E4571 Personalization - Homework 2

Deepak Ravishankar (dr2998), Yu Han Huang (yh3093)

Objective

In this project, we used the Dating Agency data set which contains 17,359,346 anonymous ratings of 168,791 profiles made by 135,359 LibimSeTi users as dumped on April 4, 2006. We built two brute-force collaborative filtering algorithms to recommend items, which in our case is user profiles, to users. As a dating agency, we wanted to learn how these algorithms can effectively recommend our users to people that they might be interested in meeting. We try to optimize our recommendations so that our users may be more likely to like the profiles they are recommended to. Since we have both a lot of users and a lot of items, we are also interested in knowing how well these algorithms scale with the data size.

Data Exploration

We wrote a script to understand our users' preference by the profiles they rated. We define their preference as follow:

Female

- Straight: Female users that have only rated male users and never rated a female user
- Bi: Female users that have rated both male and female users
- Les: Female users that have only rated female users and never rated a male user

Male

- Straight: Male users that have only rated female users and never rated a male user
- Bi: Male users that have rated both male and female users
- Gay: Male users that have only rated male users and never rated a female user

Result

	STRAIGHT	BI	LES/GAY
FEMALE	0	59258	0
MALE	0	58498	0

We discovered that the website LibimSeTi is more of a friend-making website than dating website, since all of our users have rated both users of the same and opposite sex. This led to our business decision to recommend a more general profile to our users without considering the user's and the profile's gender.

First Algorithm - neighborhood-based collaborative filtering

We built both a user-based and an item-based collaborative filtering algorithm to understand which of the approach works better with our data. Instead of using libraries such as Surprise for training, we built and implemented our models with the formulas of neighborhood-based collaborative filtering algorithms, which will be later explained in description.

Objective

Our objective is to understand which approach, user-based or item-based, works better with our data and is able to give the most accurate predictions. In our cases, the approaches are defined as:

- User-based:
  Our system will recommend users profiles that he/she might want to make friends with based on similar users that have liked the same thing as him/her.
- Item-based:
  Our system will recommend users profiles that he/she might want to make friends with based on what he/she has liked and profiles that are similar to the profiles users have liked.

Cross-validation setup

We use the same cross validation set up as in this paper that we divide our data into a training set and a test set, where 80% of the data was used as our training set and the rest 20% as test set. Since different users have rated different numbers of profiles, our methodology is that we randomly selected 20% of the profiles each users have rated in the user-based approach, and 20% of the users that have rated the profile in the item-based approach. We also asserted in our code that the training set and the test set are disjoint.

In [5]:
```
#split data into training and testing set
def train_test_split(data):
    test = np.empty(data.shape)
    test[:] = np.nan
    train = data.values.copy()
    data_mat = data.values.copy()

    for user in range(data.shape[0]):
        nonnaindex = np.argwhere(~np.isnan(data_mat[user,:])).T[0,:]
        nonnaindex_len = int(len(nonnaindex)/5)
        if nonnaindex_len>0:
            test_ratings = np.random.choice(nonnaindex,size=nonnaindex_len,replace=False)
            train[user, test_ratings] = np.nan
            test[user, test_ratings] = data_mat[user, test_ratings]

    # Test and training need to be disjoint
    assert(np.isnan(train*test).sum()==data.shape[0]*data.shape[1])
    return train, test
```

ACCURACY - PRIMARY ACCURACY METRIC AND A SECONDARY ACCURACY METRIC

In our evaluation, we choose Mean Squared Error (MSE) as our primary accuracy metric and Mean Average Error (MAE) as our secondary accuracy metric, where MSE penalizes outliers more heavily and MAE doesn't overweight outliers. Both of the calculation are implemented with functions in package sklearn.

```
In [10]: #calculate mean squared error and mean absolute error
from sklearn.metrics import mean_squared_error, mean_absolute_error
def get_mse(pred, actual):
    # Ignore nan terms.
    pred_nr = pred[np.where(~np.isnan(actual))]
    actual_nr = actual[np.where(~np.isnan(actual))]

    if len(np.where(np.isnan(pred_nr))[0])>0:
        pred_nr = pred_nr[np.where(~np.isnan(pred_nr))]
        actual_nr = actual_nr[np.where(~np.isnan(pred_nr))]

    if len(pred_nr)>1 and len(actual_nr)>1:
        mse = mean_squared_error(pred_nr, actual_nr)
    else:
        mse = np.nan

    return mse

def get_mae(pred, actual):
    # Ignore nan terms.
    pred_nr = pred[np.where(~np.isnan(actual))]
    actual_nr = actual[np.where(~np.isnan(actual))]

    if len(np.where(np.isnan(pred_nr))[0])>0:
        pred_nr = pred_nr[np.where(~np.isnan(pred_nr))]
        actual_nr = actual_nr[np.where(~np.isnan(pred_nr))]

    if len(pred_nr)>0 and len(actual_nr)>0:
        mae = mean_absolute_error(pred_nr, actual_nr)
    else:
        mae = np.nan

    return mae
```

COVERAGE ON TRAINING AND TEST DATA

In our case, we set our item coverage to 5 profiles/user since we value quality more than quantity when we recommend potential dates to our users. While we certainly can recommend more options to our users, we configured that our users may not be able to evaluate all of the recommendations if it goes up to a high list. Therefore, our algorithm would recommend 5 profiles to each user based on users with similar tastes or items that they previously have rated.

Recommended list for user 90280

User-based

Data size/Error Matric	100	200	500
MSE	[33639,64438,720,24402,131976]	[114979,126991,33639,720,108318]	[114979,97383,607,108497,20949]
MAE	[33639,64438,720,24402,131976]	[114979,33639,720,126991,64438]	[607,114979,97383,21667,108497]

Item-based

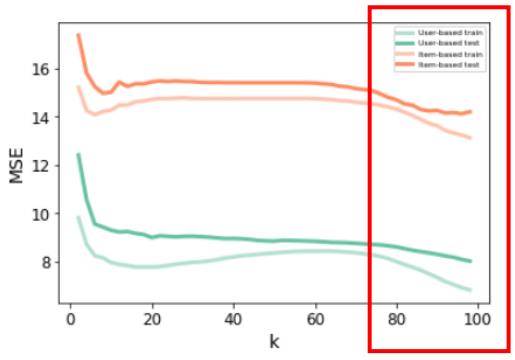
Data size / Error Matric	100	200	500
MSE	[104712,77713,71873,71099,56792]	[108318,78756,6796,34631,77713]	[61587,63343,129753,89474,76153]
MAE	[104712,77713,71873,71099,56792]	[108318,78756,6796,34631,77713]	[108318,76153,96095,36278,89474]

We can see from the result above that our recommendation list are pretty consistent between the two error matrics, where apparently the list in User-based have more outliers.

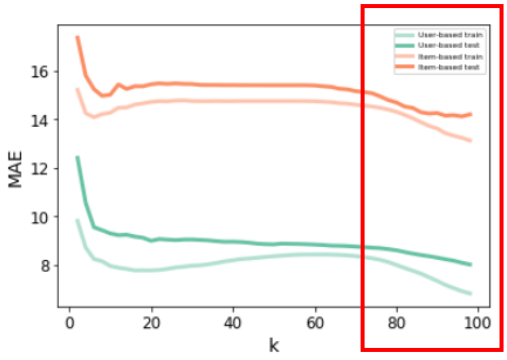
HYPERPARAMETER TUNING – THE DESIGN CHOICE WE CONSIDERED DUE TO OVERFITTING ISSUES

In both of our models, we systematically tried a range of hyperparameters of neighborhood size to see which leads to a lowest training error. The maximum neighborhood size is set to be 50% of the number of users. The reason is that we discovered as we increase the neighborhood size, both user-based and item-based methods will encounter overfitting problems which leads to a low error.

MSE



MAE



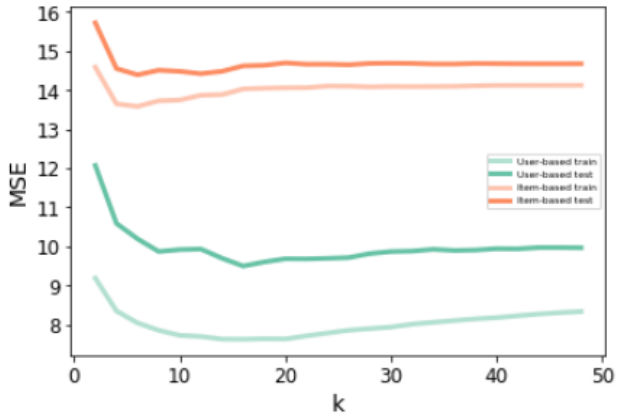
The optimized neighborhood size for different size of subset data is as follow:

100 Users

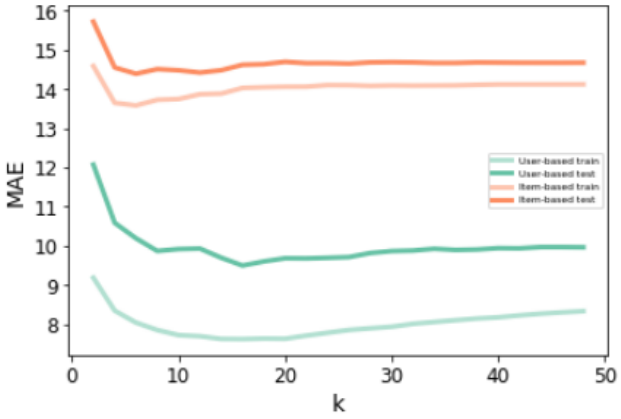
```
In [18]: RecommenderSystem(ratings, 100, 50, 2, 5)

-------Recommender system starts now--------
Sparsity of user training data: 11.33%
===User-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 16 is 7.61589609797
Test data's MSE with the optimal_k 16 is 9.4930108814
---MAE---
Training data's MAE with the optimal_k 16 is 2.26801028182
Test data's MAE with the optimal_k 16 is 2.55716303593
===Item-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 6 is 13.5828854247
Test data's MSE with the optimal_k 6 is 14.3928179159
---MAE---
Training data's MAE with the optimal_k 6 is 3.0112164593
Test data's MAE with the optimal_k 6 is 3.11449140374
```

Color reference: Light Green: User-based train / Dark Green: User-based test / Light Orange: Item-based train / Dark Orange: Item-based test

MSE



MAE



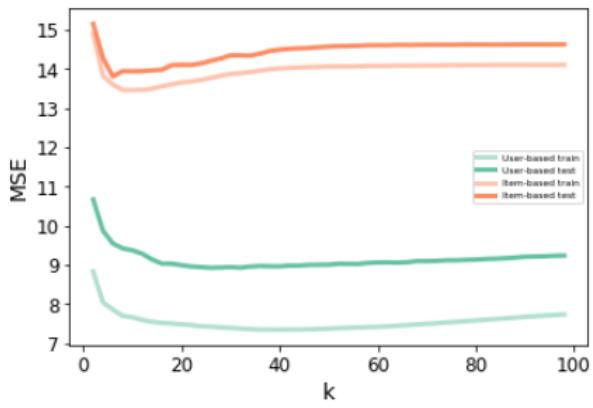
200 Users

```
In [19]: RecommenderSystem(ratings, 200, 100, 2, 5)
```

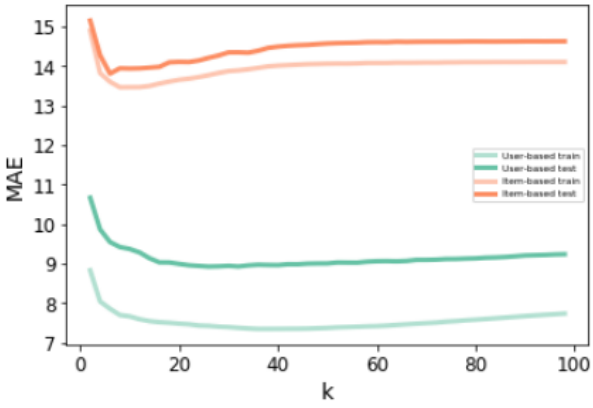
```
-------Recommender system starts now--------
Sparsity of user training data: 9.04%
===User-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 36 is 7.34735021338
Test data's MSE with the optimal_k 36 is 8.97052452917
---MAE---
Training data's MAE with the optimal_k 46 is 2.2184793898
Test data's MAE with the optimal_k 46 is 2.44723849822
===Item-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 8 is 13.457112475
Test data's MSE with the optimal_k 8 is 13.9378639623
---MAE---
Training data's MAE with the optimal_k 8 is 3.00143988564
Test data's MAE with the optimal_k 8 is 3.06591329567
```

Color reference: Light Green: User-based train / Dark Green: User-based test / Light Orange: Item-based train / Dark Orange: Item-based test

MSE



MAE



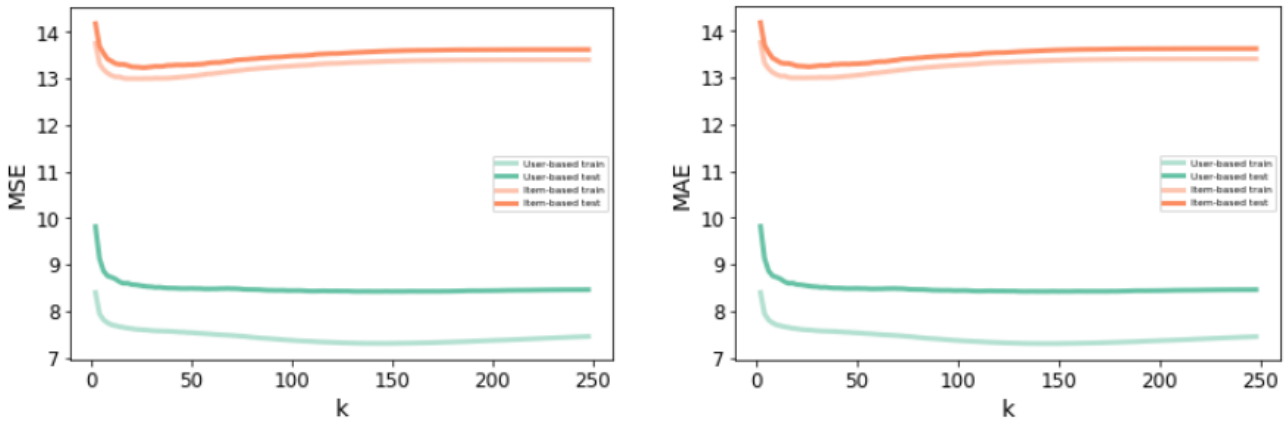
500 Users

```
In [22]: RecommenderSystem(ratings, 500, 250, 2, 5)
```

```
-------Recommender system starts now--------
Sparsity of user training data: 5.82%
===User-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 144 is 7.302049821
Test data's MSE with the optimal_k 144 is 8.41970092842
---MAE---
Training data's MAE with the optimal_k 148 is 2.20891476349
Test data's MAE with the optimal_k 148 is 2.38675703371
===Item-based Collaborative Filtering Model===
---MSE---
Training data's MSE with the optimal_k 24 is 12.9987128736
Test data's MSE with the optimal_k 24 is 13.2437454956
---MAE---
Training data's MAE with the optimal_k 38 is 2.96420876014
Test data's MAE with the optimal_k 38 is 2.98933745822
```

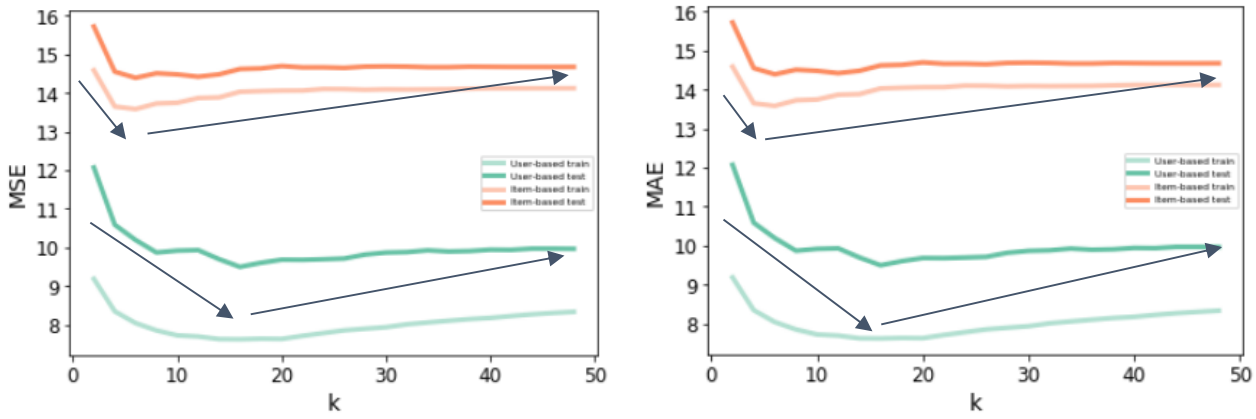
Color reference: Light Green: User-based train / Dark Green: User-based test / Light Orange: Item-based train / Dark Orange: Item-based test

MSE **MAE**



We can discover the same pattern of how evaluation metrics varies with the neighborhood size. The evaluation metrics would first descend, and then at some k, the evaluation metrics would ascend. The k is therefore the optimal hyperparameter we are in search of to get the minimum error.

MSE **MAE**



SCALABILITY

We sampled our user size from 100, 200 to 500, and then we test on the accuracy and run time.

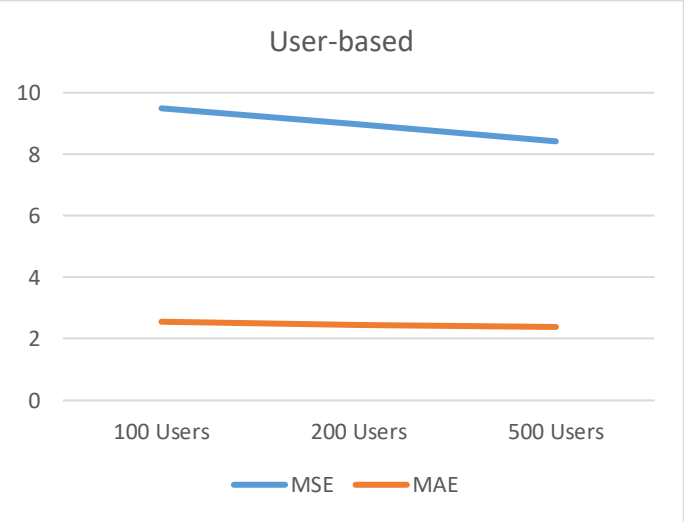
Accuracy

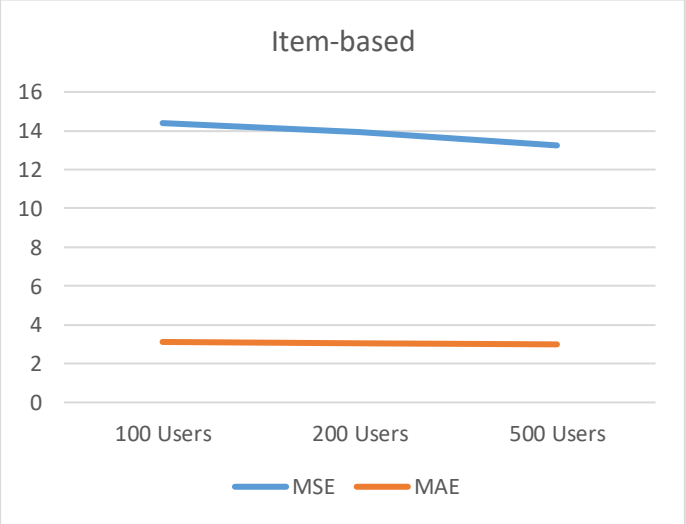
User-based

Data size/Error Matric	100	200	500
MSE	9.4930108814	8.97052452917	8.41970092842
MAE	2.55716303593	2.44723849822	2.38675703371

Item-based

Data size / Error Matric	100	200	500
MSE	14.3928179159	13.9378639623	13.2437454956
MAE	3.11449140374	3.06591329567	2.98933745822





We can clearly observe that in both User-based and Item-based approach, the error decreases as the sample data size increases.

Run-time

Data size	100	200	500
Run time	39s	4m 12s	1hr 3min 40s

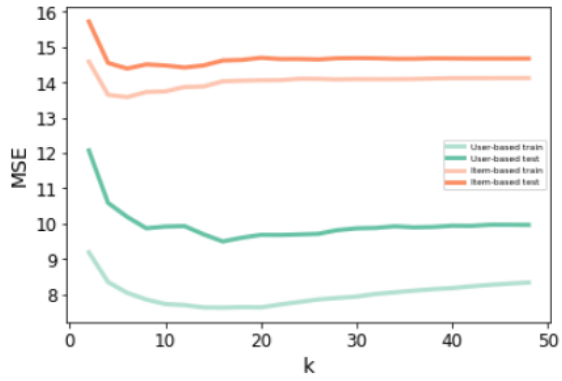
However, the run-time doesn't scale well with data size. When our data size increases from 100 to 500 users, the run time increased by an hour, which is a considerable difference that can affect the business.

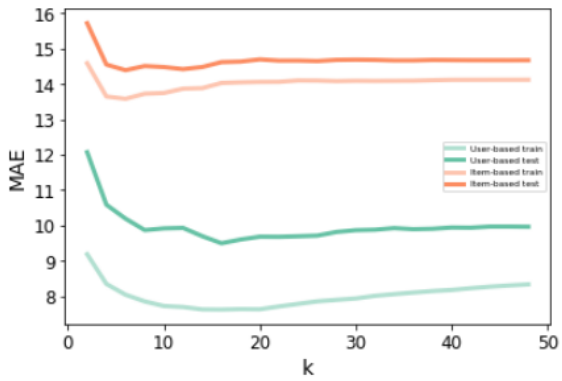
EVALUATION

Our objective was to understand which approach, user-based or item-based, works better with our data and is able to give the most accurate predictions. We evaluate it by seeing which approach gives us the least error.

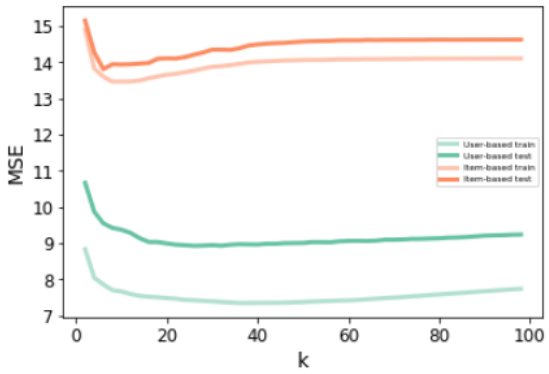
Color reference: Light Green: User-based train / Dark Green: User-based test / Light Orange: Item-based train / Dark Orange: Item-based test

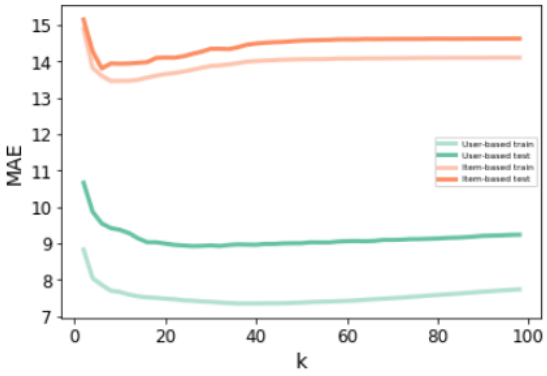
100 Users



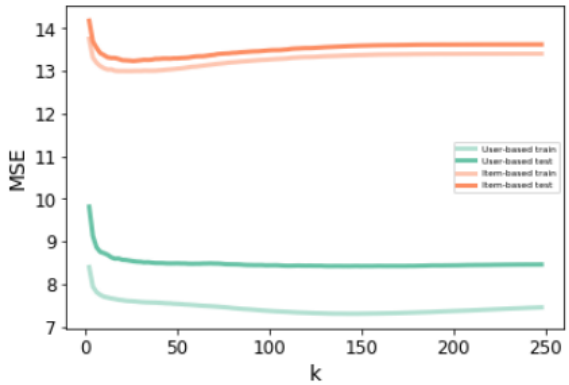


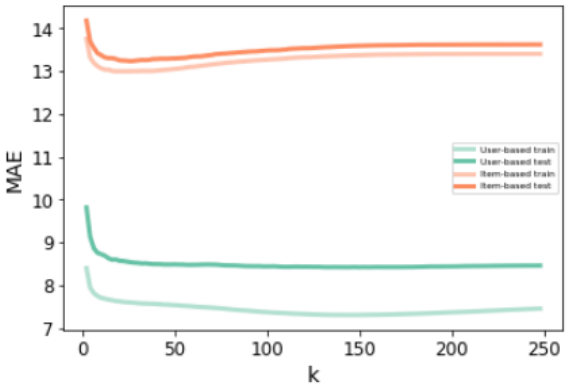
200 Users





500 Users





We can clearly observe that in all data sizes and all evaluation metrics, user-based approach has a much better performance and less error than item-based approach. This matches with our business-intuitive since it is reasonable for people to have a wide variety of tastes in possible dates. Therefore as a dating agency, we should try to recommend users with dates that other people like him/her have gone out with, other than keep on recommending similar profiles (ex: tall blondes) to the user once he/she has liked a user with such features.

CONCLUSION

By our study, we conclude that the neighborhood-based collaborative filtering method is usable in recommending possible dates to our users. We discovered that we will be able to determine the optimal neighborhood (i.e. similar users/items we should consider) by evaluation metrics since it has a pattern to decrease and increase as the neighborhood size increases. We also discover that the overall accuracy would increase as we increase our data size.

However, the runtime of algorithm doesn't scale well with the data size, and evaluating for 500 users already take more than 1 hour. If to deploy in real business scenarios, we may need to sacrifice a little accuracy to divide users into smaller batches in order to have better runtime. This leads us to our study in part two, where we test on another approach - Matrix Factorization to see whether it would outperform our results in part I.

Second Algorithm - Model-based collaborative filtering

Objective:

We are trying to understand how matrix factorization methods perform when used as a recommendation engine for the Dating Agency data set. We use two solving algorithms to factorize the matrices.

1. Alternating least squares(ALS)
2. Stochastic gradient descent(SGD)

Cross-validation setup:

As these two methods have various hyper-parameters that we can tune, we need to develop a cross-validation setup. For our cross-validation setup, we use a 60-20-20 split for the train, cross-validation, and test respectively. This is important as if we choose the hyperparameters using our test set then we will be fitting our test set and the error on the test set will not be a true reflection of the generalization of the algorithm. We also assert in our code that the three sets are disjoint as well. In our notebook the function `train_cv_test_split` splits the data into three equal parts.

Accuracy - a primary and a secondary metric:

We use two accuracy metrics to evaluate the models that we have developed.

1. **Mean squared error:** This is used as the primary error metric as it is useful in finding the optimal latent vectors as it gives a smooth error surface which can be easily optimized over using SGD and ALS. This is implemented in the `get_mse` function in the notebook.
2. **Mean Absolute error:** This was used as a secondary evaluation metric as the error that is returned is more understandable in terms of the range of the data set. This is implemented in the `get_mae` metric in the data

Other error metrics like normalized discounted cumulative gain(nDCG) were not considered as we had ~10 ratings per user after the subsetting and the test split would be very small ~2 for the ordering of the final results to have a huge effect on the results.

Methodology:

Method I (Alternating least squares)

In this method, the Algorithm keeps either the user matrix or the item latent vectors constant at a given step and solves the for the non-constant latent vector. The error metric used to optimize is the mean squared with regularization. The error metric is calculated over a set A which contains all the User u and Item i pairs which have been rated

$$Error = \sum_{u,i \in A} (r_{ui} - {U_u}^T I_i)^2 + \lambda_U \sum_u ||U_u||^2 + \lambda_I \sum_i ||I_i||^2$$

Where,

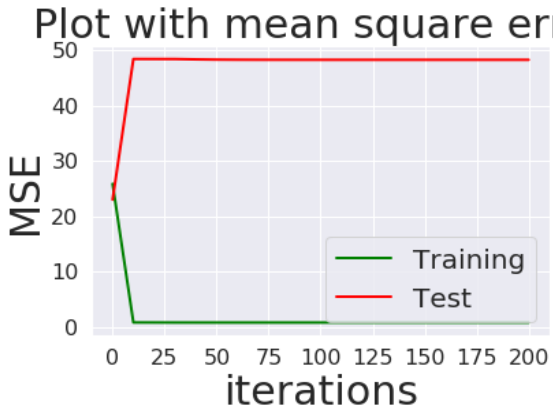
r_{ui} *is the rating for user u and item i*
U_u *is the user latent vector for user u*
I_i *is the item latent vector for item i*
λ_U *is the regularization term user latent vector*
λ_I *is the regularization term for the item latent vector*

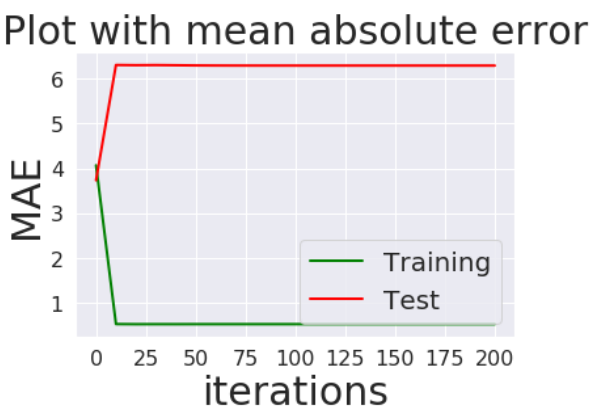
Optimizing the error metric keeping Either User or Item latent vectors constant we get the equation

$$(I^T I + \lambda_U E)U^T = rI$$
$$(U^T U + \lambda_I E)I^T = rU$$

Which are of the form AX=B where X is U^T *and* I^T . We solve the equations using the `numpy.linalg.solve` function. Using which we get the values of U *and* I . The class `MatrixFactorization` has a method `als_train` which implements this method

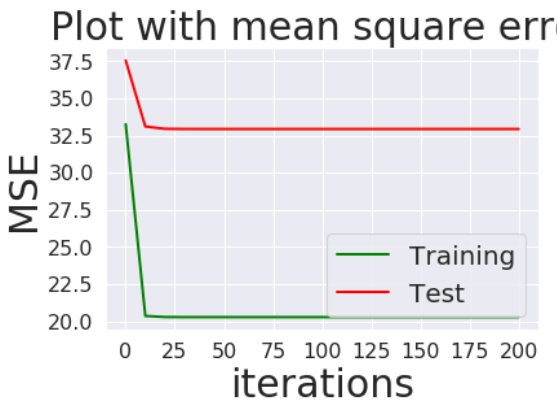
Solving for the user and item latent vector with no regularization we get the following plots for the error metrics which are plotted using the functions `plot_mse_learning_curve` and `plot_mae_learning_curve`

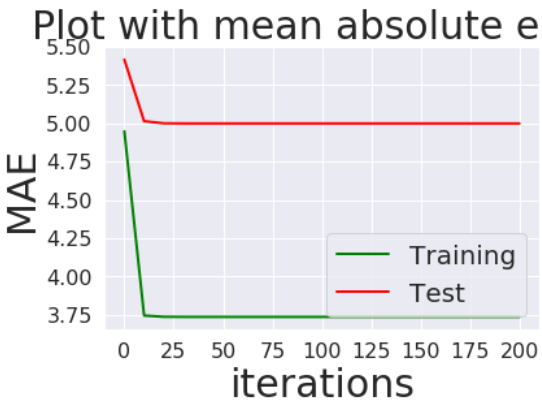




Observation from the plot: We can see that the training accuracy nearly reaches 0 whereas the test accuracy is very high. This is clearly indicative that the data set is being over fit and that we Need have some form of regularization that we should use to improve the generalization in the model.

Now if we run the model again with regularization terms set to 5 for both the user and the item latent vectors we observer the following plots.





Observation from the plot: We can notice that there is a huge improvement as now the algorithm seems to be generalizing well as the test MSE has reduced from 50 to 32 and the test MAE has reduced from 6.5 to 5.

Hyperparameter tuning:

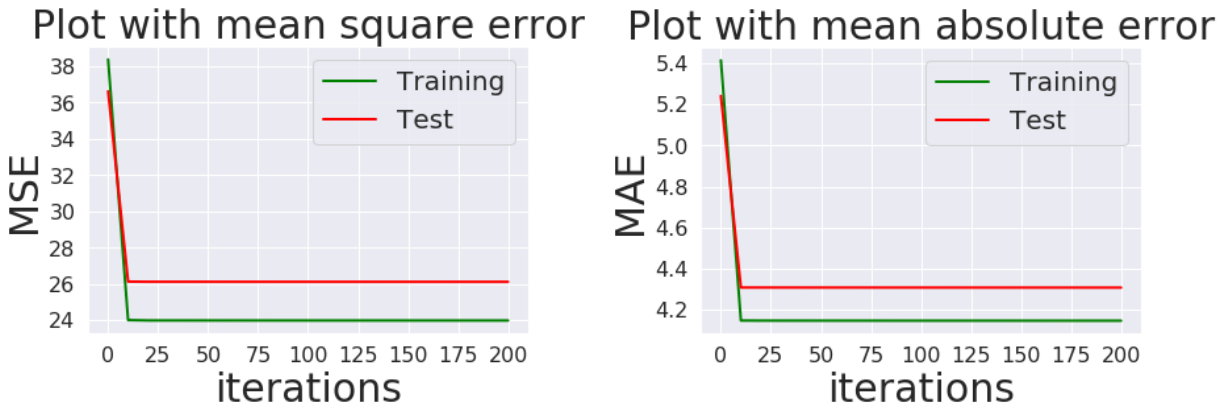
Now that we have established that ALS does need regularization to avoid overfitting we use can do some hyperparameter tuning to obtain the best settings for the model. The hyper-parameters are tuned using the cross-validation set and the best are obtained. The cross-validation is done over the following data.

- Length of latent vectors from [2 ,4 ,5, 10, 20, 40, 80]

- User regularization term from [0.1, 1., 10., 100.]
- Item regularization term from [0.1, 1., 10., 100.]

	User_regularization	item_regularization	latent_factor	test_mae	test_mse
0	0.1	0.1	2	4.334464	26.483381
1	1.0	1.0	2	4.465039	27.437142
2	10.0	10.0	2	4.487100	27.836269
3	100.0	100.0	2	5.743096	41.940746
4	0.1	0.1	4	4.378304	27.005513
5	1.0	1.0	4	4.412795	27.293121
6	10.0	10.0	4	4.521172	28.164408
7	100.0	100.0	4	5.742908	41.938193
8	0.1	0.1	5	4.533237	29.067664
9	1.0	1.0	5	4.429322	27.608967
10	10.0	10.0	5	4.594254	29.087980
11	100.0	100.0	5	5.742768	41.936283
12	0.1	0.1	10	4.771201	31.955191
13	1.0	1.0	10	4.792943	32.295230
14	10.0	10.0	10	4.802973	31.856198
15	100.0	100.0	10	5.743074	41.940453
16	0.1	0.1	20	5.221841	36.862318
17	1.0	1.0	20	5.209443	36.664616
18	10.0	10.0	20	5.088041	34.785455
19	100.0	100.0	20	5.743473	41.945874
20	0.1	0.1	40	5.638063	41.079155
21	1.0	1.0	40	5.603071	40.614186
22	10.0	10.0	40	5.291345	36.781188
23	100.0	100.0	40	5.743827	41.950690
24	0.1	0.1	80	5.740350	41.911973
25	1.0	1.0	80	5.694305	41.384500
26	10.0	10.0	80	5.298304	36.844923
27	100.0	100.0	80	5.743774	41.949971

Running the tuning on the cross-validation set we get the best hyperparameters to be 2,0.1,0.1 for latent vectors, user regularization, and item regularization respectively. Training the best hyper-parameters on the train and cross-validation set we get the following plots for error.



Observation from the plot: We can see that the error has clearly reduced and it is one of the lowest error that we have received yet both from the MSE and MAE. With a MSE of 26 and MAE of 4.3 for the test set, the model seems to be performing well.

Method II (Stochastic gradient descent)

In this method, the algorithm finds the best user and item latent vector removing the error gradient from the user and item latent vectors and multiplying it by a learning rate and then updating it till it reaches minima where the error is low. In this, the values of the error metrics are updated using the algorithm

$$U_u = U_u + \sum_{u,i \in A} \alpha \left((r_{ui} - {U_u}^T I_i) * I_i - \lambda_U * U_u\right)$$

$$I_i = I_i + \sum_{u,i \in A} \alpha \left((r_{ui} - {U_u}^T I_i) * U_u - \lambda_I * I_I\right)$$

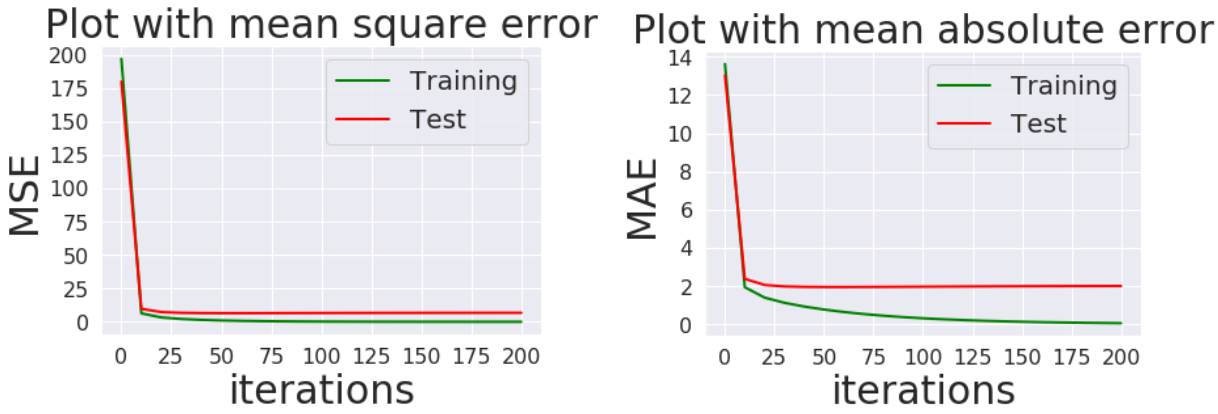
Where,

r_{ui} *is the rating for user u and item i*

U_u *is the user latent vector for user u*

I_i *is the item latent vector for item i*
λ_U *is the regularization term user latent vector*
λ_I *is the regularization term for the item latent vector*
α *is the learning rate of the algorithm*

Running the Algorithm we get the following results.



Observations from the plot: We can see that the SGD algorithm seem to perform better on the data even without regularization compared the ALS. Also that the MAE is around 10 when the algorithm reaches convergence which is much lower than the 26 reached by the ALS algorithm after tuning.

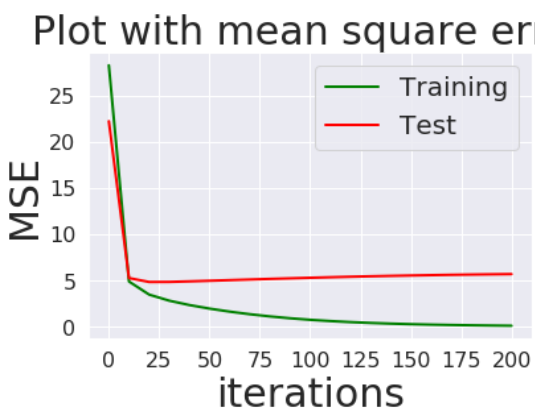
Hyperparameter tuning: Next we tune the hyperparameter of the algorithm to improve the accuracy even further. The hyperparameters that we check for are:

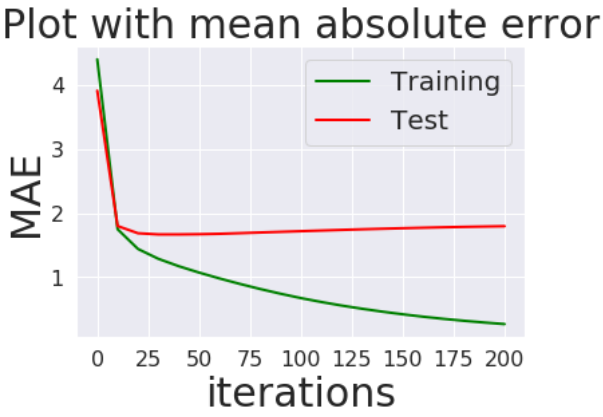
- Learning rate from [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]
- Latent of vectors length from [5, 10, 20, 40, 80]
- User regularizations from [0.001, 0.01, 0.1, 1.]
- Item regularization from [0.001, 0.01, 0.1, 1.]

	latent_factor	User_regularization	item_regularization	alpha	test_mse	test_mae
0	5	0.001	0.001	0.001	5.629446	1.774052
1	5	0.010	0.010	0.001	5.393264	1.704704
2	5	0.100	0.100	0.001	5.550021	1.756418
3	5	1.000	1.000	0.001	6.152843	1.854561
4	10	0.001	0.001	0.001	5.409097	1.741107
5	10	0.010	0.010	0.001	5.457148	1.791856
6	10	0.100	0.100	0.001	5.352246	1.756356
7	10	1.000	1.000	0.001	6.125220	1.860785
8	20	0.001	0.001	0.001	5.250780	1.825336
9	20	0.010	0.010	0.001	5.207429	1.770462
10	20	0.100	0.100	0.001	5.002533	1.769608
11	20	1.000	1.000	0.001	6.105917	1.892287
12	40	0.001	0.001	0.001	5.180908	1.799184
13	40	0.010	0.010	0.001	5.394756	1.776063
14	40	0.100	0.100	0.001	5.247426	1.790902
15	40	1.000	1.000	0.001	5.716833	1.870451
16	80	0.001	0.001	0.001	7.879685	2.178551
17	80	0.010	0.010	0.001	7.941349	2.217210
18	80	0.100	0.100	0.001	6.260629	1.959960
19	80	1.000	1.000	0.001	6.095501	1.870588

This tuning is done on the cross-validation set where we obtain the best hyperparameters which are 0.001 for learning rate, 0.01 for user and item regularization and 40 as the length of the latent vectors.

Running sgd with the best hyperparameter and plotting against error we get the following results.





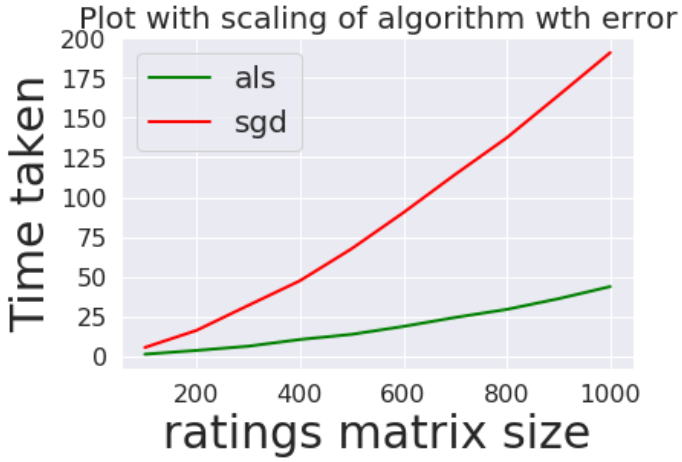
Observations from plot: We get some of the lowest errors on the tuned SGD algorithm with an MSE of 5 and MAE of ~2 on the test data, which is a huge improvement over the previus method of ALS

Coverage on training and test data

Similar to the previous case we set our item coverage to 5 profiles/user since we want to have better quality then quantity. As our test error is really low with the matrix factorization with SGD, we are confident that the user will find someone he likes among the top 5 recommended users per day.

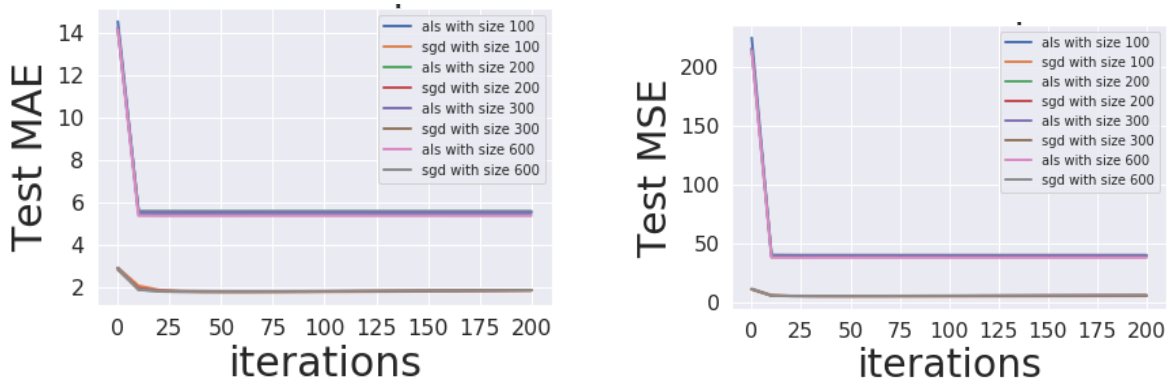
Scaling with data:

Too see how the algorithm scales with respect to the data we run the algorithm from with ratings matrix of size 100, 200, 300 and 600 to see the total amount of time that it takes as well as the change in error. Ploting the time taken for the two algorithms we get the following plot.



Observartions from plot: From this plot we can see that the sgd algorithms scales much worse compared to the als algorithm though both of them seem to be scaling nearly linearly with the size of the ratings matrix

Scaling of error with size: Plotting the interaction of error with scaling with data we get the following plot.



Observations from plot: The plot shows that with the increase in the the rating's matrix size the error metrics are not effected at all after being tuned on the hyperparameters and thus the error metrics scale well with an increase in the size of the ratings matrix

Conclusions with the bussiness usecase:

In the terms of the dataing company we can use the SGD algorithm as it scales much worse than the ALS algorithm but it is still linear and offers much high accuracy. We could run the users best matches in a batch , as it is really important to be giving useful recommendation to the user in order to retain him with the business. Also to retain even though all the data we have had bi-sexual users we can still just give the top rated males and females based on the users sexual orientaion so that he gets the best recommendations

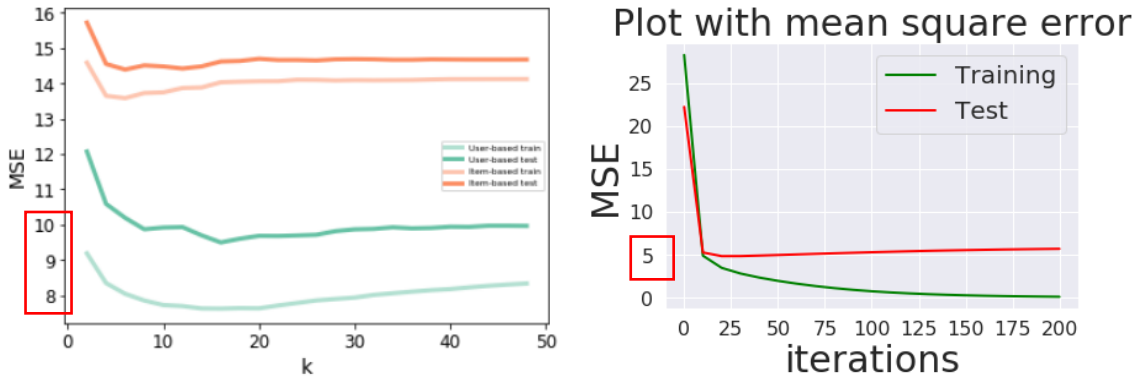
CONCLUSION

In this project, we used the Dating Agency data set which contains 17,359,346 anonymous ratings of 168,791 profiles made by 135,359 LibimSeTi users as dumped on April 4, 2006. We built two brute-force collaborative filtering algorithms to recommend items, which in our case is user profiles, to users, and two solving algorithms to factorize the matrices, which are Alternating least squares(ALS) and Stochastic gradient descent(SGD).

Our business objective was to find out which approach can recommend profiles that matches the preference of our users the most, which in other terms is the approach that is able to generate the least error when predicting the profiles that the user has already rated. We used two different error matrics in our studies, Mean Squared Error (MSE) and Mean Average Error (MAE). We also highly valued the runtime scability of the user size since we have a large user base in the business case.

We discovered in the first part that we will be able to determine the optimal neighborhood (i.e. similar users/items we should consider) by evaluation metrics since it has a pattern to decrease and increase as the neighborhood size increases. However, the runtime of algorithm didn't scale well with the data size, and evaluating for 500 users already took more than 1 hour, which makes the algorithm less applicable to real business scenarios. These issues are resolved in part two as we saw great improvements in decreasing errors and runtime using matrix factorization. We then further improved the matrix factorization with different solving algorithms. We concluded that in the terms of the dataing company, we can use the SGD algorithm as it is scaling much worse than the ALS algorithm but it is still linear.

100 Users evaluation metric in user-based and in stochastic gradient descent matrix factorization

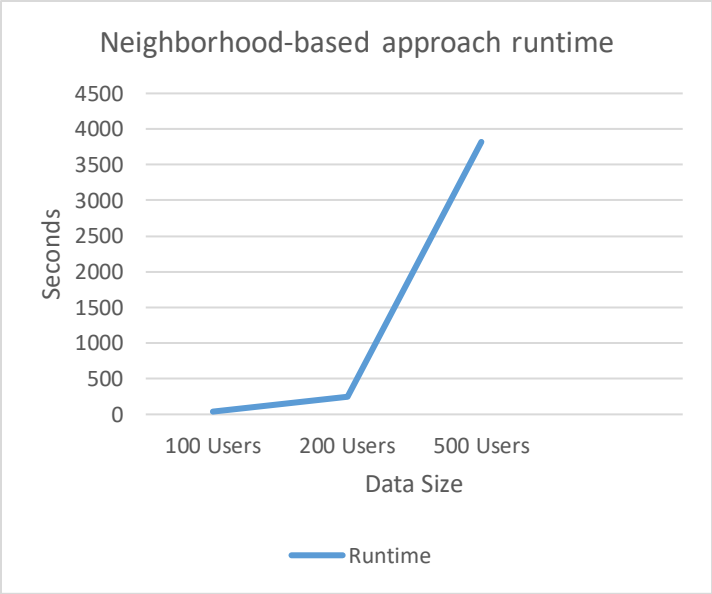


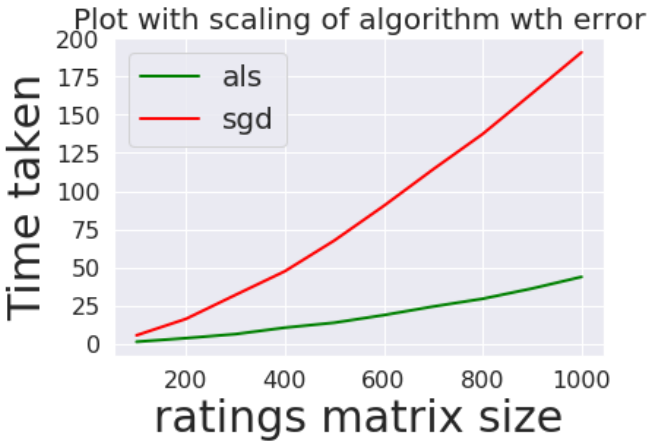
We can see that the minimum error we have in neighborhood-based approach is around 8, while it is around 5 in matrix-factorization approach.

100 Users runtime scalability in user-based and in stochastic gradient descent matrix factorization

User-based

Data size	100	200	500
Run time	39s	4m 12s	1hr 3min 40s





We can see that the maximum runtime in matrix-factorization approach with 1000 users is around 200 seconds, while in in neighborhood-based approach, the runtime goes up to 4000 seconds with just 500 users.

Therefore, in considering which approach we should use in our business case, we decided to use stochastic gradient descent matrix factorization algorithm which gives the most accurate results and with great runtime scalability.