Project Introduction

Group member: Yang xi, Chen Wang

1. The motivation

Our project is a parallelized movie recommendation system base on big dataset. Nowadays, people have increasingly higher requirements for movies and each person has its personal preference. Many excellent machine-learning algorithms could solve this issue properly. According to a certain amount of training data, the algorithm is able to give a reliable recommendation to different people.

However, the classification accuracy is highly affected by how much training data we have. To ensure an ideal recommendation result, we normally need to process big size of data. In this case, parallel computing is a very powerful and effective way for us to enhance the efficiency.

In our project, a parallelized recommendation system based on Hadoop and Hive is proposed. We are responsible for the whole design and implementation of the architecture and algorithms in this system. When we have a new user's personal information like gender, age and occupation, many suitable movies would be output. More importantly, both training and testing processes are parallelized and therefore a much better efficiency can be achieved.

2. Data description

Our test data contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. Basically there are three files and they have the following information respectively:

ratings.dat	UserID::MovieID::Rating::Timestamp
users.dat	UserID::Gender::Age::Occupation::Zip-code
movies.dat	MovieID::Title::Genres

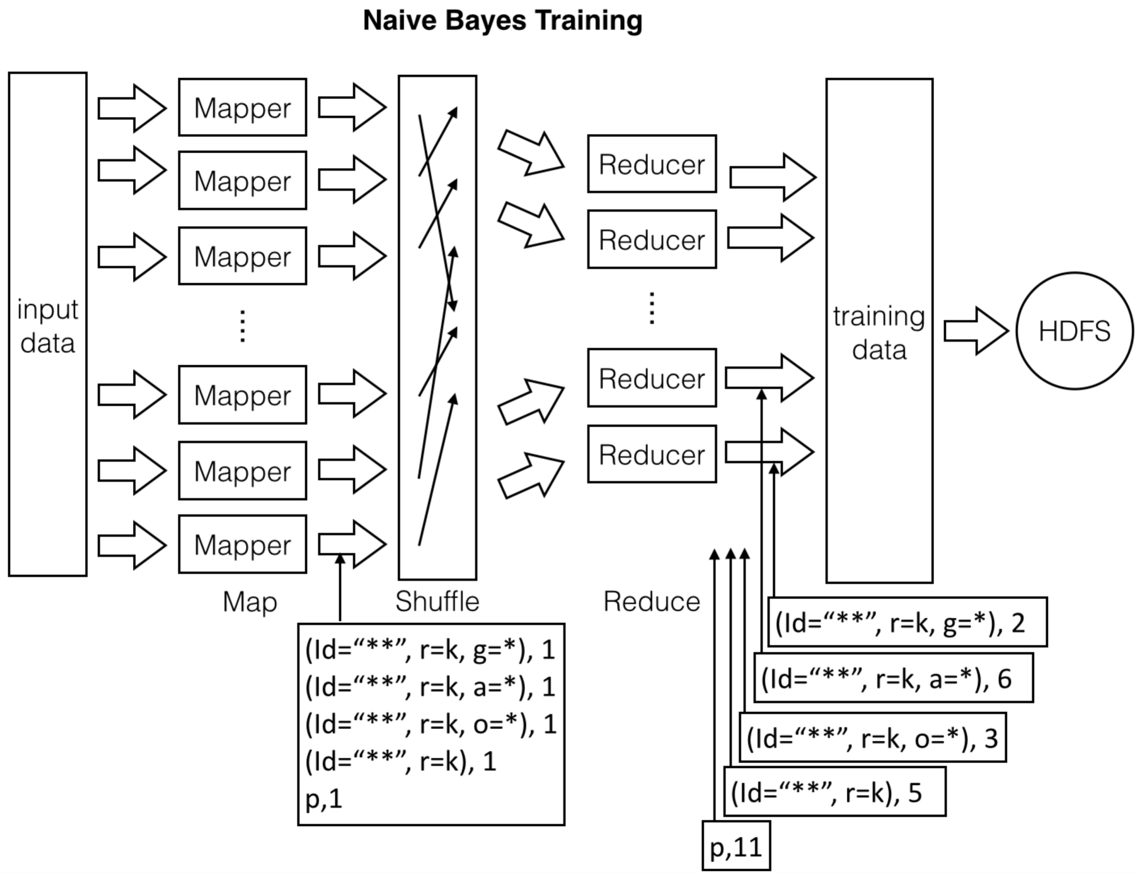
More detailed information about the data can be found here:
http://grouplens.org/datasets/movielens/

3. Project Description

First, we extract data from the raw data set through Hive. Obviously, these three files are quite similar to three relational tables, and therefore we want to extract the useful data by Hive. To implement the recommendation, we only need the information from ratings.dat and users.dat. We want to join these two tables in Hive, which is a distributed data warehouse built on top of Hadoop and the query is executed via MapReduce.

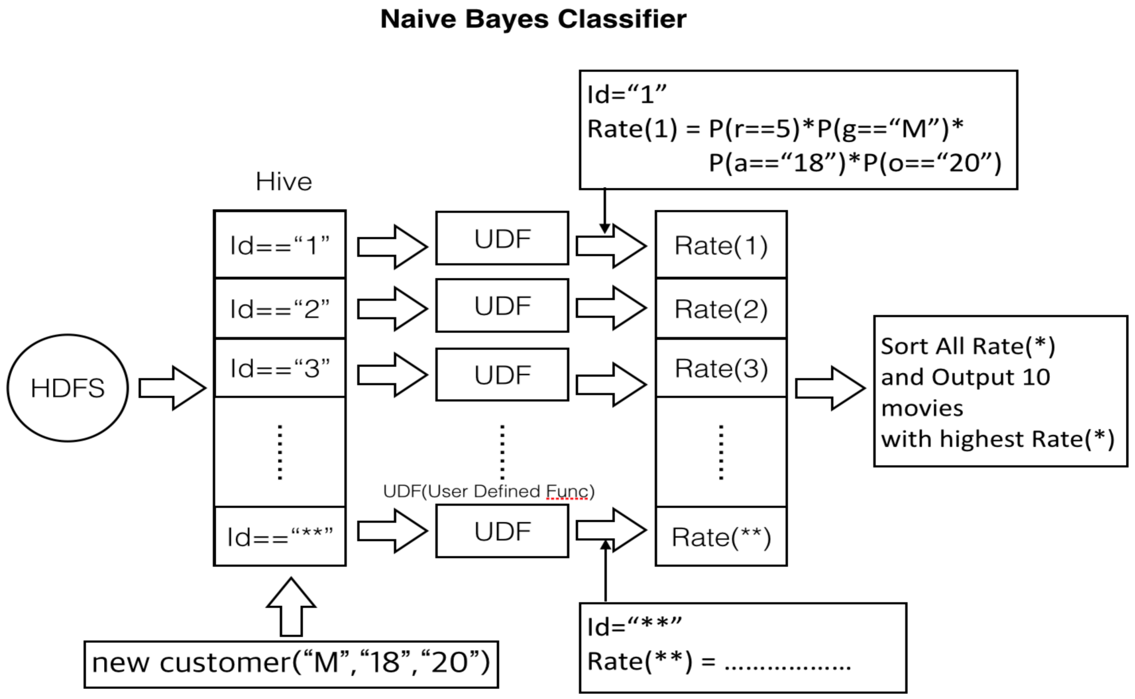
After that, by applying Naive Bayes, for each movie (distinguished by "Id"), we have to estimate it's $P(g = G_i | r = k), P(a = A_i | r = k), P(o = O_i | r = k)$ and $P(r)$ from the training data. Set G_i stands for gender ranging from "Male" and "Female". Set A_i stands for age ranging from "1,18,20…65". Set O_i stands for occupation ranging from "0" to "20". Set r stands for the rate of movie ranging from "1" to "5".

In order to do so, we need to sum over, $g = G_i, a = A_i, o = O_i$ for each r label in the training data to calculate $P(g = G_i | r = k), P(a = A_i | r = k), P(o = O_i | r = k)$. We specify different sets of mappers to calculate them, which is showed in the figure. Then the reducer sums up intermediate results to get the final result for the parameters.



After calculate all the possibilities by reducer, the data will be stored into HDFS, and then loaded into Hive for further Naïve Bayes classifier to read operations.

When a new customer comes in with his/her personal information, we can query the Hive database by parallelized Mapreduce. To utilize the query result and decide which movie fits user's preference best, we defined a UDF (User-Defined-Function) to calculate the probability about the user would like the movie. This is one of most powerful function of Hive, which allows us to implement their own business logic in a java interface. This UDF could be called when we do the query and take the query result as parameters. More importantly, this process is also executed via Hadoop Mapreduce!



4. What did we find?

According to our own implementation and thinking, we realize that Hive is a very handy and powerful tool to solve this type of problems. Apparently, the testing result is much faster than the serial one. Actually if we consider to solve this problem in a practical perspective, we want to rely more on Hive and its UDF. However, Hive is a highly packaged tool for parallel computing. Although it takes time to configure and learn, it has no need for user to write many MapReduce code. Therefore, for the project purpose, we isolate much of our work to implement in Hadoop MapReduce. Hadhoop also has issues for use. For example, it is hard to update a global variable in reducers and we designed a two-stage MapReduce code to solve this.