MATH 482

Matrix Factorisation Project

Code Available: https://github.com/danielbraithwt/MATH-482

Daniel Braithwaite

June 28, 2017

1 Background

1.1 Matrix Factorization

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 to approximae V when all values are present can be done by minimizing the frobenius distance $J(X) = \sum_{i,a}((WH)_{i,a} - V_{i,a}$, this is Unweighted Matrix Factorization and any crtical point that isnt a global minima is a saddle point. However the case where values are missing is called Weighted Matrix Factorisation, in this case there might be local minima [?].

1.2 Recommendar Systems

Recomender systems are widely used to identify items of usse for various users. **Collaborative Filtering** considers historial interactions alone, works by collecting user feedback. **Content-based Filtering** utilize the attributes of the iterms and users.

2 Introduction

To dicuss 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 Netflicks challenge the problem was to build a system to recomend movies to users. We have this very large martix R with the rows corosponding to a user and a column corosponding 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 recomendations we would like to predict the ratings which a user might give a movie which they havent watched.

3 Matrix Factorization Solutions

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

The first soluton we consider is that R (an uxm matrix) is actually the product of two smaller matricies U and M. Where U (a uxk matrix) represents the users in some latent feature space and M (a mxk 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 $\hat{R}_{i,j} = row(U, i)^T \cdot row(M, j)$. Now the problem becomes how do we learn these matricies U and M.

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

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

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 matricies, 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 occours in this type of matrix factorization is to use regularization on the matricies U and M. Making our optimization problem the folowing

$$\underset{U,M}{\arg\min} \quad \Big[\sum_{(i,j)\in G} (R_{i,j} - row(U,i)^T \cdot row(M,j))^2 \Big] + \lambda(\|U\|_2 + \|M\|_2)$$

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 peformance, over partial data set with regulrization

While this does provide a reduction in overfitting it is sill 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 perticular user gives mostly low ratings or a perticular movie generally recieves low ratings, these propertys arnt interactions between the users and movies, prehapse the user is just harsh or the movie is simply bad. Biases capture this idea so the model can learn what is truely 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 peformance, over partial data set with regulirzation 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 folowing. Take two nerual 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 folowing $\hat{R}_{i,j} = f_\theta(user_i)^T \cdot f_\phi(movie_j)$. Giving us the folowing optimization problem (where G is defined as before)

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

Over the same data as before we get the folowing peformance

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 peformance, over partial data set with regulirzation and biases

This method has practically no overfitting aswell as consistantly better test peformance .

3.2.2 Single Neural Network

This approach is very similar to the one just presented, how ever insted 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 folowing optimization problem.

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

4 Factorization Method Comparason

We wish to compare the peformance 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 understadning in a small environment but really we want to see how these methods peform on real data.

We will be using the MovieLens 100K data set, consiting of 943 users and 1682 movies where each user has rated atleast 20 movies.

4.1 Conclusions