similarity&psig=AOvVaw0u0BZOOJWccEijoDDwl6W7&ust=1544578028134352

System Evaluation

We feel that this does not apply to our project in the sense that there are few objective measurements to make regarding the performance of our website, as that is an aspect that, although crucial, is not paramount to the existence of our website. Additionally, to truly evaluate our system in the context of other systems would require extensive study beyond the scope of a course project.

Directions For Future Work

The future of LinguisticDB is centered in strengthening the two aspects mentioned in the survey of related work: creating a socially appealing book platform and introducing additional linguistic concepts through it. This could be done by implementing more significant algorithms from the field of computational linguistics such as Latent Dirichlet Allocation in order to perform better author analysis. In additions, new metrics to characterize linguistic aspects such as style and genre would be helpful for simultaneously increasing the computational and foundational strength of LinguisticDB while increasing its overall public appeal. Additionally, further meaning