Realtime Recommendation System using NetFlix Competition Dataset

Fernando Rocha
fmrocha@student.dei.uc.pt
2012131752
LEI

Luis Neves
luisn@student.dei.uc.pt
2012140908
MEI

***Abstract*—Nowadays, Recommendation Systems are a very important part in most of the entertainment product retailers, such as Netflix, Amazon and even Steam. These systems improve the user interaction with the services delivered by this kind of companies. With this in mind, Netflix started an open competition in 2006 that consisted in letting programmers, engineers and anyone alike to come up with the best filtering algorithm to predict user's ratings to films. Our project consists in taking this Netflix competition Dataset and develop both a Standalone and a Scalable Parallel application that would fall into this category.**

I. Introduction

Recommendation systems are software tools and techniques that seek to predict the 'rating' or 'preference' that a user would give to an item, providing them suggestions for what they would like or give use to[1]. This means that the ultimate goal of Recommendation Systems is to improve the quality of the User Experience in, mostly, entertainment platforms as NetFlix, Last.FM, Steam, YouTube but also in platforms like Amazon, eBay and so on. This type of systems is a useful alternative to search algorithms since they help the user finding and suggest things without him even know they exist.

Recommendation systems are an active research topic in Data Mining[2] and Machine Learning fields since they are systems that work with a large number of data inputs, leading into Scalability and Performance problems, which are addressed by these fields.

In this project, we will use the Netflix Competition Dataset as our main source to conceive, build and benchmark a Realtime Recommendation System as a standalone and as a scalable parallel application, test both of them and compare their results.

This paper is composed by 4 sections, besides this introduction:

1) Related work: in this section we discuss in more detail the context of our project using referenced materials from other authors and why this project is relevant to the matters at hand.
2) Description of the System: in this section we describe the details of our system and it's implementation, explaining the planning process, data transformation and further implementation of the solution.
3) Experimental Setup: here we describe our experiment was set up so that others can understand and replicate the process implemented by us.
4) Results and Analysis: this section is the main chapter of the paper, in which we show our results, discuss them and show with concrete proofs why we feel this project is important to the matter at hand.

II. Related Work

As related work to understand the essence of this project, our research consisted on 3 major topics: Recommendation Systems (what they are, what kind of algorithms do they use, where they are most used and what for), Spark's Recommendation Algorithm (Spark's Library of Machine Learning Algorithms) and, ultimately, the Netflix Prize Competition.

A. Recommendation Systems

Recommendation systems have been a independent research area since the mid-1990's but, in recent years, the interest in this field has increased not only because of entertainment platforms, as stated before, but also because new conferences and workshops related to it appeared. Maybe the most important one is ACM Recommender Systems (RecSys) as it is the premmier anual event in recommender technology research and applications. Furthermore, there have been several special issues in academic journals covering research and developments in this field. Some of these are referenced in this paper as part of our research to develop this project.

These systems provide recommendations to the users using two kinds of techniques (or a combination of them): Collaborative Filtering and Content-Based Filtering.

1) Collaborative Filtering: Collaborative Filtering is a technique that analyzes relationships between users and interdependencies among items to create new user-item associations[3]. The simplest implementation of this technique is calculating the similarity in the user's rating histories, this way, this technique is referenced as "people-to-people correlation"[1].

However, there are two primary areas of this approach: neighborhood methods and latent factor models. Neighborhood methods implement the word-of-mouth principle, where items are recommended for the user from the opinion of people with the same taste, according to his preferences[4]. Latent

factor models try to understand user's ratings by characterizing both items and users on factors inferred from the ratings patterns. Maybe the most conventional method to implement this kind of models is Matrix Factorization[3].

The development of our project will be based on this technique since Spark's Library of Machine Learning Algorithms has it implemented, using the Alternating Least Squares (ALS) algorithm. ALS algorithm is very useful when we want to parallelize the data computation and analysis[3][5], and that is exactly what we want.

2) Matrix Factorization with ALS algorithm: As stated before, Matrix Factorization characterizes both users and items by vectors of factors inferred from item rating patterns. High correspondence between user and item factors results in a recommendation[3].

Recommendation systems rely on input data, commonly placed in a matrix where one dimension represents users and the other items. This has a significant problem, the users normally just rate a small percentage of the possible items. This is where Matrix Factorization enters, incorporating additional information such as users search and purchase histories[10]. This technique associates items with a vector and users with another. For a given item, the respective vector measures the extent which the items possess those factors. For a given user, the respective vector measures the extent of interest that the user has on items with a high correspondence with those factors. The resulting dot product between the vectors gives us an approximate user rating of said item[3].

This works well when we have all the information about items and users when that is not the case we need additional information and the ALS algorithm.

When we need information that is lacking, ALS algorithm is used to predict the correlation between items and users and their respective vectors, rotating between fixing the items vector and the corresponding users vector. When one of the vectors is fully fixed, the system recomputes the other[3].

This solution has a huge impact, particularly when we want the parallelize this Matrix Factorization process since it computes each element from each vector independently of the other factors[5].

3) Content-based Filtering: Opposed to Collaborative Filtering, there is a technique that focuses more on the characteristics of the items instead of evaluating and analyzing relationships and common interests between users, this technique is called Content-Based Filtering.

With this technique, recommendations are made by assigning particular characteristics to items such as genres, actors, directors (if we are talking about movies) or musicians, bands, collaborations (speaking of music). Thus, the system recommends items to the user by analyzing its characteristics, instead of seeing what users alike enjoy.

B. Spark's Library of Machine Learning Algorithms

Spark's Library of Machine Learning Algorithms (MLlib) is a library from Apache Spark's which goal is to make practical machine learning algorithms easily scalable[6].

This library is particularly suitable for our project because makes us focus on the Data Transformation and Computation and System Scalability instead of having to learn how to implement this kind of filtering algorithms.

Mllib uses Collaborative Filtering as its primary technique for Recommendation Systems implementation, using Matrix Factorization with ALS algorithm for optimization, which we explained before[10].

C. NetFlix Prize

Netflix Prize, as stated in the Introduction, was an open competition with the goal of finding a recommendation algorithm that could improve NetFlix's recommendation system by 10%.

This contest went on for about three years and finished when Team BellKor's Pragmatic Chaos[7][8][9] got that achievement, dropping the Root Mean Square Error (RMSE) of the Netflix Cinematch system from 0.9525 to 0.8527.

The winning algorithm is a join of algorithms from three different contestants (Team BellKor[7], Team BigChaos[8] and Team Pragmatic Theory[9]). This approach makes use of Collaborative Filtering but with some twists[7]: First of all, Collaborative Filtering tries to capture the interactions between items and users but, instead of exploring this concept from the beginning, this approach starts by identifying some baseline predictors, or effects which do not involve user-item interactions.

The first baseline predictor identified was time-based, this means that the rating that on user gives to an item may vary in function of time and also that the popularity of one item varies based on time, and the ratings depend on that.

The second one was the frequency in which users rate items, more particularly, the number of ratings that one user on a specific day.

After this baseline predictors identification, the authors used matrix factorization as stated before but used the stochastic gradient descent instead of the ALS algorithm explained earlier in the paper. Stochastic gradient descent algorithm consists of predicting one user rating for a given item, compute the associated prediction error and then modify the matrix's parameters based on that[3].

Then, they incorporated a neighborhood model with temporal dynamics defined by the first baseline predictor. This model was used to, particularly, prove that the interaction between items and their ratings has a significant impact on the system's accuracy.

Finishing all this process, the authors to Gradient Boosted Decision Trees to blend all of these techniques in one algorithm. Gradient Boosted Decision Trees are a method for generating regression models, giving less effective methods shallow decision trees with few leaf nodes.

Our objective for this project is not to improve the winning algorithm but to build and benchmark a scalable solution that, hopefully, can give us good results.

III. Algorithm and System Sketch

This section describes in detail our system, and the way it works. It is divided into two main parts, describing the approach of the problem in a non-scalable and scalable way. Even though, before explaining how the system works, we will describe the dataset used and our approach to the problem.

A. Dataset

Before describing the system, we must first detail the datased used, which as referred, is the one constructed to support the participants of the Netflix Pize.

The files contain over 100 million ratings from 480 thousand Netflix customers over 17 thousand movie titles. The ratings are on a scale from 1 to 5.

The dataset is divided in three main parts, the training, qualifying and probe datasets.

The training dataset contains the files of each movie, with the respective ratings given by the users in each line. The format used by this dataset is "MovieID, UserID, Rating, Date". To notice that this isn't the original format provided for the Netflix challenge. The dataset was modified to this format by us, for a easier loading and parsing of the data.

The qualifying dataset consists of a list of movies, having each of then a list of users. This is the dataset used for prediction during the contest, and as such, these were the rating they were supposed to predict (based on the information in the training dataset). To our work, this dataset won't be needed, as we have no way to calculate how accurate our predictions were (the verification was only available online during the contest time).

Lastly, we have at our disposal a probe dataset, which follows the format of the qualifying dataset, only having movie/user pairs from the training dataset that already have a rating, so that we are able to check if our predictions are close to the real ones.This is will be dataset that we're going to use to test our prediction's RMSE with the correct ratings taken from the training dataset. Another thing to consider is that, after he collected the true ratings for each pair on the probe dataset, we needed to delete the respective lines from the training dataset so that the training of our prediction model does not get influenced by those values.

Since the dataset is already divided, we don't neet to split the training set ourselves, which would make different datasets and, therefore, disable us to compare our system's RMSE with the Cinematch one, stated in the Netflix Prize FAQ[12] as 0.9474.

B. Problem approach

As stated through the introduction and related work sections, our focus will be on using collaborative filtering as the main recommendation system approach for the rating predictions.

As such, we will use spark machine learning library, which uses the alternating least squares algorithm also referred before.

Overall, our project is going to be addressed as a Machine Learning problem so, we will require to treat it with a typical Machine Learning Workflow, as shown in Figure 1[11].

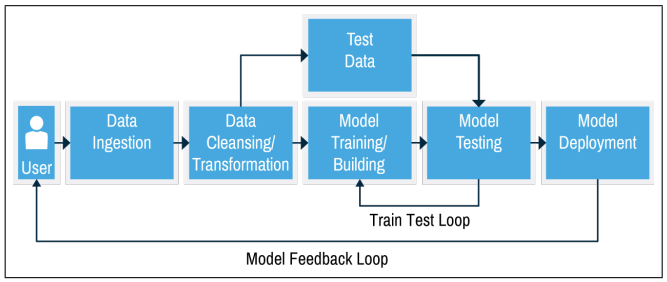

Fig. 1. Machine Learning Workflow

We will start by doing small transformations on the data, followed by the importing of the training data to resilient distributed datasets, followed by a transformation of the data, so that we can use it further on.

The data will then be used in the ALS training method, so that we can tune its parameters, which are the following:

- *numBlocks* - Number of blocks the users and movies will be partitioned into, in order to parallelize computation;
- *rank* - Number of latent factors in the model;
- *maxIter* - Maximum number of iterations;
- *regParam* - Regularization parameter in ALS;
- *implicitPrefs* - Specifies whether to use an *explicit feedback* variant or an *implicit feedback* one. The standard approach collaborative filtering using matrix factorization treats the entries in the matrix as *explicit* preferences given by the user, even though there are scenarios where that doesn't happen (e.g. page views, likes, etc.);
- *alpha* - Governs the *baseline* confidence in preference observations, applying to the *implicit* feedback variant;
- *nonnegative* - specifies the use of nonnegative constraints.

After the training, we calculate the predictions for the probe dataset (using the data without the ratings), calling the *predictAll* method using the trained model.

We then evaluate the results by computing the RMSE (root-mean square error) comparing the predictions with the probe dataset (now with the ratings). Varying the parameters will allow us to obtain different RMSE values, and to tune the results to avoid high error or overfitting. This is what is referenced in Figure 1 as *Train Test Loop*.

Finishing the development of the system, we will evaluate the results with the complete training set. After that we can test our system by adding some ratings from a new user and check if the system returns recommendations that fit into the user's profile.

IV. Non-scalable Solution

In order to approach the problem in a way that it doesn't have the ability to scale according to the size of the data being processed, we developed a solution and executed it in a single machine.

The machine referred has the following hardware and software specifications:

- MacBook Pro 13"Late 2011
- OS X 10.11.3 El Capitan
- Intel(R) Core(TM) i5 2.4GHz
- 4GB RAM
- Samsung 850 EVO 250GB SSD
- PySpark with Python 2.7.10 over Spark 1.6.0 without Hadoop

As this is a non-scalable standalone solution, we had the need to create a subset of the training dataset **using only 1500 of a total of 17770 movies** because the original dataset has a tremendous amount of data for one machine to process at a time, with the computer easily running out of memory while trying to train our prediction model.To consider that for the probe dataset we are only testing entries with MovieID between 1 and 1500.

Testing the training dataset with variable length subsets, we came to the realization that 1500 was a good number to test our system since 2000 movies made the system performance greatly drop and, therefore, would make us wait a long time to have conclusive results.

Load training and probe datasets into RDD's
Map the files splitting the fields by commas
Parse the fields to the respective types
Define starting ALS parameters
while *parameters are in the defined interval* **do**
  Trains the model using ALS with given parameters
  Predicts the ratings for the probe dataset using the model
  Joins the results of the predictions and original ratings for comparison
  Calculates RMSE based on that information
end

Algorithm 1: Overview of the solution for the standalone approach

Even though we were not able to run the training algorithm on all the dataset available with the standalone version, as referred, we used the solution on a subset of the data. That way, we were able to validate our implementation as we wanted.

Using the algorithm represented before, we tuned our prediction model with a variable number of latent factors (rank), choosing after the one that would give us the best predictions, or in another terms, the minor RMSE.

In the table I, we can see the results obtained varying the rank between 4 and 11, first varying the results between 5,7,9 and 11 but, noticing that the rank 5 had the smallest error, we trained the model with rank 4 and 6, to see if 5 was the exact value for the perfect rank.

The Accuracy values were calculated by rounding every prediction made by the model using the most common **round half up** rule and checking if the result is equal to the correct rating given by the user.

TABLE I
DATA OBTAINED WHILE TESTING THE STANDALONE VERSION

Iterations	Rank	RMSE	Accuracy
5	4	1.29905429393	35,21%
	5	1.28427734966	35,30%
	6	1.31915064738	34,27%
	7	1.36161019417	33,18%
	9	1.41552104554	31,72%
	11	1.49141205203	30,03%

By analyzing the RMSE and accuracy of the results, we can see that the **rank 5** is, indeed, the most accurate, since it's RMSE is the smallest of the group, meaning it's the one with the predicted ratings closer to the real ones.

As an example, in the table II we can see a few lines of the joined RDD containing a pair of user/movie id's, and the respective real rating and predicted rating using the ALS model.

TABLE II
PREDICTIONS AND RATINGS FROM USER/MOVIE PAIRS

MovieID	UserID	Rating	Prediction
886	49878	2.0	4.6545185235787345
191	1073963	4.0	4.816105119493126
1110	2329131	5.0	3.370049416178089
201	1409827	3.0	1.4592385043613807
252	1671377	3.0	2.974869650268455
313	1254929	4.0	0.918098722450674
191	80014	5.0	4.40809046122685
708	440979	3.0	3.207069148651591
1470	2305834	3.0	4.248502944953375
831	173306	3.0	3.956370848727412

As expected from the RMSE and Accuracy values, the predictions and the real ratings still have a big discrepancy between them, he can only explain this by stating that we're using a pretty small training subset considering the totally of data that is available.

At this stage, we haven't yet developed the possibility for the user to rate some movies and then receive recommendations by the system. At this point, our main goal and priority is to have a effective and efficient prediction model that can give us "good" predictions. The user recommendations step will be developed later.

V. SCALABLE SOLUTION - WORK SO FAR

After the development of the standalone version, we focused on a scalable one. For this purpose, we used Hadoop to distribute our dataset through the available machines, and also Yarn, which is responsible for resource management in Hadoop.

We start by moving the dataset to Hadoop's hdfs, and proceed to adapt our current standalone version to be able to run in a cluster environment.

The next step concerns Spark's configurations tuning, since we must adapt the parameters to the available resources. The training dataset is big, so we must find the best parameters.

The main difference between the standalone version and the scalable one, in terms of code, ends up being the scope of

the spark context, which changes from *local* to *yarn-cluster*. After that change and after tuning up the cluster, the code is adapted in order to allow more parameter variations on the training algorithm (not only the *rank*)in order to analyze the obtained results.

A. Running tests

To test our scalable approach, we used the virtual machines given to us by DEI's Helpdesk, after all the installation and networking steps are finished (Those steps are described in the Install.txt file attached).

The virtual machines referred have the following hardware and software specifications:

- Ubuntu 14.04.3 LTS
- 2 VCores
- 4GB RAM
- Hadoop 2.6.4
- PySpark with Python 2.7.6 over Spark 1.6.0 using Hadoop Distributed File System and Yarn configurations for the cluster

We started by testing if our solution would have the capacity to process all of the original dataset with Spark's default settings. That didn't work out, and the process exited prematurely, accusing memory problems.

So, to prove that our scalable solution is, in fact, scalable, we decided to run the same dataset used in the standalone version, and check if there are performance improvements.

For the first ranks on the non-scalable version, we discovered that the approach took about 45 50 minutes to train ALS and compute the RMSE with that dataset, versus around 25 minutes for the scalable version has shown on Figure 2.

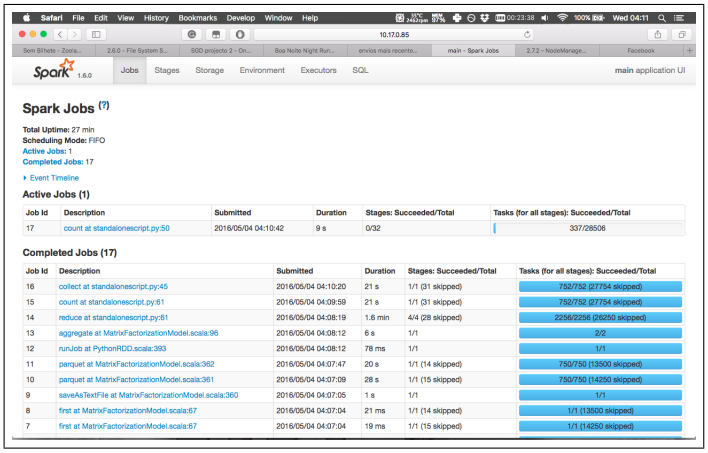

Fig. 2. Time elapsed for one training on scalable solution

So far, we haven't yet obtained any results for the complete dataset since all variable parameters given to Spark like *executor-memory*, *num-executors* and *executor-cores* didn't work and had memory problems.

We are still studying if, even with two machines, is possible to process such a high quantity of data of the complete dataset. If it isn't possible, we will verify which is the limit dataset that won't make Spark not finish the task and analyse results according to that.

Later on, we hope that using the 3VM's with 32GB of RAM will enable us to process the entire dataset without having to worry about memory problems.

VI. Questions about the challenge

References

[1] Ricci, Francesco, Rokach, Lior, Shapira, Bracha: "Introduction to Recommender Systems Handbook", in:Recommender Systems Handbook, pp. 1-35, Springer (2011)

[2] Leskovec, Jure, Rajaraman, Anand, Ullman, Jeffrey D.: Mining of Massive Datasets, Chapter 9 - Recommendation Systems, pp. 317-352, Cambridge University Press

[3] Koren, Yehuda, Bell, Robert, Volinsky, Chris: Matrix Factorization Techniques for Recommender Systems, IEEE Computer, Vol.42, pp. 30-37 (2009)

[4] Desrosier, Christian, Karypis, George:A comprehensive survey of neighborhood-based recommendation methods, in: F. Ricci, L. Rokach, B. Shapira, P.B. Kantor (Eds.), Recommender Systems Handbook, pp. 107144., Springer (2011)

[5] Y. Zhou et al., Large-Scale Parallel Collaborative Filtering for the Netflix Prize, Proc. 4th Intl Conf. Algorithmic Aspects in Information and Management, LNCS 5034,pp. 337-348, Springer(2008)

[6] MLlib Spark's Documentation http://spark.apache.org/docs/latest/mllib-guide.html

[7] Koren Y(2009) The BellKor solution to the Netflix Grand Prize, http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[8] Tscher A, Jahrer M, Bell R(2009) The BigChaos solution to the Netflix Grand Prize, http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf

[9] Piotte M, Chabbert M(2009) The Pragmatic Theory solution to the Netflix Grand Prize, http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf

[10] Benjamin Fradet: Alternating least squares and collaborative filtering in spark.ml, http://benfradet.github.io/blog/2016/02/15/Alernating-least-squares-and-collaborative-filtering-in-spark.ml

[11] Scott, James A.: "Getting Started with Apache Spark - From Inception to Production", MapR Technologies (2015)

[12] Netflix Prize Frequently Asked Questions http://www.netflixprize.com/faq