Analysis of the common reccommendation systems with the common frameworks: Spark and Flink

Final Report for the BigData project

Mirko Morandi
University of Trento
176043
mirko.morandi@unitn.it

Zhiheng Xu
University of Trento
174222
zhiheng.xu@unin.it

ABSTRACT

In this paper we provide an extensive analysis of the actual state of the art of recommendation systems.
Collaborative Filtering is the current buzzword in the world of recommendations, came to notoriety after the Netflix Prize challenge. In this paper we aim to analyze the current implementations of two different algorithms used for Collaborative Filtering: **ALS** and **Stochastic Gradient Descent** in combination with the common frameworks `Spark` and `Flink`.

Keywords

Flink; Spark; CF; Collaborative Filtering; ALS; SGD; Scala

1. INTRODUCTION

Recommender systems are now trending due to the overwhelming availability of data. These systems have the ability to discover hidden relationships between users and items, and use these patterns to improve the user's taste prediction. Reserachers discovered a "neighbourhood" of users with a similar taste which can be revealed by their previous actions: both implicit and explicit. `Collaborative filtering` is by far the most common approach adapted also by some of the biggest companies in the IT sector such as: **Amazon**, **Facebook** and **Netflix**. Altough it's massive presence in the market `CF` is not the only approach available for a recommender system, but it is actually the successor of `Content-Based filtering`. The latter aims to profile a user searching the correlation with the item's peculiarity. By item peculiarity we refer to its implicit and explicit characteristics, for example a song's genre, subgenre, writer, composer, year of composition, beats per second etc. The problem with this approach lays in the difficult of retrieving all the necessary information, which sometimes are not even available or discloable. Furthermore with the raise of the BigData paradigm some frameworks started to grow from the accademic world to the Apache Foundation: **Flink** and **Spark**. Those frameworks can be seen an extension of the Hadoop ecosystem, and both of them have their own pros and contros which will be briefly analyzed further in this paper.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. COLLABORATIVE FILTERING

The paper is structured as follow: description in more details of `Collaborative Filtering` with it's problems, what are the most commong algorithms used with `CF` and a brief introduction to both **Flink** and **Spark**.

2.1 Collaborative Filtering Approaches

Collaborative Filtering can be subsenquently defined in two different approaches:

2.1.1 *Memory-based Content Filtering*

In `memory-based CF` uses user ratings to compute similarity between user and items and subsequently make a recommendation. Usually this approach involves "neighboring" algorithms such as **K-Nearest Neighbours** to build relationships between users. The similarity between two users is calculated using the **cosine similarity**.
Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.[4]

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{t}\mathbf{e}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{1}$$

The recommendation is made by finding the top K similar users and aggregate their user-item matrices to find the appropriate recommendation. The typical problem of this approach is the difficult with scaling when the data gets bigger. Due to the `BigData` paradigm expansion this approach has been deprecated favoring the following approach.

2.1.2 *Model-based Content Filtering*

The most common approach to `CF` is through the factorization of a very big and sparse matrix.[3] For example during the Netflix Prize at the participans were given a matrix of 8.5 billions of ratings, of which only 100 millions were non zero values. *Model-based CF* uses machine-learning and data mining algorithms to uncover the latent factor model between users and items to predict the missing ratings.**latent factor models** are hidden relationships between users and items hardly discoverable in the original data; usually they

Table 1: Example of a sparse user-item ratings matrix

items/users	U1	U2	U3	U4	U5
I1	5	3	-	1	3
I2	4	-	2	1	-
I3	2	2	-	5	-
I4	4	3	-	4	2
I5	-	5	5	4	5

may for example denote the quantity of action in a movie or the complexity of the characters.These vectors are then used to create the missing values in the user-items matrix.

Furthermore the model-based content filtering can be expanded in two distinct sections: *user and item based content filtering* depending on the priority given to the prediction.

2.2 Related problems

2.2.1 Cold start problem

Due to the nature of CF, the system needs a huge amount of data in order to produce a reliable prediction. But what happens if our system hasn't collected any or not enough information yet? This problem is called **cold start** and can be tackled with some advanced machine learning solutions called *active learning.*

2.2.2 Shilling Attacks

CF can be exploited to perturbate its prediction system with a technique called *shilling attacks.* This happens when the input system (e.g. ratings) are given in the correct way trying to alter the recommendations to the one favoured by the attacker. It has also been noticed that these kind of attacks affect more user-based CF algorithms insted of item-based.[5]

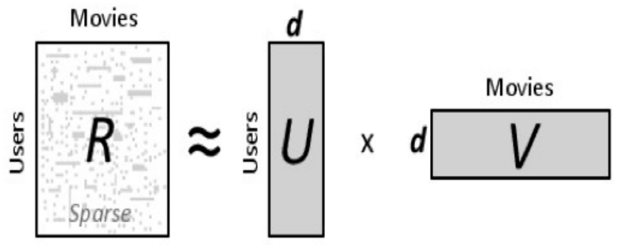
2.2.3 Sparsity

In the era of the *web 2.0* the matrices who compose the datasets are usually very sparse due to the typical proportion for which $nUsers \ll nItems$. As said previously the matrix which was given had only 100M ratings out of 8.5 billions of records. This problem is solved using matrix factorization.

3. MATRIX FACTORIZATION

During the last decade a huge effort has been applied to solve the problem of big datasets with an incredible amount of data. Let denote $R = r_{ij}$ denote a user-movie where each element r_{ij} represents a rating given by a user i to an item j with a value from 0 to 5; where 0 means non rated and 1 to 5 a rating ranging relatively from *very poor* to *awesome.* Let also define m the number of users and n the number of movies in the system. The problem of *recommender systems* is to predict the missing values of R using the known ratings.

The concept behind *matrix factorization* is to find two matrices V, W relatively $\boldsymbol{m}$ x $\boldsymbol{p}$,$\boldsymbol{n}$ x $\boldsymbol{p}$. which product can approximate a much bigger matrix R of dimensions $\boldsymbol{m}$ x $\boldsymbol{n}$. $R \approx V * W$



The process consists in a low-rank approximation of the user-item matrix using for both users and items some feature vectors which are used to model the prediction with a inner vector product of the selected user and item.
Ideally r_{ij} should correspond to the predicted rating but in reality the value will be different. Therefore we need a loss function to determine the difference between **real value** and **predicted value**. In our paper we are going to use the **Root Mean Square Error**,

$$\sqrt{\frac{1}{N}\sum_{i,j}^{N}(pr_{ij} - r_{ij})^2} \tag{2}$$

Where respectively *prij* is the prediction for the rating given by the user i for an item j and *rij* is the real rating for the former user and item. The prediction is obtained with the dot product of the user-item vectors,(ui,vj)

$$prediction_{ij} = \bar{u}_i * \bar{v}_j \tag{3}$$

The low rank approximation problem is formulated as follows to learn the factor vectors (ui,vj) [6]

$$(u_i, v_j) = \min_{u,v} \sum_{(u,i)\in K} (pr_{u,v} - r_{u,v})^2 \tag{4}$$

However solving the approximation with the former approach leads to the overfitting of data. Overfitting means that our model is learning through the noise in the training data instead of the underlying relations for which we expect to work. Altough the performance will continue improving with the training data it will get worse on unseen data. This usually happens when we are using a too complex model or we have too many free parameters, and the latter is our case. The solution is to apply regularization penzaling the magnitude of the feature vectors proportionally with a constant factor λ obtaining the following and well known formula for low-rank approximation:

$$(u_i, v_j) = \min_{u,v} \sum_{(u,i)\in K} (pr_{u,v} - r_{u,v})^2 + \lambda(u_i^2 + v_j^2) \tag{5}$$

3.1 Approaches

There are mainly two approaches to minimize equation(5) and are known as:
Alternating Least Square and **Stochastic Gradient Descent**.

3.2 Stochastic Gradient Descent

Stochastic Gradient Descent is a gradient descent optimization used to minimize a loss function, in our case equation (5). The goal of *Stochastic Gradient Descent* is to find a value $\theta^* \in R^k (k \geq 1)$ that minimizes a given loss $L(\theta)$. The algorithm makes use of noisy observations $\bar{L}'(\theta)$ of $L'(\theta)$, the function's gradient with respect to θ. Starti with some initial value θ_0, SGD refines the parameter value by iterating the stochastic difference equation[7]

$$\theta_{n+1} = \theta_n - \epsilon_n \bar{L}'(\theta_n) \qquad (6)$$

where n denotes the step number and ϵ_n is a sequence of decreasing step size.

3.2.1 SGD with Matrix Factorization

In matrix factorization we set $\theta = (V, W)$ and the loss function can be written as $Q(w) = \sum Q(u, i)$. Where w is an approximation obtained iterating through every single sample until a minimum is reached with the following step: $w = w - aplha * d\frac{d}{w}(Q_i(w))$ In our case $Q(u, i)$ is represented by equation(5), and is the equation we want to minimize. The iterative version of the algorithm can be sketched this way:[8]

Algorithm 1 Matrix Factorization with SGD

```
Require:
    R is the user-item matrix,
    V and W are the factor vectors initialized with values from 0.0 to 1.0,
    $\alpha$ is the learning rate,
    $\lambda$ the magnitude reduction,
    K is the number of iterations
    F is the number of features
 1: for $i$ to $K$ do
 2:     for user $u$ and item $i$ with a rating $r \in R[u, i]$ do
 3:         $predictedRating = V[u] * W[i]^t$
 4:         $error = R[u, i] - predictedRating$
 5:         for each feature $f \in F$ do
 6:             $V[u, f] = V[u, f] + \alpha * (error * W[i, f] - \lambda * V[u, f])$
 7:             $W[i, f] = V[u, f] + \alpha * (error * V[i, f] - \lambda * W[u, f])$
 8:         end for
 9:     end for
10: end for
```

Although this algorithm is used to optimizie equation(5) it has some scalability issues. These problem are insights of its iterative nature for which all the individual steps depends on each other and make it hard (but not impossible) to distribute it In this paper we will propose a functional implementation of this algorithm, and try to optimize it with a distributed version.

3.3 Alternating Least Square

4. TECHNOLOGIES AND CHALLANGES

Up to now we have described what is the so-called state of the art in the reserarch enviroment. Our challange in this paper will be to compare the execution of two different framework: *Spark* and *Flink*.
Both of the frameworks have an implemented version of ALS but none has a matrix factorization implemented with SGD, even though they have implementations for different purposes. We have also choose to user **Scala**, a functional programming language to develop the SGD factorization; functional programming languages seems to be better involved in the bigdata paradigm due to their ability to facilitate parallelization. Last but not least, we have used the various movielens datasets with different sizes to test our examples.

4.1 Scala

Scala, acronym of "Scalable Language", is an object-oriented functional programming language used for a large variety of tasks: from scripting to large mission critical systems (e.g. Twitter and Linkedin). Functional programming languages are well known in the bigdata environment due to one of their preculiarities: **The pure functions**. Pure functions are like mathematical functions, the return value depends only on the input value; meaning the function does not have any *side effect*. This is very important when dealing with large datasets because it's the peculiarity of functional languages that improves parallelization. The requirement for scaling horizontally is to have many independent tasks running in parallel, which goes perfectly with the latter peculiarity of functional languages.

4.2 Spark

4.3 Flink

4.4 MovieLens Dataset Structur

5. IMPLEMENTATION

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

6. RESULTS

7. CONCLUSIONS

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.