Recommendation Systems Project

Yacine Mokhtari
Lilia Izri

October 2022

Contents

1 Introduction **2**

2 Collaborative filtering **2**
2.1 Item-Item approach 2
2.2 Experiments and results 3
2.3 Possible optimizations 4

3 Bonus: UV-Decomposition **5**
3.1 Algorithm 5
3.2 Experiments and Results 5
3.3 Improvements and Overfitting 6

4 Conclusion **7**
4.1 Preview of the recommendations 7

A Appendix **8**
A.1 Results of CF 8
A.2 Results of UV-decomposition 8

1 Introduction

Throughout this project, we are asked to build a recommendation system that uses eaither the *content-based* approach or the *collaborative filttering* approach; one of the two famous approaches and most effective ones in the recommendation system field.

Here, we will implement and study two different algorithms, the collaborative filtering with a keen on the item-item approach and a *bonus* one using the UV-decomposition technique.[1]

The dataset that is used in this study is the small MovieLens dataset containing arround 100,000 ratings applied to 9,000 movies by 610 users.

---

2 Collaborative filtering

2.1 Item-Item approach

Utility matrix

In order to implement the collaborative filtering algorithm we need an utility (function) matrix r, that will associate for each pair of (*user, movie*) a rating : $user \times movie \rightarrow rating$. So the matrix will be of size $nb\_users \times nb\_movies$.

We assume that the missing values are bad grades, so in the beginning, we fill the blanks (unknown rating) with a -1 value.

Cosine similarity

We chose as a similarity metric, the cosine similarity which allows us to decide if two movies are similar independently of the size of the vector that represents movie.

We define the cosine similarity matrix *similarity_matrix* as the matrix with all the similarities. And, in order to compute the cosine similarity between movies, we replace the missed values with 0s (instead of -1).

Algorithm

The principle of the collaborative filtering algorithm is to recommend movies to an user according to their similarities to the movies already seen by the user.

Then, to predict the rating for a given user x and a given movie i, we use a weighted average over the similar movies to i seen by the user $N(i;x)$. And a kind of bias proper to the pair (user, movie) each time. So we end up with this formula:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} sim(i,j)(r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} sim(i,j)}$$

With $b_{xi} = \mu + b_x + b_i$ the baseline estimator "bias".

For the movies without any similar movie seen by an user we decided to tried different arbitrary values: 0, $\frac{1}{2}b_{xi}$, b_{xi}...etc.

To get the set $N(i;x)$ we rely on the results of the LSH algorithm. Indeed, $N(i;x)$ will be equal to the movies "similar" to the movie i (results of LSH) intersected with the movies seen by user x. Plus, we filter all of this by keeping only the movies with similarity above some **threshold**. So at end, we have a new utility matrix with all the "missing" ratings filled using the algorithm.

[1] The UV-decomposition is a matrix decomposition. This is one of the techniques of the **latent factors** that are not covered in this course, but still, we wanted to give it a try!

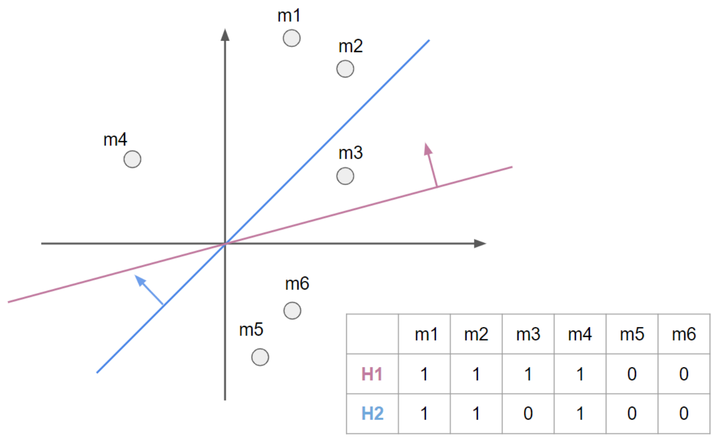
Optimization

Since it takes much time to compute the k most similar movies to a movie given a user, we use an optimization to reduce the number of movies to check for each user. Instead of checking for each user, and each item, all the movies that we have in our training set if they are similar enough or not (the cosine similarity is above some threshold). We will use Locality Sensitive Hashing (LSH) group the movies that are potentially similar together so we reduce the number of comparisons.

- **Signature matrix**

  To perform the LSH, we create a signature matrix of size n $\times$ nb_movies of our utility matrix. The point is to reduce the dimension of each movie and to be able to group the movies efficiently.

  For that, we generate n hyper-planes (i.e normal vectors of these hyper-planes, using a normal distribution) and compute the dot product between each movie and each vector that represents an hyperplane. If the dot product between the two is a positive value, we set the corresponding value in the signature matrix to 1, else to 0.



	m1	m2	m3	m4	m5	m6
H1	1	1	1	1	0	0
H2	1	1	0	1	0	0

Figure 1: Illustration of a signature matrix

- **Locality Sensitive Hashing (LSH)**

  LSH helps us to group probably similar items together into buckets, to reduce the number of comparisons we will do. To do that, we will split our signature matrix into m bands and we will consider that items/movies are similar if in a given band, their signatures are the same. So for each band, we will hash the movies that have the same partial signature to a same bucket.

  We will use a **dictionary** to keep track of the results. Therefore, for each movie i, we will have a list of the potential similar movies (i.e the movies that were hashed to the same bucket as the movie i).

2.2 Experiments and results

Before diving into the analysis of the results, we mention that we split our initial dataset into a training set we use to build the initial utility matrix and a test set to make evaluations. In addition of that, **we have studied two cases**:
A. Setting the ratings of movies not similar to the ones an user rated to 0.
B. Setting the ratings of movies not similar to the ones an user rated to $\frac{1}{2}b_{xi}$.

Accuracy evaluation and running time

To analyze our algorithm we chose to track the root mean squared error (RMSE) and the running time (whitout counting other executions like LSH or else).

The RMSE allows us to measure accuracy for continuous variables and has the particularity to penalize more biggest errors (because of the squared) and to return results in the same order of our data.

$$RMSE(\hat{r}, r) = \sqrt{\frac{1}{N}\sum_{i}^{N}(\hat{r_i} - r_i)^2}$$

Where $\hat{r}$ is the true rating of a movie and r the rating predicted by the algorithm for this movie.

We computed the RMSE on the test set and the running time for different combinations of parameters. For each of the cases mentionned before (A and B), we have:

On the one hand (plots on the right), we calculate the RMSE and running time for a fixed number of bands (used in the LSH algorithm) while varying the similarity threshold. On the other hand (plots on the left), we do the opposite: we fix the similarity threshold and vary the number of bands. We repeat the experiments two times for two different sizes of bands: 10 (Blue), 20 (Yellow).

We present the results we obtained in the figures in the appendix section. We can see that, in the case A the RMSE decreases when increasing the number of bands used in LSH but increases when increasing the threshold similarity. While in the case B we have the opposite situation.

In both cases (A and B) the running time increases with the number of bands we use in LSH and decreases with similarity threshold (figure 6).

Remark ⚠

As mentionned earlier, we gave to the movies that have no similar movies seen by an user (i.e movies i with the set $N(i, x)$ empty) arbitrary values. We noticed that the RMSE depends a lot on this particular value. And this is mostly due to the sparsity of our data.

So changing the arbitrary rating given to those movies, changes the RMSE a lot. If for example, we set this value to 0 (not recommended at all) the RMSE we can see in the plots (figure 5) is around 3. If instead we put b_{xi} or $\frac{1}{2}b_{xi}$, the RMSE is around $1 \sim 1.5$.

But having a smaller RMSE doesn't really mean that that the recommendations we make are better. For example when using b_{xi}, we will recommend more the movies well rated even if they are not similar at all to the ones an user rated (especially if the user has a big average rating and so the movie). So the recommendations don't seem to be relevant for the users.

Some issues

- Computational issues: even using LSH, it still takes time to compute ratings for all users and movies.
- Sparsity problem: How to deal with movies that have similar movies and/or users that have not watched enough movies.

2.3 Possible optimizations

- Improve the computations to reduce the running time (cause it's pretty slow).
- Compute the p@k precision in addition to RMSE to compare the relevancy of the recommendations.
- Testing more combinations of parameters: number of hyper-planes to build the signature matrix, number/size of the bands, ... etc.
- Testing other techniques to get the similar movies (clustering, dimensionality reduction ...etc) and compare the results we obtain.

---

3 Bonus: UV-Decomposition

The UV-decomposition is a matrix decomposition method in linear algebra. This decomposition will help us to find the best estimate for missing values in the utility matrix by trying to recreate a matrix which coefficient approaches the known coefficients of the Utility Matrix.

3.1 Algorithm

Let us denote the utility matrix (where in this case, the unknown ratings are set to the NaN object) as M, an $n \times m$ matrix with n being the number of users and m the number of films. This approach consist in finding too matrices U and V such that their product will give an *estimate* of (the missing values of) M. This process is called $UV-decomposition$ of M.

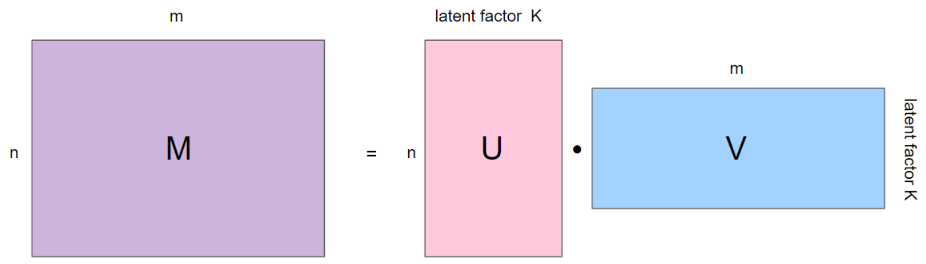


Figure 2: $UV-decomposition$ schema

The measure we used to say how close the UV product and the matrix M are is the *Root-Mean-Square-Error (RMSE)*, a quick reminder, it is the sum over the differences between each nonblank entry of M and its corresponding entry in the UV product, devided by the number of nonblank coefficient of M.

In order to find a UV-decompositoin with the least $RMSE$ value, we may start with arbitrary values for U and V. Here we chose to randomly initialize them from a *Standard Normal Distribution*.

We proceed with the optimization (single adjustment per iteration) as follows:

1. Select a random coefficient u_{rs} from U (or v_{rs} from V).
2. Optimizing that arbitrary element.
   - The general formula for u_{rs} that optimizes the $RMSE$ is: $\frac{\sum_j v_{sj}\left(m_{rj} - \sum_{k\neq s} u_{rk}vkj\right)}{\sum_j v_{sj}^2}$
   - The general formula for v_{rs} that optimizes the $RMSE$ is: $\frac{\sum_i u_{ir}\left(m_{is} - \sum_{k\neq r} u_{ik}vks\right)}{\sum_i u_{ir}^2}$
3. Back to step 1.

One may note that we can optimize a coefficient more than one time.
The stop criteria we chose was a maximum number of iterations.

3.2 Experiments and Results

We kept 10% random samples among our data set to construct the *test set*, the remaining 90% is used while optimizing the decomposition.
2 configurations were studied where the first one (A) consists in optimizing a single UV-decomposition and the second one (B) is the average of 3 different UV-decompositions, optimized independently from one another.

This operation is repeated for several values of k - the latent factor.

You can find the training loss plot of these two configuration in the appendix (Reference).

Root Mean Square Error & computation time

We gathered some data in the tables below while training such as the computation time for the optimization of each single UV-decomposition and the resulting $RMSE$ value of the optimized configuration for each combination:

Table 1: "Configuration A"

K	**MaxIter**	**opt time**	**RMSE (Test)**
2	20	1.634s	2.263
	200	15.002s	2.074
	500	37.427s	2.069
	1000	75.258s	2.269
5	20	1.615s	3.629
	200	14.842s	2.872
	500	36.787s	2.664
	1000	73.567s	2.674
15	20	1.685s	2.838
	200	15.478s	2.859
	500	41.032s	3.39
	1000	77.205s	2.975
50	20	2.0s	2.575
	200	18.161s	3.032
	500	47.903s	3.272
	1000	92.195s	3.136

Table 2: "Configuration B"

K	**MaxIter**	**opt time**	**RMSE (test)**
2	20	4.999s	2.156
	200	47.031s	1.962
	500	114.233s	2.084
	1000	233.505s	2.252
5	20	5.073s	3.452
	200	47.326s	2.671
	500	118.219s	2.953
	1000	234.852s	2.422
15	20	5.121s	2.887
	200	46.687s	2.96
	500	117.443s	2.936
	1000	240.139s	3.101
50	20	6.058s	2.578
	200	55.233s	2.9
	500	143.687s	3.105
	1000	283.464s	3.235

- One can clearly see that using multiple UV-decomposition can significantly decrease the $RMSE$ on the test set, but this naturally may cost a little bit more computation time.
- Another remark would be that a very few iterations are sufficient for these models to predict well on unseen data. Although during training the loss decreases7, it gradually increases during the test. This perfectly describes the phenomenon of overfitting.

3.3 Improvements and Overfitting

To avoid the overfitting problem and to have a UV-decomposition that approaches the "real"[2] M, we found in the *Mining of Massive Datasets*[3] book some techniques such that, moving the l first selected components to only half-way from their current value to the optimized one, or using multiple UV decompositions with different values for the latent space/factor k.

[2]By real, we meant an utility matrix where the users have rated every single movie.
[3]http://mmds.org/

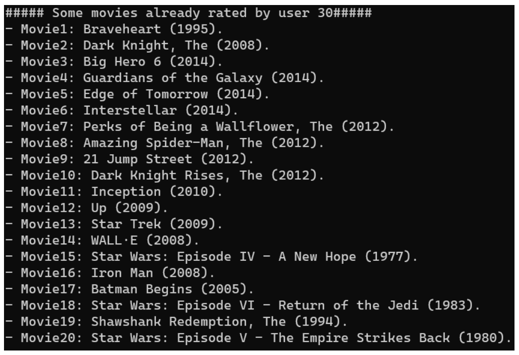
4 Conclusion

4.1 Preview of the recommendations

CF algorithm

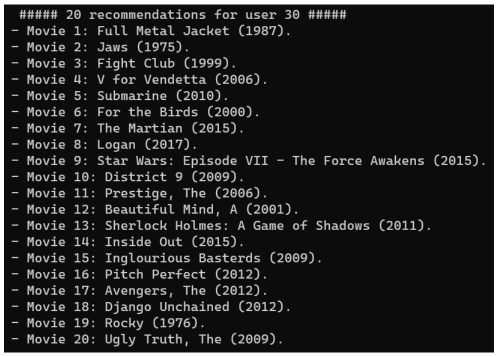
In the figure below we illustrate the recommendations we get when setting the ratings of movies that have no similar seen by a given user to 0 (The A case mentioned earlier).

We tested the user with the $id = 30$. We can see that the movies recommended are pretty similar to the ones liked by the user. For example, He liked Star wars VI, Iron man, Interstellar, ...etc. And the recommended system returned Star wars VII, Avengers, The marsian, ...etc. And this is true, even though the RMSE wasn't really good.



```
##### Some movies already rated by user 30#####
- Movie1: Braveheart (1995).
- Movie2: Dark Knight, The (2008).
- Movie3: Big Hero 6 (2014).
- Movie4: Guardians of the Galaxy (2014).
- Movie5: Edge of Tomorrow (2014).
- Movie6: Interstellar (2014).
- Movie7: Perks of Being a Wallflower, The (2012).
- Movie8: Amazing Spider-Man, The (2012).
- Movie9: 21 Jump Street (2012).
- Movie10: Dark Knight Rises, The (2012).
- Movie11: Inception (2010).
- Movie12: Up (2009).
- Movie13: Star Trek (2009).
- Movie14: WALL·E (2008).
- Movie15: Star Wars: Episode IV - A New Hope (1977).
- Movie16: Iron Man (2008).
- Movie17: Batman Begins (2005).
- Movie18: Star Wars: Episode VI - Return of the Jedi (1983).
- Movie19: Shawshank Redemption, The (1994).
- Movie20: Star Wars: Episode V - The Empire Strikes Back (1980).
```

(a) Movies well rated by the user 30.



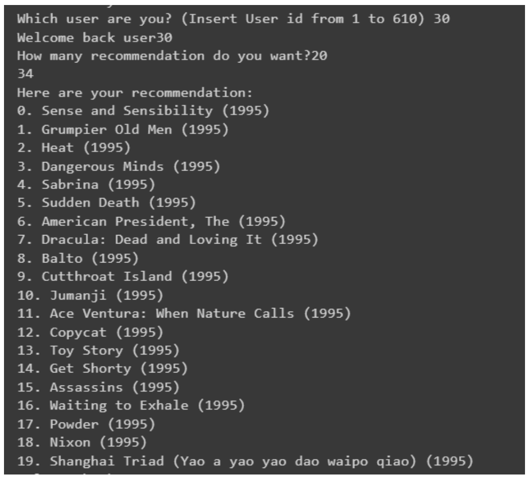
```
 ##### 20 recommendations for user 30 #####
- Movie 1: Full Metal Jacket (1987).
- Movie 2: Jaws (1975).
- Movie 3: Fight Club (1999).
- Movie 4: V for Vendetta (2006).
- Movie 5: Submarine (2010).
- Movie 6: For the Birds (2000).
- Movie 7: The Martian (2015).
- Movie 8: Logan (2017).
- Movie 9: Star Wars: Episode VII - The Force Awakens (2015).
- Movie 10: District 9 (2009).
- Movie 11: Prestige, The (2006).
- Movie 12: Beautiful Mind, A (2001).
- Movie 13: Sherlock Holmes: A Game of Shadows (2011).
- Movie 14: Inside Out (2015).
- Movie 15: Inglourious Basterds (2009).
- Movie 16: Pitch Perfect (2012).
- Movie 17: Avengers, The (2012).
- Movie 18: Django Unchained (2012).
- Movie 19: Rocky (1976).
- Movie 20: Ugly Truth, The (2009).
```

(b) Movies recommended for the user 30.

Figure 3: User 30

UV-decomposition

Regarding the UV-decomposition, for the same user (30), here are the top 20 recommendations given by the Configuration B with the lowest RMSE (see Figure ref(4)).

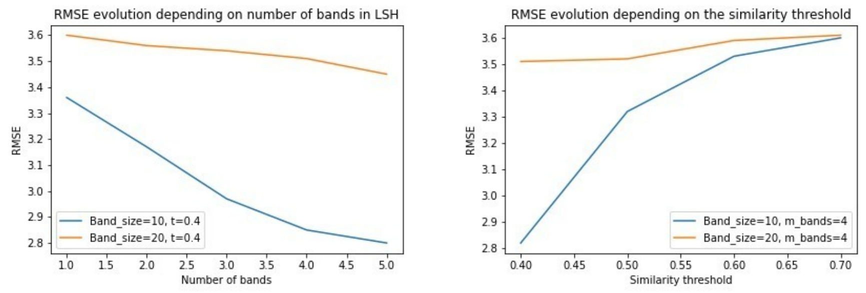


```
Which user are you? (Insert User id from 1 to 610) 30
Welcome back user30
How many recommendation do you want?20
34
Here are your recommendation:
0. Sense and Sensibility (1995)
1. Grumpier Old Men (1995)
2. Heat (1995)
3. Dangerous Minds (1995)
4. Sabrina (1995)
5. Sudden Death (1995)
6. American President, The (1995)
7. Dracula: Dead and Loving It (1995)
8. Balto (1995)
9. Cutthroat Island (1995)
10. Jumanji (1995)
11. Ace Ventura: When Nature Calls (1995)
12. Copycat (1995)
13. Toy Story (1995)
14. Get Shorty (1995)
15. Assassins (1995)
16. Waiting to Exhale (1995)
17. Powder (1995)
18. Nixon (1995)
19. Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
```

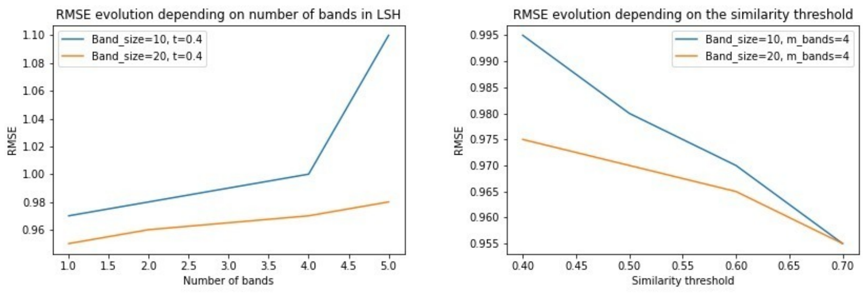
Figure 4: Top 20 recommendation for user 30

A Appendix

A.1 Results of CF



(a) A: Setting ratings of movies having no similar movies to 0.



(b) B: Setting ratings of movies having no similar movies to $\frac{1}{2}b_{xi}$

Figure 5: Evolution of $RMSE$

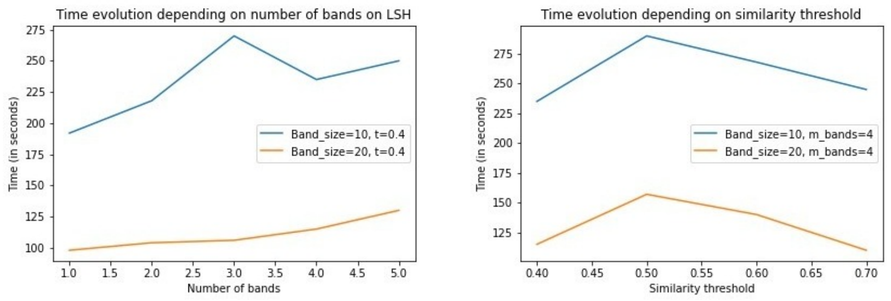
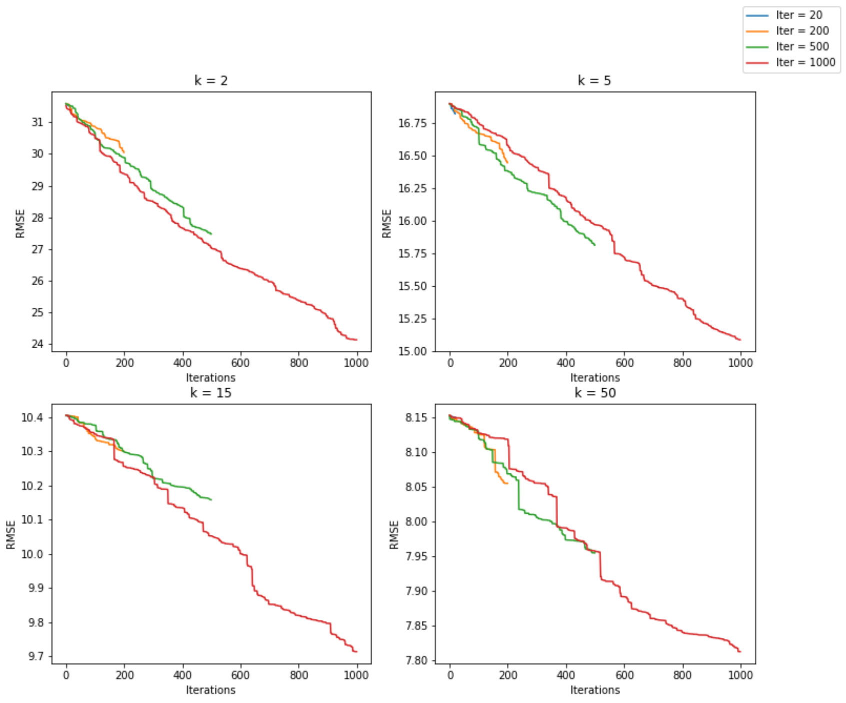
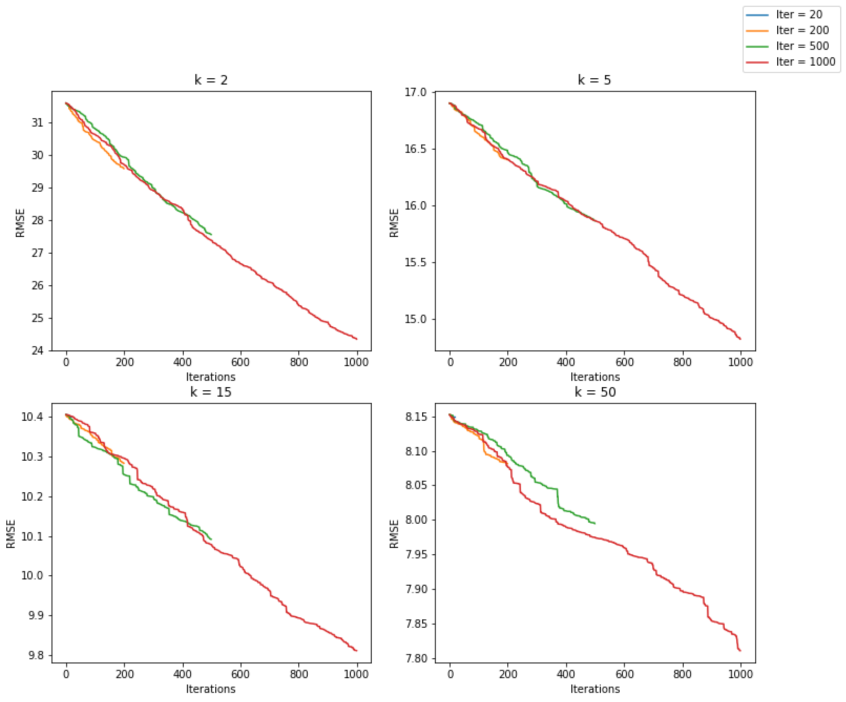


Figure 6: Evolution of running time for both A and B

A.2 Results of UV-decomposition



(a) Configuration A



(b) Configuration B

Figure 7: Training Loss ($RMSE$)