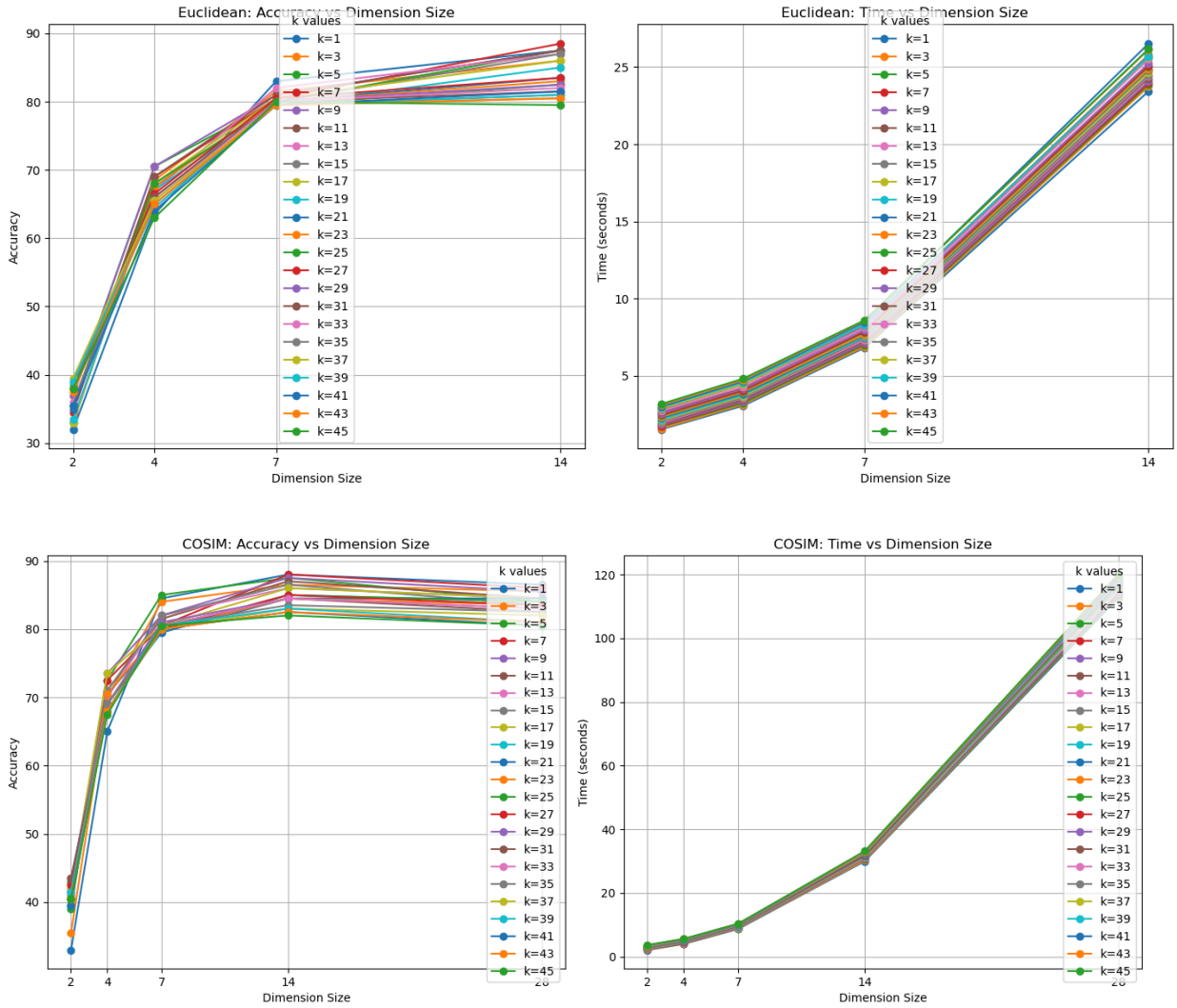
2a: Hyperparameter Selections

The first and most obvious hyperparameter I chose was K which determined how many neighbors we would compare the labels of when finding distance/similarity of a given data point. The second hyperparameter relates to dimension reduction to decrease runtime of the program.

This was done by grouping evenly sized sections of pixels together to reduce the original data from 28x28 pixel images to NxN pixel images with each of the N^2 sections being 28/N in each dimension - so downsizing to 7x7 resulted in 4x4 sections. I then found the average gray-scale value of all the pixels for each section and assigned that as the value of the new “pixel”. By doing this, the average runtime of the program with any given K decreased by 73% and 92% when using 28x28 (original) versus 14x14 and 7x7 respectively.

To choose optimal K and dimension values, I ran the model against the validation set with K values ranging from 1 to 45 in increments of 2 (odd values only), testing each possible dimension (2, 4, 7, 14, and 28) for every iteration of K. This was done for both euclidean and cosim, followed by graphing the percent accuracy for each K and dimension combination along with time to run.



Based on the data, I chose a K value of 11 and a dimension size of 7 to compromise between run time (7x7 to 14x14 shows a large jump in time for all K values) and accuracy (K=11 consistently does well for all dimension sizes).

2b: Confusion Matrices

The resulting confusion matrices when using the testing set are as follows:

Euclidean Matrix Table (k=11, dimension=7x7, acc=91%)

	0	1	2	3	4	5	6	7	8	9
0	18	0	0	0	0	0	0	0	0	0
1	0	26	0	0	0	0	1	0	0	0
2	0	0	18	0	0	0	0	1	0	0
3	0	1	0	16	0	0	0	0	1	0
4	0	0	0	0	23	0	0	0	0	2
5	0	0	0	1	0	9	2	0	1	0
6	1	0	0	0	0	0	12	0	0	0
7	0	0	0	0	0	0	0	24	0	0
8	0	2	0	0	0	1	0	0	17	1
9	0	1	0	0	1	0	1	0	0	19

Cosim Matrix Table (k=11, dimension=7x7, acc=91%)

	0	1	2	3	4	5	6	7	8	9
0	18	0	0	0	0	0	0	0	0	0
1	0	26	0	0	0	0	1	0	0	0
2	0	0	18	0	0	0	0	1	0	0
3	1	1	0	15	0	0	0	0	1	0
4	0	0	0	0	23	0	0	0	0	2
5	0	0	0	1	0	9	2	0	1	0
6	1	0	0	0	0	0	12	0	0	0
7	0	0	0	0	0	0	0	24	0	0
8	0	2	0	0	0	0	0	1	18	0
9	0	1	0	0	1	0	1	0	0	19

The meat of the analysis is that the dimension reduction worked very well in decreasing time (92% decrease for 7x7) while still maintaining a high accuracy, which, when compared to the untransformed data for cosim, actually outperforms it. This suggests that the dimension reduction was able to keep details in the data needed for distinguishing between different values. However, arguably more complex values such as 5, 8, and 9 continued to be hard to identify compared to “easier” numbers like 0, 1, and 2. It may be possible to improve these values by changing K when the model is less certain - more diversity in nearby neighbors - about its prediction.

3: K Means

Given the way read_data() works, it was important that we first transformed and configured the data to be usable. In our case, this meant casting all data and query values to be floats. We also removed the labels from the data since we wouldn't be using them. Query labels were removed outside of the function since the assumption is that typically query data would be unlabelled. Generally, that would also be the case with input data, but since we knew we would always be using labeled data for the training portion, it just made the most sense to handle that within the function. We also transformed the data to be 7x7 pictures instead of 28x28 just for ease of computation and to save time. When making the KNN functions, we found that this did not affect the accuracy. We used the same methods of data transformation for KNN and KMeans. We set a max number of iterations to avoid overfitting because of too many iterations and also to stop it from running too long. Our centroids are averages of the data a given distance away from the data point. We update these centroids throughout the iterations by summing up the values and changing them based on the current clusters.

Based on validation set:

k-value	Accuracy (euclidean, transformed)	Accuracy (cosim, transformed)
3	0.245	0.26
4	0.175	0.23
5	0.295	0.305
6	0.22	0.36
7	0.275	0.27
8	0.33	0.34
9	0.31	0.325
10	0.29	0.34

11	0.415	0.385
20	0.42	0.5
40	0.615	0.61
50	0.553	0.65

Based on testing set:

k-value	Accuracy (euclidean, transformed)	Accuracy (cosim, transformed)
3	0.135	0.21
4	0.245	0.26
5	0.42	0.265
6	0.325	0.215
7	0.315	0.275
8	0.39	0.345
9	0.485	0.365
10	0.435	0.295
11	0.415	0.44
17	0.53	0.455
20	0.46	0.575
40	0.59	0.695
50	0.615	0.73

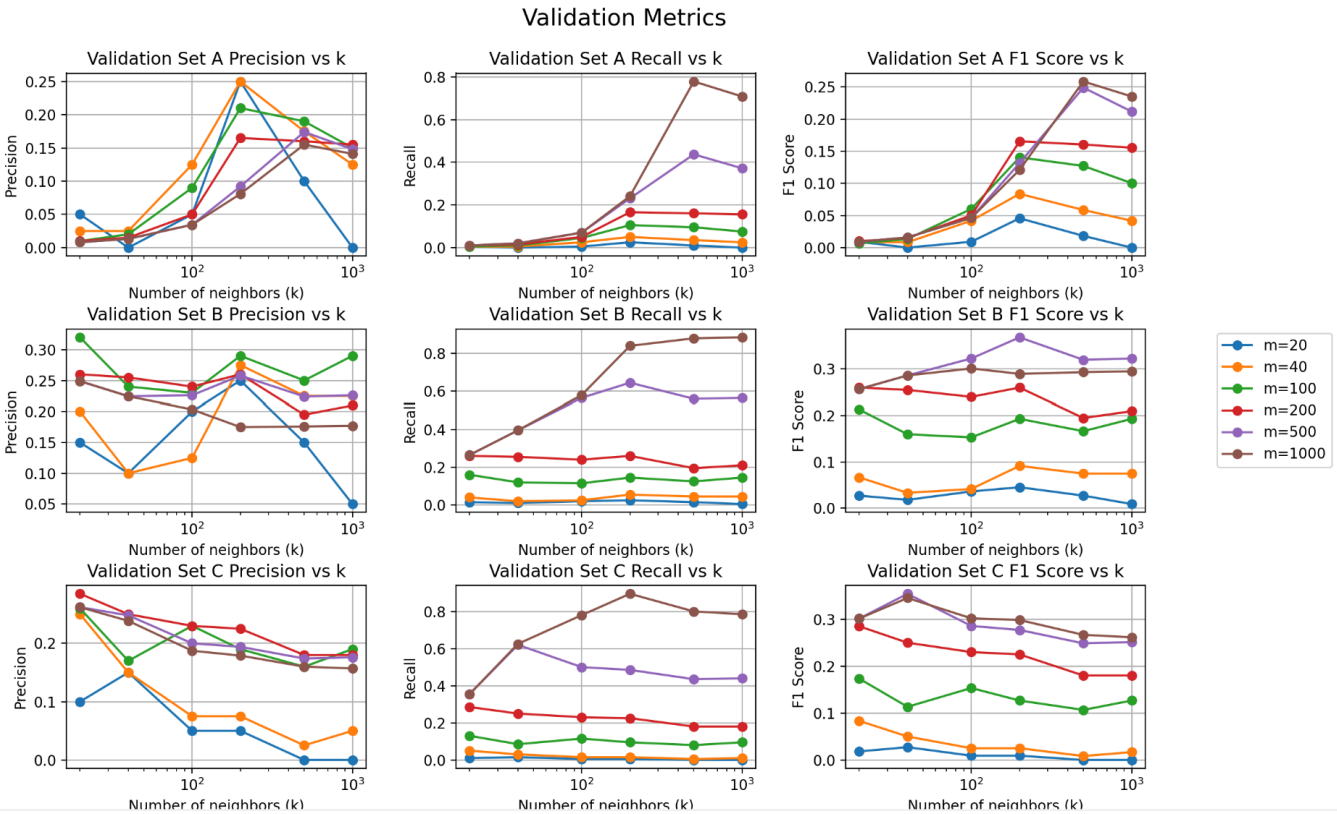
Because we constantly adjust the values of the centroids and clusters, there is not as much of a need for training against the validation set. Regardless, it's interesting to see how the k-values compare. Initially, we thought the k-value doesn't really make sense to be less than 9 when we're classifying for at least 9 different

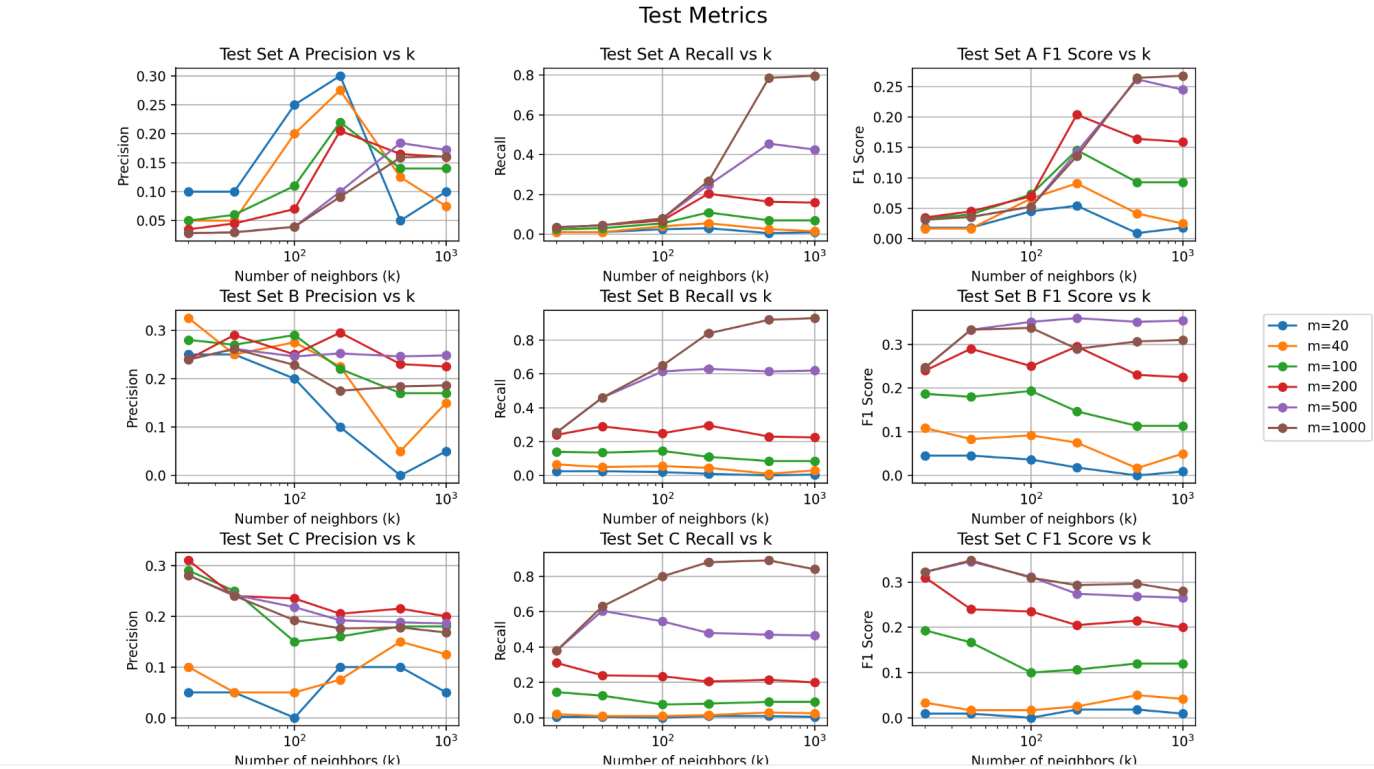
categories. By not considering the number of categories, the accuracy is reduced. On the testing table for the euclidean, the accuracy jumps up at k=9, and k=11 for the cosim. For the validation table, it jumps at around k=11 for the euclidean distance and around k=20 for the cosim. In general, the accuracy tends to fluctuate. For the testing set, when k<9, the values range from 0.135 - 0.39 for the euclidean distance and from __ - __ for the cosim. The accuracy for the validation set when k<11 for ranges from 0.175 - 0.33 for the euclidean and from 0.23 - 0.385 for the cosim when k<20. From there, the cosim seems to consistently increase while the euclidean continues to fluctuate. Though the values fluctuate, they also seem to consistently increase after the jump. Despite these differences, both cosim and euclidean accuracies peak at k=40 in the validation set. Regardless, it seems most logical to have two k-values: one for the euclidean distance and one for the cosim. We used these jumping points as k-values. We defined k=11 as our k-value for the euclidean distance and k=20 as our k-value for the cosim. Interestingly enough, for the cosim in the testing set, the accuracy continues to increase with the k-value, past the chosen k=20.

4: How M and K affect precision, recall and f1 score

In order to explain the trends in the graphs, it is important to note how the values generated by our dataset were calculated. For the purposes of this homework, a true positive was defined as a movie that was recommended and appears in the test or validation set of the user. A false positive was defined as a movie that was recommended but does not appear in the test or validation set. A false negative was defined as a movie that appears in actual ratings but was not in the test or validation set.

As can be seen from the graphs below, there are clear patterns in how precision, recall and F1 scores vary with different values of k (number of similar users considered) and m (number of movie recommendations).





From the graph above, it can be seen that on average, a value of m = 100 resulted in the greatest average precision. The fact that precision doesn't increase with lower values of m suggests that there may be an issue in how we are calculating the recommendation scores. The collaborative filter function could likely be improved by having a minimum threshold of similar users to rate the movie, as a movie could get a high recommendation score simply due to there being one similar user that rated the movie highly. On the other hand, larger values of m resulting in lower precision makes intuitive sense as the more movies recommended, the more likely it is that we will begin to recommend movies that are irrelevant and are simply being picked to fill the quota of movies to recommend, thus resulting in movies that the user won't rate positively to be recommended, decreasing our overall precision.

For recall scores, they consistently improve as m increases, reaching its highest value when m is equal to 1000. This is expected due to how recall is implemented, as it is simply measured by seeing how many of the recommended movies are in the test or validation set of the selected user. Hence, as we recommend more movies, the more likely those recommended movies will appear within the test or validation set of the selected user. However, this comes at the drawback of decreasing our precision.

For f1 scores, which tries to balance precision and recall, we can see that the best f1 scores are achieved when m is around 500 to 1000, suggesting that m values within this range provide the best trade-off between precision and recall scores. As can be seen, larger k values also tend to increase the f1 scores, which suggests that considering more similar users tends to result in the collaborative filter giving better recommendations.

Through testing our function on the validation sets, we can find when the f1 score is maximized.

Validation Set	Highest f1 score	K value	M value
A	0.259	500	1000
B	0.369	200	500
C	0.354	40	500

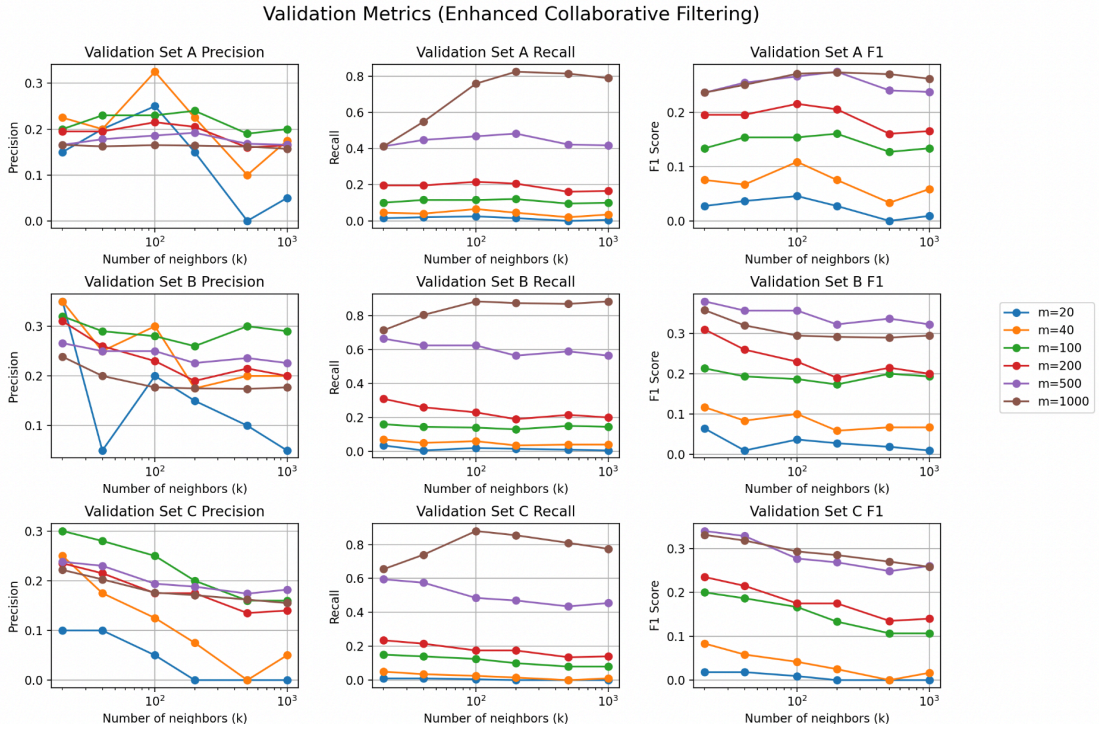
When calculating the average f1 score for our different hyper parameters, we find that a k value of 500 and an m value of 1000 gives us the best overall f1 values, resulting in an average f1 score of 0.273. These hyper parameters result in the following values for validation sets A, B and C.

Validation Set	F1 score
A	0.259
B	0.293

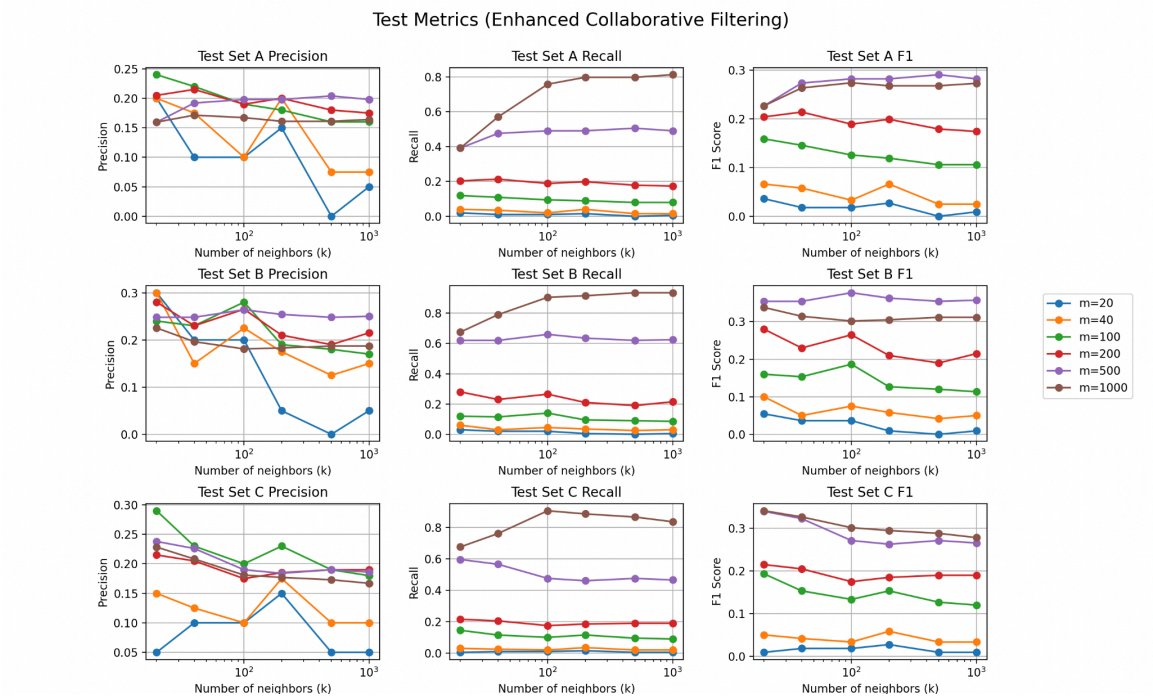
C	0.267

5. Precision, recall, and F1-score for Enhanced Collaborative Filter and it's comparative performance with the basic version

To improve on the basic collaborative filtering approach, the algorithm was enhanced by incorporating movie genre and user demographic data (age, gender, and occupation) using an adaptive weighting system. Rating similarity received a base weight of 60% with additional weight placed on common movies, while the remaining being distributed among genre and demographic similarities.



The validation metrics show strong performance across all three test sets. Precision performs best with lower k values (20-200), reaching maximum validation scores of 0.325, 0.350, and 0.300 for sets a,b, and c respectively. Recall consistently improves with larger m values, achieving maximum validation scores of 0.824, 0.885, and 0.880. F1 scores show optimal performance with moderate k values and m=500, reaching 0.275, 0.380, 0.340 for the three sets.



Test metrics confirm these patterns, with precision maxing at 0.240, 0.300, and 0.290, recall reaching 0.812, 0.935, and 0.905, and F1 scores of 0.291, 0.377, and 0.341 for sets a, b, and c respectively. These results demonstrate strong generalization of the enhanced model.

Compared to the basic collaborative filter, the enhanced version shows improvement in stability and overall performance. F1 scores improved by 17.0% for set a and 2.8% for set b, with a minimal decrease of 0.4% for set c. The additional features contribute to more reliable recommendations, particularly through better matching of user preferences and rating behaviors.

The optimal configuration varies by goal: balanced performance is best achieved with k=100-200 and m=500, maximum precision with k=20-100 and m=20-100, and maximum recall with k>=500 and m=1000. Overall, this version successfully improves recommendation quality by incorporating multiple aspects of user similarity in a balanced way.