Ranking by Alternating SVM and Factorization Machine

Shanshan Wu, Shuling Malloy, Chang Sun, and Dan Su

shanshan@utexas.edu, shuling.guo@yahoo.com,

sc20001@utexas.edu, sudan@utexas.edu

Dec 10, 2014

Abstract

In this project we focus on two collaborative ranking algorithms: Alternating SVM (AltSVM) and Factorization Machine (FM). Our project is divided into three main tasks. First, understand AltSVM and implement it for the first time, since AltSVM has not been implemented and evaluated before. We also need to come up with a method to choose good initial values and stopping criteria for this algorithm. Second, understand FM and figure out how to use the FM library (libFM). Third, performance evaluation and comparisons based on real world datasets. We show that on the MovieLens 100k dataset, AltSVM achieves 72% prediction accuracy using only 30% randomly selected pairwise comparisons in the training set. Although FM achieves a higher accuracy (78%), AltSVM has an advantage of using only pairwise comparison information instead of actual ratings. We also show that to improve the performance of AltSVM, one possible way may be to enhance its ability of handing unbalanced input data.

1. Introduction

Recommendation and ranking system is an important topic in data mining and machine learning field. Given incomplete users' preferences over a list of items, the goal is to infer a complete users' preference ranking for the entire list. So far there are many ranking techniques developed to improve ranking accuracy and efficiency. Typical ranking methods include classic nearest neighbor, matrix factorization,

support vector machines, and factorization machines. These techniques are different from each other in terms of the model, the data format, performance scalability, etc. In this project we will focus on two collaborative ranking approaches: Alternating SVM (AltSVM) [1] and Factorization machine (FM) [2].

AltSVM is a heuristic algorithm recently proposed by Prof. Sujay Sanghavi. Compared to the widely-studied ranking algorithms that use numeric ratings as the training data, AltSVM takes only pairwise preference comparisons of the type "user i prefers item j_1 over item j_2" as the input data and outputs a complete preference ranking for each user. The motivation of using pairwise comparisons instead of actual ratings is twofold. First, pairwise comparisons are more widely available. Frequently, the available data presented to us would not be explicit ratings, but in the form of comparisons which implicitly indicate the user's preference. For example, book A is purchased when both book A and book B are displayed. A particular song is added to the wish list among a collection of popular songs. Second, pairwise comparisons are more reliable. Different users have different standards of assigning numeric scores, so the same score given by a user who always gives extreme scores can be interpreted differently from another who usually gives moderate scores, indicating different degrees of preference. It is worth noting that [3] also proposes a ranking algorithm that uses only pairwise comparison information. [3] is developed based on the classical Bradley-Terry-Luce model [4] for pairwise preferences of a single user, while AltSVM is developed based on a Soft Margin SVM model. A detailed formulation of AltSVM is provided in Section 2. This part of work is mainly done by Shanshan Wu.

FM is a general regression model based on the idea of matrix factorization. Matrix factorization [6] uses the dot product to capture the interaction between user and item on latent factors to approximate the user's rating, allowing the prediction of relationship between two categorical variables. Later on, SVD++ [7], FPMC [8,10], and timeSVD++ [9] are developed to augment the matrix factorization with additional terms and also take the non-categorical variables into account. However, all of these models predict the interaction only between users and items and the latent factors are often obtained based on observed ratings only due to the sparseness in user-rating matrix. Also, some of the models are specialized for specific tasks, so every time the feature space changes, a new specialized model is probably needed to be derived. Due to the ability to solve such problems, FM is also examined with libFM in this project to verify its prediction accuracy. It provides a regression model that characterizes not only the impact of each feature on the target rating but also the impact resulting from the interactions between any two features in the feature space such as user and occupation, item and gender. Unlike polynomial regression that directly estimates the pairwise interaction parameters, the FM uses a low-rank approximation of the

parameter matrix by factorizing each into the inner product of two vectors that are easy to be estimated. Furthermore, it is actually a generic model that can be transformed to other specialized factorization models like matrix factorization and SVD++. The implementation of FM is provided by the software libFM [5]. Despite such convenience, two main tasks remain for us to use factorization machine. First, the rank k of the matrix obtained by factorizing the parameter matrix is unknown and relies on us to specify. For a fixed feature space, small k is likely to underfit the model while large k can lead to overfitting. Second, since the model is general, the responsibility is upon us to select the feature space that is most useful in predicting rankings. With different combinations of the rank values and the feature space, it is possible to generate the best prediction result on a single dataset. A detailed formulation of FM is provided in Section 3. This part of work is mainly done by Shuling Malloy and Chang Sun.

We use the MovieLens 100k dataset to evaluate the performance for both algorithms. It contains "100,000 ratings (1-5) from 943 users on 1682 movies" with 19 movie genres. There are at least 20 ratings per user. Besides the rating information, it also includes the information of timestamp, gender, occupation and age. Since AltSVM and FM require different forms of input data, a detailed explanation of how we pre-process the MovieLens dataset for each algorithm is provided in Section 2 and Section 3, respectively.

2. Alternating SVM (AltSVM)

AltSVM is a heuristic algorithm recently proposed by Prof. Sujay Sanghavi. It takes pairwise preference comparisons as the input data and outputs a complete preference ranking of each user. Our goal is to understand this algorithm, implement it, and evaluate its performance.

To achieve our goal, we have to face two challenges. First, AltSVM is a new algorithm that has not been implemented or evaluated before, so we do not have a baseline solution that can serve as a reference. Furthermore, AltSVM is not a complete algorithm with open questions such as how to choose good initial values and good stopping criterion. As a result, we need to come up with different methods and take the approach of trial and error.

This part of work is mainly done by Shanshan Wu. Chang Sun helps pre-process the MovieLens 100k data. Specifically, Chang transforms the data from actual ratings to pairwise comparisons.

2.1 Setting and Algorithm

Setting and notations. We consider the following problem. For n users and m items, suppose there is an underlying low-rank rating matrix $L^* \in \mathbb{R}^{n \times m}$ with rank r. Each entry L^*_{ij} is the rating of item j by user i. We observe k pairwise comparisons of the type "user i prefers item j_1 over item j_2". These pairwise comparison information is represented by a set of triples $\Omega \in ([n] \times [m] \times [m])^k$ such that $(i, j_1, j_2) \in \Omega$ indicates $L^*_{ij_1} > L^*_{ij_2}$. Our goal is to recover the rankings of each user, i.e., find an approximated L such that $\operatorname{argsort}(L_{i.}) = \operatorname{argsort}(L^*_{i.})$ for every user $i \in [n]$.

Algorithm and interpretation. Under the low-rank assumption, we can decompose L as UV^T, where $U \in \mathbb{R}^{n \times r}$ and $V \in \mathbb{R}^{m \times r}$. The j-th row of V can be interpreted as r attributes or features that describe item j. Similarly, user i is represented by the i-th row of U. Instead of solving L directly, AltSVM uses the idea of alternating optimization: fix V, optimize U (User's Problem), then fix U, optimize V (Item's Problem), and iterate until certain stopping criterion is reached.

The User's Problem is formulated in (1). Since the pairwise comparisons are given for each user, we can optimize each row of U independently. Specifically, the i-th row $U_{i.}$ is optimized using $\Omega_{i.}$, the pairwise comparisons associated with user i.

$$
\begin{aligned}
U_{i.} \leftarrow \arg \min_{u \in \mathbb{R}^r, \delta} \; & \frac{1}{2} u^T u + \lambda \sum_{(j_1, j_2) \in \Omega_i} \delta^{(j_1, j_2)} \\
\text{s.t. } & u^T (V_{j_1.} - V_{j_2.}) \geq 1 - \delta^{(j_1, j_2)}, \\
& \delta^{(j_1, j_2)} \geq 0, \; \forall (j_1, j_2) \in \Omega_i.
\end{aligned}
\tag{1}
$$

The above optimization problem is similar to an SVM problem: same objective functions but with different constraints. Recall that the SVM algorithm finds a maximum-margin hyperplane that separates two classes. Its constraint is given by $y_i(w^T \phi(x_i) + b) \geq 1 - \delta_i$, where y_i is the class label ("+1" or "-1") and b denotes the intersection. Compared to the SVM algorithm, we can interpret (1) as finding the maximum-margin

hyperplane for one-class data with the constraint that this hyperplane must pass through the origin, as shown in Figure 1.

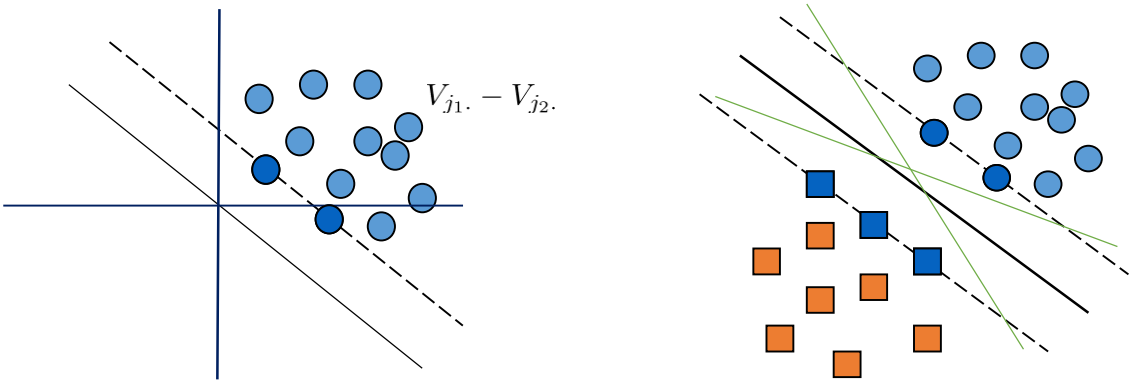


Figure 1: A comparison between the User's Problem (left) and the SVM algorithm (right).

The Item's Problem is given in (2). Unlike the User's Problem, where users are optimized independently, the items are all coupled and hence they have to be optimized together using all the pairwise comparisons. The Item's Problem can be interpreted in the same way as the User's Problem, except that all the math operations are between matrices instead of vectors.

$$
\begin{aligned}
V \leftarrow \arg\min_{V\in\mathbb{R}^{m\times r},\xi} \frac{1}{2}||V||_F^2 + \gamma \sum_{\omega\in\Omega} \xi^{(\omega)} \\
\text{s.t. } \langle V, X^{(\omega)} \rangle \geq 1 - \xi^{(\omega)}, \\
\xi^{(\omega)} \geq 0, \ \forall \omega \in \Omega \\
X^{(\omega)}_{j\cdot} = \begin{cases} U_{i\cdot} & j = j_1 \\ -U_{i\cdot} & j = j_2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{2}
$$

2.2 Implementation Issues for Large Datasets

In this section we give a detailed description of the three main issues that we have encountered when implementing AltSVM in Matlab. As mentioned previously, this is the first time that AltSVM is implemented. Although our final implementation is still not fast enough, it is yet the best implementation we can get so far.

Quadratic programs. The User's Problem (1) and Item's Problem (2) are two quadratic optimization programs. To implement them, we try three methods, and each time we look for an optimization solver that is more efficient in solving problems of the specific form as (1) and (2). Table 1 lists the time needed to run a simple case and the main issues for each method.

We first try CVX, but it is slow and suffers the out-of-memory issue when we increase the rank or the number of comparisons in the training data. This is because CVX is a general optimization solver. It uses interior point methods to solve the optimization program, making it quite slow solving specific quadratic problems in our case. Next, we try Quadprog, which is a Matlab toolbox specifically designed for solving quadratic programs. It suffers the same problem as CVX, that is, slow and memory exhausting.

To deal with the out-of-memory issue, we take a close look at the special structures of the optimization problems (1) and (2). On one hand, we find that problem (2) is the bottleneck. Unlike problem (1), which is solved independently for each user, program (2) finds the optimal V using all the pairwise information in Ω. For n users and m items, if N pairwise comparisons per user are used, with rank r approximation, problem (2) solves for $mr + nN$ variables under $2nN$ inequality constraints. Besides, even writing down all the $X^{(\omega)}$s requires $O(nNmr)$ memory, which explains why the memory issue becomes so severe for large dataset. On the other hand, we find that although each $X^{(\omega)}$ is an m-by-r matrix, it is quite sparse with only $2r$ non-zero elements. This indicates that instead of storing $X^{(\omega)}$ directly for each constraint, a more efficient method would be to re-formulate the constraint so that only the non-zero elements of each $X^{(\omega)}$ needs to be stored. To achieve this goal, we formulate the constraints as a sum of hinge loss functions and then perform gradient descent to get the optimal solution. Similar to SVM, Problem (2) can be formulated as an unconstrained quadratic program:

$$\min_{V \in \mathbb{R}^{m \times r}} f(V) = \min_{V \in \mathbb{R}^{m \times r}} \frac{1}{2}||V||_F^2 + \xi \sum_{w \in \Omega} \max(0, 1 - \langle V, X^{(\omega)} \rangle) \tag{3}$$

Let $S = \{w \in \Omega : \langle V, X^{(\omega)} \rangle < 1\}$, then the subgradient of $f(V)$ is given by

$$\partial f(V) = V - \xi \sum_{w \in S} X^{(\omega)} \tag{4}$$

As shown in (4), calculating the subgradient of a hinge loss function needs only the non-zero elements of $X^{(\omega)}$, which hence dramatically decreases the memory demand. Similar to Problem (2), we also formulate the constraints of Problem (1) as hinge loss functions and then apply gradient descent method.

Two sets of parameters need to be turned for the algorithm. The first set includes the penalty variables λ and ξ. To determine a fairly good choice, we hold out a validation set, evaluate various combinations of λ and ξ for several cases. We find that the results to be fairly robust to the choices of λ and ξ. Considering the time limit, we decide to use $\lambda = 1$ and $\xi = 1$ in our experiments. The second set of parameters are

the step sizes for the gradient descent method. After some attempts, we decide to use backtrack line search with parameters $\alpha = 0.1$ and $\beta = 0.5$.

Table 1: A comparison of three optimization solvers.

Methods	Time needed to run a simple case on the MovieLens 100k dataset (use rank 2 and 20 pairwise comparisons per user)	Main problem
CVX	5.2 hours	Quite slow; out of memory if we use more than 25 comparisons per user even for a simple rank 2 case.
Quadprog	Out-of-memory for this case	Similar problem to CVX.
Hinge loss and gradient descent with backtracking line search	32 seconds	Slow when we increase the rank value or the number of pairwise comparisons; need to tune step size for gradient descent.

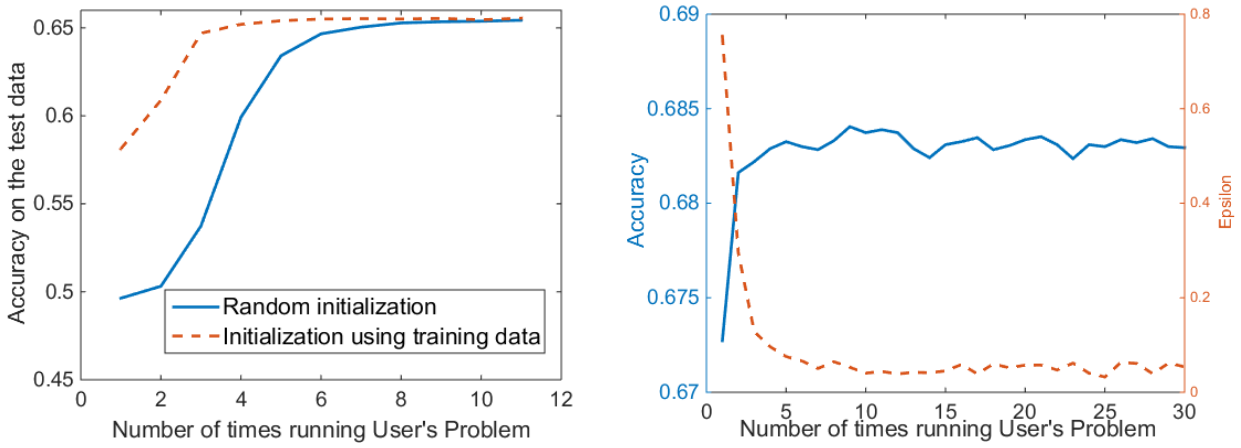
Performance metric. The MovieLens dataset contains actual ratings, so we first hold out 20% of the ratings as the test set and transform the rest 80% ratings into pairwise comparisons. Motivated by the "Kendall tau distance", we evaluate the performance by calculating the proportion of pairwise comparisons that we correctly infer user's preference. In other words, let C be the set of all possible pairwise comparisons in the test set, then the prediction accuracy is measured as

$$\text{Accuracy} = \frac{\text{Number of concordant pairs}}{|C|}$$

Since the actual ratings are given in discrete values from one to five, we are also interested in the algorithm's performance on inferring strong preferences, i.e., pairs with a rating difference greater than one. In this case, we measure the above metric and call it the "restricted accuracy".

Initial values and stopping criterion. A naive way is to initialize U and V randomly. However, this random initialization may result in a slow convergence rate. In our experiments, we try a more sophisticated approach: use the training data to initialize U and V so that the resulting low-rank matrix $L = UV^T$ satisfies as many as possible the known pairwise comparisons. Here is how we find the initial values: start with an all-zeros matrix $L \in \mathbb{R}^{n \times m}$, for every pairwise comparisons in the training data $\omega = (i, j_1, j_2) \in \Omega$, add +1 to the entry L_{ij_1} and -1 to the entry L_{ij_2}. The initial U and V are chosen as the rank-r factorization of L. The two initialization methods are compared in Figure 2a. We see that the more sophisticated initialization method speeds up the algorithm.

AltSVM is a heuristic algorithm that optimizes U and V alternatively, so an intuitive stopping criterion is to stop running the algorithm when $L = UV^T$ converges. In other words, the algorithm is stopped when the values of L between the current round and the next round does not change too much. Mathematically, we evaluate the difference of L between two consecutive rounds as ΔL, and stops the algorithm when $||\Delta L||_F / ||L||_F < \epsilon$. In Figure 2b we plot ϵ and the achieved accuracy versus the number of times we optimize U and V. We find that a smaller ϵ does not always correspond to a better performance. Besides, it is hard to tune ϵ for every case: a small ϵ would require more training time while a large ϵ may not give the best performance. Therefore, in our experiments, a different stopping criterion is used: for each round, we evaluate the accuracy on the training data and stop executing the algorithm when the accuracy starts to decrease.



(a) (b)

Figure 2: (a) A comparison of two initialization methods. (b) Relation between ϵ and the achieved accuracy.

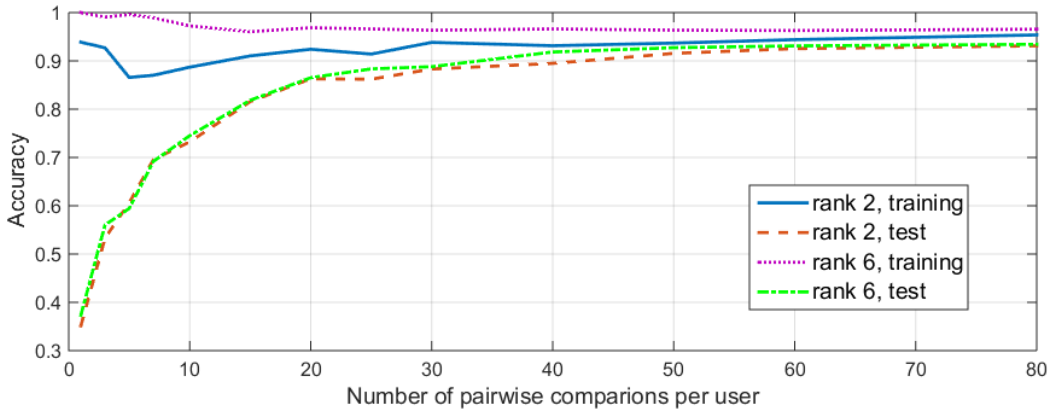
2.3 Experiments and Results

We evaluate the performance based on two datasets: artificially generated data and real-world data. The motivation behind the synthetic experiment is as follows. Because AltSVM is formulated based on the assumption that the underlying rating matrix has rank r, a natural question would be: if this assumption is true, can AltSVM correctly infer all user's preferences? To answer this question, we need to use artificial data, since it is difficult to know the actual rank of the underlying rating matrix associated with a real-would dataset.

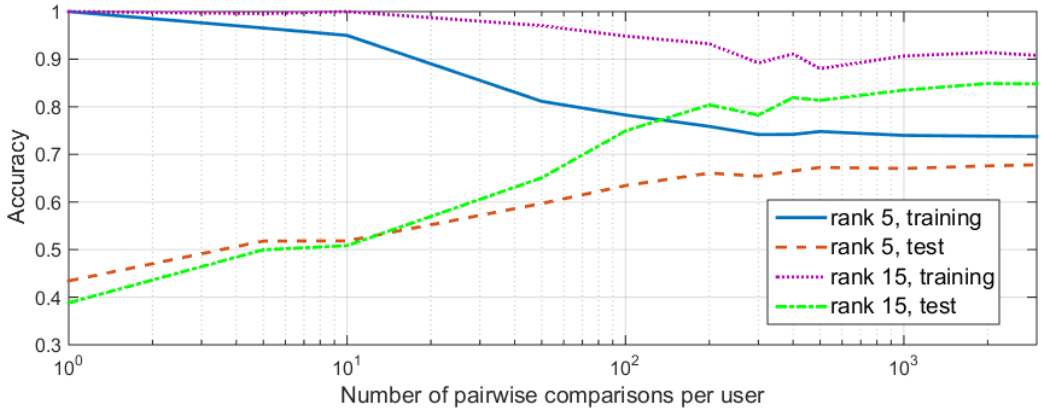
Synthetic experiments. We consider a simple case with 100 users and 100 items. The rank-r rating matrix $L = UV^T$ is generated by assigning i.i.d. Gaussian distributed values to the entries of U and V. We hold out 20% of the ratings as the test set. Two cases are considered: $r = 2$ and $r = 20$.

For the first case, we run AltSVM with rank 2 and 6 using randomly selected pairwise comparisons in the training set. As shown in Figure 3a, 90% accuracy is achieved on the test data using only 30 pairwise comparisons per user, which is a quite small number since each user has approximately 3k pairwise comparisons in total. On the other hand, since we know the actual rank is 2, using rank 6 in AltSVM would overfit the data, which is why there is a slight increase in the training accuracy in Figure 3a.

For the second case, the actual data has rank 20, while we run AltSVM with rank 5 and 15, which would underfit the data. Compared to the first case, this situation may be a more realistic model of the real-world data. As shown in Figure 3b, given the same number of pairwise comparisons per user, modelling data with a higher rank achieves a better accuracy on both the training and test data. Furthermore, as the number of comparisons used for training increases to the maximum value (about 3k), the test accuracy also reaches its maximum: 85% for rank 15 and 68% for rank 5. We also observe that the training accuracy decreases as the number of comparisons per user increases. This is because with more pairwise comparisons in the training set, find an approximated L that satisfies all the comparisons becomes more difficult. As an extreme case, if only one pairwise comparison is given for each user, then it is very easy to find an L that satisfies all the given comparisons. In fact, after the initialization process, the initial L can achieve 100% accuracy on the training data.



(a)



(b)

Figure 3: Simulations using artificially generated rating matrix with actual rank 2 (a) and rank 20 (b).

Real-world data. We use the MovieLens dataset to evaluate the performance. It contains 100k ratings from 943 users on 1682 movies. The ratings are integers from one to five, with at least 20 ratings per user. We hold out 20% of the ratings as test set. When converting ratings into pairwise comparisons, we consider pairs with different ratings and ignore pairs if their ratings are equal. We also evaluate the accuracy on the "restricted" pairs whose ratings differ by two or more.

According to the synthetic experiments, two parameters would affect the prediction accuracy: the rank number used to model the underlying rating matrix, and the number of pairwise comparisons used as the training data. In Figure 4 we evaluate the accuracy versus rank using 50 and 3000 randomly selected pairwise comparisons per user. We observe a nice underfit-overfit tradeoff for both cases: a small rank would underfit the data, resulting in a low accuracy both on the training and test data; as rank increases, the model complexity also increases, hence a large rank tends to overfit the data, resulting in a high accuracy on the training data and a low accuracy on the test data. One way to deal with overfitting may be to increase the size of the training data, i.e., increase the number of pairwise comparisons. This actually does not work for our case (see Fig. 6 and the detailed explanation). Nevertheless, we observe from Figure 4 that for a small training size, a rank number smaller than 10 would be sufficient to model the MovieLens dataset.

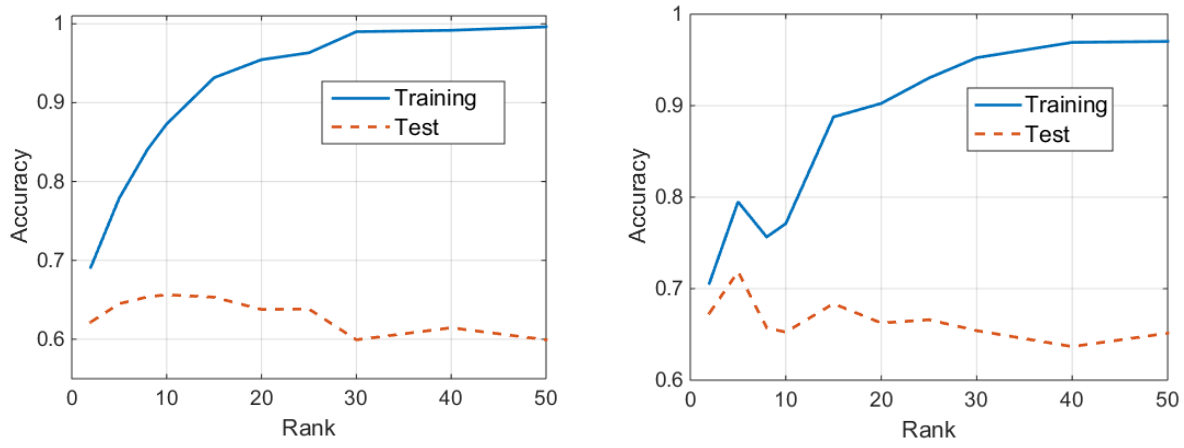


Figure 4: Accuracy versus rank using 50 (left) and 3000 (right) pairwise comparisons per user.

In Figure 5 we plot the prediction accuracy versus the number of randomly selected pairwise comparisons per user. The rank number is tuned from 2 to 10. In other words, given a training set, we fit the data with a rank from 2 to 10 and report the best performance. We also evaluate the restricted accuracy using pairs indicating strong preferences in the test data, i.e., pairs that have a rating difference greater than one. The performance of AltSVM is compared to that of a standard matrix completion algorithm. For each pairwise comparison, the matrix completion algorithm takes the difference between their actual ratings as the training data. The prediction accuracy achieved by matrix completion on the same dataset is borrowed from [3]. As shown in Figure 5, AltSVM performs quite well: although it only uses the sign of each pairwise comparison (i.e., information that whether user i prefers item j_1 over j_2), it achieves a better prediction accuracy than the case when we know the actual rating difference.

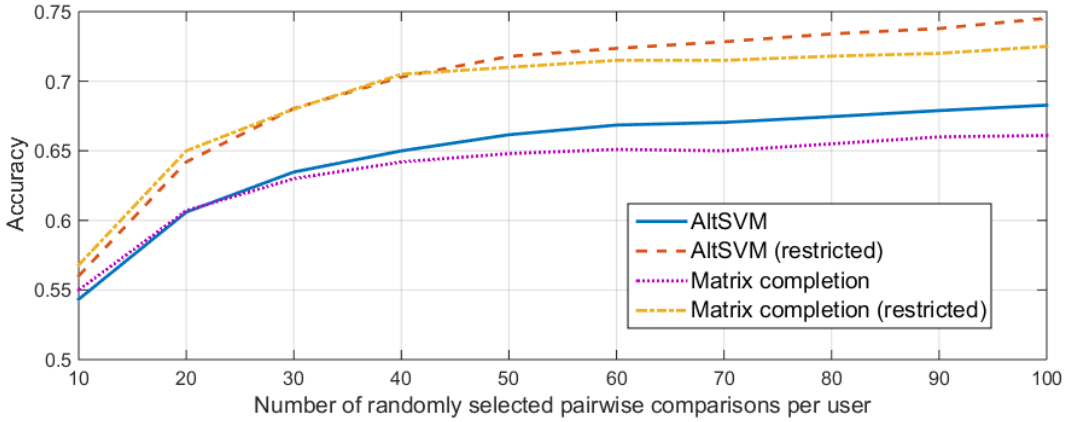


Figure 5: Prediction accuracy for different number of pairwise comparisons per user.

As shown in Figure 5, AltSVM can achieve a prediction accuracy of 68% on the test data (75% accuracy on the restricted test data). We also observe that the prediction accuracy increases when we use more pairwise comparisons. Given this promising trend, we are then interested evaluating the performance with a larger size of training data. A case of large-size training data is shown in Figure 4b, in which an accuracy of 72% (80% for the restricted case) is achieved when we use rank 5 and 3000 randomly selected pairwise comparisons per user. Then a natural question would be if we use all the available pairwise comparisons, can we achieve a better prediction accuracy? The answer is, unfortunately, no. In Figure 6 we use all the pairwise comparisons to train the model with different ranks. As shown in Figure 6, the best prediction accuracy is 68% on the test data, which is lower than the case we use only 3000 pairwise comparisons per user.

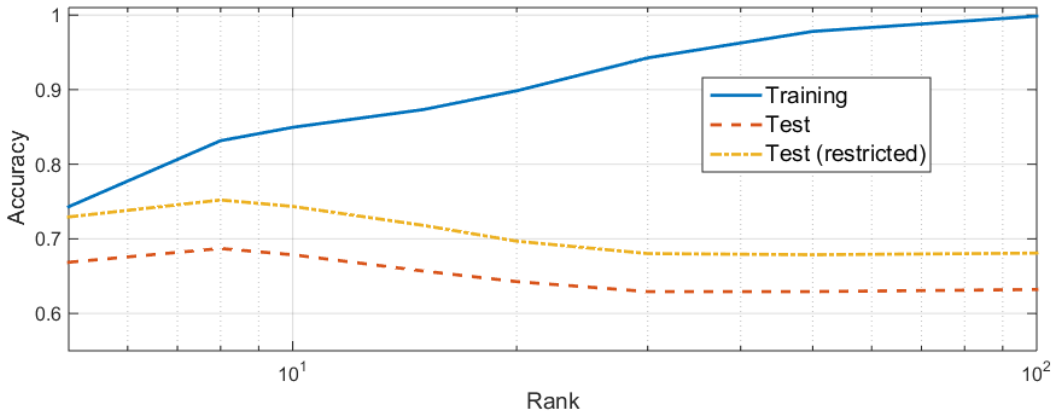


Figure 6: Accuracy achieved when all the pairwise comparisons in the training set are used.

There are two possible reasons why using all the available pairwise comparisons does not improve the prediction accuracy. First, more pairwise comparisons do not necessarily mean more information. As an simple example, (i, j_1, j_2) and (i, j_2, j_3) already indicates that user i prefers item j_1 over j_2, so adding one more pair of (i, j_1, j_3) will not increase the amount of information that we already had. Second, the available pairwise comparisons are quite unbalanced from user to user. For the MovieLens 100k dataset, after picking 80% of the ratings as the training set and converting them into pairwise comparisons, we get more than 4 million pairs in total. However, they are quite unbalanced for each user: user 13 has about 105k pairwise comparisons while user 849 has only 32 pairs (note that although each user has rated at least 20 movies, but we ignore equal ratings when converting them into pairwise comparisons). When running AltSVM, all the pairwise comparisons are treated equally in the User's Problem and Item's Problem. Therefore, by increasing the number of pairwise comparisons per user in the training set, we are actually giving a larger weight on users with more ratings, which possibly causes a performance

degradation. Note that for a real-world dataset, this unbalance characteristic always exists. However, compared to using the actual ratings to infer user's preferences, this unbalance problem becomes severer when the ratings are converted to pairs (e.g., 105k versus 32).

3. Factorization Machine (FM)

This part of work is mainly done by Shuling Malloy and Chang Sun. Dan Su is involved in some of the discussions.

3.1 Algorithms

FM is a general regression model that can incorporate any real-valued features. Its model contains three parts: the constant term, also known as the global mean; the input variables with associated weights, which are exactly the same as that in a linear regression model; the pairwise interactions of all input variables. A second-order FM model can be constructed as follows:

$$\hat{y}(x) = w_0 + \sum_{i=1} w_i x_i + \sum_{i=1} \sum_{j=i+1} \langle v_i, v_j \rangle x_i x_j$$

where x_i refers to each input variable, $x_i x_j$ represent the interaction between any two variables.

The important difference with polynomial regression is that the pairwise interaction parameter is not directly estimated, but factorized into a dot product $\langle v_i, v_j \rangle = \sum_{f=1}^{k} v_{i,f} * v_{j,f}$, where k is the dimensionality of the factorized matrix V. Thus, the interaction parameter matrix can be represented as $\mathrm{V} * V^t$ if k is sufficiently large. Such way of factorization is one of the biggest novelties of FM since the parameter of the interaction is usually hard to estimate, especially in higher-order variable interaction circumstances.

The algorithm can be proved to be computed in linear time if we rewrite it as

$$\hat{y}(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \frac{1}{2} \sum_{f=1}^{k} [\left(\sum_{j=1}^{p} v_{j,f} x_j \right)^2 - \sum_{j=1}^{p} {v_{j,f}}^2 x_j^2$$

As can be seen, the number of parameters of the model is $1 + \mathrm{p} + \mathrm{kp}$, where p is the number of input variables. So the total number of parameters needed to estimate is linear to both number of variables and k. Thus the complexity is $\mathrm{O}(\mathrm{kp})$ instead of $\mathrm{O}(\mathrm{k}p^2)$ in polynomial model without the factorization.

To obtain the ultimate rating, there are three groups of parameters needed to estimate: $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^p$ and $\mathbf{V} \in \mathbb{R}^{p*k}$. The FM algorithm aims to estimate such parameters using the following steps:

a. Construct the feature space;

b. Choose optimization function and regularization method;

c. Choose learning algorithm to infer over FM model parameters.

3.1.1 Feature space construction

FM constructs the feature space differently from the typical factorization models such as SVM. In a feature matrix, both the users and items can be regarded as features. Each row in the matrix represents a feature vector indicating that a particular user rated a particular item with specific background information for features such as gender, zip code and rating time point. Each column represents an input variable.

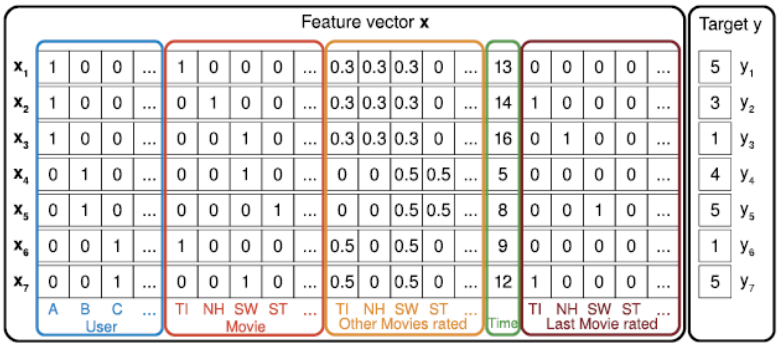


Figure 7: An example of the feature space of FM (this plot is borrowed from [2]). Input variables are grouped into User, Movie, Other Movies Rated (normalized so that they sum up to 1), Time and Last Movie Rated. The first row can be read as: user A rated TI movie to score 5 at time 13 with three movies rated in total and TI as the first rated movie.

3.1.2 Optimization function and regularization method

The optimization of model parameters begins with the loss function which aims to minimize the total loss over the observed data,

$$\mathrm{OPT(S)} = \underset{\Theta}{\operatorname{argmin}} \sum_{(\boldsymbol{x},y \in S)} l(\hat{y}(x|\Theta), y)$$

where $\boldsymbol{x}$ is the input variables, Θ is the model parameters required to estimate and $\hat{y}(x|\Theta)$ is the estimated rating and y is the observed rating.

Depending on the task, three kinds of loss functions are used by Rendle [2]. For regression,

$$l^{ls}(yhat(x), y) = (y - yhat(x))^2$$

For binary classification (y $\in$ {−1,1}),

$$l^{C}(yhat(x), y) = \ln\left(e^{yhat(x)*y} + 1\right) - yhat(x) * y$$

For ranking, a pairwise classification loss is used based on Bayesian Personalized Ranking (BPR-OPT) [2],

$$l^{R}\left(yhat(x), x^{(1)}, x^{(2)}\right) = \prod_{(x^{(1)}, x^{(2)}) \in S} \sigma\left(yhat(x), x^{(1)}, x^{(2)}\right) P(\Theta)$$

where σ is the logit link $\sigma(x) = \frac{1}{1+e^{-x}}$ and $P(\Theta)$ is the prior of model parameters. $x^{(1)}, x^{(2)}$ are pairs of instance vectors $(x^{(1)}, x^{(2)}) \in S$.

To avoid the overfitting problem caused by fitting the model with large k, L2 regularization is applied.

$$\mathrm{OPTREG}(\mathrm{S}, \lambda) = \underset{\Theta}{\operatorname{argmin}}\left(\sum_{(\boldsymbol{x}, y \in S)} l\left(\hat{y}(x|\Theta), y\right) + \sum_{\theta \in \Theta} \lambda_\theta \theta^2\right)$$

Note that since we have three kinds of parameters in the model, different λ_θ should be used for different parts of the model. Also, parameters are grouped in libFM, which means each feature has its own parameter for each of three kinds of parameters. Having a lot of λ_θ can be good or bad. On the one hand, the prediction quality may highly depend on the choices of λ_θ in the certain learning algorithm such as SGD, so setting different λ_θ for different parts of model and features could enhance the prediction accuracy. On the other hand, with many parameters already in the model, it may take a long time for the learning algorithm to search for the best λ_θ with cross-validation.

Despite a large number of parameters and λ_θ to estimate, one advantage of FM is that the gradient for each parameter when direct optimizing the loss function is very easy to compute. For example, if the regression loss function is used, then

$$\mathrm{h}(\theta) = \frac{\partial(\hat{y}(x|\Theta), y)}{\partial\theta} \begin{cases} 2(\hat{y}(x|\Theta) - y) * 1 + 2\lambda_0 w_0 & if\ \theta \text{ is } w_0 \\ 2(\hat{y}(x|\Theta) - y)x_i + 2\lambda^w_{\pi(i)} w_i & if\ \theta \text{ is } w_i \\ 2(\hat{y}(x|\Theta) - y)(x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{j,f} {x_i}^2) + 2\lambda^v_{f,\pi(i)} v_{j,f} & if\ \theta \text{ is } v_{i,f} \end{cases}$$

where λ_0 is the regularization parameter for the constant part, $\lambda^w_{\pi(i)}$ is for each single input variable and $\lambda^v_{f,\pi(i)}$ is for factorized vector element

3.1.3 Learning algorithms

Three learning algorithms are proposed for FM: Bayesian posterior estimate, Stochastic Gradient Descent (SGD) and Alternating Least-Squares. For this project, we only focus on the first two learning algorithms and choose the first one in practice to optimize the parameters in different FM models.

Bayesian posterior estimate. Basically, Bayesian posterior estimate constructs the posterior probability of parameters given observed data. Since the parameters are hard to sample from the full posterior distribution, MCMC algorithm is used with Gibbs sampling to derive the converged conditional distribution of each parameter, which in essence is the same as optimizing the loss function with gradients.

$$\mathrm{P}(\theta|\mathrm{X},\mathrm{y}) \propto \mathrm{P}(\mathrm{y}|\mathrm{X},\theta)\mathrm{P}(\theta)$$

where θ is all parameters needed to estimate in the model.

Each parameter in the model follows a normal distribution with variance to be the regularization parameter corresponding to the feature. This makes sense because the regularization parameter controls the flexibility of model parameter values when observed data changes. So w_0 follows $\mathrm{N}\left(\mu^0, \frac{1}{\lambda^0}\right)$, w_j follows $\mathrm{N}\left(\mu^w_{\pi(j)}, \frac{1}{\lambda^w_{\pi(j)}}\right)$, and $v_{j,f}$ follows $\mathrm{N}\left(\mu^v_{f,\pi(j)}, \frac{1}{\lambda^v_{f,\pi(j)}}\right)$. Then hyperpriors are set for $\mu^w_{\pi(j)} \sim N\left(\mu_0, \gamma_0^{\lambda^w_{\pi(j)}}\right)$, $\lambda^w_{\pi(j)} \sim \Gamma(\alpha_\lambda, \beta_\lambda)$, $\mu^v_{f,\pi(j)} \sim N\left(\mu_0, \gamma_0^{\lambda^v_{f,\pi(j)}}\right)$, and $\lambda^v_{f,\pi(j)} \sim \Gamma(\alpha_\lambda, \beta_\lambda)$. Finally, the prior is set on y, $\mathrm{y} \sim \mathrm{N}(\hat{y}(x|\Theta), \frac{1}{\alpha})$ where α follows a gamma distribution $\Gamma(\alpha_0, \beta_0)$. The values of hyperparameters for w_0, w_j and $v_{j,f}$ can be found by sampling from their corresponding conditional posterior distributions, saving time for searching the best regularization parameters. Even though there are numerous priors, Rendle believes that these priors have little effect on hyperparameters values [5]. One of the reasons may be that the proportion of conditional posterior mean that is explained by prior mean is very small compared with that explained by maximum likelihood estimate when the number of input variables is quite large.

Stochastic Gradient Descent. SGD is popular in optimizing the loss function in various forms since it has low computational and storage complexity. Due to the easy computation of gradients of FM loss function, the algorithm should run fast. Basic algorithm is:

$$\theta = \theta - \eta\left(\frac{\partial(\hat{y}(x|\Theta), y)}{\partial\theta} + 2\lambda_\theta\theta\right)$$

There are three kinds of hyperparameters to estimate in order to run SGD. First, the learning rate η has to be set suitably since the convergence highly depends on η. Second, regularization parameters are required to estimate through grid search or cross-validation. However, Rendle erases the group difference to make λ_θ the same for all features. Such revision effectively reduces the searching time for best λ_θ, but may increases the bias for some parameter estimates if the best λ_θ for all features is larger than the best one with group difference or increases the variance for others if the best λ_θ for all features is smaller than the best one with group difference. That is probably one of the reasons why the ranking result by using SGD is not as good as that by using MCMC in Rendle's experiment [5]. The third one is the parameter for the factorized interaction (V). Initialized values are sampled from $\mathrm{N}(0, \sigma)$ where σ is small.

Based on Rendle's prediction error rate with different algorithms [5], in our project, SGD algorithm for FM is studied but not used. Bayesian posterior estimate with MCMC is used instead to obtain the best prediction result and compare it with that obtained by AltSVM.

3.2 Experiment and Results

3.2.1 Data preprocessing

In the MovieLens 100k dataset, the input features we can select are: UserID, MovieID, Movie Genre, Age, Gender, Occupation and Zip code. In this project, Moive Genre is not used. All demographic information about users in the MovieLens 100k is transformed to numerical values instead of strings. Gender is coded as 0 and 1 for male and female and occupation is coded from 0 to 19. All features seem to have no outliers and wrong records except the Zip code. Most of the values in Zip code are four or five digit number, but some are combinations of letters and numbers like "T8H1N" and "V3N4P", which obviously don't look like zip code. To make them in the same form as others, these values are coded as "00000".

Then, the movie dataset is randomly divided into training data and test data by the ratio of 8:2, so theoretically, each user should have 80% of rankings in the training dataset and 20% of rankings in the test dataset.

Eight different feature spaces are selected from MovieLens 100k to build FM models in this project:

Table 2: Different features used to build FM models

Selected features
User + Movie

User + Movie + Age
User + Movie + Gender
User + Movie + Occupation
User + Movie + Zip code
User + Movie + Gender + Occupation
User + Movie + Age + Gender + Occupation
User + Movie + Age + Gender + Occupation + Zip code

3.2.2 Evaluation metric

The root mean squared error (RMSE) is used to evaluate each FM model:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

3.2.3 Experiment procedure

In this project, MCMC algorithm is used on MovieLens 100k. There are two hyperparameters needed to tune in order to obtain the best prediction result. The standard deviation for factorized interaction V has to be initialized in order to generate the Gibbs sampling. Since the initial value is trivial to the conditional posterior distribution after the convergence, usually small value is used to speed up the convergence [5]. The k value, that is, the dimensionality of the factorized matrix V has to be determined. If k is set too small, there will be a high bias between $\mathrm{V} * \mathrm{V}^{\mathrm{t}}$ and the interaction parameter matrix while if k is too large, the model tends to be overfitted.

For each of the eight FM models with different feature space, the initialized standard deviation is determined first by selecting the optimal one from different values which will result in the fastest convergence, with k kept constant as the default value in LibFM. We use the training and test RMSE after the first 100 iteration as a measure of converging rate. A smaller RMSE indicate a faster convergence.

Once the optimal value of initialized standard deviation is determined, it is set constant for all the following tuning of the factorization dimensionality. Different k values are tried on each model and for each k, MCMC runs for 2000 iterations. The corresponding RMSE is recorded with each k. By observing

the RMSE of both training and test data, the optimal dimensionality for FM models is selected as the one which leads to the smallest test RMSE.

After all eight models are fitted with their respective optimal initialized standard deviation and k values, their respective smallest test RMSE can also be generated. The performances of eight models are compared based on the test RMSE so that the feature space that is most useful to predict the movie rating could arise.

3.4.4 Main Results

We first use UserID and MovieID as the predictor variables in the FM model. Table 3 shows all initialized standard deviation we have selected and the corresponding training and test RMSE. Figure 8 shows relationship between selected initialization values and corresponding RMSE.

Table 3: Initialized standard deviation and the corresponding RMSE.

Dimensionality (k)	Initialized standard deviation	Training RMSE	Test RMSE
8	0.01	0.84211	0.939953
8	0.1	0.82	0.92
8	0.5	0.796	0.9
8	1	0.795	0.9
8	2	0.796	0.9
8	3	0.803575	0.929342
8	5	0.830218	0.967751
8	8	0.908367	1.02205
8	10	0.9086	1.04

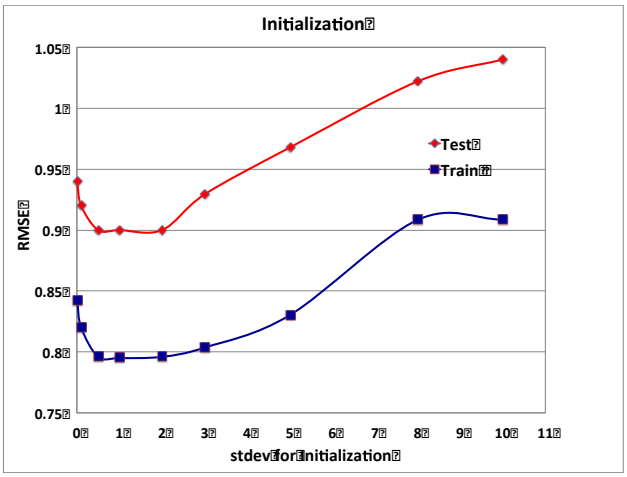


Figure 8: Selected initialization values and the corresponding RMSE.

As can be seen from above, the training and test RMSE have similar trend when the initialization value increases from 0.01 to 10. The training RMSE goes down quickly to 0.795 as the smallest when initialization value is 1. And the test RMSE decreases to 0.9 as initialization value is 0.5, 1 and 2, then goes up quickly. Combining the training and test RMSE results, we set the initial standard deviation to be 1.

Table 4 shows all k values we have tried and the corresponding training and test RMSE. Figure 9 shows the changing trend of RMSE with the increase of dimensionality k.

Table 4: Different dimensionality of factorized interaction and the corresponding RMSE.

Dimensionality (k)	Initialized standard deviation	Training RMSE	Test RMSE
0	1	0.923743	0.949935
1	1	0.885847	0.923279
2	1	0.867476	0.916586
4	1	0.836922	0.910053
8	1	0.795	0.9059
10	1	0.77628	0.9049
12	1	0.764545	0.907677
16	1	0.735331	0.909
32	1	0.653082	0.911651

48	1	0.585849	0.915848
64	1	0.52507	0.919282
92	1	0.449133	0.925375
128	1	0.370692	0.924

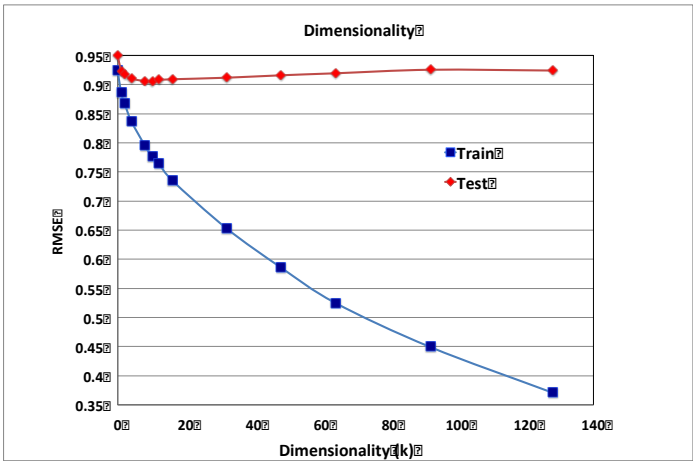


Figure 9: RMSE influenced by the number of dimensionality.

When the dimensionality of factorization increases, the training RMSE goes down gradually. The test RMSE reaches 0.9049 when k is 10. As the dimensionality increases from 10 to 128, it goes up slightly since the model becomes more complex and is prone to be overfitted. So the best dimensionality is 10 for FM model with only UserID and MoiveID and the corresponding ranking accuracy is 78%.

For each of the other seven FM models, we first find the optimal initialization of standard deviation and then tune the factorization dimensionality k. So each model is tuned to have the smallest RMSE with a specific optimal factorization dimensionality. The performances of FM models with different predictor variables are compared in the Table 5 and Figure 10.

Table 5: Factorization dimensionality and the prediction error of FM using different predictor.

Predictor	dimensionality(k)	RMSE
User,Movie	10	0.9018
User,Movie,Age	20	0.8984
User,Movie,Gender	15	0.9003
User,Movie,Occupation	16	0.8996
User,Movie,Zipcode	20	0.9654
User,Movie,Gender,Occupation	15	0.9001
User,Movie,Age,Gender,Occupation	20	0.9006
User,Movie,Age,Gender,Occup,Zipcode	20	1.3397

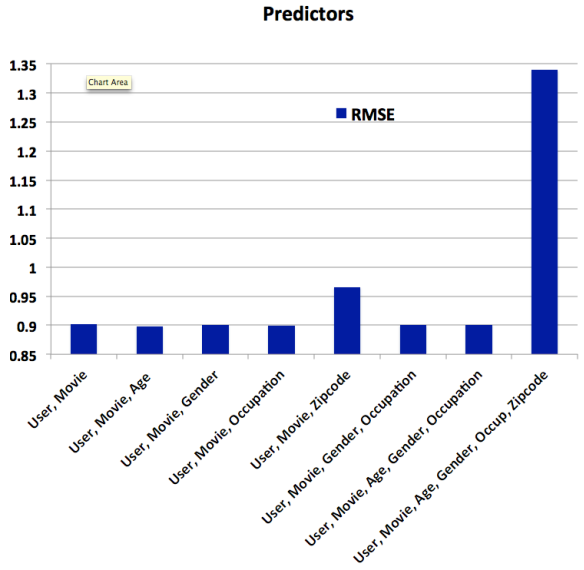


Figure 10: RMSE influenced by different feature sets.

Adding one or more predictor variables was expected to improve model performance. However, the result shows that there is little difference in test RMSE in models without zip code, indicating that adding more features doesn't necessarily improved the ranking prediction. The test RMSE for these models is about 0.9. When zip code is included in two models, both models have substantially high test RMSE, which indicates that zip code is not a useful predictor feature.

Despite that there are two hyperparameters needed to tune, the hyperparameter k is the only parameter that decides the prediction result. As we can see from the result above, even though different number of features is added to the model, the optimal k that leads to the smallest RMSE generally ranges from 10 to 20 without a certain increasing or decreasing pattern, which means the interaction parameter matrix can be approximated very well by the factorized matrix V with a 10 to 20 rank, regardless of how many features are added into the model. However, it's interesting to find that certain features may have an impact on the optimal dimensionality. Age is likely to increase the optimal k since the model with UserID,

MovieID and Age has higher k value than the model with either Gender or Occupation besides the UserID and MovieID. Still, more proof is needed to verify the relationship between the features and the dimension of the factorized interaction.

4. A comparison of AltSVM and FM

Table 6 lists the main strengths and weaknesses we have found for AltSVM and FM. One of the major differences between the two algorithms is that AltSVM uses only pairwise comparison information as the training data while FM needs to use the actual ratings. This difference can correspond to a strength and a weakness for either algorithm. On one hand, pairwise comparisons are more reliable and more widely available than actual ratings, so designing algorithm based on this special input data is quite crucial for learning user's preferences. On the other hand, since AltSVM works only with data in the form of pairwise comparisons, this would restrict its application to other scenarios (e.g., when we have different features). In contrast, FM provides a general regression model which can incorporate any real-valued features. In terms of performance, AltSVM performs quite well using only limited number of pairwise comparisons per user. As shown in Section 2.3, AltSVM achieves a higher prediction accuracy than that of matrix completion. The best prediction accuracy is 72%, achieved by using at most 3k randomly selected pairwise comparisons per user. This corresponds to only 30% of the total pairwise comparisons in the training set. However, the performance of AltSVM degrades when we use more pairwise comparisons. Possible reasons have been analyzed thoroughly in Section 2.3. In particular, this performance degradation indicates that AltSVM is not robust in handling unbalanced data. In contrast, FM achieves a higher prediction accuracy (78%) than AltSVM (72%) on the same MovieLens dataset. This result is not surprising since FM has an advantage over AltSVM because it sees the actual ratings and hence has more information.

Table 6: A comparison of AltSVM and FM.

Algorithms	Strength	Weakness
AltSVM	1) Uses only pairwise comparison information as the training data. 2) Performs quite well with a small number of pairwise comparisons per user (even achieves better performance than matrix completion).	1) Performs relatively bad when we use more pairwise comparisons (possible reason is that the model is not robust when dealing with unbalanced data). 2) Works only for data in the form of pairwise comparisons.
FM	1) Provides a general regression model. 2) Performs quite well in terms of prediction accuracy and RMSE.	Needs to use real-valued features (e.g., actual ratings) as the training data.

5. Conclusion

In this project we have explored two collaborative ranking algorithms: Alternating SVM (AltSVM) and factorization machine (FM). For FM, our main focus was to understand its model and figure out how to use libFM. We have demonstrated its strong prediction ability by applying FM on the MovieLens 100k dataset. In addition to the movie ratings, we also investigated the impact of features such as age, gender, occupation, etc. Unlike FM, whose implementation is already given by the libFM package, AltSVM is a new algorithm that has not been implemented or evaluated before. To implement AltSVM in Matlab, we have tried three optimization solvers, two initialization methods, and two stopping criteria. Our final implementation used the most efficient combination among them. Specifically, we solved the optimization programs by formulating the constraints as hinge loss functions and then apply gradient descent method with backtracking line search. The performance of AltSVM was evaluated on the synthetic dataset and the MovieLens 100k dataset. For both datasets we investigated the performance variation with respect to two parameters: rank and the number of pairwise comparisons per user. Strengths and Weaknesses of the two algorithms were analyzed thoroughly in Section 4. Although the prediction accuracy achieved by AltSVM (72%) is lower than that of FM (78%), we still think that AltSVM is a promising algorithm, especially for the case when actual ratings are not available. One key finding of AltSVM was that its performance did not improve even when we used all the pairwise comparisons in the training set. As analyzed in Section 2.3, one possible reason is that AltSVM is not robust in handling unbalanced data. This may be a future research direction that would further improve the performance of AltSVM.

References

[1] S. Sanghavi, personal communication, 2014.

[2] S. Rendle, “Factorization Machines,” in *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010)*, Sydney, Australia, 2010.

[3] J. Neeman and S. Sanghavi, personal communication, 2014.

[4] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, pages 324–345, 1952.

[5] S. Rendle, “Factorization machines with libFM,” ACM Trans. Intell. Syst. Technol. Vol. 3, no. 3, Article 57, 2012.

[6] Yehuda Koren, Robert Bell, Chris Volinsky, "Matrix Factorization Techniques for Recommender Systems", Computer, vol.42, no. 8, pp. 30-37, August 2009.

[7] Y. Koren, “Factorization meets the neighborhood: a multifaceted collaborative filtering model,” in KDD ’08: *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2008, pp. 426–434.

[8] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, “Factorizing personalized markov chains for next-basket recommendation,” in *WWW ’10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 811–820.

[9] Y. Koren, “Collaborative filtering with temporal dynamics,” In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’09)*. ACM, New York, NY, 447–456, 2009.

[10] S. Rendle and L. Schmidt-Thieme, “Pairwise interaction tensor factorization for personalized tag recommendation,” in *WSDM ’10: Proceedings of the third ACM international conference on Web search and data mining*. New York, NY, USA: ACM, 2010, pp. 81–90.