Online Matrix Completion Techniques for Recommender Systems

Anirudh Acharya, Vamsi Meduri, Vignesh Narayanan, Pradeep Mani

May 3, 2015

1 Abstract

Now a days consumers are offered a huge list of choices by retailers and content providers and they try to meet the consumers needs and preferences. Selling right products to the consumers increases the user satisfaction and consumer loyalty towards the retailers. Therefore, retailers use computer based recommendation system to analyze the users interest to provide product recommendation based on individual user preferences. Collaborative filtering technique is one of the most widely used method in building a recommendation system. But the problems we often face with Collaborative Filtering method is data sparsity and cold start problem. In our project we have proposed a solution to overcome this problem through Online Matrix completion technique and we have built a recommendation system based on this approach.

2 Introduction

Every day in our life we make certain decisions based on choices and options What songs to hear? What movie to watch? What to eat? What restaurants are good? What news to read? Often these decisions had to be made from a massive amount of choices available for each domain. For example, Netflix contains over 17000 movie titles and Amazon ecommerce website has over 410,000 titles in kindle store alone. A simple decision What movie to watch? can sometimes be difficult because the knowledge on that subject might be limited to the person. It is a significant challenge to discover information manually from such a large data space. So people usually rely on recommendations or advice from their peers or from experts to support their decisions and to find new items or products. But even such recommendation has some limitations in information discovery. For example, there may be a new product introduced in the market that a person would be interested in but none of his friends or acquaintance knows about that product yet. So in such cases a person may never come to learn about such products.

Computer based recommendation systems can be used to overcome such limitations. Using a computer based recommendation system the user set can be expanded to obtain recommendation from a wide range of users. So a user have a better chance of getting new products recommended to them. It also enable the user to set preferences and also by mining users past behavior the system can recommend provide the users a more fine-tuned experience. The recommendation system not only helps the users but also the companies who has spent money on building such a system. Using recommendation system, a company could increase the sales of different brands of similar product and could sell more diverse items together. Understanding the customers preferences and selling the right products to them increases the customer satisfaction which in turn increases customers fidelity.

Over the past two decades there has been a significant amount of research done on this subject and a wide variety of computer based recommendation methods has been proposed to automatically recommend products to the users. The various methods proposed for building a recommendation system can be broadly classified in to two categories Content Based Filtering method and Collaborative Filtering method. Content Based filtering techniques builds a profile for each object (users and products) based on certain features (genre, demographic, gender, etc.) and the recommendation is done by matching their profiles. Collaborative Filtering looks at the past user behavior such as product reviews and buying history, analyzes the relationships between users and interdependencies among products to generate user-product associations. In our project we have developed a recommendation systems to recommend a product to a user based on the Online Matrix Completion method, a type of Collaborative filtering technique.

3 Recommendation Engine

A Recommendation Engine can be defined as a system that attempts to predict a rating or a preference that a user would give to an item such as movies, songs, books, consumer products, etc. Using this preference, the system would suggest similar products having similar rating given by different users. That is when a user interacts with a Recommendation systems, the system applies knowledge discovery techniques to understand the problem and makes a personalized recommendations. Different companies has employed different techniques to build the recommendation system. For example Spotify uses Metadata/Content Analysis and Collaborative Filtering techniques, Pandora uses Manually Tag Attributes and Songza uses Manual Curation techniques. And as introduced earlier, the recommendation methods can be classified into two broad categories Content Based filtering and Collaborative filtering. Content Based Filtering In this method the systems uses a set of data that are previously rated by the user manually and the system analyzes this data to build a profile for the user. Using the profile, the recommendation system matches the data with contents profile and makes suggestion of the contents having similar features.

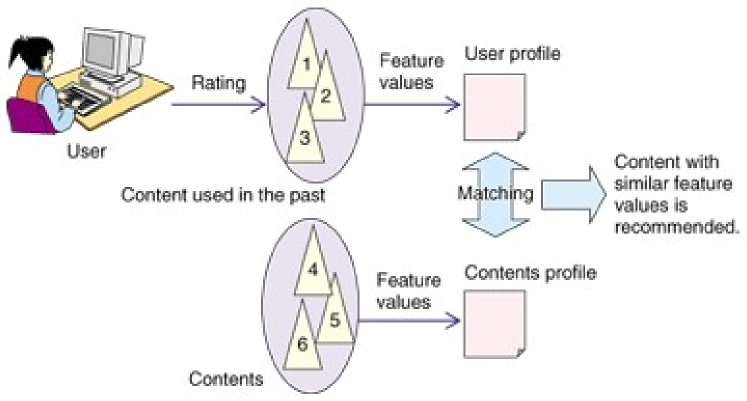


Figure 1: Content Based Filtering Technique

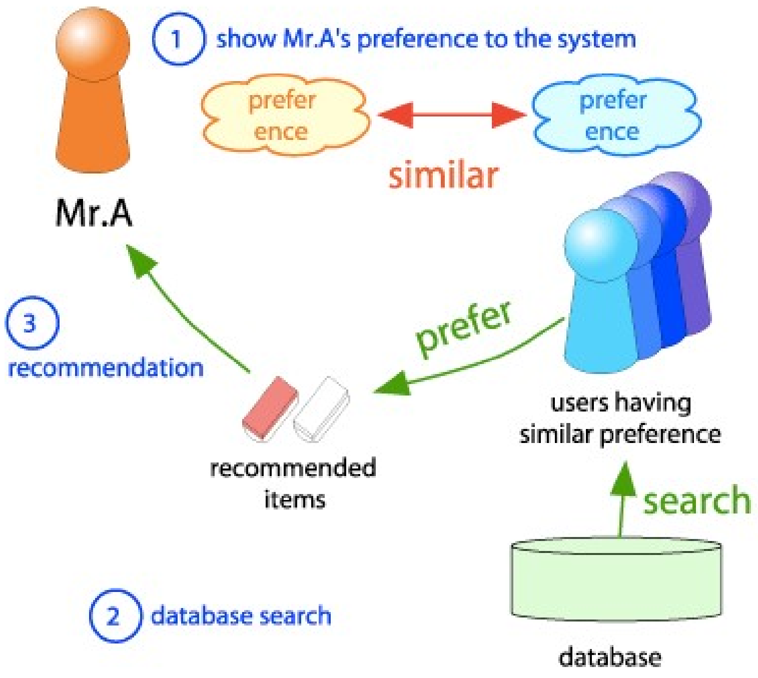


Figure 2: Collaborative Filtering Technique

The advantage of this technique is user independence, that is, the system does not have to depend on other user profiles and most importantly no cold start problem. The disadvantages of this technique includes limited content analysis and over specialization. Also when new user uses the system for the first time then the system cannot make efficient recommendation since the user profile does not have enough information.

In Collaborative Filtering, the systems analyze users past behavior and matches similar user preference with different user profiles and suggest new items based on the similarity to other users. For example, if person A likes items P, Q, R S and person B like items Q, R, S T, the system would analyze and match these preferences and recommends item P to person B and item T to person B.

The problem with Collaborative Filtering techniques is that the user-item matrix that is generally used for analyzing past user behavior could be extremely large and sparse, which will pose challenges to the performances of the recommendation engine. One quintessential problem caused by the data sparsity is the cold start problem. As collaborative filtering techniques need the users past preferences, the engine might find it hard to recommend products to new users, who will have to rate sufficient number of products before any new product could be recommended to them. Similarly for new products, it will have to be reviewed by sufficient number of users before it can be recommended to any user based on its associations with other products.

4 Related Work

Many approaches have been proposed to solve the problem of building recommending systems. The paper by Koren et al [2] talks of one of the most popular approaches to building recommendation engine i.e. to model the problem as a matrix factorization problem. Matrix Factorization maps both users and products into a joint latent factor space with dimensionality f <min (users, items). The challenge for such a method is to compute the factor vectors for each item in the user product matrix. This problem is then solved by using optimization techniques such as alternating least squares or by minimizing the recommendation value and its factors while avoiding over fitting through regularization. This work also discusses interesting aspects like temporal dynamics - which reflects the varying taste of the user and varying rating of an item over time, user biases - where there are users who are relatively tough critics and assign controlled rating to even popularly accepted items, additional input sources- like the user click pattern indicating implicit feedback and user demographics indicative of the popularity of an item in a particular region, inputs with varying confidence levels- which are weighted recommendations. All these additional factors are expressed as terms in the optimization problem formulation, thus making it more robust to these real world possibilities.

The paper by Yao et al [4] addresses two aspects of user-item recommendations. One is the issue of possible recommendations by a user in the case of items which have not been explicitly endorsed by him. Another is the case of cold start. The paper talks about a One-class collaborative filtering (OCCF) where users have not explicitly rated items from 1 to 5 as in a multiclass collaborative filtering (MCCF), but their implicit behavior and clicking patterns can be exploited to tackle user recommendations to unseen or missing entries in OCCF. The paper talks about coupling two ideas - imputation and weighting. Imputation is the task of assigning a non-zero probability to an item that has not been recommended, because it may be a possible recommendation by a user once he sees it. Weighting is to assign controlled weight to an example which is unseen, thereby assigning more weight to seen items, hence controlling the level of confidence in the recommendations provided. These two methods work together to not discount any unseen item and at the same time, to assign it an importance that is slightly lesser than a seen item.

To tackle cold start, side information about items and users are captured through a user-user and an item-item matrix. Two users closer to each other are captured as having a stronger association leading to a strong recommendation; same applies to two closely related items being chosen together. This in turn means that if we know a recommendation value for a user item combination, it is easier to predict the value for another (user, product) matrix entry as long as the user or item is overlapping. This is captured through user homophily and item homophily. This is also coupled with regularization parameters to control the magnitudes of the low rank approximation matrices of the original recommendation matrix while formulating the optimization problem. One shortcoming of this approach is that it might not always be possible to have information about user-user or product-product proximity. And even when present it might not always be a good assumption to recommend items based on feature proximity alone. For instance two friends who are marked with a close proximity based on any social metric might actually have diametrically opposite tastes when it comes to buying products that being recommended. This is something we hope to look into during the course of the project.

Social networks which is a collection of social actors and the relationship, including nodes (social actors), the edge between the nodes (the actors association) and the weights of the edges (the impact between the actors) can be used to overcome the cold start problem. Each node, not independent individuals, is interdependent by sides. Sides, the channels of resource flowing, provide guidance for individual actions, greater weight being stronger guidance.

There are works that propose solution to the cold start problem based on homophily in social networks: we can use social networks' information in order to fill the gap existing in cold-start problem and find similarities between users. There are works that show that communities could be used which are extracted from different dimensions of social networks (friendship network, item similarity network, commenting network), to capture the similarities of these different dimensions and accordingly, help recommendation systems to work based on the found latent similarities. This is proven to be way more effective than normal social network based recommendations.

Ontology is a shared conceptual model explicit formal specification and its goal is to capture the knowledge of related fields, to provide a common understanding of the domain knowledge to determine the terms of mutual recognition in the field, and give a clear definition of the mutual relations between these terms and terminology from the different levels of formalization model.

Another potential solution to the cold-start problem in group recommender systems is to use the information about previous group recommendation events and copy ratings from a user who played a similar role in some previous group event. It is also shown that copying in this way, i.e. conditioned on groups, is superior to copying nothing and also superior to copying ratings from the most similar user known to the system.

Literature works shows that this can be effectively used to solve the cold-start problem. There are works that presents a method combining social sub-community division and ontology decision model to solve the new user cold-start problem in collaborative filtering algorithm, which builds relationships between user static information and dynamic preferences by learning. Sub-community whose points have strong relationship with each other means nodes impact to others is larger than those outside. When multiple edges between the nodes, it can be changed to the matrix for data analysis and refining side weight in order to simplify the network.

Another potential solution proposed by Quijano-Sanchez et al in [3] to the cold-start problem in group recommender systems is to use the information about previous group recommendation events and copy ratings from a user who played a similar role in some previous group event. It is also shown that copying in this way, i.e. conditioned on groups, is superior to copying nothing and also superior to copying ratings from the most similar user known to the system.

For any recommendation in general, it would be desirable for us to accommodate in an online fashion each new incoming (user, product) tuple while mapping the rating to the factor space while performing the matrix factorization. The Online Matrix Factorization [1] by Blondel et al. propose a method to achieve this in their paper. With each new entry in the user-product matrix the corresponding factor vectors from the factor matrices have to be modified to ensure that the matrix factorization still holds even after the entry of the new data point.

5 Challenges with Collaborative Filtering

Some of the issues we face while using collaborative filtering for recommendation systems are - Data Sparsity, and Updating our recommendations based on new user-product tuples that come in.

- For most recommendation systems, the number of product runs in tens of thousands and the number of users go into millions. But every user probably uses a very small subset of all the products and rates even fewer products. This makes our user-product matrix, based on which we are making the recommendations, a very sparse matrix. When user-product matrix is very sparse it makes it challenging for us to recommend products based on this sparse data.

- Another issue we often face is that of updating our recommendations when we get new user-product tuple. Typically in collaborative filtering systems we factorize the user-product matrix and project the user and product vectors onto a lower dimensional latent space. We later use this latent feature space to match similar products and users. But if new entries are being updated in the user-product matrix then we will likely have to re-factorize the whole matrix to obtain the new latent feature vectors of users and products. This could be very inefficient and time consuming to perform as we expect the user-product matrix to be very big and the new user-product tuples arrive quite often. So re-factorizing the matrix can be a very step.

In our proposed solution we attempt to address both these issues by finding factor vectors of part of the user-product matrix instead of re-factorizing the whole matrix.

6 Proposed Solution

6.1 Concepts and Vocabulary

In this section we will describe the basic concepts and terminologies used in the proposed solution. Many of these are applicable to other collaborative filtering based recommender systems in general.

The information for a collaborative filtering system consists of *users* who express their interests and preferences in certain *products* in the form of *rating* given to the products. Each value in the *user-product* matrix corresponds to a rating given by that *user* to the particular *product*. These ratings are on an integer scale of 0-5. These set of matrices form a sparse matrix. Each rating is a value in the matrix indexed by the user-product tuple. For example figure 1.1 shows one such matrix with ratings and missing values denoted by 0.

With this setting, a recommender system has two tasks - the first is to predict a missing value in the matrix i.e given a user and a product, what would be the likely rating that the user would give to that product? The ratings matrix can be viewed as a small sample from a complete user-product matrix, then the prediction of a particular product rating can be seen as a matrix completion problem. The second is, given a user we should present a ranked list of recommended products to which the user is likely to give a high preference.

In this project the user-product rating matrix is referred as R and each element in row u and column i is referred by $r_{u,i}$ where u is the user id and i is the product id. The matrix R is factorized into two factor matrices P and Q. Every value $r_{u,i}$ in the rating matrix R corresponds to the dot product of the two factor vectors p_u and q_i of matrices P and Q respectively.

$$R = P.Q$$

$$r_{u_t,i_t} = p_{u_t}.qi_t \quad \forall u, i$$

The following figure 1 gives a good illustration of how factor matrices will accomplish matrix completion task.

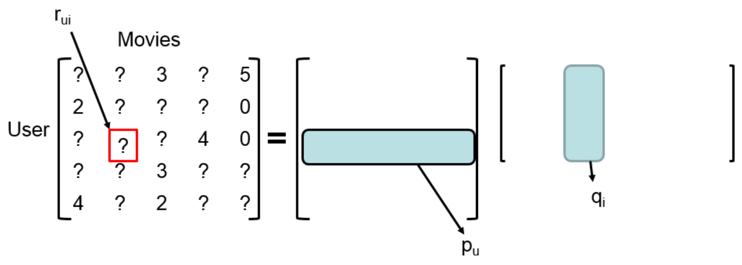


Figure 3: User-Product matrix, R with its factor matrices P and Q

Every missing value in the R matrix can be obtained by multiplying the corresponding factor vectors from the matrices P and Q. We would also require that given a new rating value in the R we would want to update the factor vectors p and q without having to factorize the whole matrix entirely. Our project gives a formulation to achieve this.

Such a technique of updating factor matrices without having to re-factorize the whole matrix is very useful when we want to update the our recommendations based on more recent product ratings that the user has given. This makes our system of On-line Matrix Completion work very well and update recommendations in real-time.

Also, given a user we can traverse the corresponding row in the user-product matrix and fill the corresponding missing values in the row and then rank them by their predicted rating. It avoids having to predict all the missing values in the matrix to recommend products to a particular user. Hence this method also improves the latency of the recommender system.

The problem formulation can be stated as follows - The user-product matrix denoted by R factorized into matrices P and Q. The dimensionality of the matrices is
R - $numOfUsers \times numOfMovies$
P - $numOfUsers \times featureSpace$
Q - $featureSpace \times numOfMovies$
where $featureSpace < min(numOfUsers, numOfMovies)$

A particular user u rates a movie i with the rating $r_{u,i}$ on a $0 - 5$ scale. The current values of the feature vectors are p_u^t and q_i^t. We now have to compute the new values the new values of the feature vectors p_u^{t+1} and q_i^{t+1} such that the factorization of the matrix holds true even with the inclusion of the new rating value

$r_{u,i}$ in the rating matrix R. We accomplished this by setting up the following optimization problem and then alternatively solving the minimization problem to obtain p_u^{t+1} and q_i^{t+1}.

$$p_u^{t+1} = argmin_p \quad 0.5 \times ||p - p_{u_t}^t||^2$$

$$\text{s.t} \qquad |p.q_{i_t}^t - r_{u_t,i_t}| = 0$$

where p_u^t is p_u at iteration t.

A similar formulation is made for calculating the factor vector q_i^{t+1}.

$$q_i^{t+1} = argmin_q \quad 0.5 \times ||q - q_{i_t}^t||^2$$

$$\text{s.t} \qquad |q.p_{u_t}^t - r_{u_t,i_t}| = 0$$

where q_i^t is q_i at iteration t.

The above formulations can be seen as follows, what is the minimal change we can make in the magnitude of the vectors p_u and q_i such that the product of the new feature vectors will be equal to the new product rating $r_{u,i}$

The above problem was solved as a Quadratic Programming problem. In the following section we will give an a brief overview of Quadratic Programming and Convex Optimization and the software packages we used. Following that we will elaborate on how our our problem formulation was solved using the above techniques.

7 Convex Optimization and Quadratic Programming

7.1 Definitions and Concepts

Often in machine learning where we would have to optimize the value of some function. That is, for a function $f : R^n \rightarrow R$, we want to determine $x \in R^n$ that minimizes (or maximizes) $f(x)$. There are several examples of optimization problems: least-squares, logistic regression, and support vector machines can all be framed as optimization problems. Though in the general case, finding the global minima or maxima of a function can be a challenging task, for a specific class of optimization problems known as convex optimization problems, we can find the global solution in most cases polynomially time.

More formally, a convex optimization problem is of the form

$$minimize \ f(x)$$

$$subject \ to \ g_i(x) \leq 0, \ \forall i = 1, ..., m$$

$$h_i(x) = 0, \ \forall i = 1, ..., p$$

where f is a convex function, g_i are convex functions, and h_i are affine functions, and x is the optimization variable.

A convex optimization problem is a Quadratic Program (QP) if the objective function f is a convex quadratic function and the inequality constraints g_i are still all affine. In other words, the problems have the form,

$$minimize \ \frac{1}{2}x^T Px + q^T x + d$$

$$subject \ to \ Gx \leq h$$

$$Ax = b$$

where again $x \in R^n$ is the optimization variable, $c \in R^n$, $d \in R$, $G \in R^{m \times n}$, $h \leq R^m$, $A \in R^{p \times n}$, $b \in R^p$ are defined by the problem, but we also have $P \in S^n_+$, a symmetric positive semi-definite matrix.

7.2 Solving the Optimization Problem

In this project we used a freely available open source scientific computing package for solving the quadratic problem. CVXOPT is a free software package for convex optimization based on the Python programming language. It can be used with a Python interpreter, or integrated in other software via Python extension modules. It makes good use of Pythons extensive standard library and on the strengths of Python as a high-level programming language, and makes scientific computing more simpler. It requires the mathematical package *numpy* to be installed beforehand for it work.

The CVXOPT QP framework expects a problem of the form defined in the previous section, defined by the parameters P, q, G, h, A, b where P and q are required, the other parameters are optional. If our optimization formulation is not in the above form then they must be manipulated to conform to the above form for us to be able to use the software package. For example, if a particular inequality constraint was expressed as $Gx \geq h$, then it should be rewritten as $-Gx \leq -h$ before we could use the package for our computation. Also, as we have used in our project, if we want to specify lower or upper bounds on x, an identity matrix can form part of G, since $x \leq u$ is equivalent to $Ix \leq u$. Here x is an internal variable being optimized and the solver has no explicit knowledge of x itself; everything is implicity defined by the supplied parameters. It is essential that the same variable order is maintained for the relevant parameters (e.g., q; h; b should correspond to variable x).

A generalized formulation of the optimization problem for our recommendation engine can be stated as follows.

$$w_{t+1} = argmin_{w \in R^m} \ \frac{1}{2}||w - w_t||^2$$

$$s.t \quad |w.x_t - y_t| = 0$$

$$and \ w \geq 0$$

The last condition ensures that none of the ratings we predict will be less than 0, which ensures that our results are more intuitive and more accurate. The above formulation can be interchangably used for calculating both the vectors p_u and q_i.

With n as the dimensionality of the vector w_t, the following mapping will be able to solve the above formulation using python's cvxopt package.

$P = I_n$(n-dimensional identity matrix), $q = -1 \times w_t$, $G = -1 \times I_n$, $h = 1 \times n$ dimensional matrix with every element equal to 0, $A = x_t$, $b = y_t$.

The following function call will yield the desired results for the optimization variable -

from cvxopt import solvers

sol = solvers.qp(P,q,G,h,A,b)

8 Implementation of the Recommendation Engine

A subset of the Netflix dataset has been used for experimentation purposes. We used 427 movies out of 17770 movies from the actual dataset. The number of users has been varied to observe the change in performance of the recommendation system with respect to the accuracy of the ratings and the number of recommendations provided to the end users.

The movie titles are stored in a text file which has the following fields in a comma separated format as shown below for each movie title:

<Movie_id, year of release, title of the movie >

The training set contains a set of files for each movie where the numerical_id is the integer id corresponding to the movie. Each such file consists of a series of entries for ratings from a variety of users as follows:

<User_id, rating, Date of rating >

Upon the <user, item >adjacency matrix that is constructed, we proceed to perform matrix factorization on this in order to obtain the user and item representations in the latent dimensional space. This is necessary because, conceptually we are trying to find the semantic space mapping of both the original user and item dimensions based on which we can cluster the item and user features respectively in the latent space.

The underlying principle is similar to k-nearest neighbor techniques based on Pearsons coefficient try to quantify user proximity and item proximity from the transactions recorded in the ¡user, item¿ matrix and grouping users together based on the overlap in the itemsets of transactions or items together based on the overlap of users who purchased a given item. However, they use a set based intersection measure to define overlap and construct a distance measure from it.

K-nearest neighbor techniques isnt appropriate for our system with a large <user, item >matrix because a very primitive experimentation showed that they are computationally expensive because in an n x n dimensional adjacency matrix, the pairwise distances are computed for each of the combinatorial pairs of users and items.

Matrix factorization, on the other hand, captures the property of functional synonyms. It measures the co-occurrence of items implicitly for each user row and the co-occurrence of user_ids for each item column based on which the factor dimensions are actually constructed. Algorithms like SVD (Singular Value Decomposition) can effectively capture the factor spaces from the Eigen space of the matrix (and its transpose).

However, traditional SVD algorithms are not used for this system owing to the following reasons: a) SVD can operate only on complete matrices without any sparsity. b) SVD recomputation may be expensive each time an update is done to the <user, item> adjacency matrix.

In our case, sparsity is induced from the fact that a user doesnt rate all possible items and recomputation of the factor space is necessary because more users keep rating more movies with time leading to constant updates to the <user, item> adjacency matrix.

Quadratic programming (QP) takes care of these issues as it factorizes an element from the user-item matrix at a time using the alternative least squares method. It takes an existing user vector and item vector as input and generates a refined user vector and an item vector. The functional synonymy or user/item correlation is captured by QP as follows:

- The user and item vectors fed to QP for an element ¡u, i¿ factorization are those vectors that have been obtained from the output of QP factorizations done on other entries in the matrix like <x, i>or <u, y>where x corresponds to any other user who has rated the same item i and y corresponds to other items which the user u has rated. This works in capturing functional synonym because every time a user or item vector is updated by QP, the output vector is retaining the properties it obtained out of the factorization of earlier elements while getting updated by the factorization of the current element.
- Alternative Least squares method tries to keep the updated user and item vectors as close as possible to the input vectors while doing the factorization which helps in the aggregate correlation property of the matrix being incrementally captured by the output user and item vectors.

The incremental update of the user and item vector by QP easily helps in capturing any online changes to the user-item matrix. QP helps in performing recommendation in the following way:

- The set of user vectors and item vectors it emits from the factorization of the existing ratings in the user-item adjacency matrix are stored.

- When a rating has to be given for a new <u, i>combination, we multiply the respective user vector u with item vector i and the scalar value obtained from their dot product is stored back in the adjacency matrix.

An implementation of this system consists of several steps:

- Construction of the sparse user-item matrix: The user-item matrix needs to be constructed from the dataset. Each entry of this matrix <u,i>corresponds to the rating a user u has given to a movie i . By parsing the training set files, these entries are created for all the users and movies.

- User_vector and item_vector initialization: After constructing the user-item matrix, the user and item vector are initialized to all 1s. They are not initialized to 0 because QP works by approximating an existing rating while factorizing it to the dot product of the user and item vectors. A 0 initialization of the vectors will make the dot product 0 and the intended factorization can never be done. Similarly, a fractional initialization is not done because while factorizing a rating value r, if one vector is a fraction, the other vector is correspondingly much larger to reach the constant product, r. But this may lead to very large recommendations. This can be explained with an example. Suppose a user u_2 has rated an item i_1 with a rating of 4. With a latent dimensionality 2, we could initialize u_2 and i_1 vector to <1,1>and <2,2>respectively or <0.1, 0.1>and <20, 20>respectively. In the latter case, if another element <u3, i3>entry of 2 has been factorized to a u_3 vector, <10, 10>and i_3 vector, <0.1, 0.1>respectively. The recommendation for an unseen combination $<u_3,i_2>$, where we are predicting the rating the user u_3 would give to item i_2, will be computed as dot product of u_3 vector and i_2 vector which would be <10,10>.<20,20>= 400. Now the usual ratings given to movies lie between 0 and 5. This constraint can be imposed by quadratic programming itself, but given its idea to stay as close to the input vectors as possible while applying alternative least squares method, we need to choose the seed vectors reasonably; hence we choose a vector of all 1s.

- Iteratively calling QP over non-sparse elements and updating vectors Quadratic Programming is iteratively called over all the entries in user-item adjacency matrix and the seed user and item vectors are constantly updated.

- Filling up the blanks for a given user from the dot product of the corresponding vectors: As mentioned before, in order to fill in an expected value of rating a user u would give an item i, we compute the scalar product of the user vector u and item vector i

9 Evaluation

All the experiments have been run on an Intel i5- 2.3 GHz, 4 GB 64 bit Windows 7 machine. We evaluated our system w.r.t

- The accuracy of the ratings,
- The number of recommendations

while varying the number of users and keeping the number of items constant. We also ran experiments with varying dimensionality of the latent space.

For example, in the 427 item dimensional (user x) movie matrix, a user # 97 has rated the following movies:

- 83,1983,Silkwood (biography, drama, thriller)
- 167,2004,The Chorus (drama, music)
- 175,1992,Reservoir Dogs (crime, drama)
- 270,2001,Sex and the City: Season 4 (drama, romance, comedy)
- 275,2002,Evelyn (drama)
- 353,2002,Life or Something Like It (comedy, romance)
- 413,2002,Igby Goes Down (comedy, drama)

We can see based on a genre observation that user #97 predominantly likes drama followed by comedy. With a latent space dimensionality of 5, and 100 x 427 matrix, following movies get a rating close to 5 among the recommended movies to user #97:
155,1975,The Strongest Man in the World (comedy, fantasy)
235,1996,Unhook the Stars(drama, romance)
271,1943,Saludos Amigos(animation, short, family)
293,2003,Lennon Legend: The Very Best of John Lennon(documentary, music)
336,1978,The Driver(action, crime, thriller)
Among the recommended movies, we have all the movies from the genres user #97 prefers. However, the fact that he likes drama the most out of all genres is not yet captured. So we increased the user dimensionality to 150. This is because, more user space creates room for a significant item overlap with other users. By the

property of collaborative filtering, we will be able to recommend more relevant movies as the correlated user space to user# 97 will be significantly more now.

With a latent space dimensionality of 5, and 150 x 427 matrix, following are the top recommendations:
8,2004,What the #$ Do We Know!?(documentary, comedy, drama)
28,2002,Lilo and Stitch (animation, adventure, comedy)
52,2002,The Weather Underground (documentary, history, war)
55,1995,Jade (action, crime, drama)
58,1996,Dragonheart (action, adventure, drama)

While preserving the diversity in the top genres user# 97 prefers, the system is also able to maintain the majority genre, drama, in the recommended set of movies.

We also ran experiments while varying the latent space dimensionality. An observation we made is that the set of recommendations never change because of varying the dimensionality from 5 to 10 or so. What varies is the confidence with which recommendations are made. To be precise, the value of the predicted rating for the recommended movies gets refined as we increase the dimensionality. This result makes sense because an increase in the latent feature space dimensionality cannot create more overlaps among the user or item sets. It can only create more entries in the user vector and item vector space eventually leading to a more accurate rating being predicted.

With a 100 x 427 dimensionality for the <user x item>adjacency matrix, there were initially only 59 entries indicating a high level of sparsity. The time to factorize this matrix using QP took 3.56 seconds indicating the speed of using QP as against a k-nearest neighbor technique which would have taken much longer owing to its quadratic computational complexity. QP based factorization is close to linear as the factorization of a given element is independent of the size of the matrix. By applying the dot product method to create unseen rating entries, the system took 0.096 seconds to fill in 3733 entries. The remaining entries are not filled because the existing 59 entries were not enough to create a user vector or an item vector for them. A non-sparse entry <u, i>can be induced only when both the user and item vectors for u and i can be created.

With a 150 x 427 matrix, it can be noted that there is a slight rise in the factorization and recommendation time owing to a larger matrix but it is still indicative of the quickness of the usage of QP.

The hit rate determines the extent to which movies from relevant genres have been made to a user. As shown, a larger user dimensional matrix (in this case, a user dimensionality of 150) has led to a hit rate of 80

Overall, our results indicate that QP based matrix factorization techniques, when used for movie recommendations to users, give a large set of induced entries (ratings), while being able to work with a very high level of sparsity in the user x item adjacency matrix. The hit rate based estimate gives a clear measure of how many relevant genres have been captured by the recommendation set. Latent space dimensionality can be used like an adjustable meter to get ratings with a higher confidence level. A possible extension to this system would be to build a feedback (loop based system) where a user can actually rate the unseen movie after the recommendation has been made and based on the feedback the system received from the user, matrix factorization can be re-run to refine the user and item vectors.

10 Evaluating Recommender Systems By Movie Genres

10.1 Need for such evaluation?

There has been a huge increase in the number of users who use internet and there is also a wide increase in the web searches that are being performed on a day to day basis and this can be accounted due to the increase amount of processed data available online. Millions and trillions of new offline data and information are being transferred online regularly. Having said these, it is not always the case that all information available online are totally reliable and valuable. This makes the process of fetching relevant and user satisfactory results from the web more difficult. This will also increase the number of times a particular user needs to refine his search to find what exactly he is looking for.

Recommendation systems were mainly suggested as a cure for this problem. Such systems proposes more relevant and useful results to users rather making them search for the same information again and again many times. These systems rely heavily on collaborative filtering techniques as discussed in the previous sections. As a brief review, a collaborative filtering technique for a recommender system that is based mainly on user information such as age, geographic location, or genre preference has resulted to be more effective and efficient than other systems.

The only reason that could make this system fail sometimes would be the cold-start problem or the information sparsity problem which actually is our current problem in our hand that we happen to solve using online matrix factorization techniques. Collaborative filtering mainly requires metadata about both the user and the product to start throwing out recommendations. In recent times there has been several attempts made to solve the data sparsity issue in such collaborative filtering techniques and ours in one such solution.

Once we have a recommendation engine that recommends movies (even with sparse initial information), then the next immediate pressing need would be a way to evaluate such recommendation engine. It makes more sense to think in a fashion that the newly recommended movies would be closely correlated in terms of many attributes to the previously rated movies by a particular user and one such attribute would be the movie genre. Every movie has genre information provided by movie experts, directors and movie reviewers.

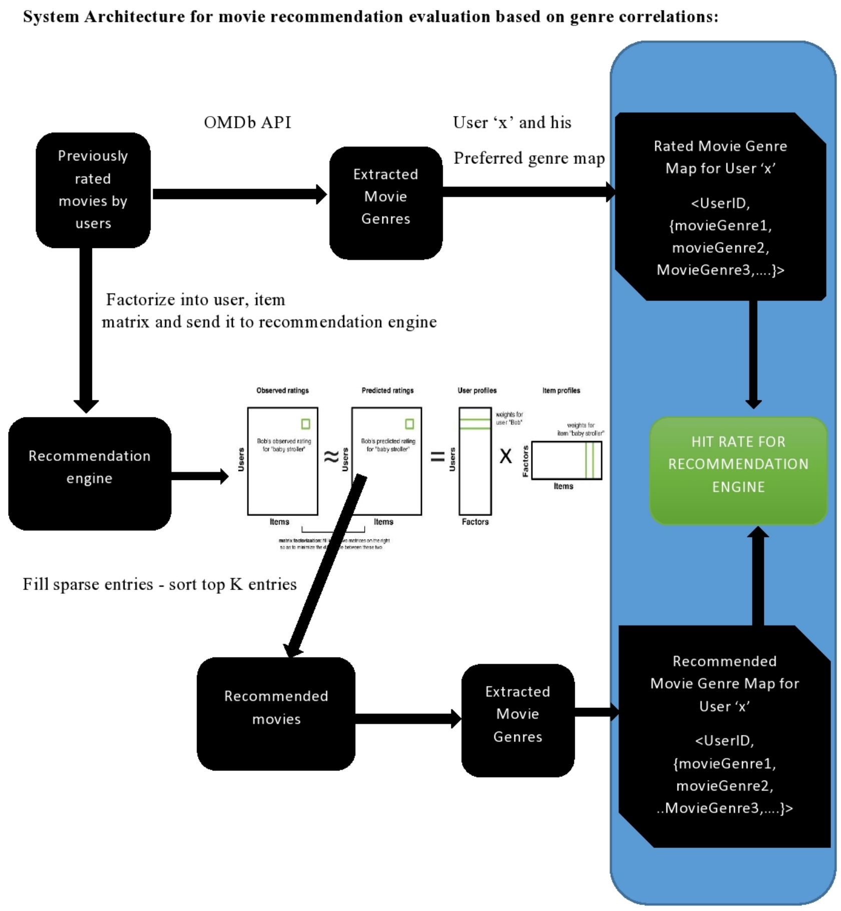

System Architecture for movie recommendation evaluation based on genre correlations:
OMDb API
User 'x' and his
Preferred genre map
Previously rated movies by users
Extracted Movie Genres
Rated Movie Genre Map for User 'x'
<UserID, {movieGenre1, movieGenre2, MovieGenre3,....}>
Factorize into user, item matrix and send it to recommendation engine
Recommendation engine
Observed ratings
Predicted ratings
User profiles
Item profiles
Users
Items
Factors
HIT RATE FOR RECOMMENDATION ENGINE
Fill sparse entries - sort top K entries
Recommended movies
Extracted Movie Genres
Recommended Movie Genre Map for User 'x'
<UserID, {movieGenre1, movieGenre2, ..MovieGenre3,....}>

Figure 4: System Architecture for movie recommendation evaluation based on genre correlations

The information contained in this attribute can be exploited to evaluate the goodness of our recommendation engine.

Our evaluation system first calculates genre correlations based on the genre combinations of each movie. It then applies the genre combination of all movies and user-preferred genres to the average rating of each movie based on the genre correlations. Finally, it computes the hit rate of the recommendation according to the computed recommendation points. Figure 4 illustrates the architecture of our system.

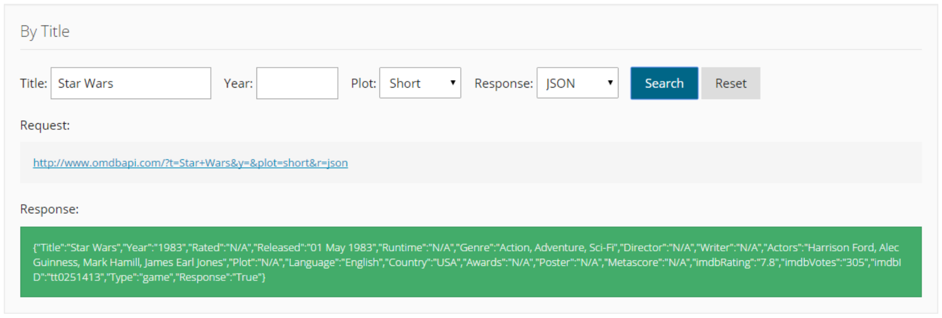
Steps to evaluate the recommendation engine by computing movie genre correlation:

- Extract all the movie genres using OMDb API- the Open Movie Database.

- for the newly recommended movie IDs for one particular user, compute the overlap with the movie genres using his previously rated movies.
- Compute Hit rate and thus evaluate the performance of recommendation system through movie genres.
- Extend the same technique for computing weighted hit rate, where we assign weights according to the degree of overlap in movie genres. For example:
  4.1) for a user X, the genres of movies which he has already rated: Action, Comedy, and Drama.
  4.2) A newly recommended movie to him has the following genre : Action, Crime, then we have 1 hit!
  4.3) If the newly recommended movie to him has the following genre : Documentary, Animation, then we have 0 hit as there is no overlapping genres!

11 Extracting movie genres using OMBb API

This is an excellent open database for movie and film content. The API usage is very simple.



Title: Star Wars Year: Plot: Short Response: JSON

Request:

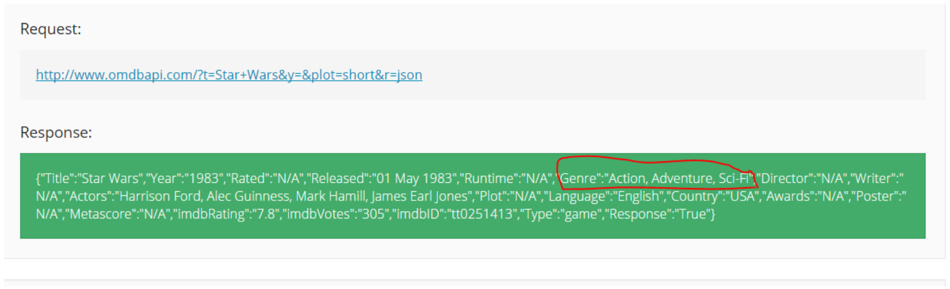
http://www.omdbapi.com/?t=Star+Wars&y=&plot=short&r=json

Response:

{"Title":"Star Wars","Year":"1983","Rated":"N/A","Released":"01 May 1983","Runtime":"N/A","Genre":"Action, Adventure, Sci-Fi","Director":"N/A","Writer":"N/A","Actors":"Harrison Ford, Alec Guinness, Mark Hamill, James Earl Jones","Plot":"N/A","Language":"English","Country":"USA","Awards":"N/A","Poster":"N/A","Metascore":"N/A","imdbRating":"7.8","imdbVotes":"305","imdbID":"tt0251413","Type":"game","Response":"True"}

Figure 5: OMDB url construction

Figure 5 shows the way in which the URL gets generated for a particular movie title. You can also see the movie genre attribute encoded in the response JSON Object generated. See Figure 6



Request:

http://www.omdbapi.com/?t=Star+Wars&y=&plot=short&r=json

Response:

{"Title":"Star Wars","Year":"1983","Rated":"N/A","Released":"01 May 1983","Runtime":"N/A","Genre":"Action, Adventure, Sci-Fi","Director":"N/A","Writer":"N/A","Actors":"Harrison Ford, Alec Guinness, Mark Hamill, James Earl Jones","Plot":"N/A","Language":"English","Country":"USA","Awards":"N/A","Poster":"N/A","Metascore":"N/A","imdbRating":"7.8","imdbVotes":"305","imdbID":"tt0251413","Type":"game","Response":"True"}

Figure 6: OMDB JSON response

Java module to extract the JSON object as a response given the Movie name as input is shown in Figure 7.

```
try {

        String selectedItem =
jListFilms.getSelectedValue().toString().replace("\\s+", "+");

        InputStream input = new URL("http://www.omdbapi.com/?t=" +
URLEncoder.encode(selectedItem, "UTF-8")).openStream();
        Map<String, String> map = new Gson().fromJson(new InputStreamReader(input,
"UTF-8"), new TypeToken<Map<String, String>>(){}.getType());

        String title = map.get("Title");
        String year = map.get("Year");
        String released = map.get("Released");
        String runtime = map.get("Runtime");
        String genre = map.get("Genre");
        String actors = map.get("Actors");
        String plot = map.get("Plot");
        String imdbRating = map.get("imdbRating");
        String poster = map.get("Poster");

        testForm tf = new testForm(title, year, released, runtime, genre, actors,
plot, imdbRating);
        tf.setVisible(true);

    } catch (JsonIOException | JsonSyntaxException | IOException e){
        System.out.println(e);
    }
```

Figure 7: Java Snippet to parse the JSON

11.1 Increase in dimensionality leads to increase in hit rate

The hit rate seems to increase significantly from 60% to 80% with increase in the number of users and this could be accounted for the fact that, genre corelation increases with the increase in the number of users.

12 Results

In our project we have developed a recommendation engine based on Collaborative filtering method. We used Online Matrix Completion technique to overcome the data sparsity problem of collaborative filtering method. We used Netflix Prize competition dataset to make recommendations. As mentioned previously, the recommendation system was evaluated with respect to two important aspects. The accuracy of the ratings and the number of recommendations were the two aspects. The evaluation was performed by varying the number of users and by keeping the number of items constant. The results section also outlines about the experiments that were performed by varying dimensionality of the latent space. All these results were outlined by means of bar graphs which are quite intuitive and all the results were as expected. Experiments were also performed by varying the latent space dimensionality. The results made it evident that the set of recommendations never changed by the change in the dimensionality. Confidence with which recommendations are made is the only thing that changes. The value of the predicted rating for the recommended movies gets improved as the dimensionality increase. The results also demonstrated speed of the quadratic programming against a k-nearest neighbor technique. The results also shows that QP based factorization is close to linear as the factorization of a given element is independent of the size of the matrix. The results also discusses about hit rate that determines the extent to which movies from relevant genres have been made to a user.

Overall Evaluation results:

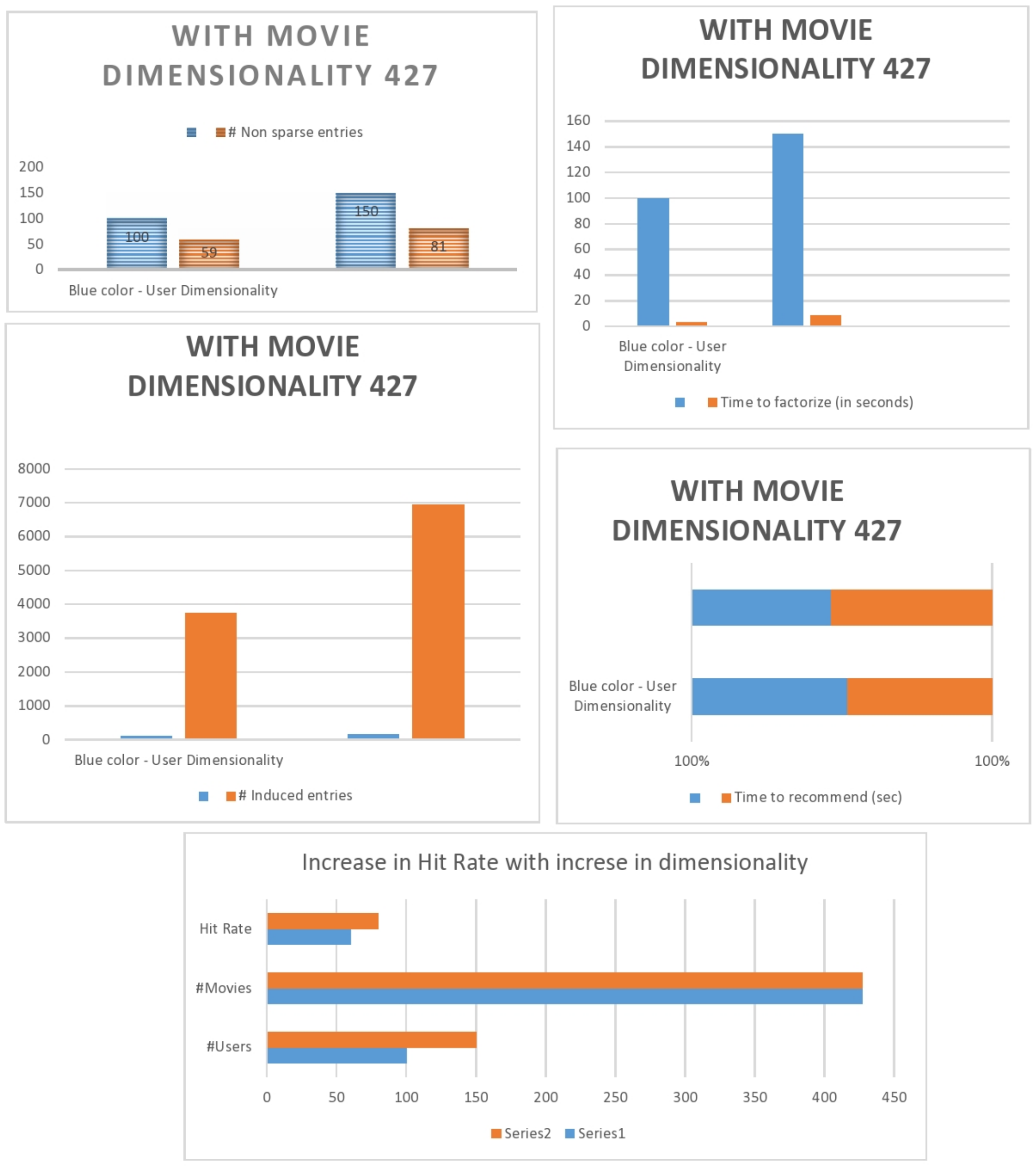


Figure 8: Overall Evaluation Results

13 References

[1] Mathieu Blondel, Yotaro Kubo, and Ueda Naonori. Online passive-aggressive algorithms for non-negative matrix factorization and completion. In Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, pages 96104, 2014. 3

[2]Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):3037, 2009. 2, 3

[3]Lara Quijano-Sanchez, Derek Bridge, Belen Daz-Agudo, and J. A. Recio-Garca. A case-based solution to the cold-start problem in group recommenders. In 20th International Conference on Case Based Reasoning, ICCBR12, pages 342356, 2012. 3

[4]Yuan Yao, Hanghang Tong, Guo Yan, Feng Xu, Xiang Zhang, Boleslaw K Szymanski, and Jian Lu. Dualregularized one-class collaborative filtering. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pages 759768. ACM, 2014. 2