

MATH 482
Matrix Factorisation Project
Code Available: <https://github.com/danielbraithwt/MATH-482>

Daniel Braithwaite

June 28, 2017

1 Background

Non-negative matrix factorization is the problem given a non negative matrix V , find non negative factors W and H such that $V \approx WH$. If all entries of V are present then the problem can be solved with SVD. If entries of V are missing then SVD is undefined, if one where to ignore the missing values and use SVD anyway then the model will be prone to overfitting [?].

Consider that finding optimal WH can be done by minimizing the frobenius distance $J(X) = \sum_{i,a} ((WH)_{i,a} - V_{i,a})^2$

The problem of matrix factorisation when information is missing is equivalent to the problem of Weighted Matrix Factorization which

2 Introduction

To discuss the idea of matrix factorisation and methods to solve it first we must understand the motivation for wanting to solve such a problem. In the case of the Netflix challenge the problem was to build a system to recommend movies to users. We have this very large matrix R with the rows corresponding to a user and a column corresponding to a movie. The entry $R_{i,j}$ is the rating that user i gave movie j , in practice we would find that a very small percentage of this matrix would be filled in. To make recommendations we would like to predict the ratings which a user might give a movie which they haven't watched.

3 Matrix Factorization Solutions

3.1 Solution 1: $R = U \cdot M$

The first solution we consider is that R (an uxm matrix) is actually the product of two smaller matrices U and M . Where U (a uxk matrix) represents

the users in some latent feature space and M (a $m \times k$ matrix) represents the movies in the latent feature space. We consider $M_{i,j}$ to be the ammount movie i has feature j , likewise we consider $U_{i,j}$ to be how much user i is interested in movies with feature j . Then we can take the rating user i gives movie j to be $\hat{R}_{i,j} = \text{row}(U, i)^T \cdot \text{row}(M, j)$. Now the problem becomes how do we learn these matrices U and M .

We consider the folowing optimization problem, where G contains all pairs (i, j) for which we know $R_{i,j}$

$$\arg \min_{U, M} \sum_{(i,j) \in G} (R_{i,j} - \text{row}(U, i)^T \cdot \text{row}(M, j))^2$$

This optimization problem can be solved with gradient decent. We generate a matrix R which is (20x23) by multiplying two randomly generated matrcies U (20 x 15) and M (23 by 15). Initially the training set and test set are the same and we train over the entire data set. We train each situation 10 times each from random initial conditions.

latent factors	peformance (SSE)	95% CI
5	10.675	(10.648, 10.702)
10	0.767	(0.767, 0.767)
15	0.053	(0.039, 0.066)

Table 1: Table for latent features vs peformance, over entire data set

However in a real world situation we know that the problems which utilize matrix factorization usially involve factorizing sparse matrices, so what happens when we remove say half the entries. Now our data is split in two, half is our training data and the other half is our test data. The results below are generated by training with the same hyperparamaters as before

latent factors	training (SSE)	training 95% CI	test (SSE)	test 95% CI
5	1.11765105354	(1.027, 1.208)	211.201	(182.722, 239.680)
10	2.368e-07	(5.744e-09, 4.679e-07)	68.927	(63.564, 74.288)
15	2.083e-19	(-8.646e-20, 5.031e-19)	62.868	(59.319, 66.417)

Table 2: Table for latent features vs peformance, over partial data set

We observe that the test error is higher than the training error, and these differences are stastically significant which indicates that there is overfitting occouring.

One common approach to solve the overfitting that occurs in this type of matrix factorization is to use regularization on the matrices U and M . Making our optimization problem the folowing

$$\arg \min_{U, M} \left[\sum_{(i,j) \in G} (R_{i,j} - \text{row}(U, i)^T \cdot \text{row}(M, j))^2 \right] + \lambda(\|U\|_2 + \|M\|_2)$$

latent factors	training (SSE)	training 95% CI	test (SSE)	test 95% CI
5	19.783	(19.692, 19.875)	37.965	(37.830, 38.100)
10	19.423	(19.375, 19.473)	38.133	(38.002, 38.264)
15	19.347	(19.323, 19.372)	38.171	(38.117, 38.225)

Table 3: Table for latent features vs performance, over partial data set with regularization

While this does provide a reduction in overfitting it is still a significant problem. The next concept we can implement is that of biases, our model that we are learning is suppose to capture the interactions between the users and movies however we might find that in some cases a particular user gives mostly low ratings or a particular movie generally receives low ratings, these properties are interactions between the users and movies, perhaps the user is just harsh or the movie is simply bad. Biases capture this idea so the model can learn what is truly important.

latent factors	training (SSE)	training 95% CI	test (SSE)	test 95% CI
5	15.674	(15.633, 15.716)	35.282	(35.242, 35.323)
10	15.539	(15.519, 15.559)	35.288	(35.253, 35.322)
15	15.491	(15.479, 15.503)	35.298	(35.281, 35.316)

Table 4: Table for latent features vs performance, over partial data set with regularization and biases

3.2 Solution 2: Using Neural Networks

In this section we present two similar solutions each using neural networks, only difference being whether we use two neural networks or one.

3.2.1 Two Neural Networks

In the same set up as before there is a matrix R with rows representing users and columns representing movies. Our aim is to optimize the following. Take two neural networks f_θ which takes a row of R to some latent feature space and f_ϕ which takes columns of R to some feature space. Then we compute the ranking user i gives movie j by the following $\hat{R}_{i,j} = f_\theta(\text{user}_i)^T \cdot f_\phi(\text{movie}_j)$. Giving us the following optimization problem (where G is defined as before)

$$\arg \min_{\theta, \phi} \sum_{(i,j) \in G} (R_{i,j} - f_\theta(\text{user}_i)^T \cdot f_\phi(\text{movie}_j))^2$$

Over the same data as before we get the following performance

latent factors	training (SSE)	training 95% CI	test (SSE)	test 95% CI
5	24.767	(24.767, 24.767)	27.334	(27.334, 27.334)
10	17.004	(16.822, 17.185)	16.566	(16.397, 16.734)
15	5.080	(5.0312, 5.128)	4.978	(4.905, 5.051)

Table 5: Table for latent features vs performance, over partial data set with regularization and biases

This method has practically no overfitting as well as consistently better test performance.

3.2.2 Single Neural Network

This approach is very similar to the one just presented, however instead now we only have one neural network f_ψ , which takes some row of R representing a user and some column of R representing a movie and outputs a rating. Making our approximation of ratings $\hat{R}_{i,j} = f_\psi(user_i, movie_j)$, and finally giving us the following optimization problem.

$$\arg \min_{\theta, \phi} \sum_{(i,j) \in G} (R_{i,j} - f_\psi(user_i, movie_j))^2$$

4 Factorization Method Comparison

We wish to compare the performance of these methods against each other and identify the tradeoffs between them. Testing our factorization methods on randomly generated data is a good way to develop an understanding in a small environment but really we want to see how these methods perform on real data. We will be using the MovieLens 100K data set, consisting of 943 users and 1682 movies where each user has rated at least 20 movies.

4.1 Conclusions