TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JONNE PETTERI PIHLANEN

BUILDING A RECOMMENDATION ENGINE WITH APACHE SPARK

Master of Science thesis

Examiner: ????
Examiner and topic approved by the
Faculty Council of the Faculty of
xxxx
on 1st September 2014

ABSTRACT

JONNE PETTERI PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampere University of Technology
Master of Science thesis, xx pages
September 2016
Master's Degree Program in Signal Processing
Major: Data Engineering
Examiner: ????
Keywords:

The amount of recommendation engines around the Internet is constantly growing.

This paper studies the usage of Apache Spark when building a recommendation engine.

TIIVISTELMÄ

JONNE PETTERI PIHLANEN: Suosittelijajärjestelmän rakentaminen Apache Sparkilla
Tampereen teknillinen yliopisto
Diplomityö, xx sivua
syyskuu 2016
Signaalinkäsittelyn koulutusohjelma
Pääaine: Data Engineering
Tarkastajat: ????
Avainsanat:

PREFACE

ASDASDASDASDASD

Tampere,

Jonne Pihlanen

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

Recommendation Engine	System that tries to predict the items that a user woud like
Collaborative	Users collaborate with each other to recommend items
Spark	Fast and general engine for large-scale data processing
Information retrieval (IR)	Activity of obtaining relevant information resources from a collection of information resources.
SDK	Software Development Kit

1. INTRODUCTION

Recommender systems have been successfully utilized to aid customers in decision making. In fact, they are constantly present in our everyday life. Whether a customer is shopping online, watching a movie from Netflix, browsing the Facebook or simply reading the news. All of these tasks involve a presence of a recommendation engine. Basically all parts of our daily life include recommendations of some sorts. However, the most basic type of recommendation is the one from human to human and happens completely without computers.

Recommendation can be divided into two major categories: item based and user based.

This thesis is structured as follows. Chapter 2 describes recommendation systems. In Chapter 3, the data for this project and related tasks are discussed. Chapter 4 discusses Apache Spark, an open source framework for building distributed programs. Chapter 5 discusses the results. Finally, in Chapter 6 the evaluation is presented along with conclusions.

2. RECOMMENDATION SYSTEMS

Recommendation denotes a task in which a tool is measuring similarity of items. It refers to an algorithm that calculates the similarity of the items to be compared. This can be performed by comparing users to other users or comparing items into other items.

Recommender Systems are a set of techniques and software tools that provide suggestions to users about potentially useful items. [9] Item denotes the general subject that the system recommends to users.

A recommendation engine is a system that includes all the necessary steps to retrieve the actual recommendation result. This includes acquiring, cleaning and preprocessing of the data, recommendation algorithm and finally the interpretation of obtained results.

A recommendation system must balance between the needs of service provider and the customer. [9] A recommendation system can be used in tasks such as increasing the number of items sold, selling more diverse items, increasing user satisfaction, increasing user loyalty or understanding better what the user wants [9]. In addition, a number of different functions exist that the user might expect a recommendation system to offer. Some of these are similar to the ones mentioned above, thus considered as core functions. Others include tasks such as recommending a sequence of items. [9]

GroupLens was a pioneer in recommendation systems along with BookLens and MovieLens. In addition to this they also released data sets which, aside of recommendation, was also pioneering in the field since data sets were not that common for benchmarking or just trying out new technologies. [2]

Lately, a number of service providers, like Telegram and Microsoft, have started to introduce bot frameworks for their services. A bot is a web service that uses

a conversational format to interact with users. Users can start conversations with the bot from any channel the bot is configured to work on. Conversations can be designed to be freeform, natural language interactions or more guided ones where the user is provided choices or actions. It is possible to utilize simple text strings or something more complex such as rich cards that contain text, images, and action buttons. [8]

Already for a long time, companies have had some sort of SMS that have been accepting feedback from customer or ordering a new data package for you mobile subscription. IRC channel bots have been around even longer. Idea is not new but now there are popular platforms for the bots. As any other idea, also a recommendation engine could be implemented in a way that it can be used via a bot.

2.1 Collaborative filtering

Collaborative Filtering Recommendation Systems (CF, CFRS) are based on the collaboration of users. They aim at identifying patterns of user interests in order to make targeted recommendations [1]. First a user provides ratings for items. Next the method will find recommendations based on other users that have purchased similar items or based on items that are the most similar to the user's purchases. Collaborative filtering can be divided into two sub categories which are item based collaborative filtering and user based collaborative filtering. Collaborative filtering has been studied the most thus being considered as the most popular technique in recommendation systems [4] [9] [3]. It is, also, usually easy to implement.

Collaborative filtering analyzes relationships between users and interdependencies among products to identify new user-item associations [6] For example, deciding that two users may both like the same song because they play many other same songs is an example of collaborative filtering. [10]

Item based collaborative filtering (IBCF) starts by finding similar items from the user's purchases [4]. Next step is to model the preferences of a user to an item based on ratings of similar items by the same user [9].

The following snippet presents the idea in IBCF for every new user. Popular similarity measures are cosine distance and Pearson correlation. Similarity measures will be described in section 3.4.

Program 2.1 *Item-Based Collaborative Filtering algorithm [4]*

1. For each two items, measure how similar they are in terms of having received similar ratings by similar users
2. For each item, identify the k−most similar items
3. For each user, identify the items that are most similar to the user's purchases

User-based collaborative filtering (UBCF) starts by finding the most similar users, rate items purchased by similar users, pick top rated items. The similarity in taste of two users is calculated based on the similarity in the rating history of the users [9].

The steps for every new user in user-based collaborative filtering are as follows:

Program 2.2 *User-Based Collaborative Filtering algorithm [4]*

1. Measure how similar each user is to the new one. Like IBCF, popular similarity measures are correlation and cosine.
2. Identify the most similar users. The options are:
   - Take account of the top k users (k−nearest_neighbors)
   - Take account of the users whose similarity is above a defined threshold
3. Rate the items purchased by the most similar users. The rating is the average rating among similar users and the approaches are:
   - Average rating
   - Weighted average rating, using the similarities as weights
4. Pick the top−rated items.

Collaborative filtering algorithms suffer from the new user and new item problems [4], which originates to the fact that it is based only on user's recommendations. If

user has not given any reviews, the algorithm is not able to produce any recommendations either.

Collaborative filtering algorithms typically suffer from issues such as cold start and sparsity. Cold start denotes that a relatively large amount of data is required in order to be able to provide accurate recommendations for a user. Sparsity means that the number of items typically exceeds the number of users. This makes the relations extremely sparse since most users have rated or purchased only a small subset of the total items. [1]

Amazon.com, the biggest Internet retailer in the United States, has previously been using item-to-item collaborative filtering method. However, the current status of Amazon recommendation seems to be unknown to the public, which, in fact, is better for the business. In their implementation the algorithm builds a table containing similar items by finding ones that users tend to purchase together. The algorithm then finds items similar to each of the user's purchases and ratings, combines those items, and returns the most popular or correlated items. [7]

3. APACHE SPARK

- Driver

- Executor

Apache Spark is an open source framework that combines an engine for distributing programs across clusters of machines with an elegant model for writing programs. [10] It provides high-level APIs in Java, Scala, Python and R.

Spark can be introduced more easily by describing its predecessor, MapReduce, and the advantages it offers. MapReduce offered a simple model for writing programs that could execute in parallel across hundreds of machines. MapReduce achieves nearly linear scalability as the data size increases. The execution time is maintained by adding more computers to handle the task.

Spark preserves MapReduce's linear scalability and fault tolerance while extending it in three important ways. In MapReduce the intermediate results between the map and reduce tasks must be written into memory where as Spark is able to pass the results directly to the next step in the pipeline. Spark also treats the developers better by offering a rich set of transformations which enables users to represent complex pipelines in a few lines of code. (EXAMPLE?) Spark also introduces in-memory processing by introducing Resilient Distributed Dataset (RDD) abstraction which offers a way for developers to materialize any step in a processing pipeline and store it into memory. This means that future steps do not need to calculate the previous results again. Previously this kind of feature has not been available within distributed processing engines. [10]

Spark programs can be written using Java, Scala, Python or R. However, using Spark with Scala instead of Java, Python or R has a couple of advantages to it. Performance overhead is reduced, since tasks such as transferring data across different layers or performing transformations for data may result in weaker performance.

Spark is written with Scala, which denotes that user has always access to latest and greatest features of the framework. Spark philosophy is easier to understand when Spark is used with the language it was built with. There is still one, maybe the biggest benefit of using Scala with Spark, and it is the developing experience that comes with the fact that user is using the same language for everything. Importing data from database, data manipulation, shipping the code into clusters. [10]

Spark is shipped with a read eval print loop (REPL). Usually when a application developed in REPL has matured enough, it is a good idea to move it into a compiled library (JAR). This way it is possible to prevent code and results from disappearing.

3.1 Resilient Distributed Dataset (RDD)

RDD is an abstraction for a collection of objects in spark that can be distributed across multiple machines in a cluster. When a new RDD is created, nothing is actually done, it means that spark knows where the data is when the time comes to do something with it.

RDD can be created in two ways: parallelizing an existing collection in the driver program or referencing an external dataset in an external storage system, such as a shared filesystem, HDFS, HBase or any data source offering a Hadoop InputFormat.?? [13]

3.2 Dataset API

Dataset, DS, is the replacement for RDD in Spark.

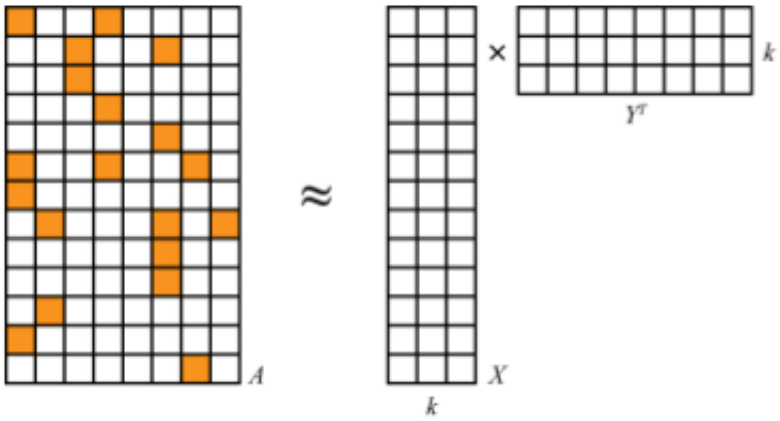
3.3 Matrix Factorization

Matrix factorization denotes a task in which a matrix is decomposed into a product of matrices. There are many different matrix decompositions. The following chapter will describe matrix factorization in general and the Alternating Least Squares algorithm which is the matrix factorization algorithm that is implemented in Spark. It is based on same idea as Netflix prize winner, matrix factorization models.

Matrix factorization belongs to a vast class of algorithms called latent-factor models. Latent-factor models try to explain observed interactions between a large number

of users and products through a relatively small number of unobserved, underlying reasons. For example, they can try to explain why people would buy a particular album out of endless possibilities by describing users and albums in terms of tastes which are not directly available as data. [10] A latent factor is not available for direct observation. For example health of a human being is a latent factor. Health can not be observed as a variable such as blood pressure.

Figure 3.1 *Matrix factorization [10]*



Matrix factorization algorithms treat the user and product data as if it was a large matrix A. Each entry in row i and column j represents a rating the user has given to a specific product. [10]

Usually A is sparse, which denotes that most of the entries of A are 0. This is due to the fact that usually only a few of all the possible user-product combinations exist.

Matrix factorization models factor A as the matrix product of two smaller matrices, X and Y, which are quite tiny. Since A has many rows and columns, both of them have many rows, but both have just a few columns (k). The k columns match to the latent factors that are being used to explain the interactions of the data. The factorization can only be approximate because k is small. [10]

The standard approach to matrix factorization based collaborative filtering treats the entries in the user-product matrix as explicit preferences given by the user to the product, for example users giving ratings to movies. Implicit data denotes for example page views or a value representing if a user has listened to a artist. Explicit data means actual ratings that a user has given to a product. Spark ALS can handle both implicit and explicit data. [12] [10]

Usually many real-world use cases have access only to implicit feedback data such as views, clicks, purchases, likes or shares. However, instead of trying to model the matrix of ratings directly, the approach in Spark MLlib treats the data as numbers representing the strength of the observations such as the number of clicks, or the cumulative duration someone spent viewing a movie. Instead of explicit ratings, these numbers are related to the level of confidence in observed user preferences. Based on this data, the model tries to find latent factors that can be used to predict the expected preference of a user for an item. [12]

Sometimes these algorithms are referred to as matrix completion algorithms. This is because the original matrix A may be sparse while the product XY^T is dense. Hence, the product is only an approximation of A. [10] In this thesis, the original user-item matrix A is actually very dense since we have a value in every entry of the matrix.

3.3.1 Alternatig Least Squares (ALS)

Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. Spark MLlib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. Spark MLlib uses the Alternating Least Squares (ALS) algorithm to learn these latent factors. [12]

Spark ALS attempts to estimate the ratings matrix A as the product of two lower-rank matrices, X and Y. [11]

$$A = XY^T \tag{3.1}$$

Typically these approximations are refered to as factor matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix. [11] Spark ALS enables massive parallelization since it can be done separately, it can be done in parallel which is an excellent feature for a large-scale computation algorithm. [10]

Spark ALS is a blocked implementation of the ALS factorization algorithm. Idea is to group the two sets of factors, referred to as *users* and *products*, into blocks. Grouping is followed by reducing communication by only sending one copy of each user vector to each product block on each iteration. Only those user feature vectors are sent that are needed by the the product blocks. Reduced communication is achieved by precomputing some information about the ratings matrix to determine the out-links of each user and in-links of each product. Out-link denotes those blocks of products that the user will contribute to. In-link refers to the feature vectors that each product receives from each user block they depend on. This allows to send only an array of feature vectors between each user block and product block. Consequently the product block will find the users' ratings and update the products based on these messages. [11]

Essentially, instead of finding the low-rank approximations to the rating matrix A, it finds the approximations for a preference matrix P where the elements of P are 1 when r $>$ 0 and 0 when r $<=$ 0. The ratings then act as confidence values related to strength of indicated user preferences rather than explicit ratings given to items. [11]

$$A_i Y (Y^T Y)^{-1} = X_i \tag{3.2}$$

Alternating Least Squares operates by rotating between fixing one of the unknowns u_i or v_j. While the other is fixed the other can be computed by solving the least-squares problem. This approach is useful because it turns the previous non-convex problem into a quadratic that can be solved optimally [1]. A general description of the algorithm for ALS for collaborative filtering taken from [1] is as follows:

Program 3.1 *Alternating Least Squares algorithm [1]*

1. Initialize matrix V by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.

2. Fix V, solve U by minimizing the RMSE function.

3. Fix U, solve V by minimizing the RMSE function.

4. Repeat Steps 2 and 3 until convergence.

Minimizing the Root Mean Square Error RMSE function denotes a task in which line is plotted. EXPLAIN RMSE.

4. IMPLEMENTATION

- MovieLens ml-latest-small dataset.

- What is MovieLens?

5. RESULT

6. EVALUATION

6.1 Conclusion

There are a number of possible implementations for a recommendation engine. This was selected because Apache Spark could be a good tool to know in future and in addition learning Scala programming was another thing that was considered.

Existing recommendation or analytic engines should be evaluated before making a decision about the recommendation engine.

In the end the most difficult thing was to find the right approach for this task. By trial and error the right combination of technologies and an actual working example was found.

6.2 Future work

Actual parallelization?

BIBLIOGRAPHY

[1] C. Aberger, "Recommender: An analysis of collaborative filtering techniques," 2014. [Online]. Available: http://cs229.stanford.edu/proj2014/Christopher%20Aberger,%20Recommender.pdf

[2] C. C. Aggarwal, *Recommender Systems*. Springer International Publishing, 2016.

[3] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370. [Online]. Available: http://dx.doi.org/10.1023/A:1021240730564

[4] S. K. Gorakala and M. Usuelli, *Building a Recommendation Engine with R*, 1st ed. Packt Publishing, 2015.

[5] D. Harris, "Twitter open sourced a recommendation algorithm for massive datasets," 2014. [Online]. Available: https://gigaom.com/2014/09/24/twitter-open-sourced-a-recommendation-algorithm-for-massive-datasets/

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," 2009. [Online]. Available: https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[7] G. Linden, B. Smith, and J. York, "Amazon.com recommendations," *IEEE INTERNET COMPUTING*, pp. 76–79, 2003. [Online]. Available: http://www.cin.ufpe.br/~idal/rs/Amazon-Recommendations.pdf

[8] Microsoft, "Bots." [Online]. Available: https://docs.botframework.com/en-us/

[9] F. Ricci, L. Rokach, B. Shapira, and P. B. Kanto, *Recommender Systems Handbook*, 1st ed. Springer, 2011.

[10] S. Ryza, U. Laserson, S. Owen, and J. Wills, *Advanced Analytics with Spark*. O'Reilly Media, Inc., 2015.

[11] Spark. (2014) ALS. [Online]. Available: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS

[12] ——. (2014) Collaborative filtering - rdd-based api. [Online]. Available: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

[13] ——. (2014) Spark programming guide. [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html