Fast and Better Algorithms for Matrix Completion

Bo-Yao Lin, Chun-Yu Chen, Jun-Kun Wang, Alireza Alizadegan
{boyaolin,chunyuc,jimwang,aralizad}@umich.edu

Abstract

Matrix completion is about inferring the missing entries of a matrix. A well known example is "Netflix Prize", which was a competition for predicting user ratings to movies. In this project, we extensively study and compare several algorithms for the problem. We consider three common objective functions in the literature for the problem as well as some optimization methods for each objective function. In total, we implement 8 algorithms in this project. We compare the implemented algorithms on two real datasets, which are Netflix dataset and MovieLens dataset. Analysis for the strengths and weaknesses of the algorithms are also provided.

1 Introduction

We study the matrix completion problem in this project. In this problem, a movie rating matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ containing many missing entries is given, where an entry (i,j) in the matrix means a rating of movie i by customer j. The goal is to recover the missing entries by leveraging information from the observed ones. The matrix below is a toy example, where the number on an entry means a rating and $*$ represents unobserved ones. Thus, the objective is to predict or recover the ratings on those $*$'s.

$$\begin{bmatrix} 1 & * & 1 & 2 & * \\ 3 & 4 & * & * & 4 \\ * & * & 1 & * & * \\ * & 5 & * & 4 & * \end{bmatrix} \text{ A movie rating matrix } \mathbf{M} \in \mathbb{R}^{4\times 5}$$

A typical modeling strategy is to minimizing the predicting error over the observed entries subject to a constraint

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} \frac{1}{2}\|\mathbf{X}-\mathbf{M}\|_{OB}^2 \text{ s.t. } \|\mathbf{X}\|_* \le k, \tag{1}$$

where we use the notation that $\|\mathbf{X}-\mathbf{M}\|_{OB}^2 \equiv \sum_{M_{i,j} \text{ is observed}} (\mathbf{X}_{i,j}-\mathbf{M}_{i,j})^2$ and the nuclear norm $\|\cdot\|_*$ is the sum of singular values. Using nuclear norm as the constraint is due to a common modeling assumption that the rating matrix is low-rank. The intuitive interpretation is that each entry in the rating matrix M can be explained by only k latent factors, where $k < \min(m,n)$. This is reasonable because there may only be some factors (e.g. gender, age, area) that determine a user preference in real life. Therefore, a constraint that encourages the predicted matrix $\mathbf{X}$ to be low-rank is expected. However, if we directly use the matrix rank as the constraint, then the objective becomes non-convex. Instead, a common way to get around this issue is using a surrogate, which is the nuclear norm here. It can be shown that the nuclear norm is an upper bound of the matrix rank in some sense.

In additional to (1), we also consider the un-constrained form,

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} \frac{1}{2}\|\mathbf{X}-\mathbf{M}\|_{OB}^2 + \lambda\|\mathbf{X}\|_*, \tag{2}$$

as well as a non-convex form introduced by [10],

$$\min_{\mathbf{A}\in\mathbb{R}^{m\times r},\mathbf{B}\in\mathbb{R}^{n\times r}} \frac{1}{2}\|\mathbf{A}\mathbf{B}^\top - \mathbf{M}\|_{OB}^2 + \frac{\lambda}{2}(\|\mathbf{A}\|_{Fro}^2 + \|\mathbf{B}\|_{Fro}^2), \tag{3}$$

where $\|\cdot\|_{Fro}^2$ represents the square of Frobenius norm, which is the sum of squares of all the entries in a matrix. The meaning of (3) is that if the matrix is indeed low rank, then it can be represented or factorized into two thin matrices, where $r << \min(m, n)$. The objection is not jointly convex of the matrices $\mathbf{A}$ and $\mathbf{B}$. But, fixing one matrix and updating the other becomes a convex problem and actually the resulted problem shares the same form as the regression problem.

A good algorithm of matrix completion allows us to build a recommendation system that exploits the users preference to maximize profit. Other application includes recovering the missing value of some sensors in a sensor network, which is also an important task. Because of its importance, there have been many works in the literature for matrix completion. The works either come with a new objective/modeling strategy or a new optimization method for an existing objective function. Thus, some natural question are *What is the pros and cons of a proposed method?* and *Which method should I choose?* Yet, a thorough comparison among these works is lacked. Therefore, we decide to systematically and extensively study this subject and try to answer the questions.

2 Algorithms

We extensively study and compare several algorithms in the literature for this project. Specifically, the algorithms that we investigate and implement in this paper can be classified into three categories. The first class of algorithms is about dealing with objective (1), which are Frank-Wolfe algorithm [8, 9], stochastic Frank-Wolfe algorithm [6], and rank-1 regularized Frank Wolfe algorithm [3]. The second class of algorithm aims at objective (2). For (2), we select [7], which is the state-of-the-art algorithm for the unconstrained optimization problem. The third class of algorithm is for objective (3). Yet, (3) is non-convex and the iterate may stuck in the local optimum. Recently, [4] proposed an algorithm that alternatively minimizes between objective (3) and objective (2) and show global convergence to the optimum. Therefore, we select [4] to represent the third class of the algorithms.

2.1 Frank-Wolfe style algorithms for objective (1)

Frank-Wolfe algorithm has recently enjoyed a surge popularity due to its ability to solve structured constraint problem like the matrix completion problem (1) here. To show the merit of Frank-Wolfe algorithm and its variants, let us first explain how to use projected gradient descent to solve (1) (we also implement projected gradient descent in the experiment). Then, we will describe the Frank-Wolfe algorithms and address how they avoid an issue that occurs to projected gradient descent when solving (1).

In each iteration t of projected gradient descent, a gradient descent step is conducted, followed by a projection to the constraint set (i.e. $\|\mathbf{X}\|_* \leq k$), which can be expressed as $\mathbf{X}_{t+1} \leftarrow \Pi_{\|\cdot\|_*\leq k}(X_t - \eta\nabla f(X_t))$, where η is the step size and the (i, j) entry of the gradient at $\mathbf{X}_t$, $\nabla f(\mathbf{X}_t) \in \mathbb{R}^{m\times n}$, is $(\nabla f(X_t))_{i,j} = (X_t)_{i,j} - M_{i.j}$ if (i, j) is observed, or 0 otherwise. The projection $\Pi_{\|\cdot\|_*\leq k}$ is conducted by singular value decomposition (SVD); pairs of leading singular vectors are collected as many as possible until the sum of the corresponding singular value of the collected pairs reaches the bound of the nuclear norm (i.e. k in (1)), Then, the new update X_t is set to the sum of the collected components, which means the other components are discarded. The concern here is that the SVD computations is very expensive, the computational complexity of an SVD is cubic of the matrix size, $\mathcal{O}((m + n)^3)$. For matrix completion, m and n would be the number of users and number of movies respectively. So, the iteration cost of the projected gradient descent is very high.

Frank-Wolfe style algorithms can avoid SVD computations and there are sometimes called "projection-free" in the literature. We study the original Frank-Wolfe algorithm and its two variant here.

Frank-Wolfe algorithm [2, 8, 4]

The standard Frank-Wolfe algorithm [2, 8, 4] is shown in Algorithm 1, where $f(\cdot)$ is the objection function, which is $\frac{1}{2}\|\mathbf{X} - \mathbf{M}\|_{OB}^2$ in this paper.

Note that line 3 in Algorithm 1 is a linear program over the constraint set. It has been shown that optimizing the linear program is equivalent to finding the first leading singular vectors (chapter 7

Algorithm 1 Frank-Wolfe algorithm for solving (1)

```
1: Initialize: any $\mathbf{X}_0$ satisfying the constraint $\Omega : \|\mathbf{X}\|_* \leq k$.
2: For $t = 1, 2, 3 \ldots, T$
3:     $\mathbf{V}_t = \arg\min_{\mathbf{V} \in \Omega} \langle \mathbf{V}, \nabla f(\mathbf{X}_{t-1}) \rangle$
4:     $\mathbf{X}_t = (1 - \gamma_t)\mathbf{X}_{t-1} + \gamma_t \mathbf{V}_t$.
```

in [5]).

$$\min_{\mathbf{V} \in \mathbf{R}^{m \times n} : \mathbf{V} \in \{\|\mathbf{X}\|_* \leq 1\}} \langle \mathbf{V}, \nabla f(\mathbf{X}_{t-1}) \rangle \equiv \min_{\mathbf{u} \in \mathbf{R}^m : \|\mathbf{u}\|_2 \leq 1, \mathbf{v} \in \mathbf{R}^n : \|\mathbf{v}\|_2 \leq 1} \mathbf{u}^\top \nabla f(\mathbf{X}_{t-1}) \mathbf{v}^\top.$$

Thus, line 3 is about computing the first pair of singular vectors of the gradient matrix $\nabla f(\mathbf{X}_{t-1})$ [5]. After computing the singular vectors, the solution of the original problem is set to $\mathbf{V}_t = k\mathbf{u}_t\mathbf{v}_t^\top$. The complexity of finding the leading pair of singular vectors is at the order of the number of non-zero entries, while performing SVD requires $\mathcal{O}((m+n)^3)$ for a matrix in $\mathbb{R}^{m \times n}$. So, the iteration cost is improved over the projected gradient method. The new update in line 4 is the convex combination of the current update and $\mathbf{V}_t$, which means that the update is always in the constraint set. Thus, it avoids the annoying $\Pi_{\|\cdot\|_* \leq k}$ projection operator.

Stochastic Frank-Wolfe [6] For stochastic Frank-Wolfe, it simply replaces the $\nabla F(\mathbf{w}_{t-1})$ in line 3 of Algorithm 2 with stochastic gradient. For our matrix completion case, it is conducted by sampling mini-batches of entries to get an estimate of the gradient. The advantage of this method is that iteration cost is cheaper.

Rank-1 regularized Frank-Wolfe [3]

Another variant is that a quadradtic term is added to the original linear program to solve $\mathbf{V}_t$, with additoinal constraint that the rank of $\mathbf{V_t}$ is one. Solving this program is still by finding the first singular vectors. The solution of line 5 in Algorithm 2 is the rank-one matrix consisting of the first pair of the singular vectors of $-\nabla f(\mathbf{X}_t) + \eta\beta\mathbf{X}_t$.

Algorithm 2 Rank one regularized conditional gradient

```
1: Initialize: any rank-one matrix $\mathbf{X}_1 = \mathbf{x}_1\mathbf{x}_1^\top$ satisfying the constraint $\Omega : \|\mathbf{X}\|_* \leq k$.
1: Parameter: sequnce of step size $\{\eta_t\}$, smooth parameter $\beta$.
2: For $t = 1, 2, 3 \ldots, T$
3:     Let $\mathbf{X}_t = \Sigma_{\tau=1}^{t} a_i \mathbf{x}_\tau \mathbf{x}_\tau^\top$.
4:     Sample $i_t \in [t]$ according to the probability distribution $(a_1, a_2, \ldots, a_t)$.
5:     $\mathbf{V}_t = \arg\min_{\mathbf{V} \in \Omega \text{ and } \mathrm{rank}(\mathbf{V})=1} \mathbf{V} \cdot \nabla F(\mathbf{X}_t) + \frac{\eta_t\beta}{2}\|\mathbf{V} - \mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top\|^2$
6:     $\tilde{\eta}_t = \eta_t/2$ if $a_{i_t} \geq \eta_t$ or $\tilde{\eta}_t = a_{i_t}$ if $a_{i_t} < \eta_t$.
7:     $\mathbf{X}_{t+1} = X_t + \tilde{\eta}_t(\mathbf{V}_t - \mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top)$.
```

2.2 Active Subspace Selection for objective (2) [7]

To solve the unconstrained objective (2), the naive way is using gradient descent. We have shown that the gradient of $\frac{1}{2}\|\mathbf{X} - \mathbf{M}\|_{OB}^2$ when we introduced the projected gradient descent in the previous section. The tricky part is the nuclear norm, it is shown in [11] such that the subgradient of $\|\mathbf{X}\|_*$ is the set

$$\{UV^\top + W : U^\top W = 0, WV = 0, \|W\|_2 \leq 1\},$$

where U and V is the singular value decomposition of $\mathbf{X}$. Therefore, we can apply subgradient descent to solve objective (2). The concern is that the iteration cost of SVD is high, $\mathcal{O}((m+n)^3)$ for a matrix in $\mathbb{R}^{m \times n}$, as we described in the previous section.

Different from the Frank-Wolfe style algorithms in the previous section that totally avoid SVD. The algorithm that we consider in the subsection, [7], performs smaller SVD in each iteration to reduce the computational cost. The idea of [7] is to identify some active row and column subspaces and perform subgradient descent on those subspaces. Because the subgradient descent is performed on the subspaces, it involves a smaller SVD and saves some computations. The issue is how to identify the *active row and column subspaces*.

The idea is that any matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ can be represented as a sum of rank one matrices: $\mathbf{X} = \Sigma_{ij} \sigma_{ij} \mathbf{u}_i \mathbf{v}_j^\top$. Therefore, the goal of the active subspace selection is to eliminate rank-one subspaces/matrices which are likely to have zero weights in the final solution, which it calls *fixed rank one subspace*, and then focus on the remaining rank one subspaces, which it calls the *active rank one subspace*. Given a rank one matrix $\mathbf{u}\mathbf{v}^\top$, if we want to update the current X by it, the optimal step size is

$$\sigma^* = \arg\min_{\sigma} f(\mathbf{X} + \sigma \mathbf{u}\mathbf{v}^\top) + \lambda \|\mathbf{X} + \sigma \mathbf{u}\mathbf{v}^\top\|_*, \tag{4}$$

where we denote $\frac{1}{2}\|\mathbf{X} - \mathbf{M}\|_{OB}^2$ as $f(\mathbf{X})$. An observation is that the subgradient with respect to the singular value σ_{ij} is $[\mathbf{u}_i^\top \nabla f(\mathbf{X})\mathbf{v}_j - \lambda, \mathbf{u}_i^\top \nabla f(\mathbf{X})\mathbf{v}_j + \lambda]$. Then, if $\mathbf{u}\mathbf{v}^\top$ is not the active subspace, the corresponding σ^* should be 0. Yet, if 0 is the solution for the (4), by the standard convex analysis we learned in class, $\|\mathbf{u}_i^\top \nabla f(\mathbf{X})\mathbf{v}_j\| \leq \lambda$. Therefore, [7] defines *active rank one subspace* as $\mathcal{A} \equiv \{\mathbf{u}\mathbf{v}^\top : |\mathbf{u}^\top \nabla f(\mathbf{X})\mathbf{v}| \geq \lambda \text{ or } \mathbf{u}^\top \mathbf{X}\mathbf{v} = 0\}$. Define soft-thresholding operator as $S_\lambda(\mathbf{X}) = \mathbf{U}(\Sigma - \lambda I)_+ \mathbf{V}^\top$. Using the idea and definition, Theorem 4 in [7] shows a way to use the operator to find out the active subspace, which leads to Algorithm 3. We refer the readers to [7] for details.

Algorithm 3 Matrix Completion via Active Subspace Selection

```
Input: Regularization parameter $\lambda$, initial $X = U\Sigma V^\top$
Output: The optimal solution $X^* \in \mathbb{R}^{m \times n}$
1: for $t = 1, 2 \ldots, T$
2:     $[\bar{U}, \bar{\Sigma}, \bar{V}] = \text{ApproxSVD}(X - \nabla f(X))$
3:     $U_G = \{\bar{\mathbf{u}}_i | \bar{\Sigma}_{ii} > \lambda\}, V_G = \{\bar{\mathbf{v}}_i | \bar{\Sigma}_{ii} > \lambda\}$
4:     $U_A = \text{QR}([U_G, U]), V_A = \text{QR}([V_G, V])$
5:     $S = \arg\min_S f(U_A S V_A^\top) + \lambda \|S\|_*$
6:     $[U_S, \Sigma, V_S] = \text{ThinSVD}(S)$
7:     $U = U_A U_S, V = V_A V_S, X = U\Sigma V^\top$
```

2.3 SoftImpute-ALS for objective (2) and (3) [4]

SoftImpute-ALS[4] is an algorithm designed specifically for solving matrix completion problem. SoftImpute-ALS gathers the features from two matrix completion algorithms that are both solving matrix completion problems but with different objective functions. The two algorithms are SoftImpute [1] and alternating least squares algorithm (ALS) [10], which are designed for objective (2) and (3), respectively. SoftImpute-ALS leverages the fact that (2) and (3) will have the same solution if the solution to (2) has rank $q \leq r$ and vice versa to speed up the computation. It solves A and B in objective (3) alternatively while utilizing soft-thresholded SVD. It first fixes A, solves B, then fixes B, and solves A. The algorithm is summarized in Algorithm 4.

2.4 Converting the matrix completion problem to a supervised learning problem

In the class, we have learned many algorithms for supervised learning. Yet, in our matrix completion problem, there is **no feature** provided. To leverage the algorithms we learned from class, we try to convert the matrix completion problem to a supervised learning problem by writing it as an empirical risk minimization problem: $\min_{\mathbf{w} \in \mathbb{R}^d,\, b \in \mathbb{R}} \frac{1}{N} \sum_{k=1}^N (y_k - \mathbf{w}^T \mathbf{x}_k - b)^2 + \lambda R(\mathbf{w})$, where R is a regularization term. The notations are that $\mathbf{w}$ and b are parameters of linear regression function to be learned, N is number of data points, $\mathbf{x}_k$ and y_k are feature vectors and its label of instance k, and λ is the regularization parameter.

In our matrix completion problem, N corresponds to the number of observed entries in the movie rating matrix $\mathbf{M}$ and y_k corresponds to an observed entry $\mathbf{M}_{i,j}$. The issue is how to generate the features $\mathbf{x}_k$ for the entry. Our method to form $\mathbf{x}_k$ is obtained by conducting singular value decomposition of $\mathbf{M}$ in which zero is assigned to replace the unobserved entries. By doing singular

Algorithm 4 SoftImpute-ALS

```
Input: Input Data matrix $M$ and projection function for observed data $P_\Omega$ (the projection function
    from complete data to the observed data)
Output: Complete matrix $\hat{M}$
  1: Initialize $A = UD$, where $U(m \times r)$ is randomly chosen matrix with orthonormal columns and
  $I(r \times r)$ , and $B = VD$ with $V = 0$
  2: Given $A = UD$ and $B = VD$, solve $\min_{\tilde{B}} \frac{1}{2}||P_\Omega(M - A\tilde{B}^T)||^2_{Fro} + \frac{\lambda}{2}||\tilde{B}||^2_{Fro}$
  to update B.
      • $M^* = P_\Omega(M) - P_\Omega(AB^T) + AB^T$
      • $\tilde{B}^T = (D^2 + \lambda I)^{-1} DU^T M^*$
      • Compute $\tilde{B}D = \tilde{U}\tilde{D}^2\tilde{V}^T$
      • $V \leftarrow \tilde{U}$ and $D \leftarrow \tilde{D}$
  3: Given $B = VD$ and solve $A$ like in step 2, with $M^T$ replacing $M$, and $B$ and $A$ interchanged.
  4: Repeat step 2 and 3 until converge
  5: Compute $\hat{M} = M^*V = UD_\sigma R^T$. Output $U$, $V \leftarrow VR$ and
  $D_{\sigma,\lambda} = diag[(\sigma_1 - \lambda)_+, \ldots, (\sigma_r - \lambda)_+]$
```

value decomposition, we obtain left/right singular matrices U and V respectively, where $M = USV^\top$. We can view each row on the left singular matrix U as the feature vector of a movie, while each column on the right singular matrix $V^\top$ as the feature vector of a user. Thus, $\mathbf{x}_k$, which is the feature vector for an entry $\mathbf{M}_{i,j}$, is obtained by concatenating the i_{th} row of U and j_{th} column of $V^\top$. This is our way to convert it to a supervised learning problem. For the regularization term, we try square of L2 norm (ridge regression) and L1 norm (lasso) in the experiment. For the training, we use stochastic (sub)-gradient descent to solve the problem due to large amount of entries (i.e. large N).

3 Experiments and Evaluation

3.1 Dataset statistics

We conduct our algorithms in two datasets, which are Netflix [1] and MovieLens100k [2].

For Netflix dataset, there are user ratings for 17,770 movies and a total of 480,189 users. The ratings are from 1 to 5 points. Due the the huge size of the matrix, in this project, we use the first 1000 movies with the most user rating counts as target movies and choose 1,000 users with most frequently commenting rates within these movies as our target users. We randomly drop the non-zero entries with 50% probability to construct our training set. For MovieLens dataset, there are user ratings for 943 movies and a total of 1,682 users. There are 100,000 observed entries, we use 80/20 split to construct the training/testing set. The basic properties of datasets are shown in Table 1.

Properties	Netflix Dataset	MovieLens
Size	1,000,000	1,586,126
No. of non-zero entries in complete dataset	816,842	100,000
No. of non-zero entries in training set	408,752	80,000

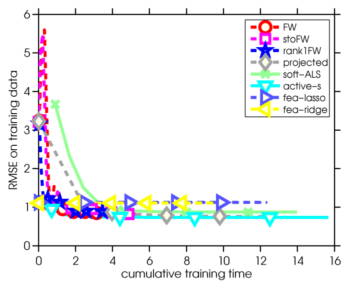
Table 1: Dataset Properties

3.2 Algorithms and its abbreviation

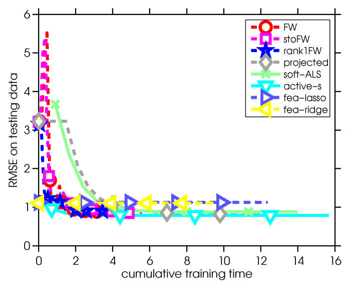
We compare eight algorithms that we have described in the experiment. They are Frank-Wolfe algorithm (FW), Stochastic Frank-Wolfe algorithm (stoFW), rank-1 regularized Frank-Wolfe algorithm (rank1FW), projected gradient descent (projected), SoftImpute-ALS (soft-ALS), Active Subspace

[1] Netflix datatset is available on `https://archive.org/details/nf_prize_dataset.tar`

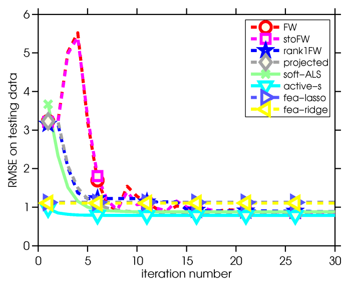
[2] MovieLens100k is available on `http://files.grouplens.org/datasets/movielens/ml-100k.zip`



(a) RMSE in training entries vs. training time

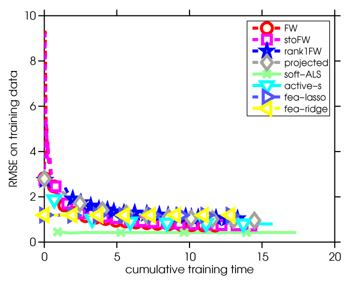


(b) RMSE in testing entries vs. training time

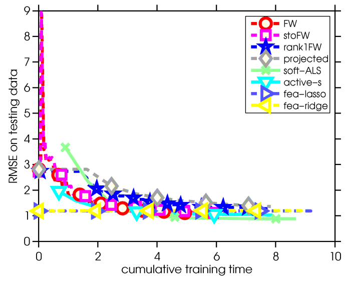


(c) RMSE in testing entries vs. iteration number

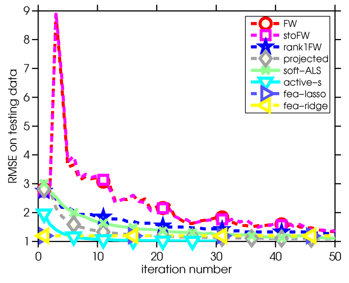
Figure 1: RMSE on Netflix Dataset



(a) RMSE in training entries vs. training time



(b) RMSE in testing entries vs. training time



(c) RMSE in testing entries vs. iteration number

Figure 2: RMSE on MovieLens Dataset

Selection (active-s), as well as lasso (fea-lasso) and ridge regression (fea-ridge). They all have been described in this paper.

In the following, we report the performance of the algorithms with respect to different measures.

3.3 RMSE in training entries vs. training time

The first measure that we consider is the root mean square errors (RMSE),

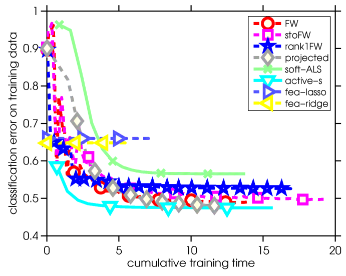
$$\mathrm{RMSE}(A) = \sqrt{\frac{\sum\limits_{\forall (i,j)\in A} (M_{ij} - X_{ij})^2}{|A|}}, \text{for a set of entries } A.$$

To be specific, in this subsection, we report the RMSE in training entries versus training time. To plot the figure, during training, we save the model of an algorithm at the end of each iteration and record the running time so far. Then, we compute the corresponding root mean square error (RMSE) by each saved model later. Figure 1(a) and Figure 2(a) show the result on Netflix and MovieLens respectively. Please be noted that the markers on each curve are placed every certain amount of iteration, instead of every iteration.

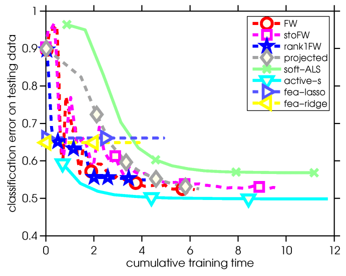
We can see that FW, stochastic FW, and active-s are highly competitive. Recall that there are the methods that can deal with the computational issue of SVD. Stochastic FW and FW avoid SVD, while active-s just needs a smaller SVD in each iteration. On the other hand, projected gradient descent and soft-ALS requires full SVD, so that they are the slowest. As for fea-lasso and fea-ridge, their curve patterns seem to be different from other methods is because that they converge extremely fast. Once they converge, the performance will remain at similar level. Hence, they have a straight line pattern.

3.4 RMSE in testing entries vs. training time

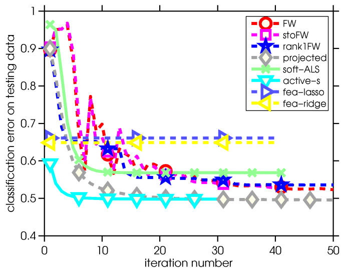
The concern of the measure in the previous subsection is that the objective of different algorithms is not the same. For example, the unconstrained objective (2) has a regularization term, which means



(a) Classification error in training entries vs. training time

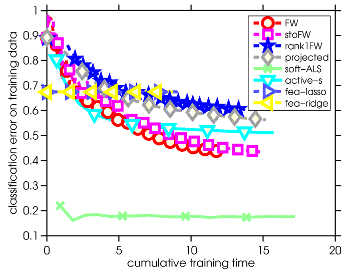


(b) Classification error in testing entries vs. training time

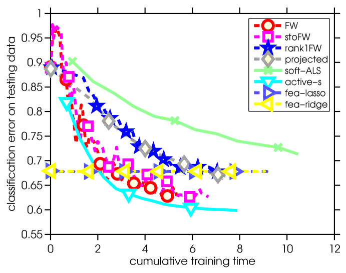


(c) Classification error in testing entries vs. iteration number

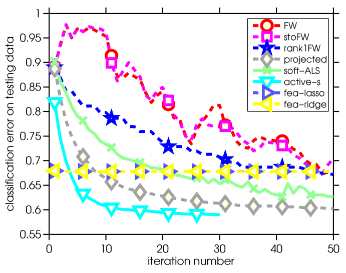
Figure 3: Classification error for Netflix Dataset



(a) Classification error in training entries vs. training time



(b) Classification error in testing entries vs. training time



(c) Classification error in testing entries vs. iteration number

Figure 4: Classification error for MovieLens Dataset

that the RMSE on training entries may not be the best measure as an algorithm may try to trade the regularization term with the training error. Therefore, we consider another measure "RMSE in testing entries vs. training time", Figure 1(b) and Figure 2(b) show the results. The way to compute the measure follows the same fashion as the previous section. This measure allows us to compare the progress of learning through time. We can see that among the methods active-s is the best under this measure. We also see that fea-lasso and fea-ridge are the worst, which may be explained by the fact that the objective of lasso or ridge regression is not designed to solve the matrix completion problem or the feature we used is not strong.

Note that we also consider a method that always outputs the same rating. The best performance of such kind of prediction is by always predicting 4, which leads to 1.1155 RMSE on testing entries for Netflix dataset and always predicting 4 which leads to 1.2297 RMSE on testing entries for MovieLens dataset. We can see that all methods are able to beat the one that always outputs the same rating. However, both fea-lasso and fea-ridge actually have performance really close to those always predict the same constant value. When we look into the predicted value, we realize it is indeed this case that they always predict value that is close to the mean of training data. These two algorithms may be regarded as underfitting to the matrix completion problem.

3.5 RMSE in testing entries vs. iteration number

The reasons why we consider the measure regarding to iteration number are that the running time may be affected by the way of our implementation, and that different optimization methods have different convergence rate in terms of number of iterations and have different cost per iteration. Therefore, we also plot Figure 1(c) and Figure 2(c) based on this measure. In the figure, each point on a line is a RMSE on testing entries by the model learned at the end of an iteration. We see that the projected gradient descent is one of the fastest methods under this measure, while it converges slower in terms of time. We explain this observation as follows. It is known that for a smooth and strongly convex objective, gradient descent enjoys exponential convergence rate (therefore it is fast). For a twice differentiable function, smooth means $\nabla^2 F(\mathbf{X}_t) \preceq LI$; while strongly convex means $\nabla^2 F(\mathbf{X}_t) \succeq \alpha I$ for a positive number L, α For our matrix completion problem, if all the movie entries are observed, then it would clear be strongly convex. Yet, we know that some entries are missing, so that the strongly convexity is 0 at these entries. But, we still can say that the objective is strongly convex in some sense. The message here is that though projected gradient descent and soft-ALS have fast convergence rate in iteration number, the iteration cost is high, so that they are outperformed by other projection-free algorithms or active-s in running time.

3.6 Classification error

In this subsection, we treat the problem as a five-class multi-class classification problem, since the rating is a number among $\{1, 2, 3, 4, 5\}$. To predict a "label" of an entry, the output of an algorithm is rounded up to the nearest integer, which allows us to calculate the classification error. Fig. 3 and Fig. 4 show the classification result. Most of the algorithms except fea-lasso and fea-ridge can achieve about 50% classification error, which means that they are effective to some degree under this measure, as it is a (difficult) multi-class classification problem. For fea-lasso and fea-ridge, the both training and testing classification error are about 65%. They show consistent error rate, which may again indicate that these methods are not designed to solve the matrix completion problem or the feature we used is not strong.

3.7 Comparison of different objectives

In this subsection, we compare the best RMSE by different objective functions. Specifically, we report the best RMSE on the testing entires in the following table. From the table, it seems the the

Objective	best RMSE in Netflix	best RMSE on MoviewLens
objective (1)	0.7910	1.0123
objective (2)	0.7822	1.0025
objective (3)	0.8741	0.9982

unconstrained objective (2) can achieve the best performance. The algorithm we choose for the unconstrained objective (2) is active-s [7]. From the previous section, active-s is also the fastest algorithm to converge. Therefore, we recommend the readers to use active-s [7] in practice, which has best performance in terms of time and prediction performance.

4 Conclusions

We have studied, analyzed, and compared several algorithms for matrix completion in practice under different measures. We also have described some interesting results and provided the analysis in the experimental section. Perhaps the most important lesson we have learned is that a good algorithm should leverage some trade-offs. For example, different optimization algorithms have different trade-off between number of iterations and computational cost in each iteration. The algorithm Active Subspace Selection [7] can exploit the trade-off by reducing the cost per iteration while maintaining fast convergence rate in terms of iteration number, which results in the best method in our experiment.

5 Appendix: Works done by individual member

All members analyzed the experimental results and wrote the report. The individual works are: **Bo-Yao Lin** implemented Active Subspace Selection, and conducted the implemented algorithm on the datasets. **Chun-Yu Chen** wrote codes to parse the Netflix dataset, implemented SoftImpute-ALS, and conducted SoftImpute-ALS on the datasets and collected the results. **Jun-Kun Wang** collected references for this project, implemented projected gradient, Frank-Wolfe, stochastic Frank-Wolfe, and rank-1 regularization stochastic Frank-Wolfe algorithm. **Alireza Alizadegan** implemented ridge regression and lasso, and conducted the implemented algorithms on the datasets.

References

[1] J-F Cai, E. J. Candes, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. on Optimization*, 2008.

[2] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistic*, 1956.

[3] Dan Garber. Faster projection-free convex optimization over the spectrahedron. *NIPS*, 2016.

[4] Trevor Hastie, Rahul Mazumder, Jason Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *JMLR*, pages 3367–3402, 2015.

[5] Elad Hazan. Introduction to online convex optimization. 2016.

[6] Elad Hazan. Variance-reduced and projection-free stochastic optimization. *ICML*, 2016.

[7] Cho-Jui Hsieh and Peder A. Olsen. Nuclear norm minimization via active subspace selection. *ICML*, 2014.

[8] Martin Jaggi. A simple algorithm for nuclear norm regularized problems. *ICML*, 2010.

[9] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. *NIPS*, 2015.

[10] Nathan Srebro, Jason Rennie, and Tommi Jaakkola. Maximum margin matrix factorization. *NIPS*, 2005.

[11] G.A. Waton. Characterization of the subdi erential of some matrix norms. *Linear Algebra and its Applications*, 1992.