S17-18645 How to Write Fast Code
Term Project Report - Team009
Recommendation System of Contemporary Music

Team member: Jianfeng Guo(jguo3), Chun-Liang Kuo(chunliak), Chien-Hua Wang(chienhuw)

Part 1: Overview

Music lovers should be familiar with the song list by choosing and adding a song one by one. To help users more efficiently find out their favorite from tons of songs, we're going to implement the music recommendation system. The content-based recommendation can share with users the featured music and save users lots of time on searching the song.

We use Million Song Data Dataset (https://labrosa.ee.columbia.edu/millionsong/) as our raw data set, which contains millions of information on not only the cover, lyrics, tags but also users who follow or purchase these songs. Our main achievement is building a simple web service architecture shown in the following picture:

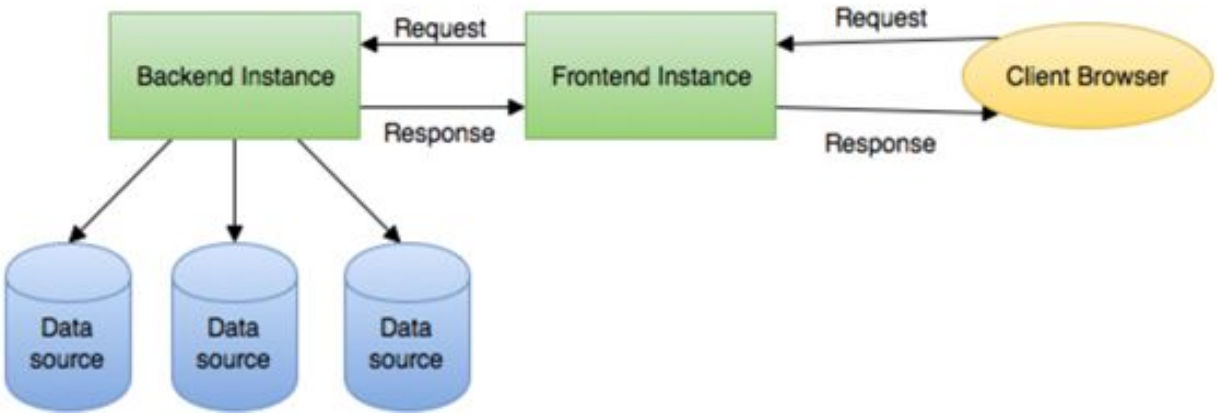

Figure 1. Web service architecture

Where the frontend instance receives the client-side request, parse them and delivers these requests to our backend instance. Backend instance receiver these requests and do the logic we want, fetching data that we have cleaned and loaded from data store and responding back to the frontend.

Part 2: Literature Survey

Recommendation system or recommendation engine is part of information filtering system. The information filtering is to extract useful information from large amount of data by computer, and guide to what users need through classification and featuring. Recommendation system model consists of both user and item related information. The information of users can be gender, age, race and other personal information. The information of item can be the content description, price and weight...etc. We call these information metadata. Besides, the recommendation system also records the transaction and interaction between user and item. For instance, user A bought a ticket for movie B, or user A ranked product B. After the system got the metadata and related interaction, it can analyze further. There are mainly two implementation of the recommendation system: content-based approach and collaborative filtering approach.

Content-based Approach

Content-based approach is classification for the metadata of user and item individually, and find the most similar item from corresponding metadata then recommend it. The calculation of similarity is according to metadata of item and uses different methods to compute distance. The purpose is to find the most similar k items, which is called k nearest neighbor(KNN).
Content-based approach only needs to consider the metadata of items, so it is easier to implement. But when the amount of item is large, the calculation of KNN would be time consuming.

Collaborative Filtering Approach

Compare to content-based approach, collaborative filtering approach takes more care about the transaction and interaction between users and items, and analyze these data to recommend proper items to users. We can use a matrix to represent the interaction data of users and items:

	Item1	Item2	Item3	Item4
User1				
User2				
User3				
User4				

This is User/Item matrix. Each cell represent the interaction of corresponding user and item. The value could be the rank the users gave, or the number of interaction...etc.
When the scale becomes large, the amount of user and item would be very big. In this time, the User/Item matrix should be a sparse matrix. It means most of the content of the cell is meaningless or empty of information, and these meaningless cells need to be processed by some specific methods.

MapReduce

In recent society, the data on internet becomes larger and larger. Many storage of websites arrives petabyte unit. To deal with such data, the issue is called large-data problem. In 2004, Google developed MapReduce technique into web search engine. Then Apache Hadoop leveraged from Google and proposed a framework with MapReduce programming model. MapReduce is a model for distributed data process parallelization. It distributes a large-data problem into the nodes of cluster. There are two processes of MapReduce: Map and Reduce. Their inputs and outputs are key-value pairs. In Hadoop MapReduce framework, the class of key and value can be decided by programmer according to their data type, and programmer also needs to configure Map and Reduce function to complete the work.

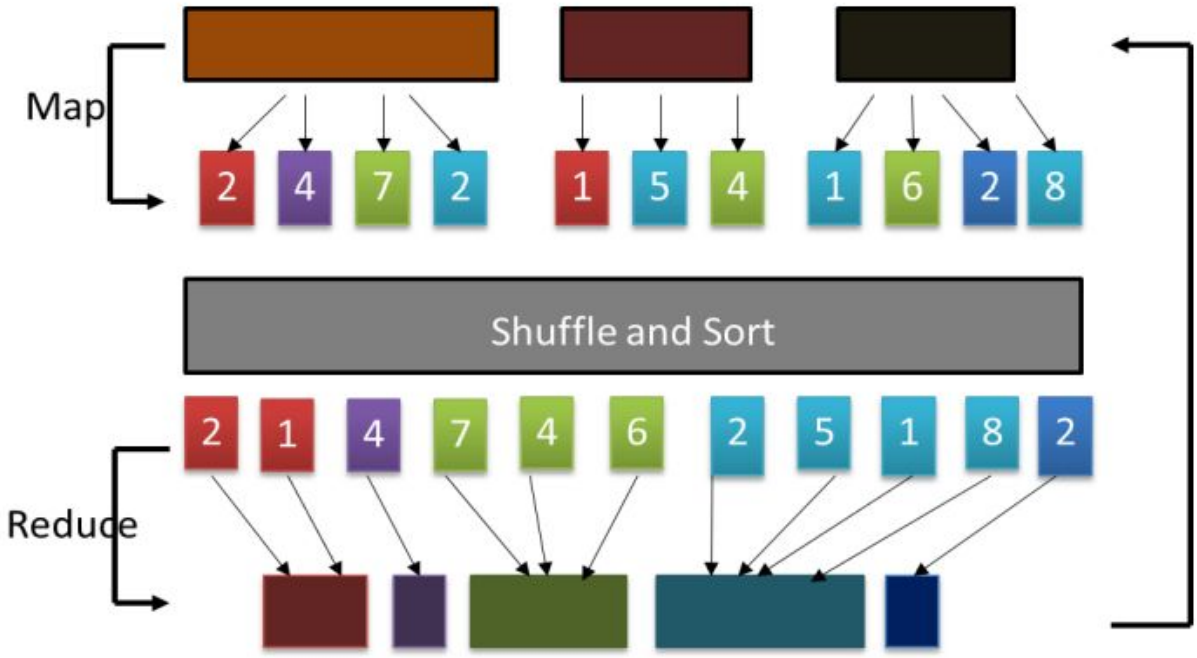

Figure 2. MapReduce Concept Illustration

Principle component Analysis (PCA):

Principal components analysis is a process of recognizing a smaller amount of uncorrelated variables, called "principal components", from a large set of data. The purpose of principal components analysis is to clarify the maximum amount of variance with the fewest amount of principal components. Principal components analysis is generally used in the market research, social science, and other industries that use large data sets. Principal components analysis is usually used as one step in a series of analyses. Principal components can be used in analysis to reduce the number of variables and avoid multi-collinearity.

Apache Spark:

Apache Spark is a open-sourced clustering calculation framework, which is developed by AMPLab from UC Berkeley. Spark is a elastic calculation framework and good at process data flow by Spark Streaming, interactive analysis by Spark SQL and machine learning by ML Lib...etc. Hence, Spark can play as a platform of general-purpose big data processing. Spark allows users to store data into cluster memory and repeat the calculation frequently, so it fits well in machine learning algorithms. Compare to Hadoop MapReduce, Hadoop needs to store the medium data into hard disk. However, the hard disk IO is always the bottleneck of efficiency. For Spark, it stores the medium data in memory and speed up the process when the repeat times become large.

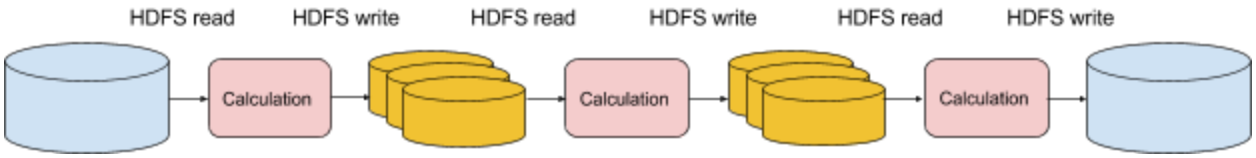

Figure 3. Hadoop MapReduce Framework

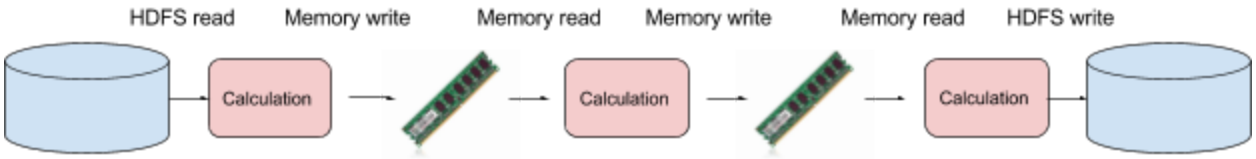

Figure 4. Spark In-Memory MapReduce Framework

Part 3: Detailed Description of Techniques

Cleaning and formatting data source:

The SHS dataset has a total of 18,196 tracks from the MSD, organized in "cliques", i.e. groups of versions of a single underlying musical work. Where possible, the cliques are referenced to "works" from the SHS site. We use MapReduce to do the formatting and cleaning (since all record processing is a one-way trip). For example, when dealing with JSON string like

{“genreid”: TRVMOOV128F92E4D89,”author”:”SFX - Cars Specific; Lamborghini”, “styles”:[game, rock, ...], followers: [xxx1,xxx2,xxx3]}

we should get a corresponding line like this: (use space instead of \t for convenience)

TRVMOOV128F92E4D89 SFX - Cars Specific; Lamborghini game#rock#... xxx1#xxx2#xxx3...

And use the following command to load our data to Hbase:

echo "create 'text_data', 'data'" | hbase shell

Problems encountered:

1. There are a lot of bad input data that are not strictly abide the JSON format, and some part may be empty or include some special characters.
2. Moreover, we encounter a lot of duplicated data with minor difference, too.

How to solve: For bad input, we encapsulate our cleaning program with a lot of try{}catch{}final{} clause. For duplication, we do not find a good solution so we just keep them in out tsv file.

Building recommendation system:

After we get our data store, we divide our data set into three parts:

1. Train Set (40%)
2. Cross Validation Set (20%)
3. Test Set (40%)

These three sets are corresponding to the three data stores in the Figure 1., where the train set is response to training our recommendation system, cross validation set is to validate our result and let us ensure our correctness. The test set is the final set for us to do the final testing on our recommendation system and make our system fully works. We use Spark framework and Scala as our primary language to implement our collaborative filtering algorithms, and the training process would not stop until we find convergence of our recommended results, i.e., for a specific input data we tried several times, the output should not be too different from each other.

Problems encountered:

1. Always get similar results regardless of input.
2. Output cannot converge with we feed the same input into system.

How to solve:

1. The reason for our first problem is that we choose the first 20% as our train set. And the data source also do some sorting on this data set, too. So we got a lot record that has a strong preference on some kinds of music while ignoring the others. So instead of use “head” command, we should randomly choose the data in data source. We just do a trivial MapReduce job which partition the whole dataset into 79 partitions (part-r-00000 --- part-r-000078) and set a random integer generator to generate a random offset of our partition file.
2. The problem is partially solved by extending our train set from 20% to 40%. After doing this, the mainstream of the dataset (also the most popular songs) can be

steadily outputted. But when dealing with some corner cases, such as a non-popular artist or a strange combination of style (rock and peace, for instance), we still cannot get a convergence result.

One thing we should mention is that we don't strictly following the standard process of dealing with machine learning problem due to limited time budget. We should also draw the learning curve of our system as well as do ceiling analysis, too. We are quite sure learning curve and ceiling analysis can give us a better understanding, but we do not get familiar with these functionalities under Spark framework.

Part4: Analysis of Results

Our system structure is as following:

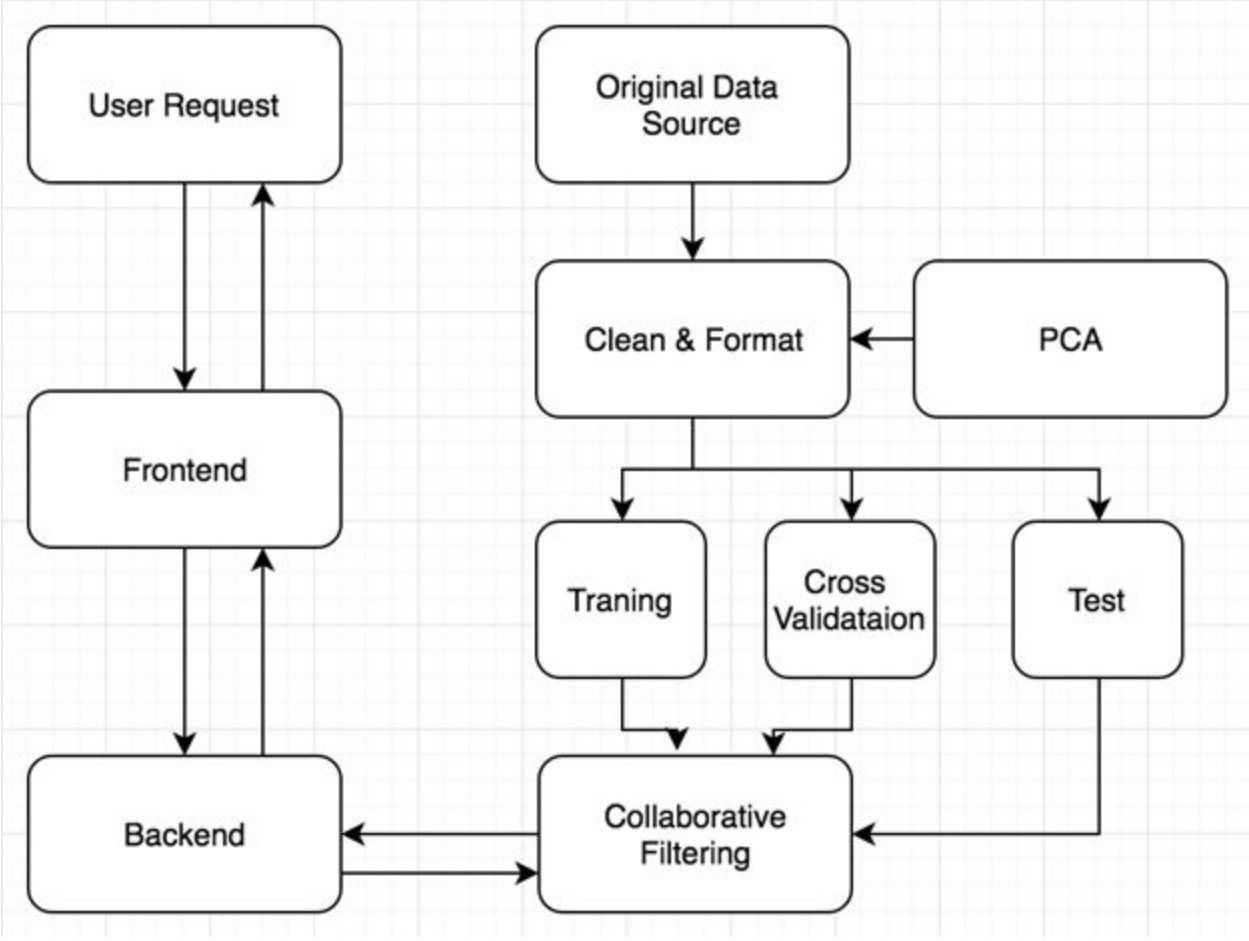

Figure 5. System structure

We use "p1=rock & p2=game" as our example here. The following two charts are the results of our testing as frontend input. The first column is genre identifier while the

second column is corresponding artist id. We list two example output here, and we can see the second commendation result is not the same as first one.

GENREID(VARCHAR)	ARTISTID(VARCHAR)
TRLBEHJ128F428F46D	ARGI5FG1187B9A1A70
IRRCLIK128F935DDD5	CK15DJQ1187FB5910C
KLKHLFK128F4229CCF	BWNQ7UN1187B98EE77

GENREID(VARCHAR)	ARTISTID(VARCHAR)
TRLBEHJ128F428F46D	ARGI5FG1187B9A1A70
DFFDULT128F4241B6C	MPBBHWW1187B9B312A
IRRCLIK128F935DDD5	CK15DJQ1187FB5910C

The randomness is due to our random initialization of our filtering matrix, which should not be all-zero. However, this randomness doesn't affect our system performance very seriously, and the result converge in the whole sense.

Also, in order to make sure our recommendation is reasonable, we list some of top albums to compare with our results, too. We believe that our recommendation results should not be too far away from the top-rating album if it is correct. In the example above, we list both top 8 rock albums and top 8 game albums. The chart below are top 8 game albums.

GENREID(VARCHAR)	ARTISTID(VARCHAR)	PREFERENCE(INTEGER)
TRTWBKF12903CD96B6	SR9JSH21187FB4C028	783868
GRAWTGT128F146ECF1	SLT50HO1187FB4D0ED	743123
UHZSMPB128F4291BDB	FVSZWK21187B9B26D7	732603
JQIQRUU128F931F86F TRTPFNO128F92FF4C5	KMTJECM1187B9930BA ARGHPIE1187FB3CCDF	730276 685404

CNATROO128F42544AB	QRZHZ8X1187B9A5C1B	632101
TRLBEHJ128F428F46D	ARGI5FG1187B9A1A70	615384
BEUHIYM128F424D618	RMI18NF1187B99FC57	592029

As illustrated above, there is an overlapping item TRTPFNO128F92FF4C5 which is the first recommended item of our two testing. Since there is no correct answer on this question and we got one that is fairly popular, we consider our result should be correct.

But there is no overlap item on rock album list even we extend the N from 8 to 50. As we expect, the top rock album preference is much larger than game album (around 5 million about) so it is more suitable to recognize “game” as our main feature.

Part 5: Conclusion

In this project, we implemented a music recommendation system by Apache Spark from designing the algorithm, establishing the architecture, and finally get the recommendation results. The algorithm that the recommendation system uses is a user-based collaborative filtering algorithm. The recommendation system can deal with large amount of data as Apache Spark is a big data handling technology that has built-in properties for processing iterative and interactive algorithms, and also faster than traditional Hadoop MapReduce architecture. Apache Spark not only helped the implementation of this project, but also as a usage of machine learning algorithms. Spark has a concept of distributed memory abstraction, which helps reduce disk writes and promotes the in memory data processing. This project makes use of the distributed memory abstraction layer of Apache Spark to help implement the machine learning algorithms for recommendation system and to help process large amount of data in relatively short time. This project could be further developed by making use of more music features for the recommendation process. Different query dependent features can be implemented to help the recommendation system to be more precise and accurate.