Matrix Methods for Recommendation Systems

Introduction

Subscription-based online movie streaming services like Netflix and Hulu have become quite popular in recent years. A major challenge that these services face is the task of personalized movie recommendations. How do we predict which movies a user is likely to watch? To answer this, we can analyze user-movie interactions. These interactions are best represented as a ratings matrix.

A ratings matrix $R \in \mathbb{R}^{u \times m}$ contains the rating a user u gave a movie m. Hence, the rows of R represent the users and the columns represent the movies. In reality, most users view only a small fraction of the entire movie universe. Hence, the matrix R is very sparse which makes it difficult to make relevant recommendations to users.

	M1	M2	M3	M4
U1	?	?	4	5
U2	3	?	4	?
U3	5	?	5	?
U4	?	2	?	?
U5	2	?	3	3

Table 1: A toy example to illustrate the structure of R.

Table 1 shows the interactions between 5 users and 4 movies where the rating scale goes from 1 to 5. The entry $R(i, j) = ?$ indicates that user i has not seen/rated movie j. The goal of movie recommendation then becomes to predict the missing '?' entries as accurately as possible and recommend movies that have the highest ratings. Thus, the movie recommendation problem can be mathematically formulated as a matrix completion task with specific constraints.

In this project, we[1] use collaborative filtering methods which use the ratings provided by multiple users to predict the missing entries in R and make a recommendation. In particular, we explore two types of collaborative filtering techniques: neighborhood models and low-rank matrix factorization models. We assess the performance of these models on the MovieLens 100K dataset.

Exploring the Data

We first look into the MovieLens 100K dataset to provide some background. The dataset contains 100386 ratings and 3683 tag applications across 9742 movies by 610 users. We only consider the ratings in this project. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars) [1].

From the statistics above, we can calculate the sparsity of R as a percentage.

[1] I use the pronoun 'we' instead of 'I' throughout this report since this is common practice in the research community. However, this report is my original work unless indicated otherwise by a citation.

$$Sparsity(R) = 100 - \frac{100386}{9742 \times 610} = 100 - 1.689 \approx 98.3\%$$

R is extremely sparse! This kind of imbalance is further exposed by the following plots.

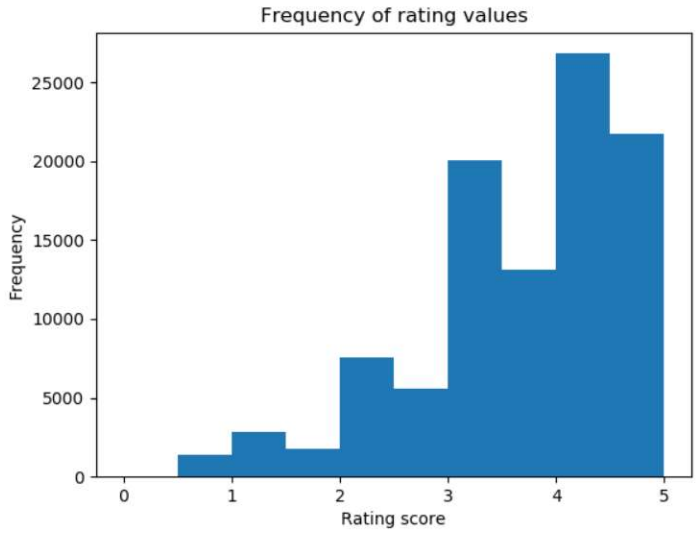

Figure 1: The histogram shows the frequency of rating values assigned by users to the movies. The distribution is slightly skewed with the average rating = 3.502. This means most users liked the movies they watched.

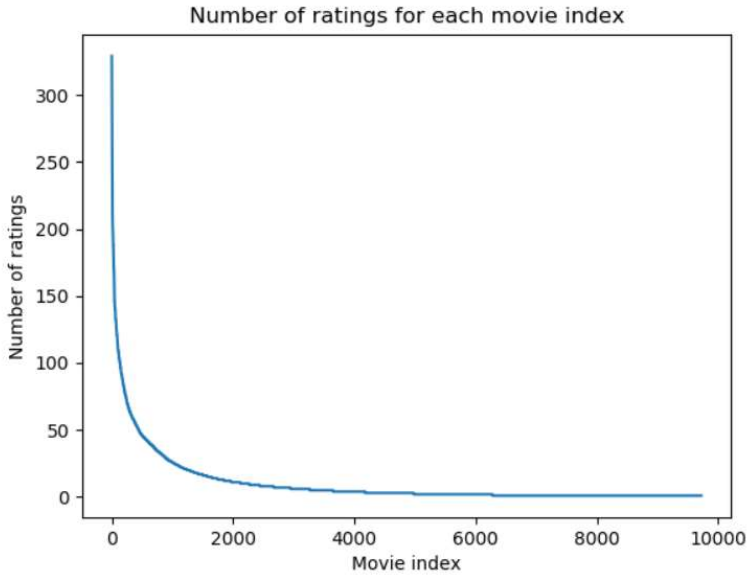

Figure 2: The curve shows the distribution of ratings among movies. The distribution is approximately heavy-tailed and hence Pareto's law applies. A majority of the movies in the dataset have not been rated which is a contributing factor to the dataset's sparsity.

From the plots above, it is clear that sparsity will be the main hindrance in achieving good, accurate rating predictions. We hypothesize that simple, naïve neighborhood models will suffer from

sparsity but can establish baseline performance. We expect more sophisticated models like low-rank matrix factorization models to handle sparsity more elegantly.

Data Pre-processing

Before we start modelling, we need to impute the missing entries in R.

1	4.0	NaN	4.0	NaN	...	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
606	2.5	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
607	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
608	2.5	2.0	2.0	NaN	...	NaN	NaN	NaN	NaN
609	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
610	5.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

Figure 3: $R(i,j) = NaN$ if user i has not rated movie j. The NaN values must be changed before any algorithms are run on R.

We plan to impute the NaN values with 0s. The motivation behind this choice is to retain the sparsity of the original matrix. If the NaN values were imputed using the mean of all the ratings, the matrix would become dense which can be troublesome for low-rank matrix factorization methods due to computational complexity constraints. We denote the imputed matrix by R.

Neighborhood Models

Neighborhood models like k-Nearest Neighbors (kNN) are intuitive and easy to understand because similar users tend to provide similar ratings on the same movie. The idea behind kNN is to use the k most similar vectors for a given vector to recommend movies where the similarity between vectors is defined by some metric.

More formally, suppose there is a feature vector a. The kNN algorithm will try to predict the missing ratings in a by looking at the k most similar vectors to a. Using a user-based collaborative filtering approach, we mathematically define the predicted rating $\hat{r}_{ui}$ for a basic kNN [2] as –

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

where the $N_i^k(u) =$ set of k users that are the most similar to user u for item i, $r_{vi} =$ rating provided by user v on item i and $sim(u,v) =$ similarity between users u and v.

Usually, there are users that rate in an extreme fashion like rating all movies either too highly or poorly. These users tend to add noise to the dataset. To reduce bias and extreme opinions, we will mean-center the ratings. Thus, the predicted rating $\hat{r}_{ui}$ for a mean-centered kNN [2] is –

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

where μ_p = mean rating for user p.

In this project, we experiment with two types of kNN algorithms: basic and mean-centered. For each kNN algorithm, we try two similarity metrics: cosine similarity and Pearson correlation coefficient. They are defined [2] as –

$$cosine(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}}$$

$$Pearson(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum_{i \in I_{uv}} (r_{uj} - \mu_v)^2}}$$

where I_{uv} = set of items ratings by both users u and v. The Pearson correlation coefficient can be interpreted as a mean-centered cosine similarity which is advantageous since mean-centering has a normalization effect. Hence, we hypothesize that Pearson correlation coefficient may outperform the cosine similarity.

To run experiments, we will use 5-fold cross validation to find the optimal k neighbors and average the root-mean-squared error (RMSE) across the folds to evaluate algorithm performance. An explanation of the Python code for this part is provided in Appendix A.

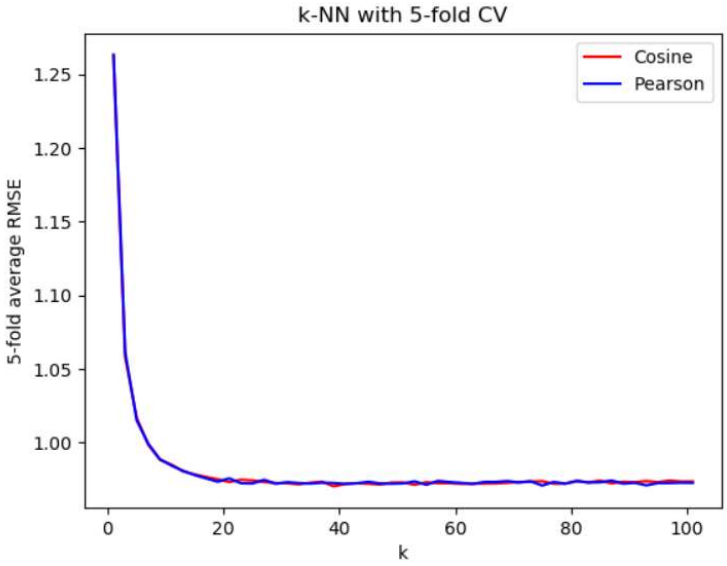

Figure 4: 5-fold average RMSE of basic kNN. Performance for both similarity metrics is comparable. Optimal kNN algorithm is $k = 39$ with cosine similarity.

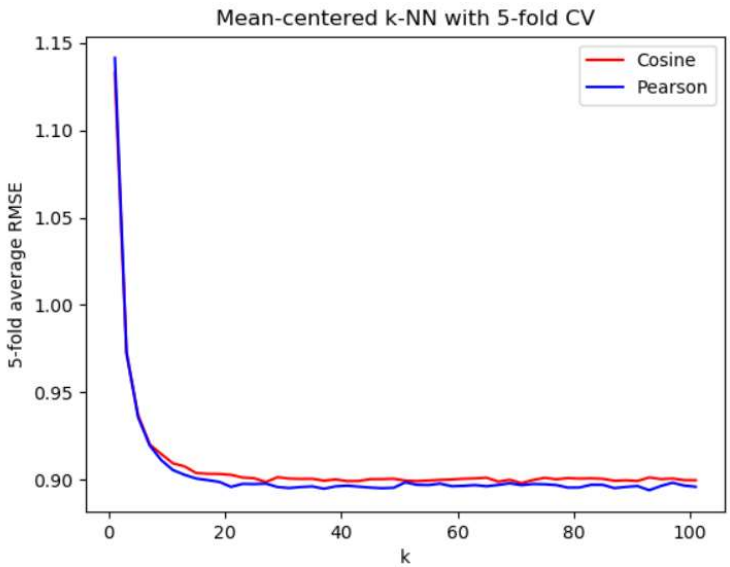

Figure 5: 5-fold average RMSE of mean-centered kNN. Pearson similarity is better. Optimal kNN algorithm is $k = 37$ with Pearson similarity.

From the plots above, the best kNN algorithm is mean-centered kNN with $k = 37$ and Pearson similarity. The empirical results verify and validate our hypothesis that Pearson similarity will outperform cosine similarity. We now try to generate movie recommendations for users who liked the movie "Iron Man (2008)" using the best kNN algorithm.

```
Top 10 movie recommendations for  Iron Man (2008) :
1. Avengers, The (2012) , Genre =  Action|Adventure|Sci-Fi|IMAX
2. WALL·E (2008) , Genre =  Adventure|Animation|Children|Romance|Sci-Fi
3. Iron Man 2 (2010) , Genre =  Action|Adventure|Sci-Fi|Thriller|IMAX
4. Dark Knight, The (2008) , Genre =  Action|Crime|Drama|IMAX
5. Avatar (2009) , Genre =  Action|Adventure|Sci-Fi|IMAX
6. Thor (2011) , Genre =  Action|Adventure|Drama|Fantasy|IMAX
7. Guardians of the Galaxy (2014) , Genre =  Action|Adventure|Sci-Fi
8. Iron Man 3 (2013) , Genre =  Action|Sci-Fi|Thriller|IMAX
9. Star Trek (2009) , Genre =  Action|Adventure|Sci-Fi|IMAX
10. Watchmen (2009) , Genre =  Action|Drama|Mystery|Sci-Fi|Thriller|IMAX
```

Figure 6: Top 10 kNN movie recommendations for the movie "Iron Man (2008)".

The movie recommendations look quite promising! However, the problem with the kNN recommender is that it only recommends popular (Marvel/Disney) movies of roughly the same genres. There is very little diversity in the movie recommendations. An explanation could be that the movie feature vectors are sparse and reside in a high-dimensional vector space. These factors (sparsity and high dimensionality) impede the kNN algorithm's ability to learn any structure/relationship between the feature vectors. This phenomenon is usually described as 'curse of dimensionality' and kNNs (or in general any distance-dependent algorithm) are known to suffer from it. Since the matrix is sparse, it is reasonable to assume that R can be expressed as a product of two smaller, low-rank matrices. Thus, to improve movie recommendations, we explore low-rank matrix factorization methods.

Low-Rank Matrix Factorization

Matrix factorization methods are attractive because they can help us discover knowledge such as user tastes and preferences in high-dimensional spaces which can consequently be exploited in movie recommendations. As depicted earlier, the matrix R is usually sparse in practice. Hence, it is reasonable to assume that R can be factored as $R = BC$ with $B \in \mathbb{R}^{u\times r}, C \in \mathbb{R}^{r\times m}$ where $rank(R) = r \ll \min\{u, m\}$. This factorization is referred to as a low-rank representation of the original matrix. In this project, we discuss two low-rank matrix factorization methods: Singular Value Decomposition (SVD) and Non-Negative Matrix Factorization (NNMF).

Singular Value Decomposition (SVD):

The (full) SVD of R is defined as $R = U\Sigma V^T = \sum_{i=1}^{\min\{u,m\}} \sigma_i u_i v_i^T$ where $U \in \mathbb{R}^{u\times u}, V \in \mathbb{R}^{m\times m}$ are orthogonal and $\Sigma \in \mathbb{R}^{u\times m}$ is 'diagonal'. Σ contains the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{u,m\}} \geq 0$. The columns of U and V are called the left and right singular vectors, respectively. The matrices U and V can be interpreted as the user features matrix and movie features matrix,

respectively. U represents how much users 'like' each feature and V represents how relevant each feature is to each movie [3] where, as an example, a feature could indicate the genre of the movie. The 'diagonal' of Σ contains the importance of each feature. However, these features are latent, and some are more informative than others. Hence, the objective is to keep the most important k features by constructing the best rank-k approximation of R. The objective can be formulated as the following optimization problem with variable $R_k \in \mathbb{R}^{u \times m}$ –

$$\begin{aligned} \text{minimize} \quad & \| R - R_k \|_F \\ s.t \quad & rank(R_k) \leq k \end{aligned}$$

The optimization problem is non-convex because of the rank inequality constraint (the rank function is non-convex). However, by the Eckart-Young Theorem, the optimal R_k is the sum of the first k terms in the SVD of R. Hence, $R_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$. This approximation is optimal for the 2-norm too.

Another approach is to use Principal Component Analysis (PCA) to obtain the approximation by projecting R onto a k-dimensional subspace in the directions that capture the maximum variance in R. Let the mean-centered ratings matrix be $R_c = R - \frac{1}{m}\mathbf{1}\mathbf{1}^T R$. Then the sample covariance matrix $S = \frac{1}{m} R_c^T R_c$. The PCA optimization objective can be formulated as the following with variable $X \in \mathbb{R}^{m \times k}$ –

$$\begin{aligned} \text{maximize} \quad & \lambda_{min}(X^T S X) \\ s.t \quad & X^T X = I_k \end{aligned}$$

Take the eigenvalue decomposition $S = Q \Lambda Q^T$ and make an orthogonal change of variables $Y = Q^T X$. Hence, we have

$$\begin{aligned} \text{maximize} \quad & \lambda_{min}(Y^T \Lambda Y) \\ s.t \quad & Y^T Y = I_k \end{aligned}$$

By the max-min characterization of eigenvalues, $Y = \begin{bmatrix} I_k \\ 0 \end{bmatrix}$ which means $X = QY = [q_1 \ q_2 \ \dots \ q_k]$. Hence, the optimal solution is the first k eigenvectors of $R_c^T R_c$. But we have $R_c^T R_c = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T$. Thus, the right singular vectors of R_c are the eigenvectors of $R_c^T R_c$ which is advantageous because 1) we can avoid the cost of constructing the Gram matrix $R_c^T R_c$ and 2) we can directly work on the data matrix R_c which is preferred for numerical stability purposes.

After taking the rank-k approximation, $R \approx U_k \Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ where $U_k \in \mathbb{R}^{u \times k}$, $V_k^T \in \mathbb{R}^{k \times m}$ and $\Sigma_k \in \mathbb{R}^{k \times k}$. Multiplying the three matrices U_k, Σ_k, V_k^T would yield as estimation for the missing ratings. However, computing a full SVD (both singular values and vectors) using traditional methods like QR algorithm on $R^T R$ and $R R^T$ have time complexity $\mathcal{O}(m^3)$ and $\mathcal{O}(u^3)$ respectively. Clearly, the cubic time complexity is not scalable to large matrices.

We solve this issue by turning to stochastic gradient descent (SGD) which enjoys linear time complexity since only one data point is needed per iteration to update parameters. Suppose each movie is represented by $q_i \in \mathbb{R}^k$ and each user is represented by vector $p_u \in \mathbb{R}^k$. We estimate the predicted rating $\hat{r}_{ui}$ by solving the following optimization problem [4] with SGD updates for p and q –

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

$$SGD\ update: \qquad q_i \leftarrow q_i + \gamma(e_{ui}p_u - \lambda q_i)$$

$$SGD\ update: \qquad p_u \leftarrow p_u + \gamma(e_{ui}q_i - \lambda p_u)$$

where $\kappa =$ the training set, $r_{ui} =$ the ratings in κ, $\lambda =$ regularization constant, $\gamma =$ learning rate/step-size and $e_{ui} = r_{ui} - q_i^T p_u$. Note that p_u represents a row of U and q_i^T represents a column of V^T. If P and Q contain all vectors p_u and q_i^T, respectively, that solve the optimization problem above, then it is evident that $P = U$ and $Q^T = V^T$. Hence, we obtain the SVD using SGD updates.

To estimate $\hat{r}_{ui}$ using SVD with SGD updates, we must decide the value of k. We do this by inspecting the singular values of R.

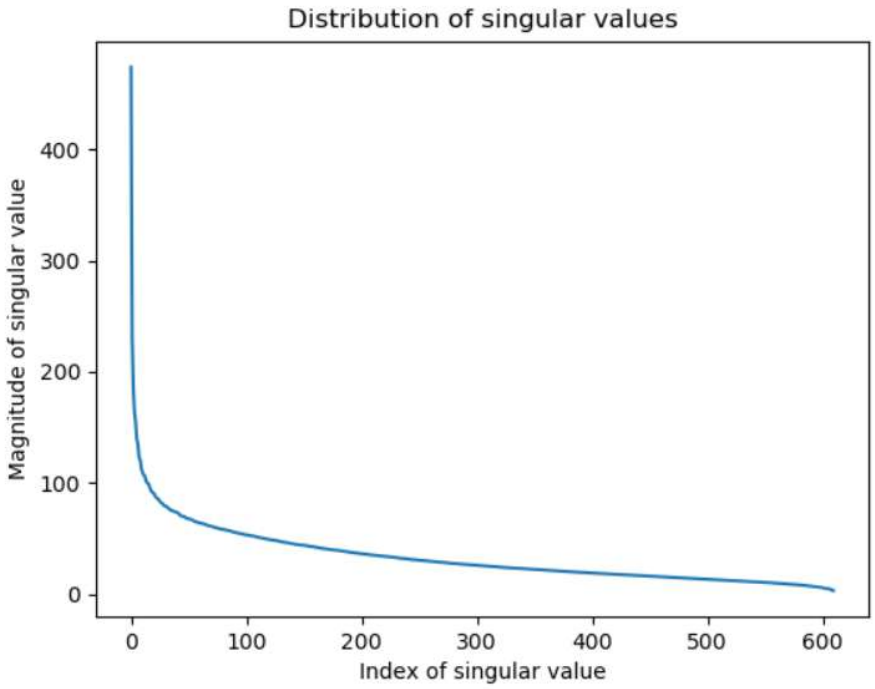

Figure 7: Distributions of the singular values of R. Keeping the top $k = 100$ singular values is a reasonable choice since the singular values decrease linearly after that.

To run experiments, we will use 5-fold cross validation to find the best k between 1 and 100. The SVD predictions will be evaluated using average root-mean-squared error (RMSE) across each fold. An explanation of the Python code for this part is provided in Appendix B.

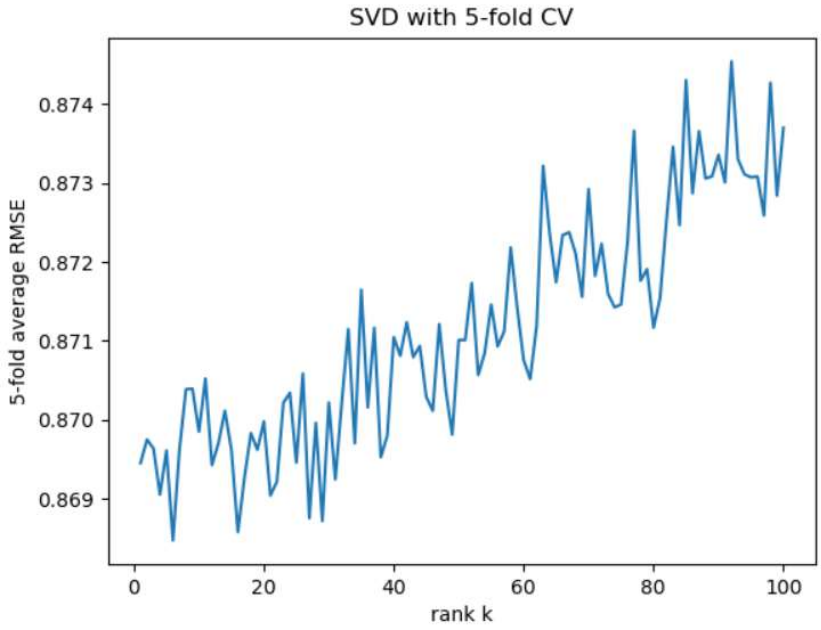

Figure 8: 5-fold average RMSE for SVD predictions. The lowest RMSE occurs at $k = 6$. As k increases, the RMSE also increases. This suggests overfitting at high values of k.

Since the average RMSE of the SVD is lower than that of kNN, we claim that the SVD movie recommendations will be superior to those of kNN. We support this claim by stating that the SVD reduces the movie feature vectors from a high-dimensional space to a lower-dimensional one in a way that captures the structure/relationships between the feature vectors. Now, we try to generate movie recommendations for users who liked the movie "Iron Man (2008)" by using SVD to construct the best rank-6 approximation of R.

```
Top 10 movie recommendations for  Iron Man (2008) :
1. Forrest Gump (1994) , Genre =  Comedy|Drama|Romance|War
2. Braveheart (1995) , Genre =  Action|Drama|War
3. Alien (1979) , Genre =  Horror|Sci-Fi
4. Inception (2010) , Genre =  Action|Crime|Drama|Mystery|Sci-Fi|Thriller|IMAX
5. Aliens (1986) , Genre =  Action|Adventure|Horror|Sci-Fi
6. Up (2009) , Genre =  Adventure|Animation|Children|Drama
7. Big Lebowski, The (1998) , Genre =  Comedy|Crime
8. Back to the Future Part II (1989) , Genre =  Adventure|Comedy|Sci-Fi
9. Schindler's List (1993) , Genre =  Drama|War
10. Avatar (2009) , Genre =  Action|Adventure|Sci-Fi|IMAX
```

Figure 9: Top 10 SVD movie recommendations for users who liked the movie "Iron Man (2008)".

The SVD recommendations look much different than the kNN recommendations! Some movies like "Inception" and "Avatar" share the same genre and popularity as "Iron Man". However, the SVD also recommends movies that are less popular and from different genres. This adds diversity to the recommendations which promotes user engagement. Hence, the empirical results verify and validate our hypothesis since the quality of the recommendations is visibly better as measured by the balance between popularity and diversity. As a result, the SVD is a major improvement over

kNN. Nonetheless, the SVD is less interpretable than kNN since the matrices U and V can have positive and negative entries even though R only has positive entries! To improve upon this, we further explore matrix factorization with non-negativity constraints.

Non-Negative Matrix Factorization (NNMF):

NNMF is a rank-k approximation tool in which $R \approx WH$ where $W \in \mathbb{R}^{u \times k}$ and $H \in \mathbb{R}^{k \times m}$. Furthermore, there are non-negativity constraints $W \geq 0$ and $H \geq 0$ on both matrices which means all the elements in both matrices are non-negative. This gives the matrices a special, interpretable meaning. The columns of W can be interpreted as user communities (features) and H indicates the affinity/importance a user has towards different user communities (features) [5].

The objective of NNMF is the following optimization problem with variables W and H –

$$\begin{aligned} &\text{minimize} && \| R - WH \|_F^2 \\ &s.t && W \geq 0, H \geq 0 \end{aligned}$$

The problem is non-linear and non-convex because of the matrix product WH. If the constraints were excluded, then the problem is equivalent to finding the rank-k truncated SVD. Clearly, this means $p^*_{SVD} \leq p^*_{NNMF}$ where p^* denotes the optimal primal value. Since the SVD provides a lower bound for NNMF, we hypothesize that the quality of NNMF recommendations may be inferior to those of SVD.

Since the problem is non-convex, convergence to the global optimum is hard to guarantee. However, a local optimal solution can be reached in an iterative, alternating fashion. Suppose W, H are initialized randomly and W is fixed. Then the problem becomes –

$$\begin{aligned} &\text{minimize} && \| R - WH \|_F^2 \\ &s.t && H \geq 0 \end{aligned}$$

The problem above is a non-negative least squares (NNLS) task [6]. Analyzing it, we get –

$$\begin{aligned} &\text{minimize} && \| r_i - Wh_i \|_F^2 \\ &s.t && h_i \geq 0, \quad i = 1 \dots m \end{aligned}$$

The problem above is a quadratic program (QP). Since QPs are convex, reaching the global optimum is provably guaranteed. The QP can be solved by standard QP solvers using interior-point and active-set methods. After obtaining H, another NNLS problem can be solved in the same manner to obtain W. This procedure can be repeated iteratively until a stopping criterion (error tolerance) is satisfied.

To predict ratings using NNMF, we solve a very similar problem [7] as in the SVD case –

$$\min_{q^* \geq 0, p^* \geq 0} \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

$$SGD\ update: \qquad p_{uk} \leftarrow p_{uk} \frac{\sum_{i \in I_u} q_{ik} r_{ui}}{\sum_{i \in I_u} q_{ik} \hat{r}_{ui} + \lambda |I_u| p_{uk}}$$

$$SGD\ update: \qquad q_{ik} \leftarrow q_{ik} \frac{\sum_{u \in U_i} p_{uk} r_{ui}}{\sum_{u \in U_i} p_{uk} \hat{r}_{ui} + \lambda |U_i| q_{ik}}$$

where $I_u =$ the set of items rated by user u and $U_i =$ the set of users who rated item i. We use SGD updates for the parameters p and q since it is easier to implement and has low time complexity.

To run experiments, we will use 5-fold cross validation to find the best k between 1 and 80. The NNMF predictions will be evaluated using average root-mean-squared error (RMSE) across each fold. An explanation of the Python code for this part is provided in Appendix C.

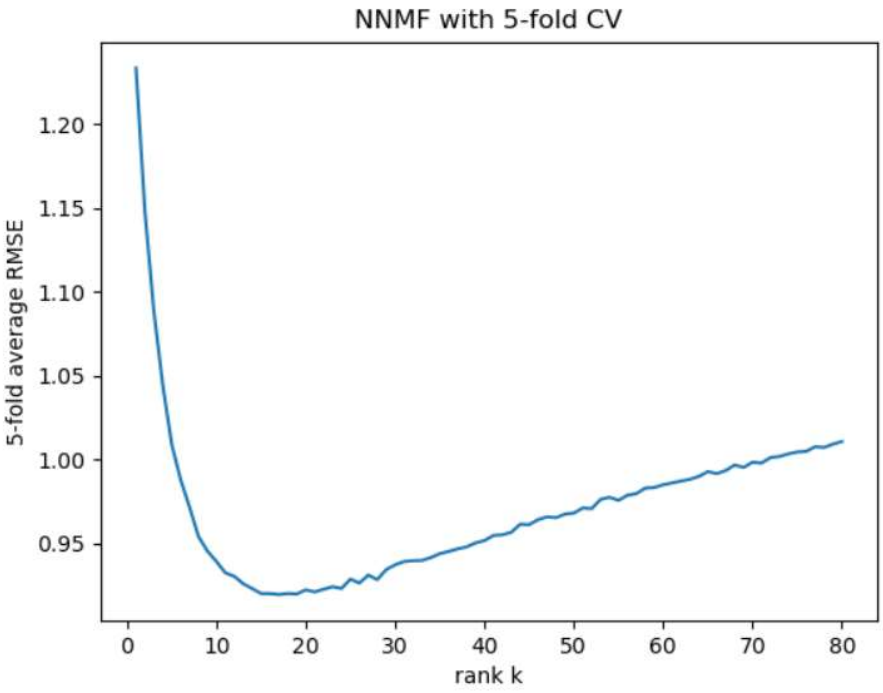

Figure 10: 5-fold average RMSE for NNMF predictions. The lowest RMSE occurs at $k = 17$. As k increases, the RMSE also increases. This suggests overfitting at high values of k. The same trend was noticed in SVD and may be generalized to other matrix factorization methods.

Now, we try to generate movie recommendations for users who liked the movie "Iron Man (2008)" by using NNMF to construct the best rank-17 approximation of R.

```
Top 10 movie recommendations for  Iron Man (2008) :
1. Forrest Gump (1994) , Genre =  Comedy|Drama|Romance|War
2. Braveheart (1995) , Genre =  Action|Drama|War
3. Unbreakable (2000) , Genre =  Drama|Sci-Fi
4. Ice Age (2002) , Genre =  Adventure|Animation|Children|Comedy
5. Meet the Parents (2000) , Genre =  Comedy
6. Back to the Future Part II (1989) , Genre =  Adventure|Comedy|Sci-Fi
7. Patriot, The (2000) , Genre =  Action|Drama|War
8. Alien (1979) , Genre =  Horror|Sci-Fi
9. O Brother, Where Art Thou? (2000) , Genre =  Adventure|Comedy|Crime
10. Green Mile, The (1999) , Genre =  Crime|Drama
```

Figure 11: Top 10 NNMF movie recommendations for users who liked the movie "Iron Man (2008)".

Again, the NNMF recommendations look much different than those of kNN. However, these recommendations are similar to those of SVD which makes sense since both are matrix factorization methods. But there are differences. Although some movies are similar, SVD recommends movies from a wider range of genres and popularity. This is probably because SVD yields an optimal solution whereas NNMF only gives an approximate solution. Hence, the empirical results verify and validate our hypothesis that the quality of NNMF recommendations is worse than those of SVD as measured by a balance between popularity and diversity. However, SVD's optimal solution comes at the price of less interpretable latent features and matrices.

Extensions

The latent features captured by NNMF can be construed as user communities that like a particular movie genre. For instance, a user community could comprise of users who only like horror movies. In this way, NNMF can be thought of as a clustering algorithm since it groups similar users together. Interestingly, k-means clustering can be interpreted as a matrix factorization method with the following optimization problem –

$$\begin{aligned} &\text{minimize} && \| A - BC \|_F^2 \\ &s.t && C_{ij} \in \{0,1\}, i = 1 \dots k, j = 1 \dots n \\ & && C_{1j} + \cdots + C_{kj} = 1, j = 1 \dots n \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and the variables are $B \in \mathbb{R}^{m \times k}$ and $C \in \mathbb{R}^{k \times n}$.

Thus, there exists a kind of duality between matrix factorization and clustering. Therefore, it seems reasonable to explore Spectral Clustering (SC) for movie recommendations. We provide a theoretical analysis below.

The role of SC in movie recommendations is to cluster users in such a way that users in the same cluster have high similarity but users from different clusters have low similarity. A simple choice of similarity could be Euclidean distance or cosine similarity. A more sophisticated similarity

function could be Gaussian or radial basis functions [8]. In some sense, the task can be thought of as maximizing between-cluster variance and minimizing within-cluster variance which is analogous to the cost function in Linear Discriminant Analysis (LDA).

Let W be the user similarity matrix created using, for example, cosine similarity. Note that $W = W^T \in \mathbb{R}^{u \times u}$ for commutative similarity metrics [i.e $sim(a, b) = sim(b, a)$] and we define $diag(W) = 0$. Now we construct an undirected weighted graph with users as vertices and similarities as weights. Next, we define the unnormalized graph Laplacian $L = diag(W\mathbf{1}) - W$.

The goal of SC is to find partitions that minimize the following optimization problem –

$$\text{minimize} \quad \sum_{k=1}^{K} \frac{cut(V_k)}{size(V_k)}$$

where $V_1, \dots, V_k =$ partitions, $cut(V_k) =$ total weight of edges between V_k and $V \setminus V_k$, $size(V_k) = \sum_{i \in V_k} d_i$ with $d_i =$ positive weight for each vertex i.

There is some freedom is choosing the clustering objective because of the value assigned to d_i. For our analysis, we choose the ratio cut objective. Hence, $size(V_k) =$ number of vertices in V_k and $d_i = 1$. Interestingly, the clustering objective can be written as –

$$\sum_{k=1}^{K} \frac{cut(V_k)}{size(V_k)} = \sum_{k=1}^{K} \sum_{i \in V_k, j \notin V_k} \frac{W_{ij}}{size(V_k)} = \sum_{\{i,j\} \in E} W_{ij} \parallel x_i - x_j \parallel^2 = trace(X^T L X)$$

where $X \in \mathbb{R}^{u \times K}$ is an indicator matrix which must satisfy the following properties: 1) X has no zero rows 2) columns of X are scaled indicator vectors and 3) $X^T diag(d) X = I$. Property (2) makes the problem combinatorial and hence we relax the problem by removing this property completely. Property (1) is not a problem since $\dim\big(null(L)\big) \neq 0$ if the graph is connected. Thus, we get the following optimization problem with variable X –

$$\begin{aligned} &\text{minimize} && trace(X^T L X) \\ &s.t && X^T X = I_K \end{aligned}$$

Take the eigenvalue decomposition of $L = Q \Lambda Q^T$ and make an orthogonal change of variables $Y = Q^T X$. Then we get –

$$\begin{aligned} &\text{minimize} && trace(Y^T \Lambda Y) \\ &s.t && Y^T Y = I_K \end{aligned}$$

The above is equal to –

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{u} \lambda_i y_i^2 \\ &s.t && Y^T Y = I_K \end{aligned}$$

Intuitively, an optimal solution would be $\lambda_1 = \lambda_2 = \cdots = \lambda_{u-K} = 0$ and $\lambda_{u-K+1} = \lambda_{u-K+2} = \cdots = \lambda_u = 1$. This yields $Y = \begin{bmatrix} 0 \\ I_K \end{bmatrix}$ which means $X = QY = [q_{u-K+1} \dots q_u]$ is the last K eigenvectors of L. This solution is indeed optimal by the min-max characterization of eigenvalues. However, X may not be a valid indicator matrix due to the relaxation made earlier. Thus, we run k-means clustering on X to get user clusters (communities) $U_1, \dots, U_k$. Obtaining user clusters is very advantageous since neighborhood models can be employed within these clusters to make very relevant movie recommendations. As a result, the quality of SC recommendations may be superior to those of SVD and NNMF.

Other Applications

The matrix factorization methods and neighborhood models explored in this project are applicable to a variety of other tasks. A couple are explained below:

1. *Document Analysis and Text Mining*. Suppose there is a collection of documents represented by a term-document matrix D where each row corresponds to a word in a dictionary and each column corresponds to a document. To find documents that are most similar to each other, we can compute the cosine similarity of the vectors d_i and d_j.

   Another way to find similar documents is by comparing the content of the documents. The content of the documents can be expressed as a set of latent features like concepts and topics. Using NNMF, we have $D(:, j) \approx \sum_{i=1}^{k} W(:, i)H(i, j)$ with $W \geq 0$ and $H \geq 0$. Due to the non-negativity constraints, the columns of W are the basis elements that are found simultaneously in different documents. Hence, these basis elements can be interpreted as topics and the (i, j) entry in H displays the importance of the ith topic in jth document [6]. A similar matrix factorization can be computed using the SVD and is commonly known as Latent Semantic Indexing (LSI) in the NLP community.

2. *Image Processing*. Suppose there is a collection of images represented by F where each row corresponds to pixel intensity and each column corresponds to a face. Performing NNMF yields $F \approx WH$ and extracts facial features. The columns of W can be interpreted as basis images which represent facial features like eyes and nose [6]. Similarly, the (i, j) entry in H displays the importance of the ith feature in jth face. More concretely, the columns of H indicate the presence/absence of features in a face. Truncated SVDs are also commonly used in image denoising tasks since noisy images are usually ill-conditioned.

Conclusion

We have evaluated the performance of different recommendation algorithms on the MovieLens 100K dataset. From empirical results, we noticed that low-rank matrix factorization methods are more effective than neighborhood models since a sparse ratings matrix R has a good low-rank representation. Thus, matrix factorization methods capture the structure present in R. However, some matrix factorization methods are more interpretable than others; the SVD is less interpretable than the NNMF but it gives a better performance in terms of the popularity and the diversity of the recommended movies. Hence, the following inequality is valid: SVD > NNMF > kNN which states that the performance of the SVD is the best.

To address the shortcomings of matrix factorization methods, we explored spectral clustering from a theoretical perspective. Spectral clustering implicitly performs matrix factorization by segregating users into groups and communities. Each community has distinct characteristics which is beneficial since it enables us to make very relevant recommendations to a user who belongs to a particular community by using simple neighborhood models like kNN in each community. However, this claim cannot be verified without running experiments. But taking a step back and assessing the situation, we realize that all these algorithms are trying to answer one of the most fundamental questions in recommendation systems: how do we recommend relevant items while still promoting diversity in the items? This relevance-diversity tradeoff might be answered using contextual multi-armed bandits from the reinforcement learning literature. Therefore, future work for this project can involve investigating the role of matrix methods in reinforcement learning for movie recommendations.

References

[1] F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*. 2015.

[2] Nicholas Hug. Surprise: A Python scikit for recommender systems. *GitHub*. 2015.

[3] Nick Becker. Matrix Factorization for Movie Recommendations in Python. *GitHub*. 2016.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *IEEE Transactions on Computers*. 2009.

[5] Sheng Zhang, Weihong Wang, James Ford and Fillia Makedon. Learning from Incomplete Ratings using Non-negative Matrix Factorization. *SIAM Conference on Data Mining*. 2006.

[6] Nicolas Gillis. The Why and How of Nonnegative Matrix Factorization. *arXiv preprint*. 2014.

[7] Xin Luo, Mengchu Zhou, Yunni Xia, and Qinsheng Zhu. An efficient non-negative matrix factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*. 2014.

[8] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*. 2007.

Appendix A

To run cross-validation for the kNN models, we used a Python library named `surprise` which has many functions to train and evaluate recommendation system algorithms. We used this library to implement two different kNN models (`KNNBasic` and `KNNWithMeans`) and create Figure 4 and 5. For cross-validation, we used the function `cross_validate` and the similarity metric could be passed in as an argument to this function.

To make the best 10 recommendations, we looked at the 10 nearest neighbors for the movie "Iron Man (2008)". The nearest neighbors were returned as indices of movie. We then queried these indices in the MovieLens dataset to get the movie names and genres.

Appendix B

The distribution of the singular values in Figure 7 was created by using the `svdvals` function from the Python `scipy` library.

To run cross-validation for the SVD models, we again used the `surprise` library. We used the `SVD` class from the library. To get the top 10 recommendations, we first decided to choose a user who rated "Iron Man (2008)" above 4.5 on the rating scale. Then, using several internal functions of the `surprise` library, we found the indices of the recommended movies and converted those to movie names and genres.

Appendix C

The cross-validation and recommendations for this part were done completely in `surprise`. The process of created recommendation with NNMF was very similar to that of the SVD and most of the code from Appendix B was reused in this part. The only difference between Appendix B and C is the algorithm that was being evaluated. We used the `NMF` class from the library.