Literari.ly

Adam Lenart, Ankith Gunapal & Sandip Palit

December 13, 2016

1 Summary

Application idea Literari.ly is a literary assistant that helps users evaluating the popularity of a queried expression. Popularity is defined by (i) the observed frequency of the expression in published volumes (books and journals) and (ii) the rating that these books received. Moreover, the rating and the number of volumes which contain the queried expression can be partitioned into categories. Furthermore, literari.ly makes future predictions about the frequency with which the queried expression is going to be used.

Data sources The Google N-grams dataset is used for describing the time trends of the queried expression. The N-grams dataset includes terrabytes of data of English and several other languages 1 to 5-gram expressions. Currently, we focus on a 3.5 TB English bigram set of the data and use a 15 GB subset of it for the examples. The ratings and categorizations of the user queries stem from Google Books and Goodreads APIs. The Google Books API contains categories for the expression which occurs in the meta-information of the book as well as ratings of these books by Goolge Books users. Similarly, the Goodreads API holds information on the ratings of the books by Goodreads users. These two data sources can be joined by common ISBN13 number.

Architecture The data intensive processes, stored in HDFS, run on Apache Spark and the smaller tables containing aggregates from a PostgreSQL database are loaded into either Python or R for serving the user. R was chosen for visualizing data on its Shiny interface on the internet using an Amazon AWS instance and for forecasting the future popularity of the queried expression. Python was used to interact with Goolge Books and Goodreads APIs. Search history is visualized by a Kibana dashboard.

Services used Google bigrams data is stored in an Amazon S3 bucket, queries from Google Books and Goodreads are stored in an Amazon PostgreSQL RDS and the visual interface of the application along with a Kibana dashboard runs on Amazon AWS instances.

2 Architecture

The application relies on three data sources: Google N-grams, Google Books API and Goodreads API (Fig. 1).

Google N-grams data contain information on expressions with different gram lengths stored in compressed CSV files separated by the initial two letters of the first word of the expression from aa to zz. Presently, we use a 3.5 GB subset of the data, the English bigrams. These tables are stored in an Amazon S3 bucket, currently a 15 GB part of it, the bigrams starting with the "ab" sequence are loaded into HDFS on an Amazon EC2 instance. This "ab" table is analyzed using the R frontend of Apache Spark. R, rather than Python, was chosen for this task because of its more advanced time series modelling packages which were used to predict the future number of occurences of the expressions. The predictions are then channeled to a Shiny interface running on an Amazon AWS instance. The Shiny interface allows user interaction by accepting input queries and outputting the results.

The Google Books allowed a different kind of interaction with the data as it can be queried real-time. The API calls result in a series of JSON files that are parsed in Python and stored in an EBS volume for the Kibana search history implementation and placed in an Amazon PostgreSQL RDS for the ratings and categorizations application. The PostgreSQL database is then accessed by the Shiny frontend and outputs the result using a combination of R, Plotly and D3 plots. For the Kibana dashboard, the results are first loaded to HDFS and then analyzed by PySpark.

2.1 Challenges

Goodreads API answers one API call for one book per second. As the Goodreads users present valuable information with their insight into rating the books, we wanted to merge their ratings with the ones of the Google Books users. In order to achieve that, we first downloaded information on 50,000 titles and stored them in HDFS tables.

The Google Books API had a looser limit; it allowed a result of a maximum 40 books per query, however, as the JSON file included the total number of results with an offset information, it was possible to send one query first, read in the total number of results, and then send as many queries as necessary with an increasing offset of 40 to avoid overlaps as to download the whole series of book information. After that, it was possible to merge the information coming from the Goodreads and Google Books APIs by the ISBN number of the books.

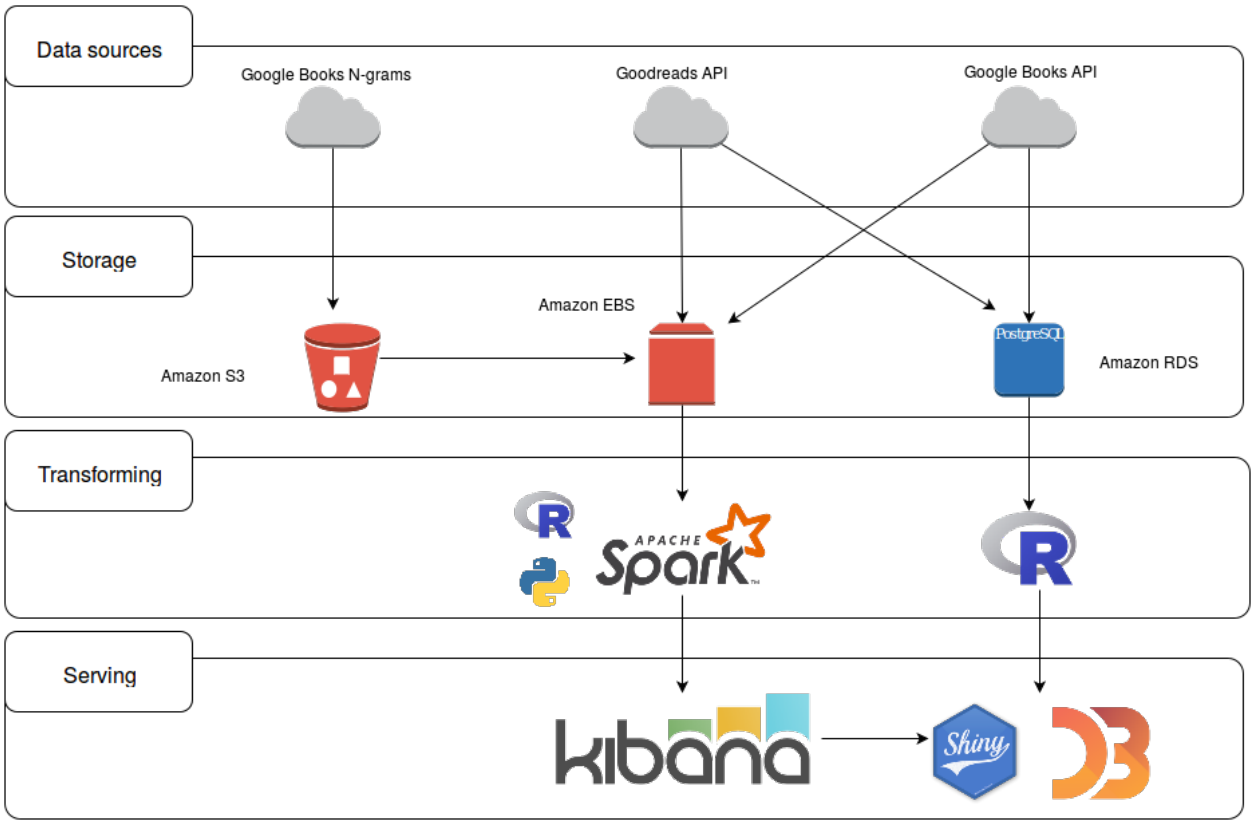

Figure 1: High-level overview of the architecture of the application

3 Data storage

Data are stored either in the attached EBS volume to an Amazon AMI, in an S3 bucket or in an Amazon RDS depending on the part of the application. The Google Books N-grams data is backed up in an S3 bucket and only a 15 GB subset of it is loaded into EBS. However, if it was necessary to be able to query the whole dataset quickly, all of the 3.5 GB could be loaded to EBS or at least stored in a RDS. The Google N-grams tables are separated by the first two letters of the first expression. For example, our example query, "aboriginal people" is located in the "ab "table. Other than the expression, the year, the number of occurrences per year and the number of volumes the expression was found in yield the columns of the Google N-grams table for forecasting (Table. 1)

Column	Description	Type
phrase	an n-gram expression	string
year	the year in which the expression is counted	integer
match count	number of occurrences	integer
volume count	number of volumes the expression occurred in	integer

Table 1: Google N-gram table

Bulk of the visualizations depend on the rating table. This table is a left join of the Google Books and Goodreads tables (Google Books on the left) as the Google Books tables generally contain more books than the Goodreads tables do and information on the categories that the volumes belong to. The rating table uniquely identifies the volumes by their ISBN13 number, includes their title, category, rating and date of publication (Table 2). The Google Books table is identical to the Ratings table.

Column	Description	Type
ISBN13	unique identifier	string
title	title of the volume	string
rating	average of Google Books and Goodreads ratings	decimal
category	category of the volume	string
date	date of publication	date

Table 2: Rating and Google Books tables

The Goodreads table differ from the Google Books only that it does not include a category column (Table 3.

Column	Description	Type
ISBN13	unique identifier	string
title	title of the volume	string
rating	average of Google Books and Goodreads ratings	decimal
date	date of publication	date

Table 3: Goodreads table

One final table (Table 4), the decade aggregates only exists for convenience, it is a weak entity, dependent on the ratings table. It counts the number of book by decades and by categories.

Column	Description	Type
decade	decade of publication	int
category	category of the volume	string

Table 4: Decades tables

An overview of the entity-relationship of the database can be seen in Fig. 2.

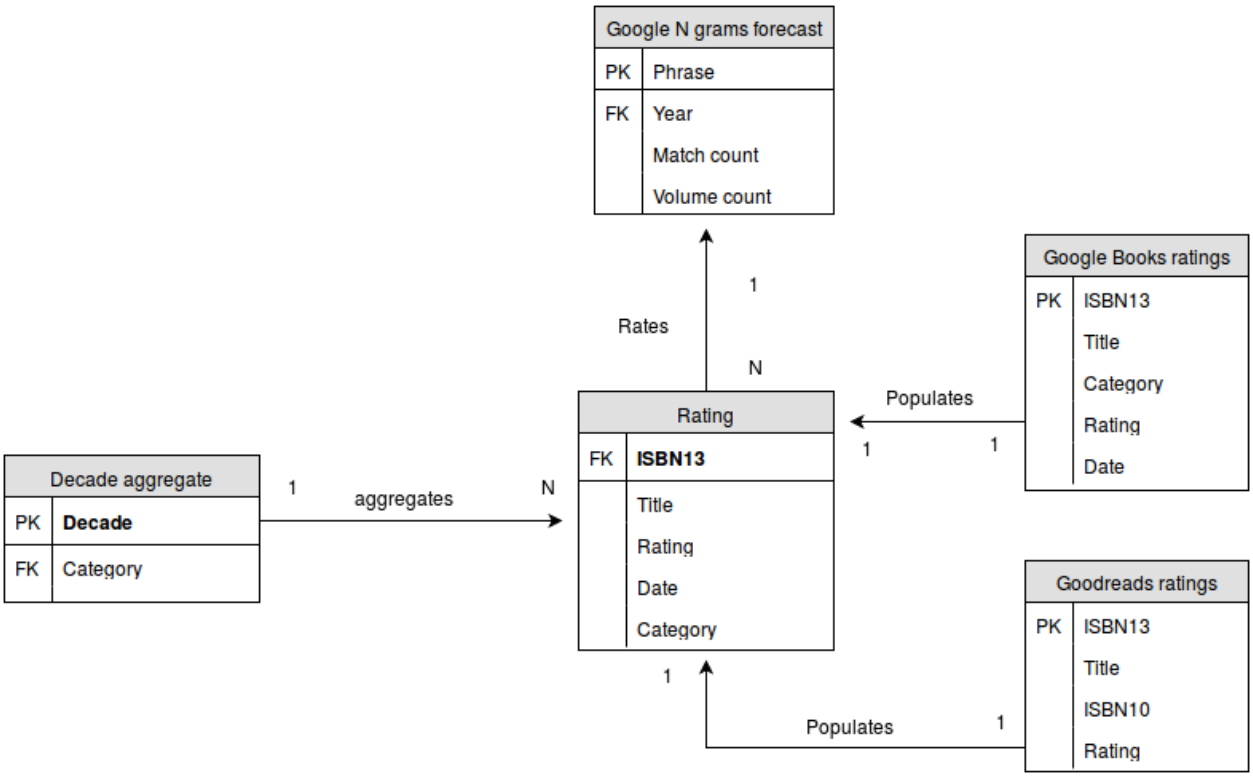

Figure 2: ER diagram

A Appendix: Sentiment Analysis of Expressions

There is a Python script to query GoodReads , once every one second, and collect information on Books and their ratings. This is stored in HDFS and we create a table called good_reads using Hive. There is a Python script to ask the user for an Expression . Ex: Hello world

```
/data/spark15/bin/spark-submit google_books_pyspark.py ''Hello world''
```

This script queries google books and obtains the list of books which contain the expression. The query result is stored in a table called google_books from pyspark's HiveContext. The two tables good_reads and google_books are merged with the common key ISBN13 (Left Join in SQL) to create a new table called query_bigram_result We store the average book rating in this merged table. The merged table (list of books) is sent to ElasticSearch. Kibana displays expressions and their query results over time. This can be useful to look at the most popular queried expressions and their results in terms of ratings per category. We can see a sample merged table when a user queries for an expression and a screenshot of kibana display in the next page.