Books Data Clustering for libraries based on Amazon reviews using distributed systems

Marwa Oussaifi, Ran Huang, Wei Wei, Ziyu Fan

Overview

The purpose of this project is to find out similar books according to the Amazon reviews. We could help the library improve its books management ecosystem and user experience by recommending those books that should've been checked out more but because of unsatisfying management, they were not noticed by people.

Our method is to get the feature vectors of each book by applying TFIDF to collections of the texts of corresponding books and use these feature vectors to cluster the books. Once we get the clusters, we would have some sense of book similarities. We can even try calculating distances between every two books in one cluster. In this way, we can recommend similar books to the library to help them purchase new books and manage existing books.

The large scale of dataset makes this pipeline and approach impractical to implement on the local machine due to the limit of memory and computing performances. Consequently, we built a data pipeline with a distributed system. Amazon Web Services (AWS) provides high availability and scalability and makes its components including storage and processing engines to be compatible. Thus, for this project, we selected it as the primary platform to store data in the data lake(S3), load data into the distributed database (MongoDB on EC2) and apply machine learning algorithms (Spark on EMR).

Exploratory Data Analysis

Motivation

The library is a public resource that every taxpayer is paying for. People are paying for this public resource not only for themselves but also for the next generation. Good operation of library helps improve people's reading and learning efficiency. However, sometimes we heard many complaints from people, especially parents, that they can not find the book they want from the library for their kids. Students also complain about the limited resources in public libraries. For example, when they want a book, either the library does not have this book at all, or they can not physically find the book due to the poor book management system. A book sometimes cannot be available because it's too popular and in short supply. There are many unsatisfying situations like this.

To help public libraries to improve the efficiency of their book management system and build a better reading environment for citizens, we are going to use clustering, a machine learning method, to quantify the similarities between books. In order to do so, we need features of the books.

Since Amazon is one of the largest online vendors of books in the world and it has a huge amount of data on customer behavior, customer purchase history, books description, and sales record, we decide to use Amazon's books data as our dataset to extract useful information and transform them into feature vectors that might help us cluster the books.

Data Overview

We got two datasets in JSON format from Amazon. One is review data, another is metadata. In both datasets, there are texts containing some information about the book.

Review data includes information on customer reviews of books. There are about 4000000 reviews of 181700 unique books from 583668 customers in our reviews dataset.

Metadata includes salesRank, categories and descriptions information of about 1685856 books.

The columns of two data set are shown below:

```
review.columns
```

executed in 25ms, finished 12:57:22 2019-01-17

```
['asin',
 'helpful',
 'overall',
 'reviewText',
 'reviewTime',
 'reviewerID',
 'reviewerName',
 'summary',
 'unixReviewTime']
```

```
meta.columns
```

executed in 22ms, finished 12:25:21 2019-01-17

```
['_corrupt_record',
 'asin',
 'brand',
 'categories',
 'description',
 'imUrl',
 'price',
 'related',
 'salesRank',
 'title']
```

Database: MongoDB

We first uploaded our data to Amazon S3. It is an object storage service that provides easy-to-use management features and connector with EC2 instances. Then we transfer the data from S3 to MongoDB to make them orderly stored and easily fetched. MongoDB is a document-oriented database. It's schema-free and it contains Database, Collection, and Document. It's appropriate to process our data in JSON format. The Amazon Elastic Compute Cloud (Amazon EC2) service enables us to launch 10 virtual machine instances with Amazon Machine Images (AMIs) and deploy the MongoDB with 2 shards, 1 configures set and 1 mongos on them.

Name	Replicas	Instance	vCPUs	Memory
Shard1	2	t2.Xlarge	4	16GB
Shard2	2	t2.large	2	8GB
Configuration	2	t2.medium	2	4GB
Mongos	0	t2.medium	2	4GB

Modeling: Spark SQL and Spark ML

Configuration

As powerful as MongoDB is on its own, the integration of Apache Spark extends analytics capabilities even further to perform real-time analytics and machine learning. Apache Spark is a unified analytics engine for large-scale data processing. Spark SQL and Spark ML are used and deployed on Amazon Elastic MapReduce (EMR) in this study. And Jupyter-notebook is built on the master node to make it easy for us to work together on the cloud.

Setting	
Master	**Core**
1 * m3.xlarge	4 * m3.xlarge

Instances	Memory	vCPUs	pricing
m3.xlarge	15 GB	4	$0.07 per Hour

Preprocessing Using Spark SQL

Among all the information of the books, we believe that the “reviewText”, which is the review for a book from customers, contains the most important characteristics representing people’s opinions on books. “Summary” is the summary of each review, which contains those representative words in the review. In metadata, “Description” is the description of the book, which is basically about the content of the book. A title of a book is often related to the content of the book. Therefore, we would extract texts from “reviewText”, “summary”, “description” and “title”. Also, “asin” is is a 10-character unique ID assigned by Amazon.com.

The next step is to create a data frame that would serve as the input of our model. Here are the steps we took:

(1) In review data, we filtered out data from 2012 to 2014, and only selected “asin”, “reviewText” and “summary”. Then, we grouped by “asin” and then concatenate the reviews and summaries.

(2) In metadata, since the book ID should be distinct, we deleted the duplicates. Then we selected “asin”, “description” and “title”.

(3) After the review data and metadata are cleaned, we inner-joined the two tables on “asin”.

(4) For each book, we got its review text, the summary of review, description, and title. We then concatenated them into one text, which becomes the representative text (reviewText) of the book.

```
# Final data set ready for modeling!!
dff.show(5)
```

```
+----------+--------------------+--------------------+---+
|      asin|               title|          reviewText| id|
+----------+--------------------+--------------------+---+
|0670869961|     The Pasta Bible|This is a wonderf...|  0|
|0671019880|Upon a Midnight C...|This is a book fi...|  1|
|0671617478|           Red Baker|I'm not sure how ...|  2|
|0671732188|Bestial: The Sava...|Interesting how t...|  3|
|0671736450|The Way of Energy...|I can't remember ...|  4|
+----------+--------------------+--------------------+---+
only showing top 5 rows
```

(5) Once we get the “reviewText” of each book, we can generate feature vectors based on the text using TFIDF, and then use the feature vectors to cluster a large number of books.

- In our case, TFIDF is the feature vectorization method to reflect the importance of a term to a reviewText in the whole collection of reviewText.
- In Spark MLlib, TF and IDF are implemented in HashingTF and IDF.
- We tokenized the text and filtered out stop words. Then we applied HashingTF and IDF separately to get sparse feature vectors. Each feature vector is corresponding to each book. These feature vectors will serve as the input of clustering algorithms directly.

Clustering

Having the feature vectors resulting from TFIDF transformer, now we will apply a clustering algorithm to get a more precise idea of the grouping topics for the most similar books. To this end, we will try two different machine learning algorithms for clustering K-means and Latent Dirichlet Allocation (LDA).
Both are unsupervised learning algorithms where we need to decide a priori value for K which is the number of the resulting clusters.

<table>
<tr><th colspan="2">K-means</th></tr>
<tr><th>K=10</th><th>K=6</th></tr>
<tr><td>

prediction	count
1	208
6	1367
3	79
5	582
9	42
4	17
8	13413
7	11
2	3951
0	60931

</td><td>

prediction	count
1	192
3	20
5	11848
4	3007
2	893
0	64641

</td></tr>
</table>

<table>
<tr><th colspan="2">LDA</th></tr>
<tr><th>K=10</th><th>K=6</th></tr>
<tr><td>

topic	count
0	973
1	809
2	1178
3	871
4	1046
5	1009
6	1297
7	733
8	878
9	632

</td><td>

topic	count
0	906
1	819
2	1171
3	869
4	1046
5	1066

</td></tr>
</table>

Based on our experiments, we got two different distribution for the books among different categories. This is reasonable when considering the difference between the two algorithms; K-means is a hard clustering one, i.e, a given book belongs to only one topic, when, LDA is a soft clustering meaning that it calculates the probabilities of a given book belonging to each topic. Considering the results we got, there are three main topics related to the K-means algorithm. While on the other hand, we got a more balanced distribution of the different categories. Also, LDA is computationally more expensive.

Lessons Learned

- If we want to run a project with a large dataset, a good choice is to use distributed computing with Spark and a flexible database that manages large data.
- To build a stable MongoDB system, we need to be very careful about the configure files in each shard and replication. Every time an error appears, we need to first check which shard or configure set is not working with “status: 1”. Once we find the problemed set of instances, we need to troubleshoot the crashed instance and restart its **mongod process** or edit its configure files then reconfigure it. We also need to **monitor the resources used** on MongoDB servers to make sure that the database would not breakdown (MongoDB is still fragile). If we find the transfer speed is too slow or the disk is full, we need to use the same method to add another shard. Another important note is that the **index** of data should be created in advance to speed up the data transferring computing.
- We tested the speed of fetching data from S3 and from MongoDB, it turns out that the structure of MongoDB significantly increases the speed of data transfer.
- It’s good to learn about how to set up the environment and Jupyter notebook on EC2. We would use it in our future group work.

Reference

https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html
https://github.com/LenzDu/Smart-Meter
https://aws.amazon.com/emr/
http://ijcsit.com/docs/Volume%206/vol6issue06/ijcsit2015060652.pdf