Project 1: Recommender System

Martyna Konopacka
Wojciech Prokopowicz

March 2021

Contents

1 Introduction **1**

2 Filling the data matrix **2**

3 SVD and NMF **3**
3.1 Nonnegative Matrix Factorization 3
3.2 Singular Value Decomposition 3
3.3 SVD2 5

4 Stochastic Gradient Descent **5**

5 Results **6**
5.1 NMF 6
5.2 SVD1 6
5.3 SVD2 6
5.4 SGD 6

1 Introduction

The aim of this project is to write a Python program which given the training data from a database consisting of movie ratings of some users will be able to predict those users' ratings for all the other movies in the best way possible. We will present and compare a couple of methods of achieving this goal.

Comparing results
Our dataset consists of tuples of movie id, user id, rating and date. First we will divide the ratings of every user in $9:1$ ratio, so that 90% of the data goes into training dataset and remaining 10% into test dataset. We will compare predictions with test data by calculating the root-mean square error between original and predicted ratings.

The training data will be inserted into matrix such that $\boldsymbol{Z}[i, j]$ is a ith user's rating of the jth movie. This will obviously result in many empty entries and our goal is to give our best guess of what could be in those entries.

2 Filling the data matrix

Some methods require us to make an initial complement of the matrix $\boldsymbol{Z}$. In this section we compare different ways of filling missing entries by comparing RMSE of obtained matrices.

Zeros
Filling entries with zeros results in a large $RMSE$ over 3.6

Average ratings
Using an average of all ratings appearing in the training set is much better - the $RMSE$ achieved in this way is 1.036.

Better movies will have on average higher ratings. Moreover, every user rates movies differently (some give on average higher rates then others). Therefore we want to consider the matrix $\boldsymbol{Z}_{movie}$ which assigns to every movie its average rating and the matrix $\boldsymbol{Z}_{user}$ which assigns to every user an average rating amongst all movies they rated. Results are better now: $RMSE(\boldsymbol{Z}_{movie}) = 0.978$ and $RMSE(\boldsymbol{Z}_{user}) = 0.936$.

The best approach: weighted averages
We might assume that it would be even better if we could take into account both of these features of our data at the same time. Let us define the following matrix:

$$\boldsymbol{Z}_{mean} = \lambda \boldsymbol{Z}_{user} + (1 - \lambda)\boldsymbol{Z}_{movie}.$$

It is the weighted average between the two matrices for some parameter $\lambda \in (0, 1)$. Since we can't use the test data for choosing the optimal parameter λ we look at the *loss* for a given matrix which is the sum of squared differences between the ratings from the training set and corresponding estimates from this matrix. We now plot the *loss* depending on λ (see Fig. 1). In theory the matrix with the smallest *loss* should be the best fit.

We obtain the optimal *loss* for $\lambda = 0.35$. The matrix $\boldsymbol{Z}_{mean}$ for this parameter gives the $RMSE = 0.909$ which is significantly better then the previous results.

Close users averages
Another approach we have tried is to use the average ratings of the closest users (in terms of the Euclidean distance between rows of the matrix $\boldsymbol{Z_{user}}$. However, the achieved RMSE was greater than 1 and therefore this method is not used.

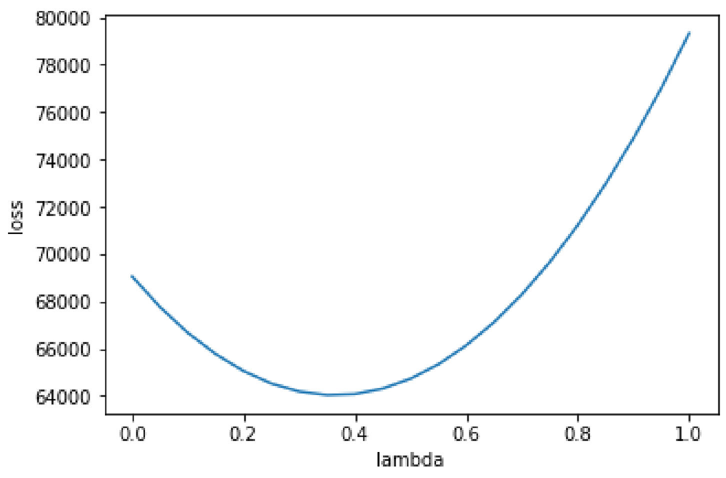

Figure 1: Loss for the matrix of the weighted averages

3 SVD and NMF

Following methods assume prior completion of the matrix.

3.1 Nonnegative Matrix Factorization

This procedure takes a non-negative matrix $(n \times d)$ (in our case $\boldsymbol{Z}$) and finds non-negative matrices $\boldsymbol{W}$ $n \times r$ and $\boldsymbol{H}$ $r \times d$, where $r << \min(n, d)$, such that $\boldsymbol{WH} \approx \boldsymbol{Z}$. Numerical methods are used to find $\boldsymbol{W}$ and $\boldsymbol{H}$ that minimize the error of approximation which in this case is the Frobenius norm of the difference between $\boldsymbol{Z}$ and $\boldsymbol{WH}$, where

$$||A||_{Fro}^2 = \sum_{i,j} A_{ij}^2.$$

The reason why performing this approximation is better then using the original matrix $\boldsymbol{Z}$ is that by looking at what constitutes a single entry of $\boldsymbol{WH}$ we can conclude that $\boldsymbol{W}$ represents the features of our data set and $\boldsymbol{H}$ - their importance (see Fig. 2), so by picking r we decide to look only at a relatively small number of the most important features during the recovery of $\boldsymbol{Z}$. This way we hope to promote looking at the patterns that appear in the data from the training set.

3.2 Singular Value Decomposition

Let Z be a matrix of size $n \times d$. Z can be decomposed into product of three matrices:

$$Z = U \Sigma V^T$$

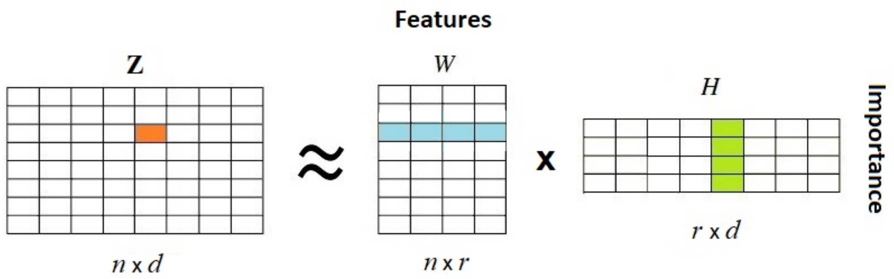

Figure 2: NMF

- U, V are orthonormal, $n \times n$ and $d \times d$
- Σ is diagonal, $n \times d$

In order to show how these matrices can be computed we assume that Z is indeed a product of U, Σ, V^T and perform eigendecomposition of ZZ^T and Z^TZ. It turns out that U and V are sets of eigenvectors of ZZ^T and Z^TZ respectively whereas entries of Σ are square roots of corresponding eigenvalues shared by both matrices. Note that ZZ^T and Z^TZ are correlation matrices of vectors and rows of Z. In terms of understanding intuitions behind the method it may be useful to notice that the correlation coefficient is the same as the cosine of angle between vectors X and Y.

$$r(X,Y) = \frac{cov(X,Y)}{\sqrt{var(X)}\sqrt{var(Y)}} = cos(\theta)$$

Next step is to sort U, Σ and V^T by eigenvalues in descending order. Now the leftmost column of U in some sense represents the most important direction in the space generated by data from matrix Z. The amount of information in first r columns can be computed as the ratio of sum of first r eigenvalues to sum of all of them. Now if we get rid of remaining $d - r$ columns and rows we obtain reduced form called truncated SVD

$$\tilde{Z} = U_r \Sigma_r V_r^T$$

which is of the same size as Z but contains less information. The reason why this approach can help us predict ratings is that now we focus on general trends in data and ignore fluctuations caused by minor differences between each individual user.

SVD is often used to perform a method called Principal Component Analysis which is in this case the same. Vectors of U are called principal components and can be thought of as axes such that the variance of points projected on them is the largest possible. Vectors of V^T multiplied by Σ are called loadings and gives coefficients of linear combination of principal components that results in corresponding vector in Z. For example in our problem PC1 (first component) might represent horror movies, PCA2 comedies and PCA3 dramas. Then if the first loading was $(0, 0.2, 0.8)^T$ it would mean that the first movie (first column) can be classified as a drama with a little bit of humor and without any horror features and all people who like such a mixture will rate it similarly.

3.3 SVD2

This method is a modification of Singular Value Decomposition, which can be described as follows:

1. Apply SVD to the matrix.
2. Fill the obtained matrix with the training data again.
3. Repeat until reaches specified number of iterations.

Note that too many iterations will result in worse results - results are too precisely fitted to training data and possibly different fit would be better for test data.

4 Stochastic Gradient Descent

Gradient descent is a method of finding local minimum of a function - for example *RMSE as a function of missing entries* in the matrix. In order to visualise how it works let $f(x)$ be a function $f : \mathbf{R^2} \to \mathbf{R}$. Our goal is to find $x \in \mathbf{R^2}$ so that the value of f is minimized - it corresponds to finding the lowest point on a 3d surface. We choose arbitrary x_0 and calculate the gradient $\nabla f(x_0)$. Moving against the direction of ∇f results in the greatest decrease of the value of f. Based on that fact we can update x so that

$$x_{n+1} = x_n - \alpha \nabla f(x_n)$$

where α determines the step size (we will call it the *learning rate*). Procedure repeats until one of following events: (1) algorithm reaches the maximal number of iterations (2) step becomes infinite (3) step length drops below chosen value called precision and thus becomes negligibly small. In the last case we know that we are close to local minimum or plateau. The idea generalises to higher-dimensional input.

Vector x is often so long, that repeated computing of all partial derivatives would be too time consuming. We can handle it by using subsets of parameter space (so called batches) in each iteration instead of the whole set. Intuitively, say, a website aims to improve some features based on users' opinions. Instead of waiting to collect many of them, it makes adjustments more frequently based on smaller groups in hope that it would be sufficiently representative. Such approach is called *Stochastic Gradient Descent*.

Some problems with simple SGD:

- choosing the right learning rate is in some cases problematic. Too small will drastically slow down the process, too big on the other hand may cause jumping over the minimum ending in oscillations or even divergence.
- algorithm will stuck in a suboptimal local minimum, plateau or saddle point due to null gradient
- moving inside the valley with sharp edges and mild slope in the direction of minimum is difficult

There are many modifications to basic SGD (such as adding momentum or using different learning rates to each parameter) attempting to manage listed problems. In this project we consider just basic version and see how good it applies to our problem.

5 Results

In all cases it was the best to use the weighted average matrix. Using SVD2 resulted in the greatest RMSE reduction. All results were under 0.96 (the worst for using NMF and SVD1 on the matrix filled with average movie ratings). Below there are results for a single partition of a dataset with parameters that gave the best RMSE. Exact values may differ due to random selection of test subset.

5.1 NMF

For matrix $\boldsymbol{Z}_{mean}$ NMF with rank 10 reduced RMSE by 0.864% from 0.909 to 0.901 (Total time: 11.66s)

5.2 SVD1

For matrix $\boldsymbol{Z}_{mean}$ SVD1 with rank 5 reduced RMSE by 1.721% from 0.909 to 0.893 (Total time: 4.78s)

5.3 SVD2

For matrix $\boldsymbol{Z}_{mean}$ SVD2 with rank 10, and 3 iterations reduced RMSE by 3.653% from 0.909 to 0.876 (Total time: 35.47s)

5.4 SGD

SGD with rank 5, learning rate 0.01 and batch size 225 result: 0.923 (Total time: 313.36s)