HW2

2024FA_COMP_SCI_349-0_SEC1
Prof. David Demeter

Group members:
Michael Bertagna
Paul McSlarrow
Medini Chopra
Benjamin Ledoux
Siddhika Swarup

Part I

K-Nearest-Neighbors

(3.0 points) Please describe all of your design choices and hyper-parameter selections in a paragraph. Run your classifier on the test set and summarize results in a 10x10 confusion matrix for each distance metric. Analyze your results in another paragraph.

K-nearest neighbors (KNN) is a machine learning technique that uses the proximity of data points within the training set to make predictions for a test example. In implementing the KNN classifier, we iterated through each example in the candidate set and calculated the k shortest distances between the candidate example and all training examples, using a heap to store these distances. We then selected the most common label among the k closest neighbors and assigned this label to the candidate example. For design choices, we employed a brute-force approach to identify the optimal distance metric and k values for our model. By using the 10x10 confusion matrices to analyze the performance of our different distance metrics, we found that cosine similarity typically did the best with regards to consistency in precision and recall (but not by a drastic amount). After experimenting with various options, we also found that KNN performed best with an average pooling (stride of two) dimensionality reduction technique and the use of cosine similarity k between three and six. We found that there were slightly more accurate cross-validation searches for the hyperparameter k, however, the trade off between test and validation accuracy was not large enough to want to increase k past six (see **Figure 1** below). On another note, because cosine similarity is a way to measure how similar vectors are by calculating the angle between them, and that the dataset has high dimensionality (784 dimensions), it is well understood why cosine similarity performs well. Our results are summarized in visualizations and tables below.

Analysis of the KNN model's performance on the final test set shows that using the Cosine distance metric with K=4 and applying dimensionality reduction achieved the best results, with an accuracy of 0.94 and both precision and recall at 0.939. Overall, the Cosine metric consistently outperformed the Euclidean metric in terms of accuracy and confusion matrix scores. As expected, the confusion matrix confirms that the Cosine metric provided a better fit to the data with dimensionality reduction and $K=4$. Notably, the accuracy of KNN with the Euclidean metric improved across all tested values of K following dimensionality reduction. This trend is expected, as the Euclidean distance metric is sensitive to the curse of dimensionality; reducing the number of input features can therefore enhance its performance.

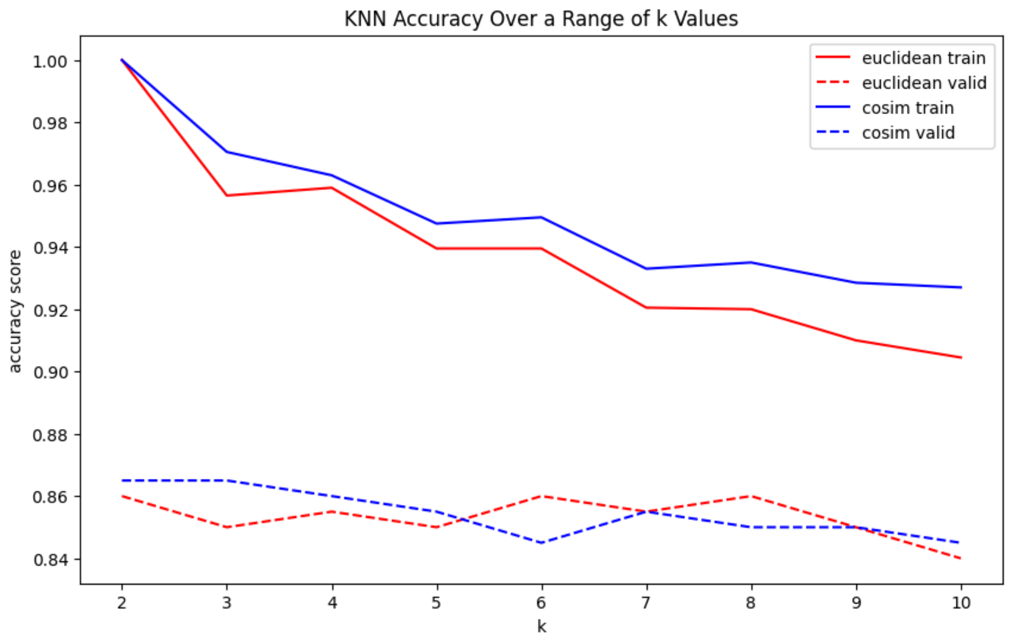


Figure 1

Test set accuracy for:

euclidean with k=6: 0.885 cosine with k=7: 0.915

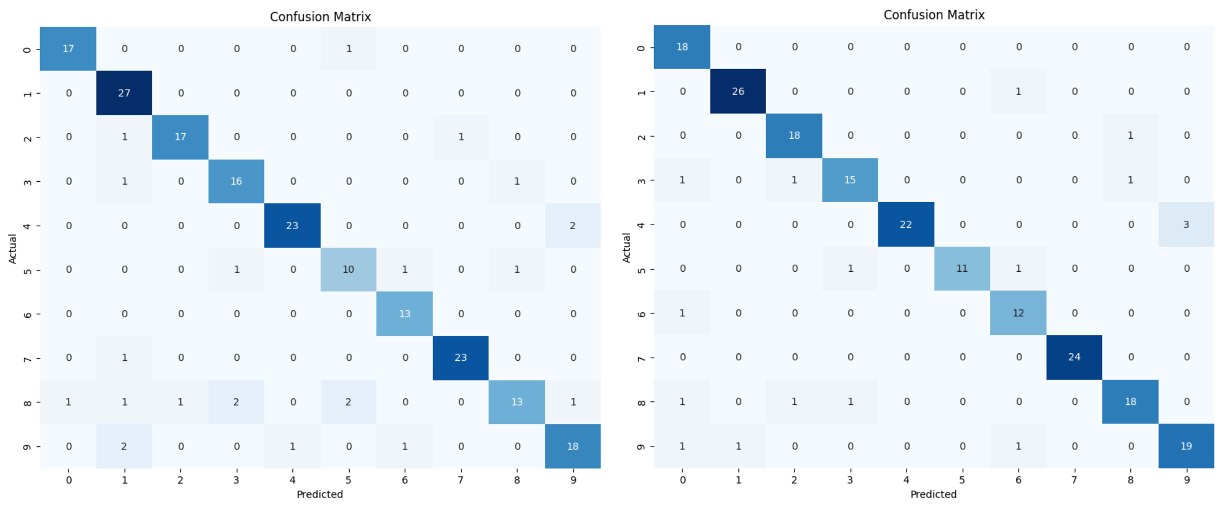


Figure 2

Test set accuracy per metric (by hyperparameter k) (without dimensionality reduction)

	K=4	**K=5**	**K=6**	**K=7**
Euclidean	0.9	0.895	0.885	0.91
Cosine	0.925	0.92	0.915	0.915

Downsampling results

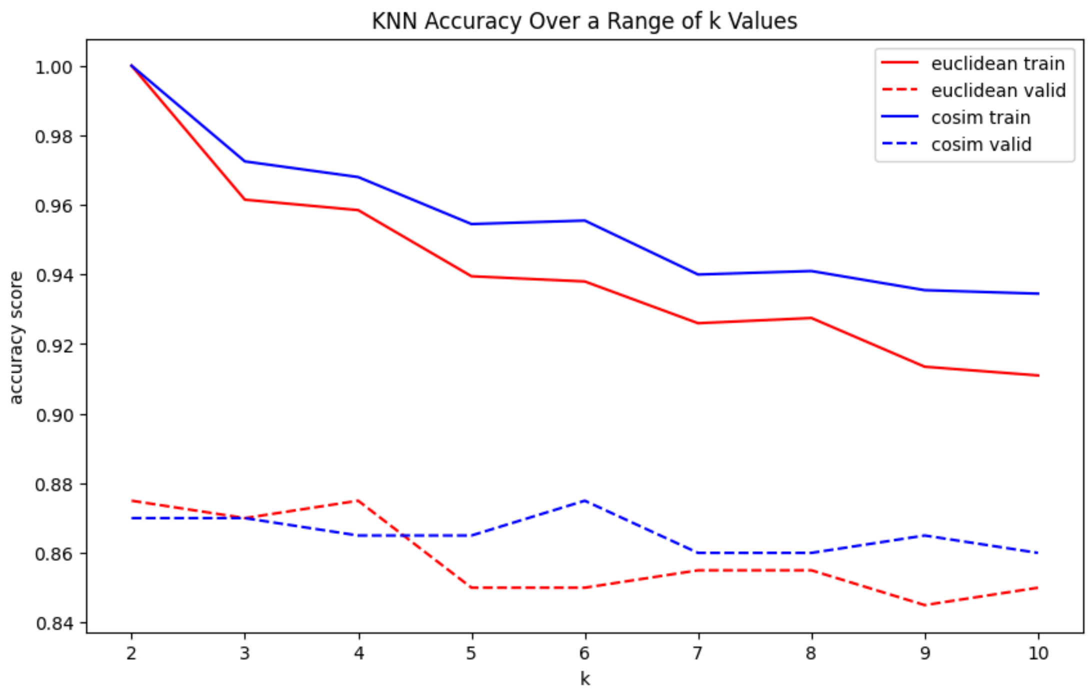
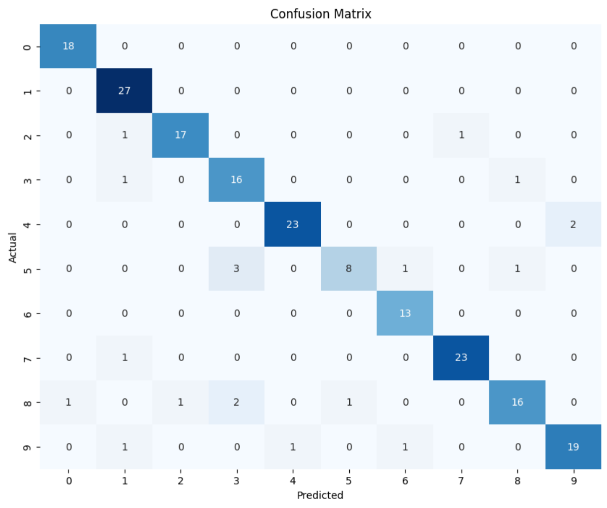


Figure 3

Test set accuracy for:

euclidean with k=4: 0.9



cosine with k=6: 0.935

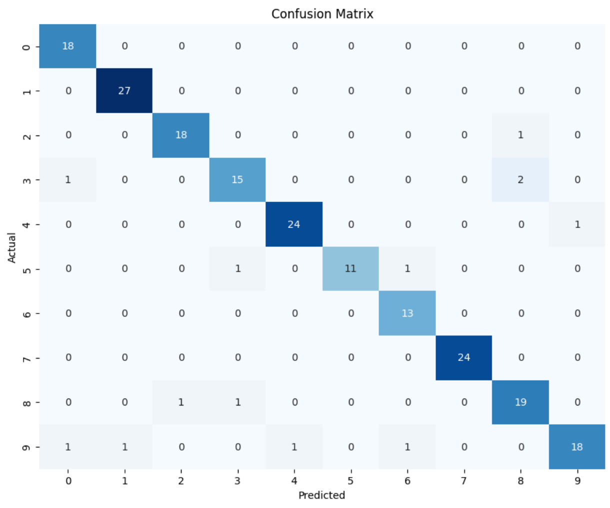


Figure 4

Test set accuracy per metric (by hyperparameter k) (with dimensionality reduction)

	K=4	**K=5**	**K=6**	**K=7**
Euclidean	0.9	0.91	0.91	0.895
Cosine	0.94	0.93	0.935	0.925

Final test set accuracies of KNN

K	**Metric**	**With Dim Reduction**	**Without Dim Reduction**
4	Euclidean	0.9	0.9
4	**Cosine**	**0.94**	**0.925**
5	Euclidean	0.91	0.895
5	Cosine	0.93	0.92
6	Euclidean	0.91	0.885
6	Cosine	0.935	0.915
7	Euclidean	0.895	0.91
7	Cosine	0.925	0.915

Final confusion matrix information

K	**Dim Reduction**	**Metric**	**Precision**	**Recall**
4	No	Euclidean	0.900	0.893

4	Yes	Euclidean	0.899	0.890
4	No	Cosine	0.920	0.925
4	**Yes**	**Cosine**	**0.939**	**0.939**
5	No	Euclidean	0.895	0.892
5	Yes	Euclidean	0.909	0.899
5	No	Cosine	0.916	0.921
5	Yes	Cosine	0.929	0.930
6	No	Euclidean	0.883	0.881
6	Yes	Euclidean	0.907	0.897
6	No	Cosine	0.913	0.915
6	Yes	Cosine	0.933	0.931
7	No	Euclidean	0.909	0.902
7	Yes	Euclidean	0.897	0.881
7	No	Cosine	0.913	0.911
7	Yes	Cosine	0.924	0.923

K–Means

(3.0 points) Present a quantitative metric to measure how well your clusters align with the labels in mnist_test.csv. Describe your design choices and analyze your results in about one paragraph each.

To measure how well the clusters align with the labels in mnist_test.csv, we used Adjusted Mutual Information (AMI) as the quantitative metric. AMI assesses the amount of information shared between the true labels and the cluster memberships. Since k-means is an unsupervised algorithm that produces clusters without explicit labels, we can treat the true labels as groups of data points. This allows us to evaluate the clustering agreement between the two sets: a perfect agreement yields an AMI value of 1, while a value of 0 indicates no agreement, and negative values suggest less agreement than would be expected by chance.

In designing the k-means clustering algorithm, we start by selecting k random centroids from the dataset. We set k equal to 10 as there were ten possible labels for the dataset; if we did not know the amount of labels in the dataset, we would have performed hyperparameter tuning on k. A copy of the data is then randomly shuffled and divided into k equally sized chunks, from which the average value for each feature (pixel intensity) is computed. Next, we calculate the distance of each data point to each centroid and assign each point to the cluster of its closest centroid. The centroids are updated by calculating the average value for each feature across all data points within each cluster. This iterative process continues until the centroids stabilize and no longer change, resulting in a final list indicating the cluster membership for each data point. A critical design choice was to maintain the original order of the data while tracking the centroid membership for each data point. This approach allows for the computation of AMI without the need for additional data manipulation post-clustering, ensuring that the evaluation process is straightforward and efficient.

Without dimensionality reduction, our KMeans clustering algorithm achieved Adjusted Mutual Information (AMI) scores of 0.49 for Euclidean distance and 0.51 for cosine distance. After applying dimensionality reduction using an average pooling technique with a stride of two, the AMI scores remained 0.49 for Euclidean distance and 0.51 for cosine distance. While cosine distance performance was unaffected, clustering with Euclidean distance showed a slight improvement after dimensionality reduction. This is expected, as Euclidean distance is sensitive to the curse of dimensionality; reducing the number of input features by half can improve its performance. Overall, the scores of 0.49-0.51 indicate that our clustering approach successfully grouped approximately 50% of the data, significantly outperforming the 10% accuracy expected by chance. The results are summarized in the following table:

Final Adjusted Mutual Information (AMI) per metric

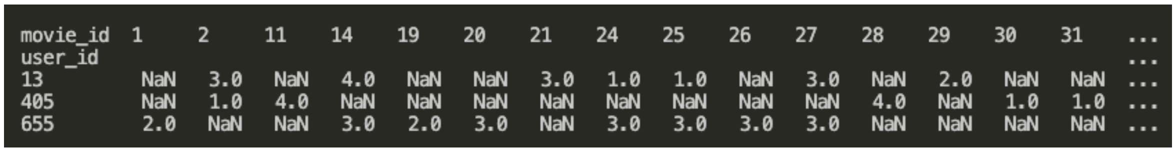
	Without dimensionality reduction	**With dimensionality reduction**
Euclidean	0.49	0.50
Cosine	0.51	0.51

Part II

<u>Collaborative Filtering Movie Recommendation System</u>

(3.0 points) Please describe how your collaborative filter works, list the hyper-parameters and describe their role. Report precision, recall, and the F1-score on the validation and test sets for users a, b, and c. Discuss how M impacts your results.

There are two most commonly used methods for collaborative filtering systems – user-based and item-based collaborative filtering. For the purpose of the task at hand in recommending movies to users, we chose to conduct a user-based collaborative filtering method to learn what to recommend to each user. A user-based method recommends items by identifying users similar to the target user, while an item-based method recommends items based on similarities to those the user has previously interacted with. To begin this process of user-based collaborative filtering, we first combined all of the training data together, then created a pivot table where the columns represent each movie (id), and the rows represent each user (id), and the values represent the rating given to that movie by each user (**Figure 5**). Please note that NaN values indicate that the user did not rate the corresponding movie.



```
movie_id  1     2     11    14    19    20    21    24    25    26    27    28    29    30    31    ...
user_id                                                                                         ...
13        NaN   3.0   NaN   4.0   NaN   NaN   3.0   1.0   1.0   NaN   3.0   NaN   2.0   NaN   NaN   ...
405       NaN   1.0   4.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   4.0   NaN   1.0   1.0   ...
655       2.0   NaN   NaN   3.0   2.0   3.0   NaN   3.0   3.0   3.0   3.0   NaN   NaN   NaN   NaN   ...
```

Figure 5

Given this information, we were able to use cosine similarity as our metric to find the similarity between each user based on the ratings they both gave to movies. Please note that we only calculated the cosine similarity between users based on the movies that both users have reviewed, which tells us how similar their tastes are. After getting the similarity between every user, we needed a way to numerically represent the likelihood of each user watching a movie they haven't seen. **Figure 6** illustrates this calculation as the estimated rating for a user *i* to enjoy a movie *m* is calculated considering all other users *j* who have rated the movie *m*. For each of the users *j* who have rated the movie, we take the cosine similarity between a user *i* and user *j* and multiply it by the rating given to the movie *m* by user *j*. By summing up these weighted contributions, we then divide by the sum of similarity scores. In simpler terms, we calculate an estimated rating for each user to watch a certain movie, weighted more heavily towards what similar users rated, and then keep only the top M of these ratings for the user's recommendations.

$$\hat{r}_{i,m} = \frac{\sum_{j \in N} S_{ij} \cdot r_{j,m}}{\sum_{j \in N} S_{ij}}$$

Figure 6

Figure 7 illustrates how precision, recall, and F1 scores vary as the number of recommended movies (M) increases. As expected, we observe a decrease in precision with increasing M. This is because as more recommendations are made, the model includes items with lower confidence levels, increasing the likelihood of irrelevant recommendations. Based on these findings, we suggest keeping recommendations under 10 to enhance user experience, though the optimal number may vary depending on design goals. We also observe consistently low recall and F1 scores due to the assumption that all movies a user rated in the validation and test sets are relevant. Since recall is defined as the number of relevant movies in the top M recommendations divided by the total number of relevant movies, it remains low as the total relevant count often exceeds M. Consequently, F1 is also low, given it is the harmonic mean of precision and recall, which explains why the recall and F1 curves nearly overlap in **Figure 7**.

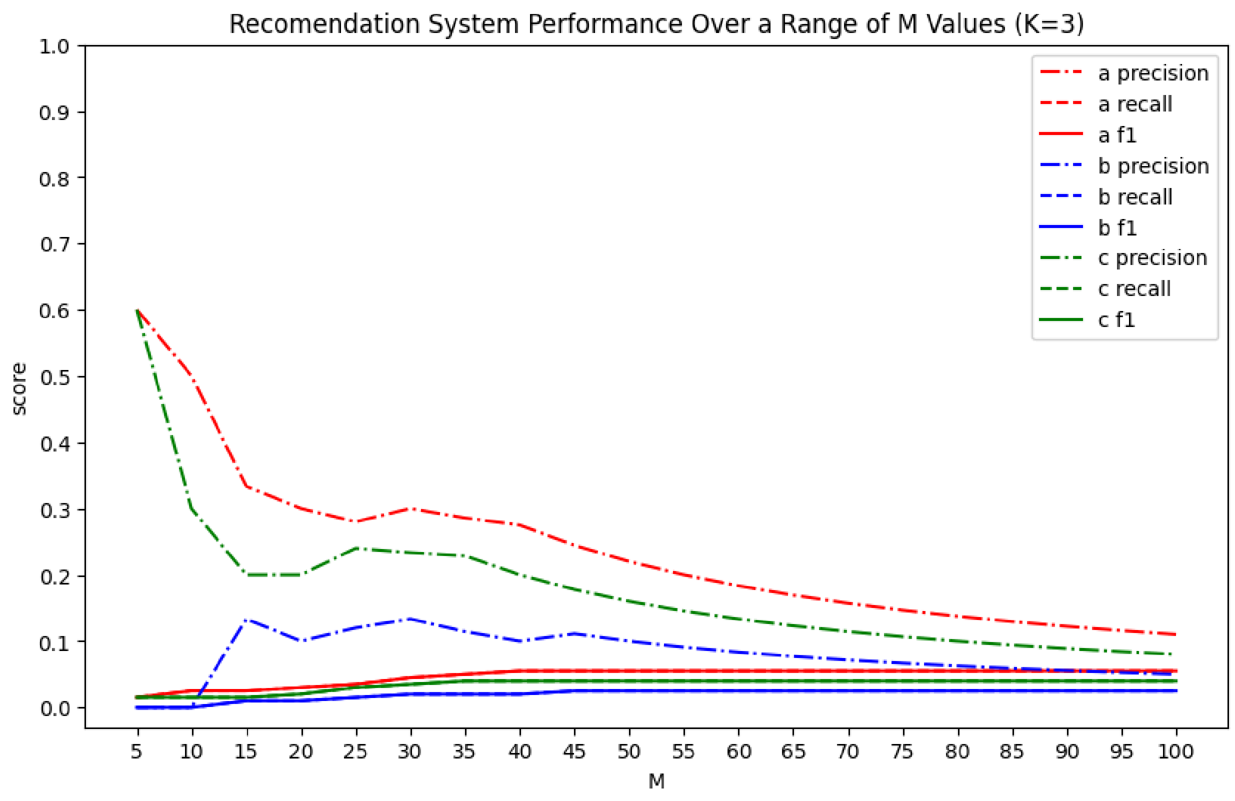


Figure 7

(2.0 points) Try to improve your collaborative filter built in Question #5 by using movie genre or user demographic data(e.g., age, gender and occupation). Report precision, recall, and the F1-score on the validation and test sets for users a, b, and c. Discuss your approach and whether or not considering additional features improved the performance of your collaborative filter.

We incorporated demographic data (i.e., age, gender, occupation) into the user-based collaborative filtering system by computing the cosine similarity of users based on these attributes. We then added this demographic similarity score to the existing rating similarity score. Specifically, S_{ij} was calculated as the sum of the movie rating similarity and the demographic similarity for each user pair, assigning equal weight to each component. Integrating these demographic features improved the system's performance, particularly increasing precision up to M=35 for users b and c (**Figure 8**). Interestingly, the results for user a were unaffected. Recall and F1 continued to behave as described earlier, so they were not particularly informative in this context.

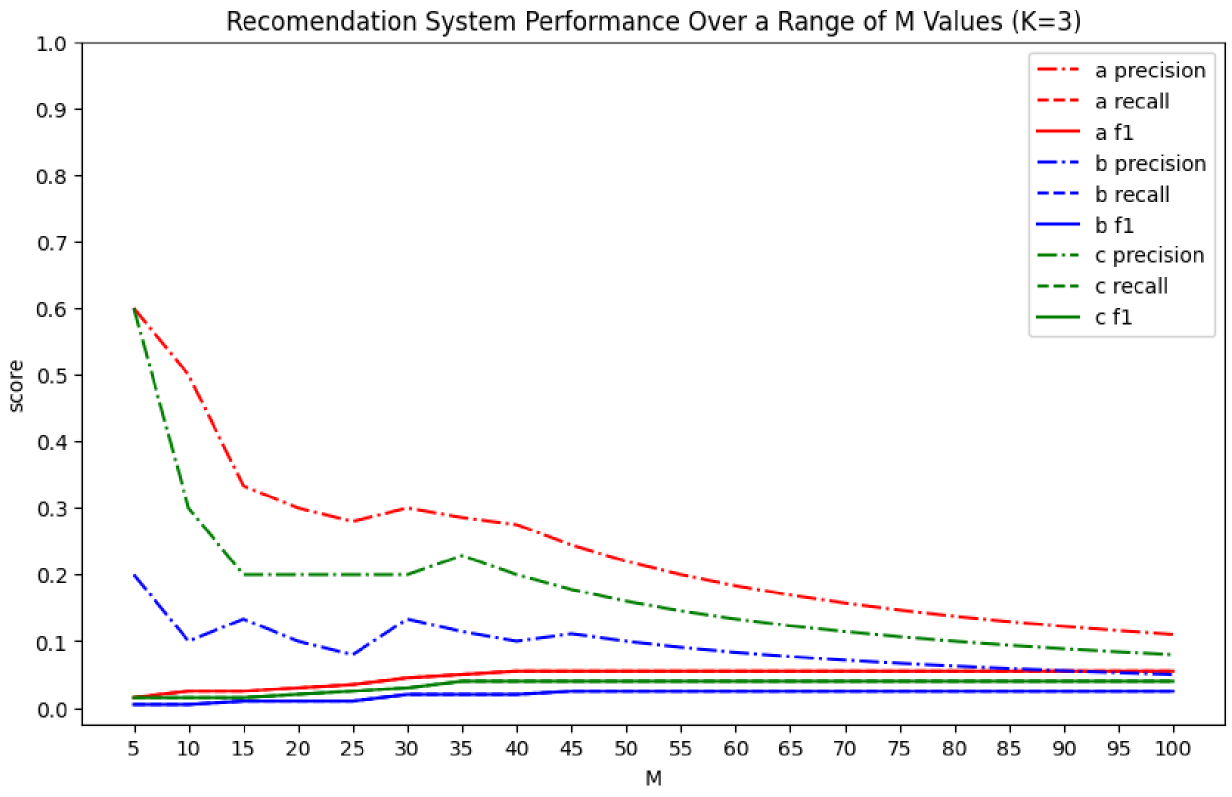

Recomendation System Performance Over a Range of M Values (K=3)
score
M
a precision
a recall
a f1
b precision
b recall
b f1
c precision
c recall
c f1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

Figure 8