CS 349 F’23 Homework 2 Report

Coel Morcott and Vatsal Bhargava

2 [KNN]) Please describe all of your design choices and hyper-parameter selections in a paragraph. Once you are satisfied with the performance on the validation set, run your classifier on the test set and summarize the results in a 10x10 confusion matrix. Analyze your results in another paragraph.

For our KNN, we utilized downsampling as our way to speed up the runtime, we took the 28x28 images and compressed them into 14x14 by averaging the values in each 2x2 grid. This greatly improved our run time by ~400% based on our empirical calculations. As for our hyperparameters, we used k = 5 because those were the best for both the cosine similarity KNN and the Euclidean distance KNN, getting >90% accuracy for both. As for our method of choosing the label based on the K-closest neighbors, we instituted a mode policy with a tiebreaker based on the average distances.

KNN Score Euclidean:

```
[[18, 0, 0, 0, 0, 0, 1, 0, 1, 0],
 [0, 27, 1, 0, 0, 0, 0, 1, 0, 1],
 [0, 0, 18, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 16, 0, 2, 0, 0, 1, 0],
 [0, 0, 0, 0, 23, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 10, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 1, 12, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 23, 0, 0],
 [0, 0, 0, 2, 0, 0, 0, 0, 17, 0],
 [0, 0, 0, 0, 2, 0, 0, 0, 0, 20]]
92%
```

92% overall accuracy, as can be seen with the vast majority of the values being across the diagonal of our confusion matrix. There was no real pattern in our misclassifications for Euclidean knn aside from the 8 value being misclassified the most.

KNN Score Cosine Similarity:

```
[[18, 0, 0, 1, 0, 0, 0, 0, 1, 1],
 [0, 27, 0, 0, 0, 0, 0, 1, 0, 1],
 [0, 0, 18, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 15, 0, 1, 0, 0, 1, 0],
 [0, 0, 0, 0, 24, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 11, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 13, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 23, 0, 0],
 [0, 0, 1, 2, 0, 0, 0, 0, 18, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 19]]
93%
```

93% overall accuracy, as can be seen with the vast majority of the values being across the diagonal of our confusion matrix. There was no real pattern in our misclassifications based on this one.

3 [KMeans]) Describe your design choices and analyze your results in about one paragraph each.

For the decision choice of algorithm, we got much inspiration from the class slides. What we do is first set our hyperparameters, which is a max iteration of 400, convergence of 10^-4 if nothing changes, and k of 10 (10 because there are 10 possible digits). We then call a helper function that returns to us our final centroids and labels for each of these centroids. To find this, we downsize our data, pick 10 random cluster points, and enter a for loop set to our max iterations. During each iteration, we go through all the data and calculate the distance for each respective one to all 10 clusters. Once this is done, we assign that data point to whichever cluster was closest (or most similar for cosim). Then, we update the clusters based on the formula provided in class and repeat the process. Once we hit max iterations or reach convergence, we pick each cluster label as the mode of the most popular label in each cluster and return our clusters and labels. Then, for each query, we calculate which cluster it was closest to, assign that cluster's label to it, and return them. One thing to note is that a cluster for 5 is missing, which we attempted to fix by going to office hours and exploring options like Kmeans++ and balanced initialization, both of which did not improve our accuracy by that much. Alos, we tried other ways of preprocessing our data, like binarization and normalization, which also did not affect the accuracy that much. The PM Declan and Professor Demeter said this was fine though, so we moved on. Below is our output for Kmeans.

```
KMeans Score Euclidean:
[[16, 0, 1, 1, 0, 0, 1, 0, 1, 0],
[0, 25, 2, 1, 0, 0, 0, 1, 0, 1],
[0, 1, 14, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 14, 0, 3, 0, 0, 1, 0],
[0, 0, 0, 0, 13, 0, 0, 0, 0, 5],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 1, 9, 0, 0, 1],
[0, 0, 1, 0, 7, 0, 0, 19, 1, 3],
[2, 0, 1, 2, 0, 7, 3, 1, 17, 4],
[0, 0, 0, 0, 5, 2, 0, 3, 0, 8]]
67.5%
```

KMeans Score Cosine Similarity:
```
[[15, 0, 2, 1, 0, 0, 1, 0, 1, 0],
[0, 22, 1, 0, 0, 0, 0, 1, 1, 1],
[0, 4, 15, 1, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 13, 0, 5, 0, 0, 1, 0],
[0, 0, 0, 0, 13, 0, 0, 0, 1, 7],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 1, 0, 0, 0, 3, 12, 0, 2, 2],
[0, 0, 1, 0, 7, 0, 0, 19, 1, 5],
[0, 0, 0, 3, 0, 5, 0, 0, 13, 0],
[0, 0, 0, 0, 5, 0, 0, 4, 0, 7]]
64.5%
```

While our classification accuracy was lower for Kmeans than KNN, this was the best we could achieve. After analyzing the confusion matrix, we do not see a pattern of misclassification aside from missing 5 cluster and still have a generally positive diagonal trend.

4) What distance metric should you use to compare user to each other? Given the k-nearest neighbors of a user, how can these k neighbors be used to estimate the rating for a movie that the user has not seen? In about one paragraph describe how you would implement a collaborative filter, or provide pseudo-code.

If we want to build a collaborative filte, we need to represent each user as a vector based on their interactions with items and movies, then based on those vectors of users, run a knn on the other people and find similar users. Based on their k-nearest neighbors, we will want to use a weighted average of items from the nearest users and, and from those results, recommend those with high interactions or ratings. We believe the most useful distance metric would be <u>cosine similarity</u> because it measures similarity rather than scalar distances, which is less relevant here.

Pseudocode:

```
Recommendations(target_user, k):
    all_users = All other users except target user
    similarity_ratings = []

    for user in all_users:
        similarity = cosim(target_user, user)
        add (user, similarity) to similarity_ratings

    k_nearest = Sort similarity_ratings based on HIGHEST value
    #cosim returns higher values for more similar

    recommendations = dict()
    for each item in possible_items(target_user):
        total_weights = 0
        total_similarity = 0
        for (neighbor, similarity) in k_nearest:
            if neighbor has rated(item) # see if the other person has viewed the current possible
item:
                total_weights += (neighbor's rating for item * similarity)
                total_similarity += similarity
            if total_similarity is not 0 #edge case where its the first item we are adding:
                rating = total_weights / similarity
                recommendations[item] = rating

    Return recommendations sorted based on rating in descending order
```