

|  |
| --- |
| RECOMMENDATION SYSTEM |
|  |
| April 5 , 2020  Mohomad Matar  Ahmad Marzurki  Sachin Jangoni  Tejas Bhor |

Table of Contents

[Introduction 3](#_Toc37011605)

[History of Recommender Systems 4](#_Toc37011606)

[Overview 5](#_Toc37011607)

[Defining Singular Value Decomposition 6](#_Toc37011608)

[Alternating Least Squares 8](#_Toc37011609)

[Building Bigdata pipeline 9](#_Toc37011610)

[1 Integration of Kafka and Zookeeper 10](#_Toc37011611)

[2 Use of Kafka producer 11](#_Toc37011612)

[3 Use of Pyspark Consumer 12](#_Toc37011613)

[4 Analysis and Model Creation 13](#_Toc37011614)

[5 Visualization 15](#_Toc37011615)

# Introduction

Every day, we are inundated with choices and options. What to wear? What movie to rent? What stock to buy? What blog post to read? The sizes of these decision domains are frequently massive: Netflix has over 17,000 movies in its selection and Amazon.com has over 410,000 titles in its Kindle store alone. Supporting discovery in information spaces of this magnitude is a significant challenge. Even simple decisions — what movie should I see this weekend? — can be diﬃcult without prior direct knowledge of the candidates.

Historically, people have relied on recommendations and mentions from their peers or the advice of experts to support decisions and dis-cover new material. They discuss the week’s blockbuster over the water cooler, they read reviews in the newspaper’s entertainment section, or they ask a librarian to suggest a book. They may trust their local the-ater manager or news stand to narrow down their choices, or turn on the TV and watch whatever happens to be playing.

These methods of recommending new things have their limits, par-ticularly for information discovery. There may be an independent film or book that a person would enjoy, but no one in their circle of acquaintances has heard of it yet. There may be a new indie band in another city whose music will likely never cross the local critic’s radar. Computer-based systems provide the opportunity to expand the set of people from whom users can obtain recommendations. They also enable us to mine users’ history and stated preferences for patterns that neither they nor their acquaintances identify, potentially providing a more finely-tuned selection experience.

# History of Recommender Systems

|  |
| --- |
| The capacity of computers to provide recommendations was recognized fairly early in the history of computing. Grundy, a computer-based librarian, was an early step towards automatic recommender systems. It was fairly primitive, grouping users into “stereotypes” based on a short interview and using hard-coded information about various stereotypes’ book preferences to generate recommendations, but it rep-resents an important early entry in the recommender systems space.  In the early 1990s, collaborative filtering began to arise as a solution for dealing with overload in online information spaces. Tapestry was a manual collaborative filtering system: it allowed the user to query for items in an information domain, such as corporate e-mail, based on other users’ opinions or actions (“give me all the messages forwarded by John”). It required eﬀort on the part of its users, but allowed them to harness the reactions of previous readers of a piece of correspondence to determine its relevance to them.  Automated collaborative filtering systems soon followed, automatically locating relevant opinions and aggregating them to provide rec-commendations. Group Lens used this technique to identify Usenet articles which are likely to be interesting to a particular user. Users only needed to provide ratings or perform other observable actions; the system combined these with the ratings or actions of other users to provide personalized results. With these systems, users do not obtain any direct knowledge of other users’ opinions, nor do they need to know what other users or items are in the system in order to receive recommendations.  During this time, recommender systems and collaborative filter-ing became an topic of increasing interest among human–computer interaction, machine learning, and information retrieval researchers. This interest produced a number of recommender systems for various domains, such as Ringo for music, the Bell Core Video Recom-mender for movies, and Jester for jokes. Outside of computer science, the marketing literature has analyzed recommendation for its ability to increase sales and improve customer experience |
| Overview *Collaborative Filtering Methods*  Collaborative filtering (CF) is a popular recommendation algorithm that bases its predictions and recommendations on the ratings or behavior of other users in the system. The fundamental assumption behind this method is that other users’ opinions can be selected and aggregated in such a way as to provide a reasonable prediction of the active user’s preference. Intuitively, they assume that, if users agree about the qual-ity or relevance of some items, then they will likely agree about other items — if a group of users likes the same things as Mary, then Mary is likely to like the things they like which she hasn’t yet seen.  There are other methods for performing recommendation, such as finding items similar to the items liked by a user using textual similarity in metadata (content-based filtering or CBF). The focus of this survey is on collaborative filtering methods, although content-based filtering will enter our discussion at times when it is relevant to overcoming a particular recommender system diﬃculty.  The majority of collaborative filtering algorithms in service today, including all algorithms detailed in this section, operate by first gen-erating predictions of the user’s preference and then produce their recommendations by ranking candidate items by predicted preferences. Often this prediction is in the same scale as the ratings provided by users, but occasionally the prediction is on a diﬀerent scale and is meaningful only for candidate ranking. This strategy is analogous to the common information retrieval method of producing relevance scores for each document in a corpus with respect to a particular query and presenting the top-scored items. Indeed, the recommend task can be viewed as an information retrieval problem in which the domain of items (the corpus) is queried with the user’s preference profile.  Therefore, this section is primarily concerned with how various algorithms predict user preference. In later sections we will discuss recommendation strategies that diverge from this structure, but in actual implementation they frequently start with a preference-ranked list of items and adjust the final recommendation list based on additional criteria. Defining Singular Value Decomposition In the traditional collaborative filtering algorithms so far described, there are hints of viewing the user–item ratings domain as a vector space. With this view, however, the vectors are of extremely high dimension: an item is a |U |-dimensional vector with missing values of users’ preferences for it (similarly, a user is a |I|-dimensional vector). Further, there is redundancy in these dimensions, as both users and items will usually be divisible into groups with similar preference pro-files (e.g., many science fiction movies will be liked to similar degrees by the same set of users). It is therefore natural to ask whether the dimensionality of the rating space can be reduced — can we find a smaller number of dimensions, ideally a constant number k, so that items and users can be represented by k-dimensional vectors?  In information retrieval, a document corpus can be represented as a term-document matrix where each cell is the number of times the given term occurs in a particular document. This results in high-dimensional representations of terms and documents, further complicated by the problems of synonymy (diﬀerent terms having the same or similar meaning), polysemy (the same term having diﬀerent mean-ings), and noise (documents or queries using terms incorrectly). Latent semantic analysis (LSA, also called latent semantic indexing or LSI) deals with these problems by using dimensionality reduction, in the form of truncated singular value decomposition (SVD), to extract the semantic relationships between documents latent in their use of vocab-ulary [16, 36]. SVD-based dimensionality reduction has since been adapted to collaborative filtering by Billsus and Pazzani , Sarwar et al. [128, 131], and many others.  For a matrix M, its SVD is the factorization of M into three con-stituent matrices such that M = UΣTT, Σ is a diagonal matrix whose values σi are the singular values of the decomposition, and both U and T are orthogonal. What this accomplishes is introducing an interme-diate vector space represented by Σ. If M is the ratings matrix, ΣTT transforms vectors from item-space into the intermediate vector space. In the pure form of the SVD, U is m × kˆ, Σ is k × kˆ, and V is  ˆ ˆ  n × k, where M is m × n and has rank k; this is not a significant gain. Σ can, however, be truncated by only retaining the k largest singular values to yield Σk .  This truncation simultaneously achieves two goals. First, it decreases the dimensionality of the vector space, decreasing the storage and computational requirements for the model. Items and users can each be represented by k-dimensional vectors. Second, by drop-ping the smaller singular values, small perturbances as a result of noise in the data are eliminated, leaving only the strongest eﬀects or trends in the model. In collaborative filtering, this noise can come as a result of other factors besides sheer preference playing a role in a user’s rating; decreasing the impact of noise improves our ability to provide high-quality recommendations.  Once the rank-k SVD of Equation (2.13) has been computed, it can be interpreted as an expression of the topic preference-relevance model. The rows of the |U | × k matrix U are the users’ interest in each of the k inferred topics, and the rows of I are the item’s relevance for each topic. The singular values in Σ are weights for the preferences, representing the influence of a particular topic on user–item preferences across the system. A user’s preference for an item, therefore, is the weighted sum of the user’s interest in each of the topic’s times that item’s relevance to the topic.   Alternating Least Squares Alternating Least Squares (ALS) is the model we’ll use to fit our data and find similarities. , however, the vectors are of extremely high dimension: an item is a |f |-dimensional vector with missing values of users’ preferences for it (similarly, a user is a |I|-dimensional vector). Further, there is redundancy in these dimensions, as both users and items will usually be divisible into groups with similar preference pro-files (e.g., many science fiction movies will be liked to similar degrees by the same set of users). But before we dive into how it works we should look at some of the basics of matrix factorization which is what we aim to use ALS to accomplish.   Building Bigdata pipeline We follow below steps to manage and adhere bigdata pipeline in spite of so many system compatibility issues.   1 Integration of Kafka and Zookeeper We successfully manage to integrate Kafka and Zookeeper with the help of docker.       2 Use of Kafka producer    3 Use of Pyspark Consumer  4 Analysis and Model Creation Using some deep learning technique, we implement ML model on our data and Accuracy is 70%           5 Visualization This is the final step where we visualize entire thing and created Dashboard in Tableau.  Below are the sample visualizations.( We attached .twbx file for reference with  dashboard )            *Thank You* |