Movie Box

Bingyu Wang, Mingzi Yi, MinHsiu Hsieh, Yifan Guo
Contract Email: mingziyi2014@u.northwestern.edu
EECS 349: Machine Learning, Northwestern University

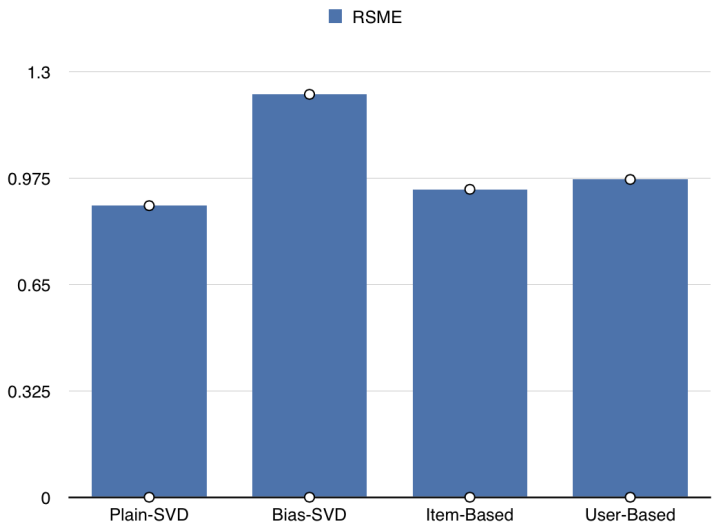
Abstract:

The main task of our project is to recommend most likely interesting movies that the user may prefer based on one's history record of movie ratings. Many online shopping websites like Amazon, eBay and Alibaba use recommendation systems to enhance user experience and guide customer to purchase potential shopping items. Thus, it's necessary and essential to generate a perfect Recommender Systems[1].

In our project, we explore the techniques of Recommender Systems with four different models - Plain Matrix Factorization Model with Singular Value Decomposition(SVD)(Model 1), Biased Matrix Factorization Model with SVD(Model 2), KNN based model with different focus: Item-Based and User-Based(Model 3 and 4). The attributes we mainly use from original dataset are user ID (discrete, numeric), movie ID (discrete, numeric) and score(discrete, ordinal) range from 1 to 5. In order to facilitate further processing, we initially map all the disordered numeric domains to integers with order.

The processed dataset consists of 200, 000 ratings from 500 users on more than 9,000 movies. After applying data into four different models, we will generate the predicted score for the unscored movies. Then we choose RMSE(Root Mean Square Error) to evaluate the system and compare the performance of different via with 10-fold cross validation. Our goal is to make RMSE close to 0.9. Finally, we would use the model with best performance to output the recommendations.

The RMSE of four model under 10-fold cross validation correspondingly is: 0.89, 1.23, 0.94 and 0.97. Expect model 2, all of the rest models are close to achieve our setted goal.



Specific Investigation:

1. Data preprocessing

We captured the original data from movie reviews from amazo and more information could be find via link: https://snap.stanford.edu/data/web-Movies.html. The first challenge we confront is the size of data - 8G. It took us more than 3 hours to parse data and split into smaller dataset. And we also tried to optimise the options of I/O during paring data in Java. Another challenge is that the user ID and movie ID are discrete disordered numeric attributes which could not be easily used in recommendation systems. We also need to construct the mapping from disordered numeric domains to integers in order with no gap and generate the mapping file for future usage.

After initially processing, we maintained three attributes we need (user ID, movie ID and score) from original data and get the clear dataset with size of 10 M. It consists of 200,000 ratings from 500 users on more than 9,000 movies. A sample of comparison between processed and original data is shown as follows:

<table>
<tr><th>Original Data</th><th>Mapping Relationship</th><th>Processed Date</th></tr>
<tr>
<td>
<table>
<tr><th>UID</th><th>MID</th><th>SCORE</th></tr>
<tr><td>4233</td><td>12</td><td>5.0</td></tr>
<tr><td>4244</td><td>6732</td><td>3.0</td></tr>
<tr><td>1549</td><td>3508</td><td>4.0</td></tr>
<tr><td>4233</td><td>3478</td><td>4.0</td></tr>
</table>
There exists gap between two nearest UID and MID.
</td>
<td>
User Mapping:
4233 -> 1223
4244 -> 1224
1549 -> 496

Movie Mapping:
12 -> 5
6732 -> 1789
3508 -> 939
3478 -> 938
</td>
<td>
<table>
<tr><th>UID</th><th>MID</th><th>SCORE</th></tr>
<tr><td>1223</td><td>5</td><td>5.0</td></tr>
<tr><td>1224</td><td>1789</td><td>3.0</td></tr>
<tr><td>496</td><td>939</td><td>4.0</td></tr>
<tr><td>1223</td><td>938</td><td>4.0</td></tr>
</table>
There is no gap between two nearest UID and MID.
</td>
</tr>
</table>

2. Modeling Construction

One approach to the design of recommender systems that has seen wide use is collaborative filtering(CF). Collaborative filtering methods are based on collecting and analyzing a large amount of information on users' behaviors, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself.

Then, based on the basic CF model, using K nearest neighbour algorithm, we construct two models with different focus: Item-Based collaborative filtering Model and User-based collaborative filtering.

Another approach referred in paper [1] is matrix factorization models, which are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels, that is determined in the Netflix Prize competition.

Then, we sequentially based on matrix factorization models with algorithm of Singular Value Decomposition (SVD), generating another two models: Plain SVD Model and Biased SVD Model. With given processed data, we could predict the the unlabeled movie's score for each user after applying SVD, then with method of stochastic gradient descent to regenerate the prediction in order to reduce the RMSE.

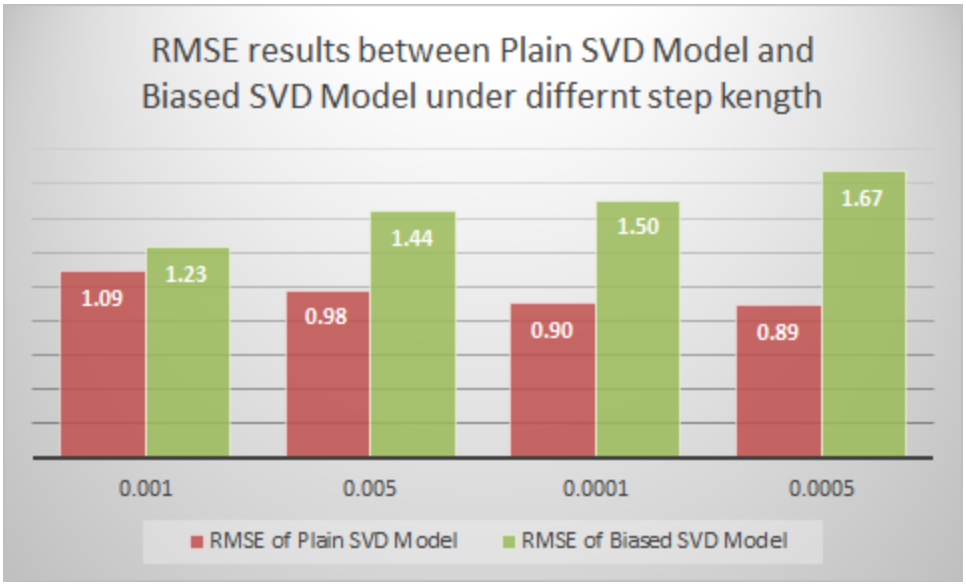
3. Details on Training

All the running codes are completely written by ourselves with Java, and could be find on our github link (https://github.com/YiMysty/SVD). The main functions are also attached at the end of this file, please feel free to see if you're interested.

With 10-fold cross validation on 200,000 samples, it means that we use 180,000 samples for training and the rest 20,000 for validation each iteration.

4. Evaluate the Modeling

Firstly, we just compared the RMSE results between Plain SVD Model and Biased SVD Model. Here we set different step length, separating dimension with 100, maximum iteration count with 1000, regularized factor donated as k with 0.02(avoid overfitting the observed data by regularizing the learned parameters) and stop threshold with 0.5.



Here, we should want to express that due to the maximum iteration with 1000 (because that it took more 3 hours to get each result), according to [1], if the iteration is larger enough, the Biased SVD Model may achieve a better result.

Next we will show the difference of using regularized factor with 0.02 and without regularized factor(set factor with 0) on two models.

Plain SVD Model when k = 0

Step Length	Stop Iteration	RMSE
0.001	70	1.03
0.0005	125	0.96
0.0001	583	0.93
0.00005	1000	0.92

Biased SVD Model when k = 0

Step Length	Stop Iteration	RMSE
0.001	1000	1.77
0.0005	1000	1.74
0.0001	1000	1.71
0.00005	1000	1.70

Plain SVD Model when k = 0.02

Step Length	Stop Iteration	RMSE
0.001	176	1.09
0.0005	292	0.98
0.0001	1000	0.90
0.00005	1000	0.89

Biased SVD Model when k = 0.02

Step Length	Stop Iteration	RMSE
0.001	1000	1.23
0.0005	1000	1.44
0.0001	1000	1.50
0.00005	1000	1.67

From the table above, we may found that both two models after adding regularized factor perform better than those without adding regularized factor.

Finally, we will present the result of using Item-Based and User-based Collaborative filtering Model.

Item - based & User - based:

Since we have 1500 users and 100w movie, when computing the similarity matrix, because of the limitation of the computer, we can not declare a matrix that is 100w*100w. So, we capture parts of data, 1500 users and 5000 movies.

Below is the RMSE result after 10-cross validation:

Item - based

	1	2	3	4	5	6	7	8	9	10
RMSE	0.932	0.974	0.953	0.946	0.952	0.931	0.932	0.961	0.92	0.9

The average: 0.9401

User - based

	1	2	3	4	5	6	7	8	9	10
RMSE	0.954	1.031	0.974	0.947	1.1	0.968	0.937	0.968	0.936	0.928

The average: 0.9743

The result is good enough, I conclude the candidates that can potentially affect my result:

1. The training data. Since we can not run our method with all of our training datas, the part we capture from the original datas is very critical.
2. Similarity calculation. We choose Cosine-Based Similarity. There are some others different similarity calculation like Pearson (correlation)-based similarity, Adjusted Cosine Similarity, which “might” improve our result.

5. Future Work

If more time is allowed, we would add more attributes into our data sets, like timestamp and pain text review. Then, for the attribute of time, we would generate the Item-Based or User-based Collaborative filtering Model considering the influence of time factor. And the plain text, we could use Bayes Network to help on recommendation in term of text reviews.

Reference:

[1] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37.
[2] Recommender Systems - Collaborative Filtering and other approaches, Xavier Amatriain Research/Engineering Director from Netflix
[3] Recommender system from Wikipedia: http://en.wikipedia.org/wiki/Recommender_system

Attachment:

Considering the length of report, here we just list two Model Functions,and another two is rather similar as these two:

```
/*1. BiasSVD Model */
package model;

import java.io.IOException;

import reader.ConfigReader;
import reader.DataProcessor;
```

```
import entity.AverageEntity;
import entity.Record;
import entity.RecordContainer;

public class BiasSVD {
	float[][] userMatrix;
	float[][] movieMatrix;
	float[][] scoreMatrix;
	float[][] userdiff;
	float[][] moviediff;
	AverageEntity[]  userAverage;
	AverageEntity[]  movieAverage;
	float[]userAverageDiff;
	float[]movieAverageDiff;
	float globalAverage = 0;
	float maxScore = 0.0f;
	float minScore = 0.0f;
	static float small = 0.0000001f;

	public void loadData(RecordContainer trainingData){
		System.out.println("initialize the score matrix.....");
		ConfigReader reader = new ConfigReader();
		DataProcessor processor = new DataProcessor();
		RecordContainer data = trainingData;
		if(data==null){
			try {
				data = processor.DataLoader();
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		userMatrix = new float[data.getUserNum()][reader.getConfigurationReader().getDimension()];
		userdiff  = new float[data.getUserNum()][reader.getConfigurationReader().getDimension()];
		movieMatrix = new
float[data.getMovieNum()][reader.getConfigurationReader().getDimension()];
		moviediff = new float[data.getMovieNum()][reader.getConfigurationReader().getDimension()];
		scoreMatrix = new float[data.getUserNum()][data.getMovieNum()];
		userAverage = new AverageEntity[data.getUserNum()];
		userAverageDiff = new float[data.getUserNum()];
		movieAverage= new AverageEntity[data.getMovieNum()];
		movieAverageDiff = new float[data.getMovieNum()];
		for(int i=0;i<data.getUserNum();i++)
			userAverage[i] = new AverageEntity();
		for(int i=0;i<data.getMovieNum();i++)
			movieAverage[i] = new AverageEntity();
		globalAverage = data.getAverageScore();
		maxScore = data.getMaxPreference();
		minScore = data.getMinPreference();
		//initialize the scoreMatrix
```

```
		for(int i=0;i<scoreMatrix.length;i++)
			for(int j=0;j<scoreMatrix[0].length;j++){
				scoreMatrix[i][j] = 0.0f;
			}
		Record u = null;
		while((u=data.getNext())!=null){
			scoreMatrix[u.getId()][u.getMovieId()] = u.getScore();
			userAverage[u.getId()].addNum(u.getScore());
			movieAverage[u.getMovieId()].addNum(u.getScore());
		}
		for(AverageEntity e:userAverage){
			e.setAverage(e.getAverage()-data.getAverageScore());
		}
		for(AverageEntity e:movieAverage){
			e.setAverage(e.getAverage()-data.getAverageScore());
		}
		System.out.println("finished intialize the score matrix...");
		System.out.println("initialize the userMatrix and movieMatrix");
		for(int i=0;i<userMatrix.length;i++)
			for(int j=0;j<userMatrix[0].length;j++){
				//userMatrix[i][j] = (float)
(Math.random()*Math.sqrt((data.getMaxPreference()-data.getMinPreference())/reader.getConfigurationReader().
getDimension()));
				userMatrix[i][j] = (float) ((float)Math.random()*Math.sqrt(2));
			}
		for(int i=0;i<movieMatrix.length;i++)
			for(int j=0;j<movieMatrix[0].length;j++){
				//movieMatrix[i][j] = (float)
(Math.random()*Math.sqrt((data.getMaxPreference()-data.getMinPreference())/reader.getConfigurationReader().
getDimension()));
				movieMatrix[i][j] = (float) ((float)Math.random()*Math.sqrt(2));
			}
		System.out.println("finished initialize the userMatrix and movieMatrix");
		reader.close();
	}
	private void resetDiff(){
		for(int i=0;i<userAverageDiff.length;i++)
			userAverageDiff[i] = 0;
		for(int i=0;i<movieAverageDiff.length;i++)
			movieAverageDiff[i] = 0;
		for(int i=0;i<userdiff.length;i++)
			for(int j=0;j<userdiff[0].length;j++)
				userdiff[i][j] = 0.0f;

		for(int i=0;i<moviediff.length;i++)
			for(int j=0;j<moviediff[0].length;j++)
				moviediff[i][j] = 0.0f;
	}
	public float doUpdate(){
		resetDiff();
```

```
		ConfigReader reader = new ConfigReader();
		int dimension = reader.getConfigurationReader().getDimension();
		float regular = reader.getConfigurationReader().getk();
		float learningrate = reader.getConfigurationReader().getLearningRate();
		float movieError[] = new float[movieMatrix.length];
		for(int  i=0;i<movieError.length;i++)
			movieError[i] = 0;
		reader.close();
		float totalerror = 0;
		int count = 0;
		for(int i=0;i<userMatrix.length;i++){
			int userError = 0;
			for(int j=0;j<movieMatrix.length;j++){
				if(Math.abs(scoreMatrix[i][j]-0.0f)>small){
					float predictValue = doPrediction(i,j);
					float error = scoreMatrix[i][j]-predictValue;
					count+=1;
					totalerror+=error*error;
					for(int k=0;k<dimension;k++){

userdiff[i][k]+=error*movieMatrix[j][k]-regular*userMatrix[i][k];

moviediff[j][k]+=error*userMatrix[i][k]-regular*movieMatrix[j][k];
					}
					userError +=error;
					movieError[j]+=error;
				}
			}
			userAverageDiff[i] = (userError-regular)*userAverage[i].getAverage();
		}
		//Calculate the movieAverageDiff

		for(int i=0;i<userMatrix.length;i++)
			for(int j=0;j<dimension;j++)
				userMatrix[i][j] +=learningrate*userdiff[i][j];
		for(int i=0;i<userAverageDiff.length;i++){
//
userAverage[i].setAverage(userAverage[i].getAverage()+learningrate*userAverageDiff[i]);
		}
		for(int i=0;i<movieError.length;i++){
//			float diff = (movieError[i]-regular)*movieAverage[i].getAverage();
//			movieAverage[i].setAverage(movieAverage[i].getAverage()+learningrate*diff);
		}
		for(int i=0;i<movieMatrix.length;i++)
			for(int j=0;j<dimension;j++)
				movieMatrix[i][j]+=learningrate*moviediff[i][j];
		return (float) (Math.sqrt(totalerror/(float)count));
	}
	public float doPrediction(int u,int m){
		ConfigReader reader = new ConfigReader();
```

```
                float score = 0.0f;
                for(int i=0;i<reader.getConfigurationReader().getDimension();i++){
                        score +=userMatrix[u][i]*movieMatrix[m][i];
                }
                score+=globalAverage+userAverage[u].getAverage()+movieAverage[m].getAverage();
                if(score>maxScore){
                        score =maxScore;
                }else if(score<minScore){
                        score = minScore;
                }
                return score;
        }
}

/*2. Item-Based Model*/
package model;

import entity.Record;
import entity.RecordContainer;

public class ItemBased{
        float[][] scoreMatrix;
        float[][] similarityMatrix;
        float threshold = 0.5f;
        int UserNum=0;
        int MovieNum=0;
        public void loadData(RecordContainer train){
                RecordContainer data = train;
                UserNum=data.getUserNum();
                MovieNum=data.getMovieNum();
                scoreMatrix = new float[UserNum][MovieNum];
                //initialize the scoreMatrix
                for(int i=0;i<scoreMatrix.length;i++)
                        for(int j=0;j<scoreMatrix[0].length;j++){
                                scoreMatrix[i][j] = 0.0f;
                        }
                Record u = null;
                while((u=data.getNext())!=null){
                        scoreMatrix[u.getId()][u.getMovieId()] = u.getScore();
                }
                System.out.println("finished intialize the score matrix...");

        }
        public void similarityMatrix(){
                //built the similarity Matrix
                float[] vec1;
                float[] vec2;
                float[] tempVec1;
                float[] tempVec2;
                float tempSimilarity=0.0f;
```

```
		int index=0;
		vec1=new float[UserNum];
		vec2=new float[UserNum];
		//initialize vec1 and vec2
		for(int i=0;i<vec1.length;i++){
				vec1[i] = 0.0f;
				vec2[i] = 0.0f;
		}
		//build similarity Matrix
		for(int i=0;i<scoreMatrix[0].length;i++){
			for(int j=0;j<scoreMatrix[0].length;j++){
				tempVec1=getCol(i);
				tempVec2=getCol(j);
				index=0;
				//remove 0, make length of two array equal
				for(int k=0;j<tempVec1.length;k++){
					if(tempVec1[k]!=0.0f && tempVec2[k]!=0.0f){
						vec1[index]=tempVec1[k];
						vec2[index]=tempVec2[k];
						index++;
					}
				}
				tempSimilarity=cosineSimilarity(vec1,vec2);
				similarityMatrix[i][j]=tempSimilarity;
			}
		}
	}
	public void doPrediction(){
		//Predict, fill out the scoreMatrix
		float weightedScoreSum=0.0f;
		float counter=0.0f;
		for(int i=0;i<scoreMatrix.length;i++){
			for(int j=0;j<scoreMatrix[0].length;j++){
				if(scoreMatrix[i][j]==0.0f){
					for(int k=0;k<similarityMatrix[0].length;k++){ //find the similar item
according to threshold;
						if(similarityMatrix[j][k]>threshold && scoreMatrix[i][k]!=0.0f ){

weightedScoreSum+=similarityMatrix[j][k]*scoreMatrix[i][k]; //similarity(weight) * known score
							counter++;
						}
					}
					scoreMatrix[i][j]=weightedScoreSum/counter; //prediction!
					weightedScoreSum=0.0f; //reset
					counter=0.0f;	//reset
				}
			}
		}
	}
	public float[] getCol(int x){
```

```
		//Get xth column from scoreMatrix
		float[] targetCol;
		targetCol=new float[UserNum];
		//initialize targetCol
		for(int i=0;i<targetCol.length;i++){
			targetCol[i] = 0.0f;
		}

		for(int i=0;i<scoreMatrix[0].length;i++){
			targetCol[i]=scoreMatrix[i][x];
		}

		return targetCol;
	}
	public float cosineSimilarity(float x[], float y[]){
		//compute cosineSimilarity of two vector
		double dotProduct=0.0;
		double magnitude1=0.0;
		double magnitude2=0.0;
		double cosineSimilarity=0.0;
		for(int i=0;i<x.length;i++){
			dotProduct+=x[i]*y[i];
			magnitude1+=Math.pow(x[i],2);
			magnitude2+=Math.pow(y[i],2);
		}
		magnitude1=Math.sqrt(magnitude1);
		magnitude2=Math.sqrt(magnitude2);
		cosineSimilarity = dotProduct / (magnitude1 * magnitude2);
		return (float)cosineSimilarity;
	}

}
```