Introduction

Recommender systems have emerged as one of the most successful tools in tackling the issue of information overload. A recommender system's objective is to predict users' preferences and recommend relevant products or services to a user by the user's characteristics or product. Recommender systems are used widely in most e-commerce platforms and online local-search services to help customers make the right decisions and save time.

Collaborative Filtering

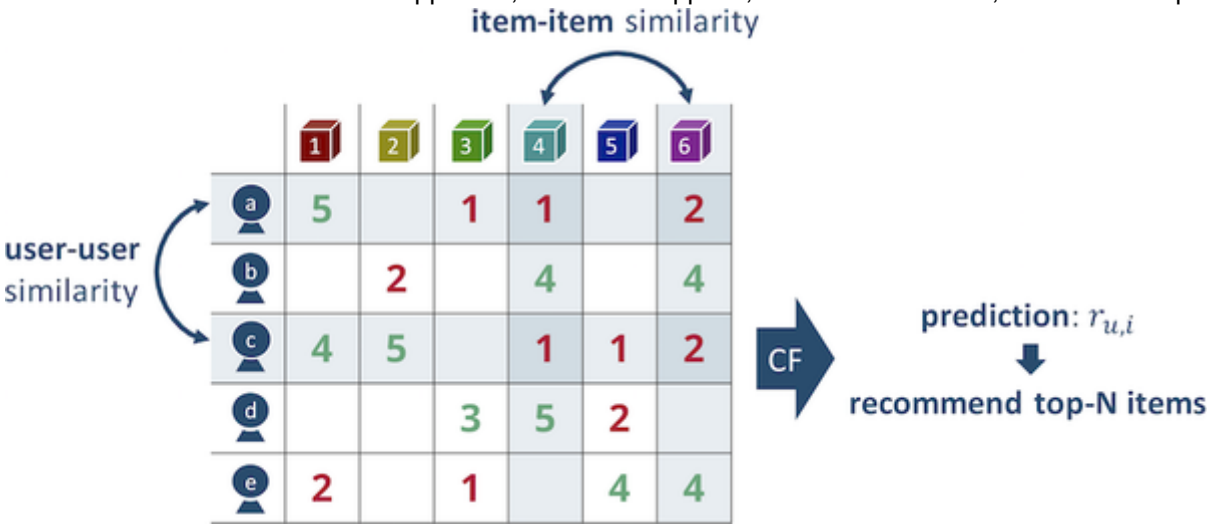
The most basic models for recommender systems are collaborative filtering (CF) models. CF models are based on the assumption that people like items similar to the items they have liked/ consumed in the past and items that other people with a similar taste like/ consume. The CF algorithms can be classified into two:

- Memory-based: which identifies the K most similar neighbors using Pearson/ Cosine similarity to a target user and then uses a weighted sum of ratings of the identified neighbors to predict the unknown ratings for the target user
- Model-based: uses the rating matrix to build a model using some data mining or machine learning algorithms to estimate the missing ratings

Memory Based CF

Memory Based CF can be further be subdivided into two:

- Item-Based CF: In an item-based approach, an item is compared to other items. The more similar the interactions of customers between these two items are, the more they fit together.
- User-Based CF: With the user-based approach, the same happens, but instead of items, users are compared.



Model-Based CF

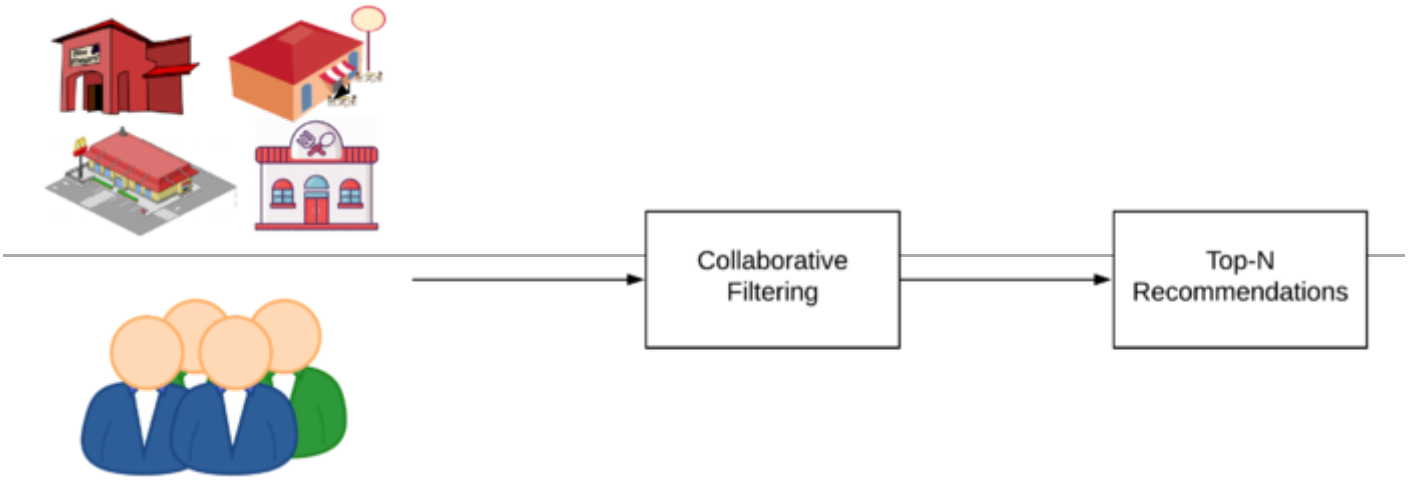
Model-Based CF approaches include matrix factorization techniques like Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and Probabilistic Matrix Factorization (PMF). In matrix factorization techniques, the standard input is user-item interaction that can be stored in a matrix with each row representing each user. Each column dedicates to each item. Since not every user is going to rate each item, the user-item interaction matrix will be sparse. Therefore, the goal of the matrix factorization method is to predict missing entries.

To predict missing entries, the primary matrix is decomposed into the product of two lower dimensional matrices. The first matrix has the same number of rows as the primary matrix, and each row belongs to each person, while the second matrix has a column for each item. Each of the two matrices expresses the power of association between a user or an item with latent factors or hidden features.

The number of latent factors can be varied. As the number of chosen latent factors increases, more hidden factors have been extracted from the data, and therefore, the recommender system's quality improves. However, too much increase in the number of latent factors causes over-fitting over the data and reduces recommendation quality.

Recommender Systems General Framework

Below is the overview for the Recommender System framework:



Experimental Settings

To evaluate the performance of the different approaches, we use different error, accuracy, and ranking metrics.

Error Metrics

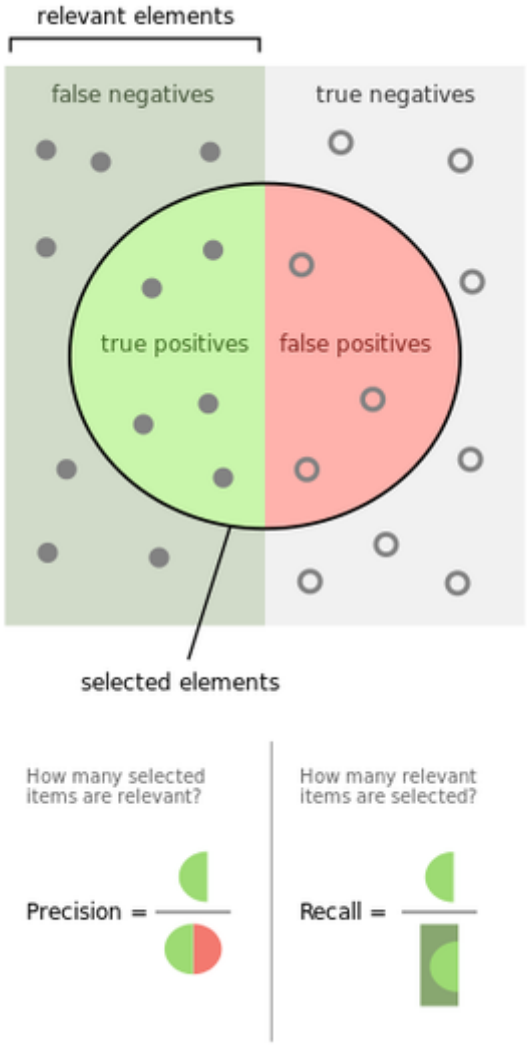
To measure the error in our predicted ratings, we use Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE):

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Accuracy Metrics

We use accuracy metrics like Precision, Recall, and F1, to evaluate Top-N recommendations' performance[1]. True Positive (TP) means a relevant product is recommended to a customer by the recommender system, while False Positive (FP) denotes the case when an irrelevant item is recommended. On the contrary, when an item of customer's liking has not been recommended, then we term the case as False Negative (FN). Precision and Recall can be easily measured with the confusion matrix's help presented below. The F1 score uses harmonic mean to combine Precision and Recall:



Ranking Metrics

Rank metrics extend recall and precision to take the positions of correct items in a ranked list into account:

- Relevant items are more useful when they appear earlier in the recommendation list.
- Particularly important in recommender systems as lower-ranked items may be overlooked by users.

The idea behind NDCG is: A recommender returns some items, and we'd like to compute how good the list is. Each item has a relevance score, usually a non-negative number. When evaluating rankings, we'd prefer to see the most relevant items at the top of the list, i.e., the first result in our search results is more important than the second, the

second more important than the third, and so on. A straightforward way to make position one more important than two (and so on) is to divide each score by the rank. Though in practice, it's more common to discount it using a logarithm of the item position, giving us:

$$DCG_k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$

It's not fair to compare DCG values across queries because some queries are easier than others, or result lists vary in length depending on the final predictions, so we normalize them. First, figure out what the best-ranking score is for this result and compute DCG for that, then we divide the raw DCG by this ideal DCG to get NDCG@K, a number between 0 and 1:

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

Cross-Validation and Grid Search

To ensure the reliability of measurements, the random split, model building, and evaluation steps are repeated several times. K-fold cross-validation is a stratified random selection procedure.

Our experiments are performed by partitioning the dataset into five disjoint sets for 5-fold cross-validation. One set is used as the test set and the other four as the training set, which results in five disjoint training/testing sets. Experiments are conducted on each set, and then the average results are reported. We also perform Grid Search for each Recommender System algorithm to choose the best set of optimal hyperparameters.

Dataset Overview

- #Ratings: 208222
- #Users 5594
- #Restaurants 2801
- Sparsity: 98.67%

Comparison

Baseline Approach

We use the Surprise package in Python to implement the different recommender system algorithms. To compare the various metrics, we use the NormalPredictor as the baseline approach. Below is an overview of the NormalPredictor approach:

The prediction $\hat{r}_{ui}$ is generated from a normal distribution $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ where $\hat{\mu}$ and $\hat{\sigma}$ are estimated from the training data using Maximum Likelihood Estimation:

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}}$$

KNN Basic

- A basic memory-based collaborative filtering algorithm:
- Parameters:
  - k (int) – The (max) number of neighbors to take into account for aggregation.
  - min_k (int) – The minimum number of neighbors to take into account for aggregation.
  - name: The name of the similarity to use, Pearson or Cosine similarity.
  - user_based: Whether similarities will be computed between users or between items. This has a significant impact on the performance of a prediction algorithm.
- Cross-Validation Results - Best Model:
  - User-Based:
    - k=60
    - min_k=20
    - name: "cosine"
    - user_based: True
  - Item-Based:
    - k=60
    - min_k=20
    - name: "pearson"
    - user_based: False

KNN Baseline

- A basic collaborative filtering algorithm that takes into account the baseline rating.

The baseline ratings are taken into account while predicting ratings:

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

or

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j)}$$

- Parameters:
  - k (int) – The (max) number of neighbors to take into account for aggregation.
  - min_k (int) – The minimum number of neighbors to take into account for aggregation.
  - name: The name of the similarity to use, Pearson or Cosine similarity.
  - user_based: Whether similarities will be computed between users or between items. This has a significant impact on the performance of a prediction algorithm.
- Cross-Validation Results - Best Model:
  - User-Based:
    - k=40
    - min_k=20

        - name: "cosine"
        - user_based: True
    - Item-Based:
        - k=60
        - min_k=20
        - name: "pearson"
        - user_based: False

Singular Value Decomposition

- The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize.
- Parameters:
    - n_factors – The number of factors.
    - n_epochs – The number of iteration of the SGD procedure.
- Cross-Validation Results - Best Model:
    - n_factors=30
    - n_epochs=20

Non-Negative Matrix Factorization

- A collaborative filtering algorithm based on Non-negative Matrix Factorization.
- This algorithm is very similar to SVD, where user and item factors are kept positive.
- Parameters:
    - n_factors – The number of factors.
    - n_epochs – The number of iteration of the SGD procedure.
    - Cross-Validation Results- Best Model:
    - n_factors=40
    - n_epochs=30

Results

Sample Prediction

The following are the ratings for reviwer_ID ="woOivdKTZdNrYOomUGAjQQ":

TRAIN SET

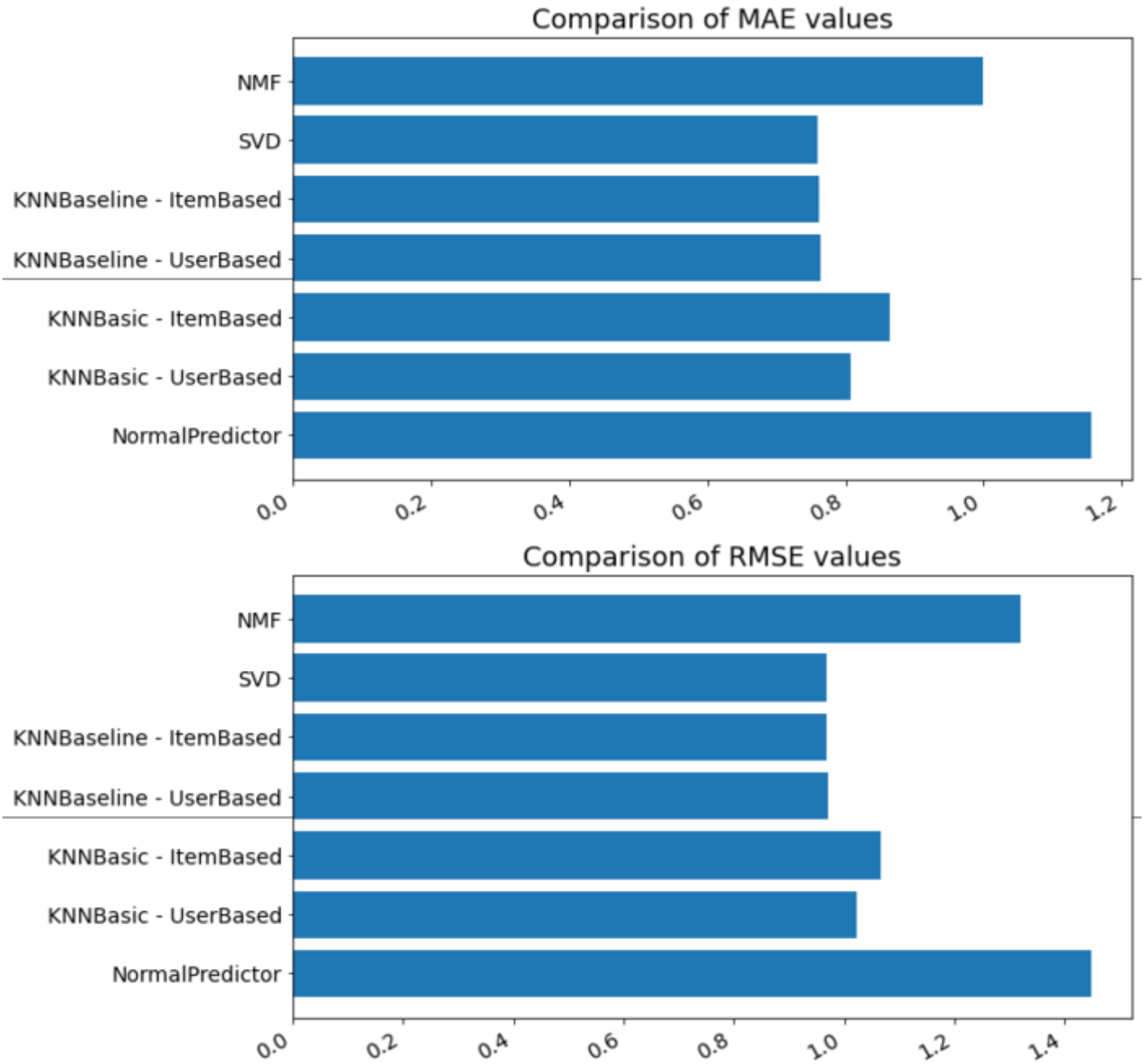
Restaurant Name	Rating Given
prezza-boston-2	5
west-side-lounge-cambridge	5
back-bay-social-boston	4
Hennessys-boston-2	4
Sweet-cheeks-q-boston	4
sunset-grill-and-tap-allston	3
papa-johns-pizza-boston-2	2
chipotle-mexican-grill-cambridge-8	2
mcdonalds-boston-17	2

The following are the predictions generated by the model:

Restaurant Name	Given Rating	Predicted Rating
tasty-burger-boston	5	5
raising-canes-chicken-fingers-boston-2	5	4.45
redbones-bbq-somerville	4	4.32

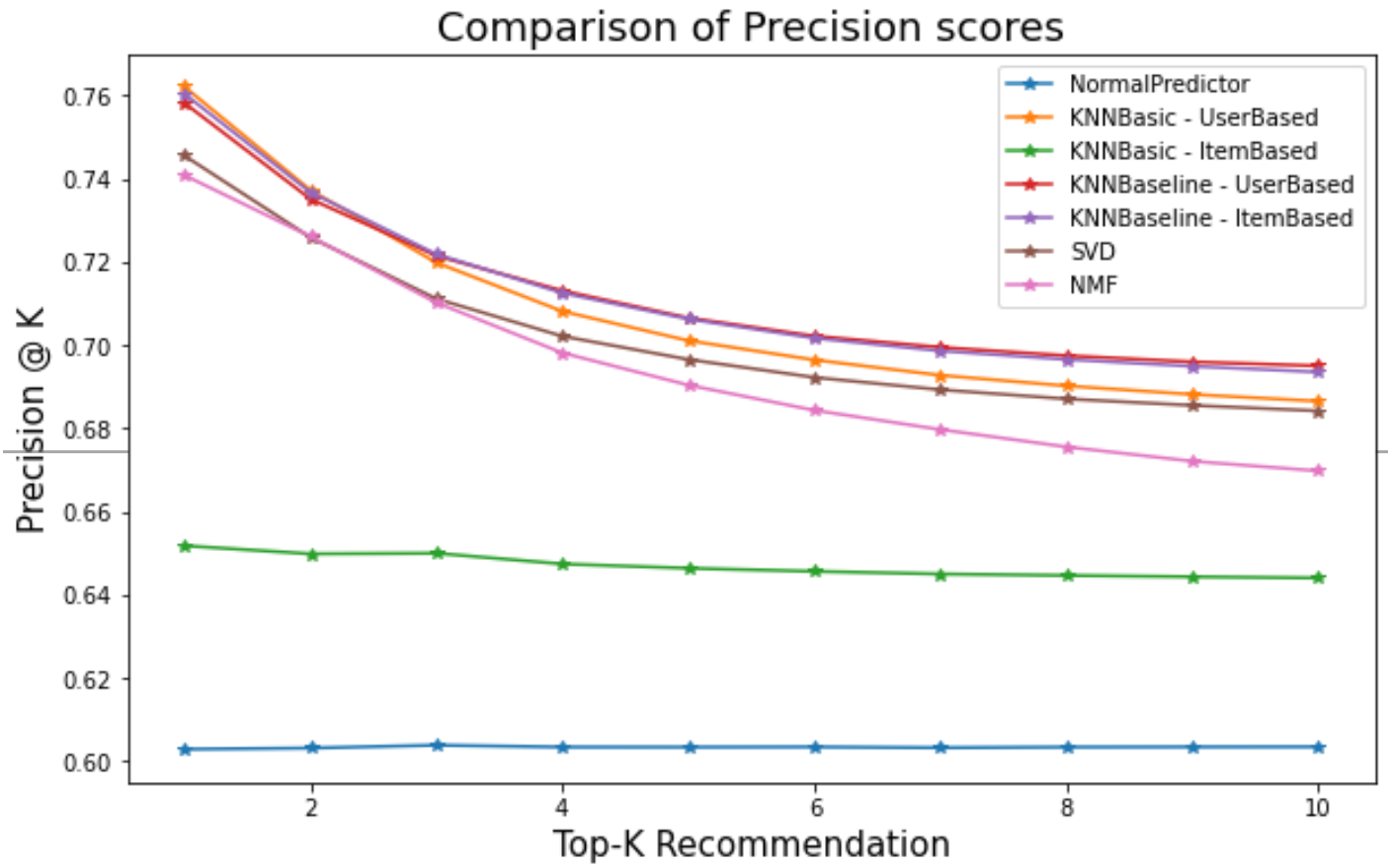
Error Metrics

SVD and KNNBasline perform the best in terms of both MAE and RMSE. Both these approaches outperform the baseline approach, NormalPredictor.

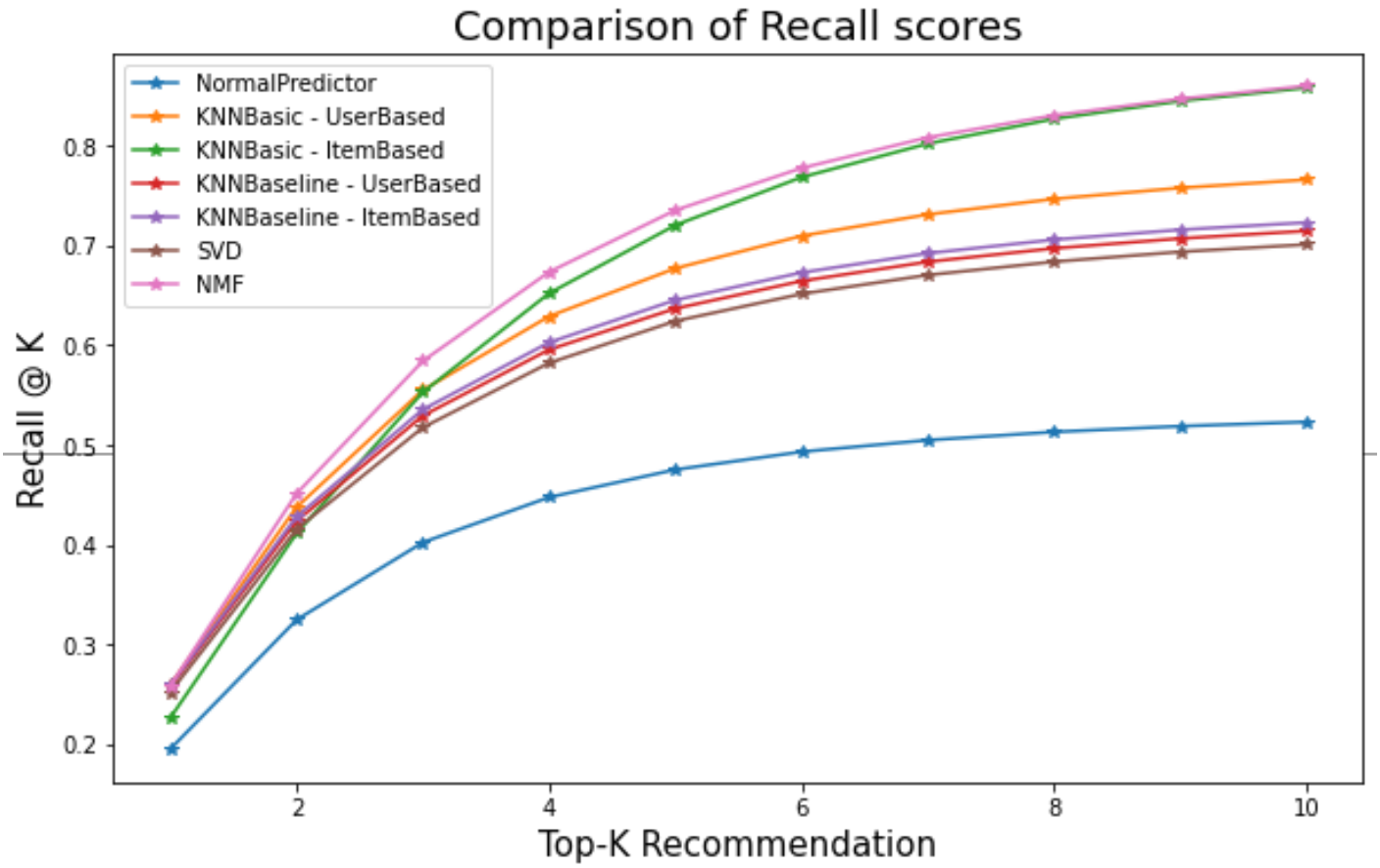


Accuracy Metrics

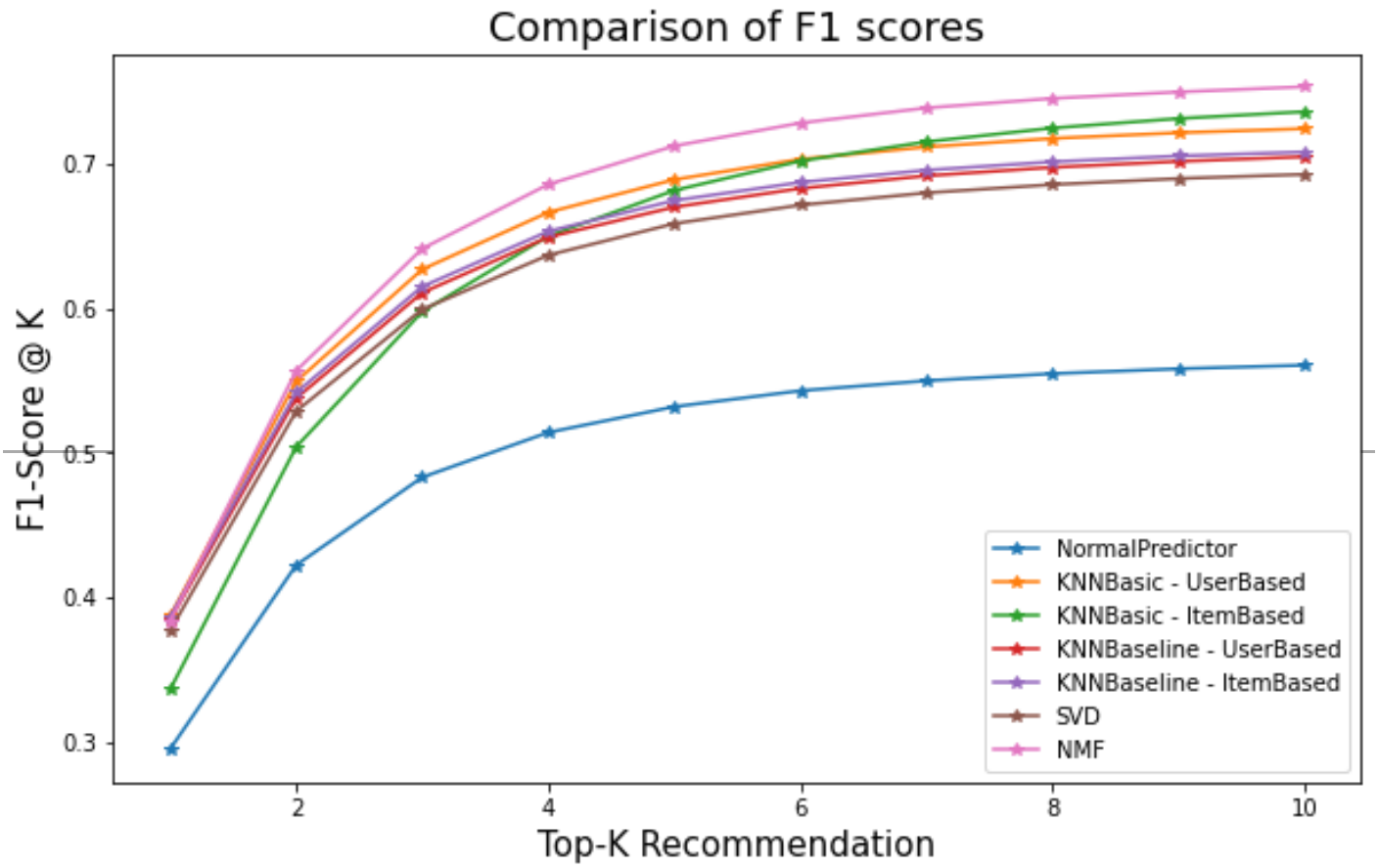
In terms of Precision, SVD and KNNBaseline give the best Precision scores for different values of K.



Unlike Precision, we get the best Recall scores for NMF and KNNBasic.

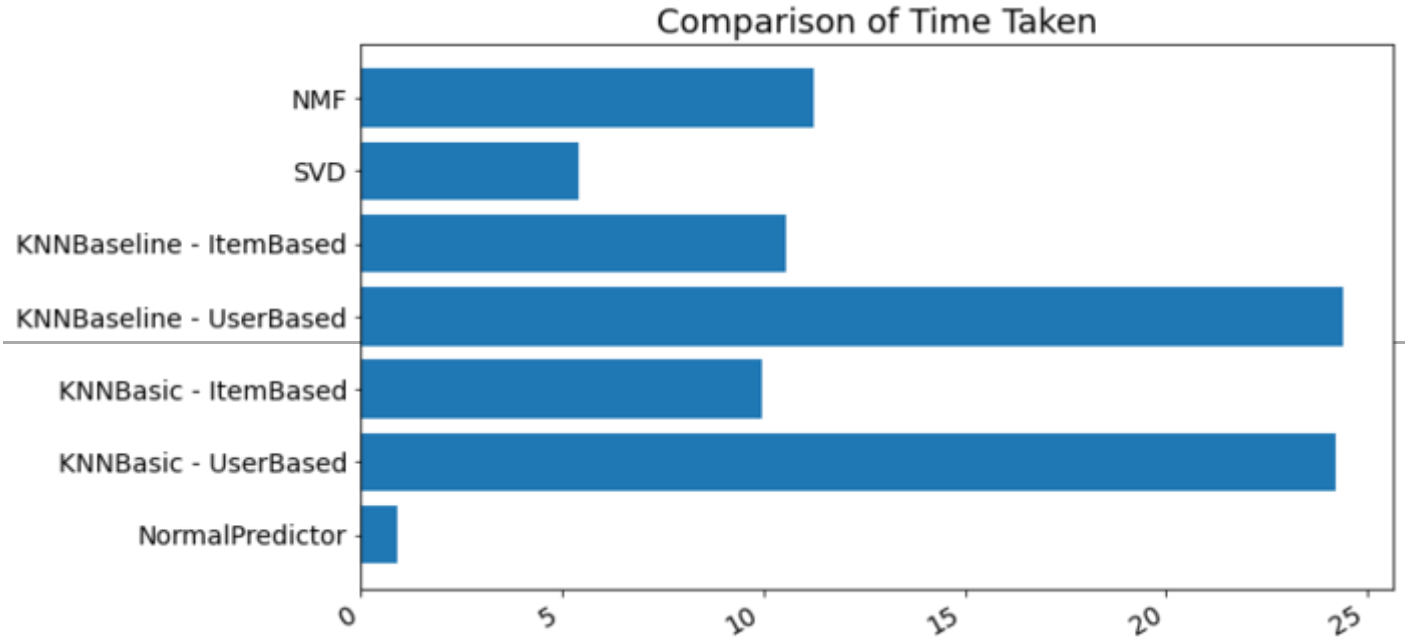


We use the F1 score to decide the best memory-based and model-based CF approaches. In the case of memory-based methods, KNNBasic outperforms KNNBaseline, while for the model-based approach, NMF scores better than SVD in terms of F1 scores.



Time Comparison

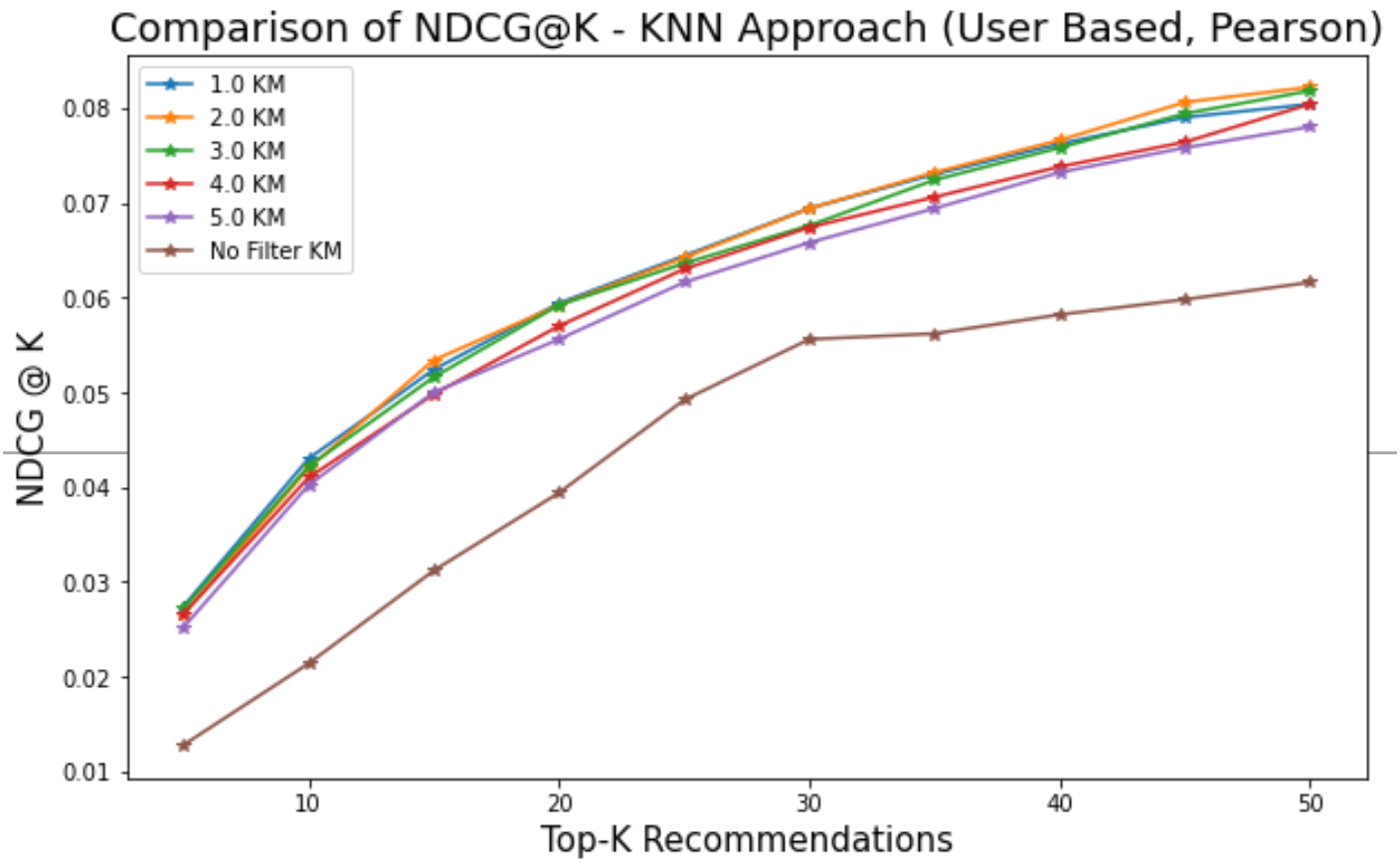
Below we compare the time taken to build the different models:

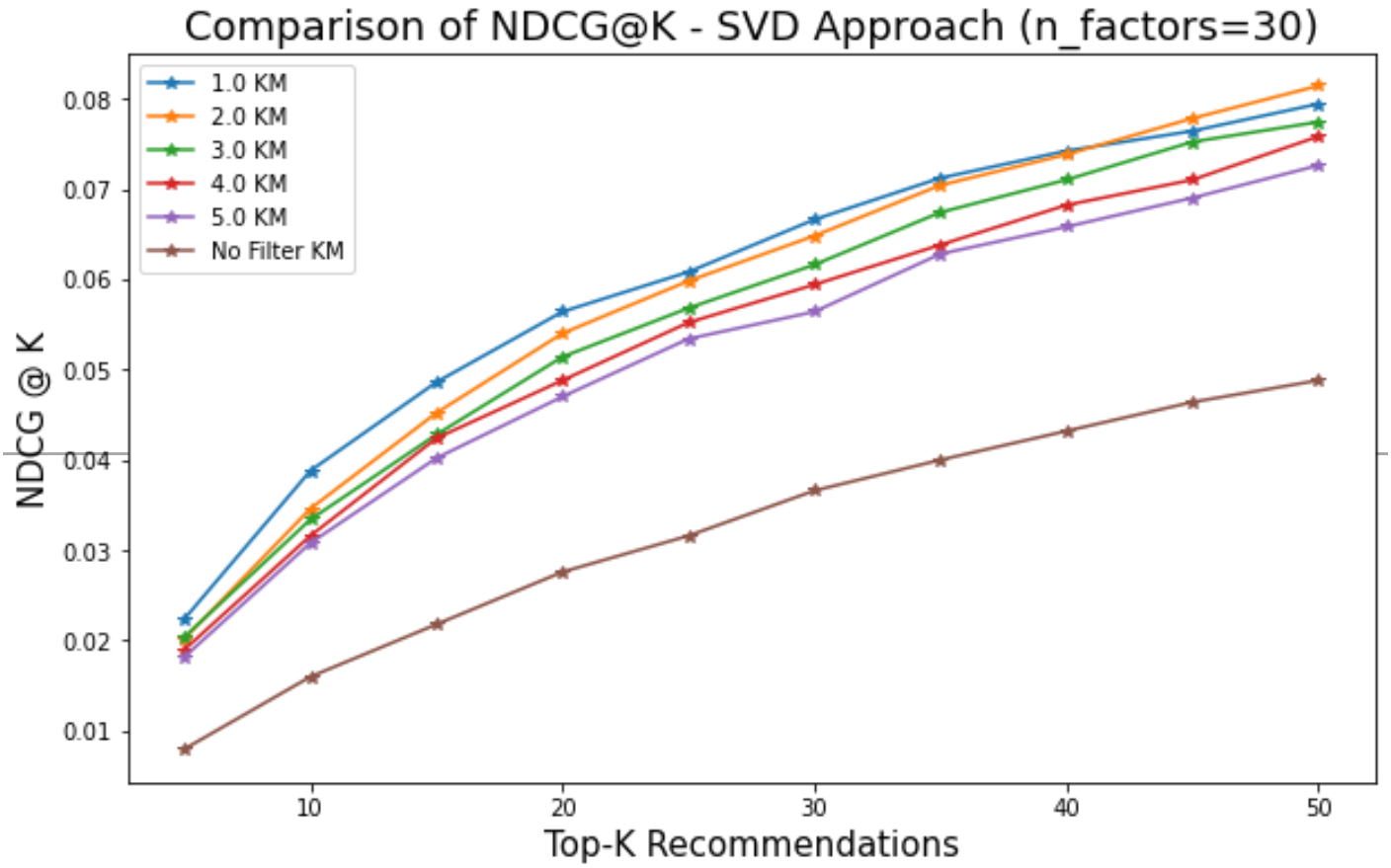


SVD takes the least amount of time in comparison to other CF-based approaches. User-Based CF takes the maximum amount of time as similarity needs to be calculated between each pair of users.

Top- N Recommendations: Geolocation Based Filtering

To improve the quality of Top-N Recommendations given to a user, we employ geolocation-based filtering. We filter the restaurants that are located in the areas the user frequents the most. It helps us recommend restaurants to the users that are convenient to reach. We vary the distance threshold from 1 Km to 5 Km and measure the change in Top-N Recommendations' quality using the NDCG metric. We test it only for SVD and KNNBasic approaches. Additionally, we use Ball Tree to quickly compute the haversine distance between the different restaurants.





Overall, geolocation-based filtering improves the quality of Top-N recommendation significantly. For KNNBasic, a distance threshold of 2 to 3 Kms yields the best NDCG score. In the case of SVD, a distance threshold of 1 Km gives the best NDCG score. Therefore, geolocation-based filtering can help make the right recommendations to the customers.