KNN and KMeans and SoftMax KMeans

Part 1 - Distance Metrics

Distance metrics can be found in distances.py and are being called while implementing KMeans and KNN. Metrics computed:

1. Euclidean distance
2. Cosine Similarity

(Comparisons against numpy implementation in main.py can be uncommented to test.)

Part 2 - KNN

Design Choices

1. Under the assumption this code will be used for the MNIST data set (features are greyscale images), we first divided all the feature values by 255, so that they now range between 0 and 1. This helps the computations run quicker. Since we know the min and max values for these features (0 and 255), we divided all values by 255 instead of using common scaling methods. Thus, all values in the test set will be guaranteed to be between 0 and 1. This is implemented in `scale_greyscale_features()`
2. We used PCA to reduce dimensionality as well. We determined from EDA on the training set that 75 was a good number of principal components to use, as it explained 88% of the variability in the training set.
   - See Appendix EDA - PCA on Training Set
3. To address ties, we created a default ordering of the labels in `set_default_label_ordering`. This gives preference to the labels that were most common in the overall data set. Second, if two labels show up with equal frequency, we arbitrarily choose the label that is larger (e.g. B > A) for consistency. This default label ordering is later used in the `get_mode_with_order()` function that returns the prediction for a sample based off its k-nearest neighbors. The `get_mode_with_order()` first checks for the mode, and if there is a tie, it chooses the mode that shows up first in the default ordering.
4. We gave the option to set the n_neighbors hyperparameter, but it can be set automatically as well (`KNearestNeighbor(n_neighbors=None)`). If it is unspecified, then the model will try a range of k's, and pick the one with the best accuracy (on the validation set). The range of k's is 2 and then up to three more evenly spaced integers between 3 and the square root of the number of samples in the training set. Our research suggested that the square root of the number of samples is an okay heuristic for choosing k, so we look for k's going up to that value. We limit to 3 more k's so that the algorithm doesn't take too long to fit.
   - Note: if auto-tuning is not used, the validation set is ignored in the main.py `run_KNN` function.
5. Note: When calculating cosine similarity, we stored 1 - the cosine similarity so that the process would be consistent with euclidean distance. In other words, with euclidean, we choose the k neighbors with the *smallest* distance values. By storing 1 - the cosine similarity, we can also choose the k neighbors with the *smallest* values.

KNN Results

In the main.py file, we demoed three different versions of the KNN.

1. KNN with auto-k selection using euclidean (best k=3)

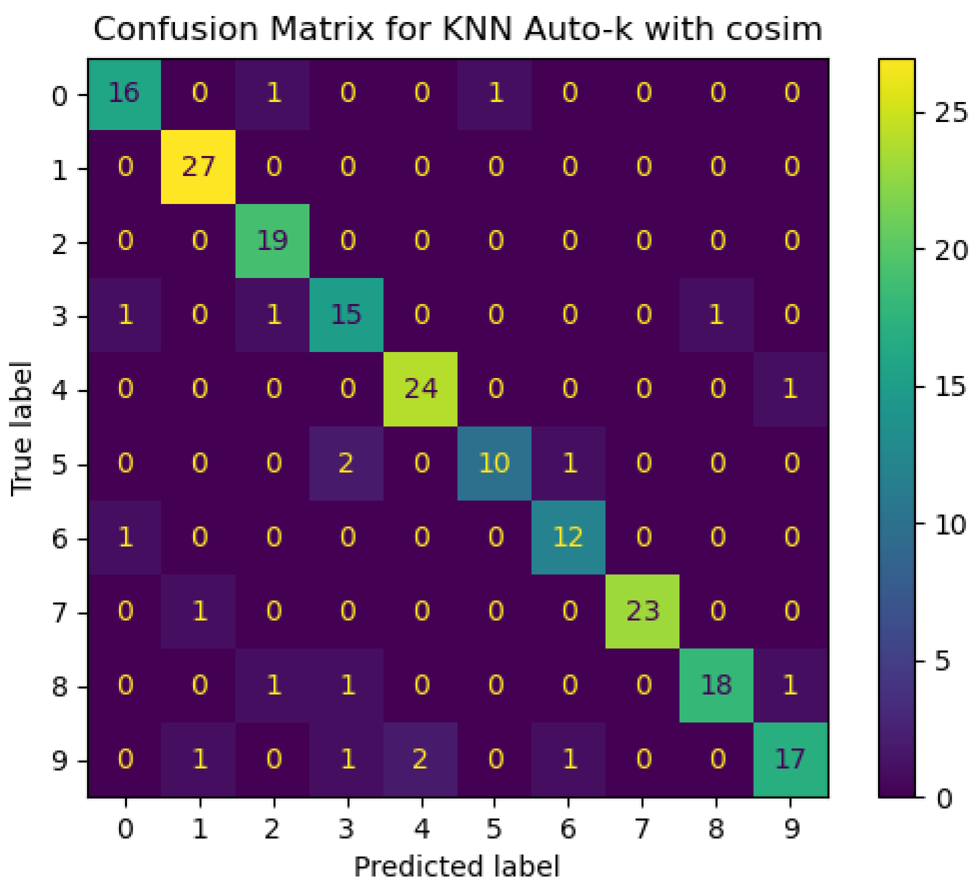
2. KNN with auto-k selection using cosim (best k=3)

3. KNN with k set to 2 using euclidean The second model gave the best overall accuracy (0.91) out the three, so we will analyze these results.

```
          ==========Classification Report==========
           precision    recall  f1-score   support

        0       0.89      0.89      0.89        18
        1       0.93      1.00      0.96        27
        2       0.86      1.00      0.93        19
        3       0.79      0.83      0.81        18
        4       0.92      0.96      0.94        25
        5       0.91      0.77      0.83        13
        6       0.86      0.92      0.89        13
        7       1.00      0.96      0.98        24
        8       0.95      0.86      0.90        21
        9       0.89      0.77      0.83        22
```

accuracy 0.91 200 macro avg 0.90 0.90 0.90 200 weighted avg 0.91 0.91 0.90 200



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	16	0	1	0	0	1	0	0	0	0
1	0	27	0	0	0	0	0	0	0	0
2	0	0	19	0	0	0	0	0	0	0
3	1	0	1	15	0	0	0	0	1	0
4	0	0	0	0	24	0	0	0	0	1
5	0	0	0	2	0	10	1	0	0	0
6	1	0	0	0	0	0	12	0	0	0
7	0	1	0	0	0	0	0	23	0	0
8	0	0	1	1	0	0	0	0	18	1
9	0	1	0	1	2	0	1	0	0	17

KNN Results Analysis

Overall, the f1-scores for each of the number images ranged from 0.81 to 0.98.

Recall We witness some issues with recall for 5 and 9 (both had low recall of 0.77).

- This means that the ability to properly identify true 5's and 9's was difficult (high false negatives).
- Out of 13 real 5's in the test set, 2 of them were misclassified as a 3 and one as a 6
- Out of 22 9's in the test set, 5 of them were misclassified as either 1, 3, 4, or 6

1's and 2's had recall of 1. This means all the real 1's and 2's were predicted as 1's and 2's, respectively. This might make sense since 1's and 2's are very simple to draw as they single lines

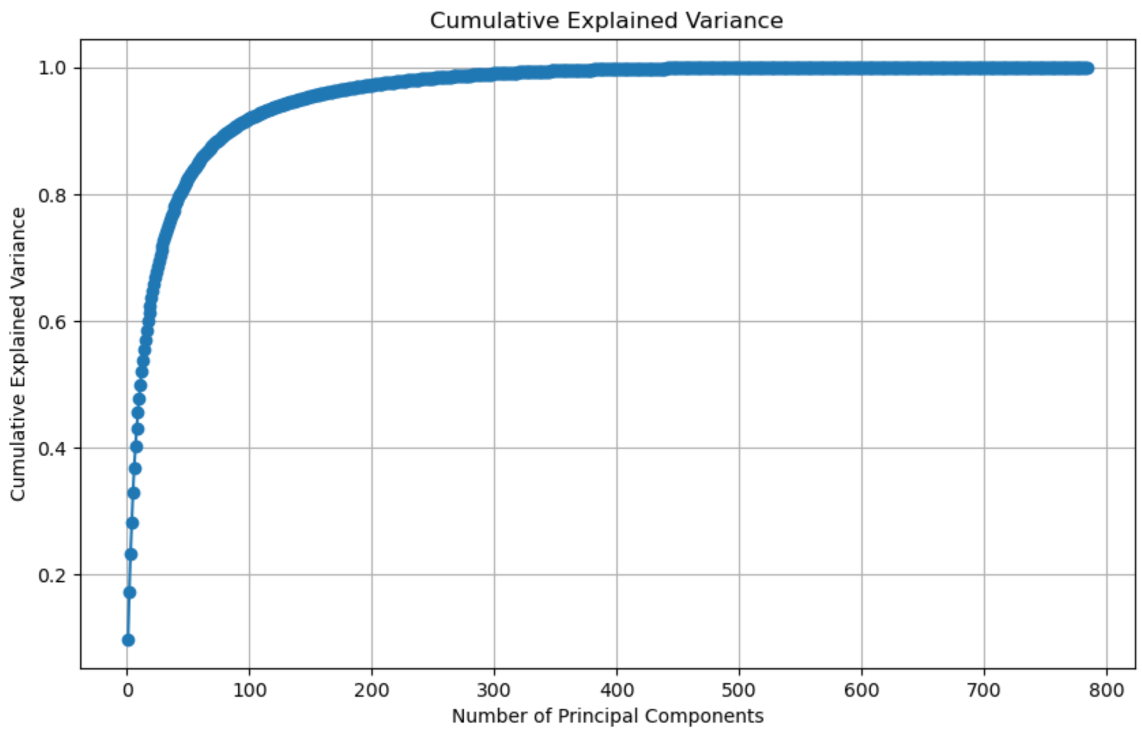
Precision The number with the lowest precision was 3 (precision=0.79).

- This means other numbers were predicted as 3s at a high rate
- Two real 5's were predicted to be 3s, and real 8 and real 9 were predicted as a 3

The number 7 had a perfect precision of 1, so no other number was predicted as a 7.

Appendix

Exploratory Data Analysis - PCA on Training Set



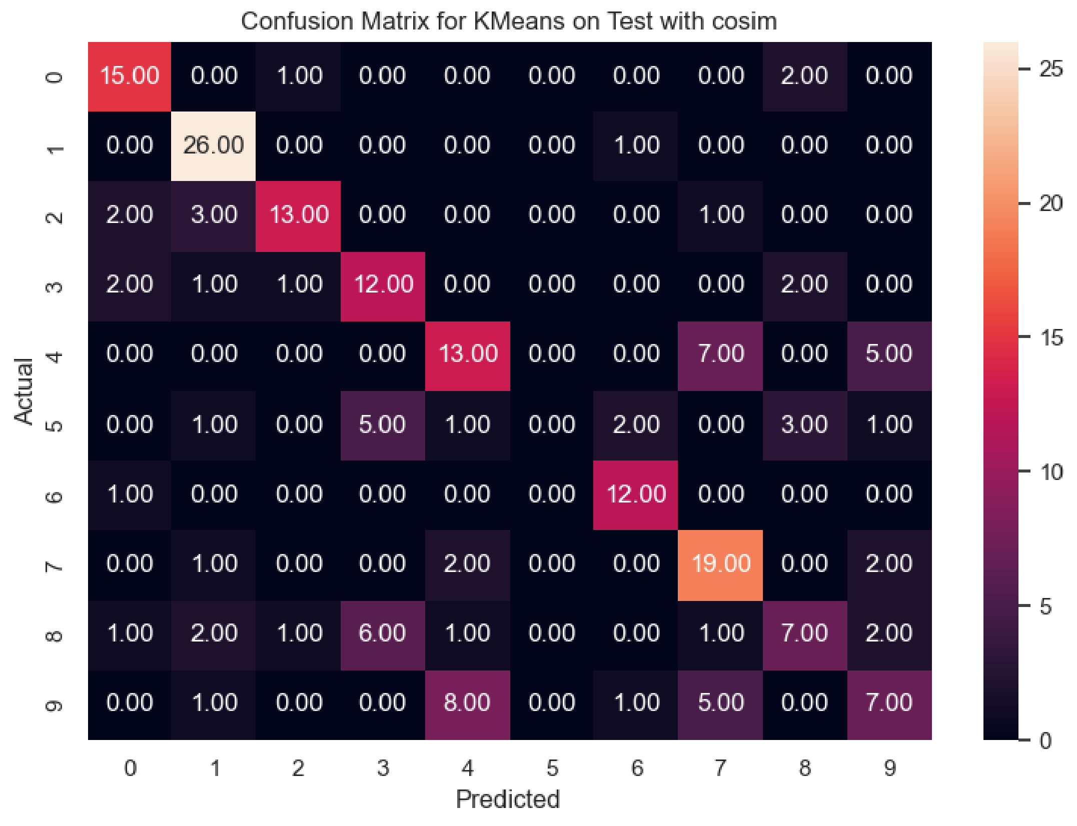
75 principal components explain about 88% of the variability in the training set.

Part 3 - KMeans

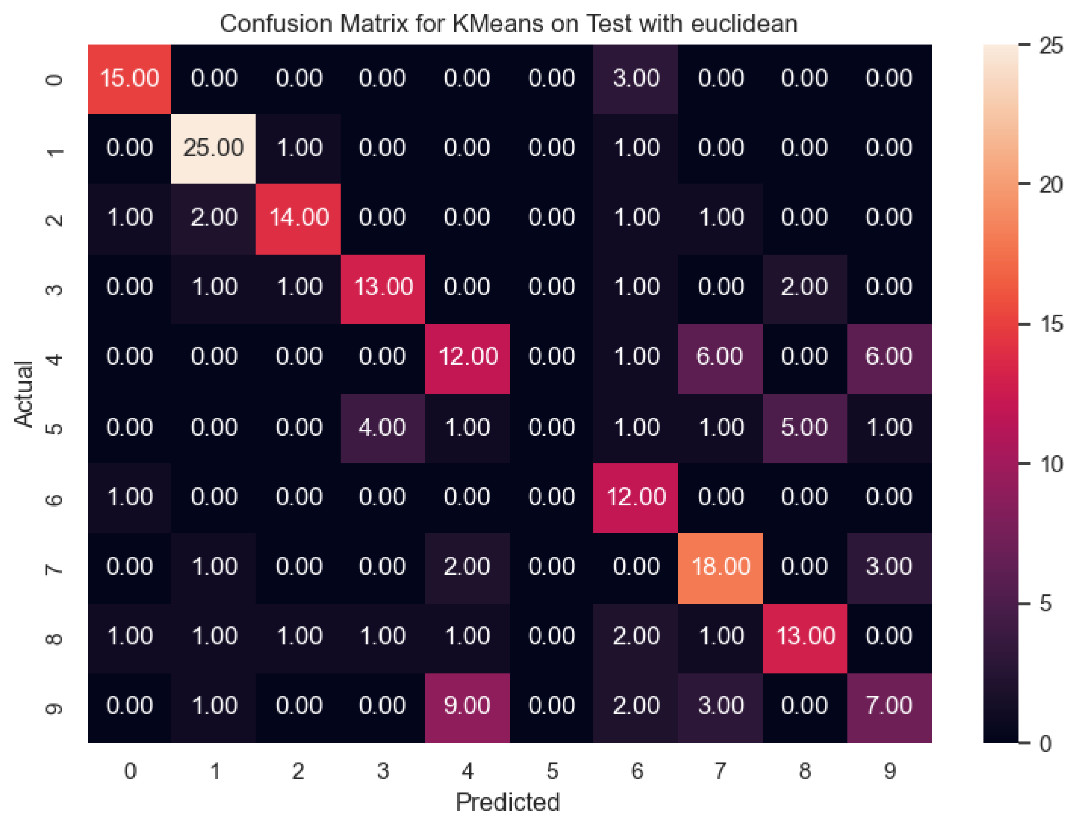
Design Choices

1. Under the assumption that it is an MNIST dataset, we have taken a default of k = 10 clusters to correspond with the numbers 0-9. Increasing the value of k improves the accuracy but since the underlying assumption was MNIST with 10 clusters, we have kept that constant
2. Hyperparameter = max_iter to ensure that the algorithm converges and time complexity is managed by this.
3. Scaling = minmax scaling which seemed to boost the performance of the clusters by increasing the accuracy when tested with the true labels (Other scaling methods tested - dividing the greyscale values by 255 to obtain them in a range of 0-1).
4. PCA (Principal Component Analysis) = To expedite convergence due to the extensive dimensionality of our distance metrics computation (x * 784), we employed PCA (Principal Component Analysis). This reduced the dimensionality, leading to a substantial decrease in clustering time. For KNN, we chose to use enough principal components to explain 95% of the variability in the training set.
5. Transform cosine similarity range = We transformed cosine similarity to a 1-cosim value to make it a more intuitive distance metric for obtaining clusters.
6. Performing validation = The classification accuracies are obtained using the true labels provided in the dataset. In essence, the logic of using the most frequent occurrence (the mode) in the validation process allows the code to assess how effectively the KMeans clustering algorithm has grouped data points of the same actual label together. If the clusters largely consist of data points with the same actual label, then the accuracy will be high.
7. Empty cluster handling = if for a particular iteration a cluster is empty i.e. no samples were assigned during training, we reinitialize the mean of the empty cluster to a random data point
8. Convergence criteria = converge if centroids are not different in 2 iterations.

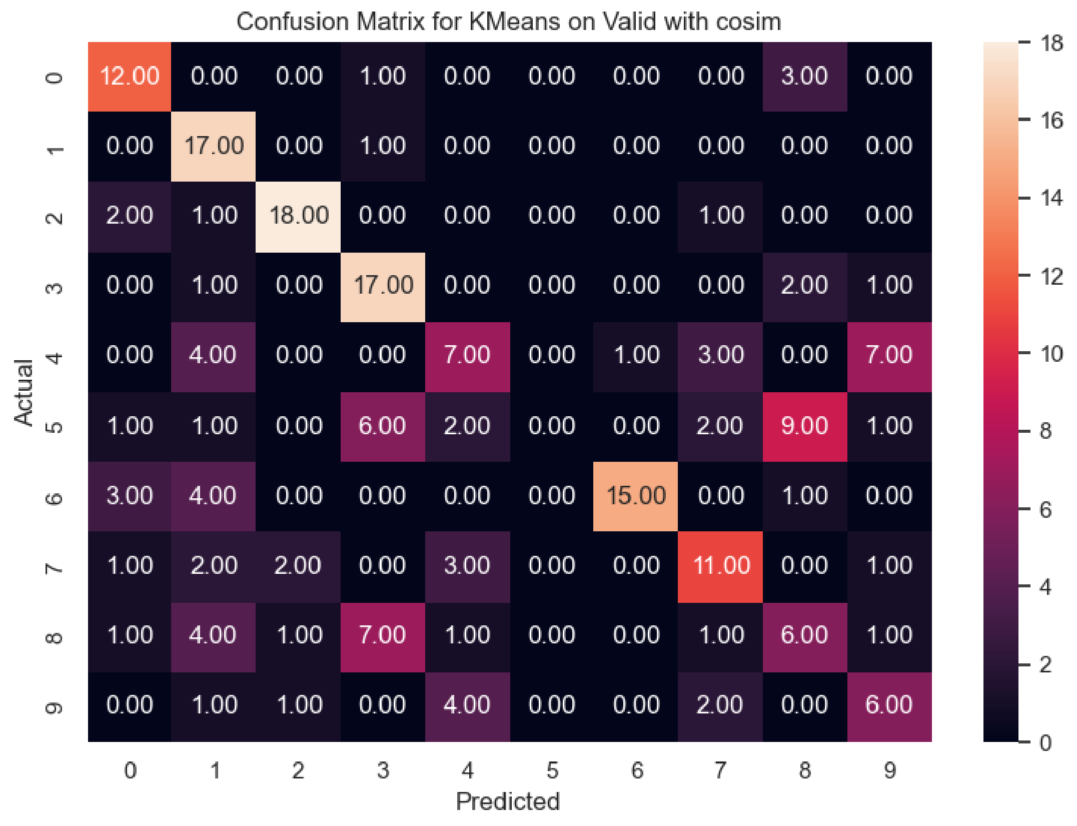
KMeans Results



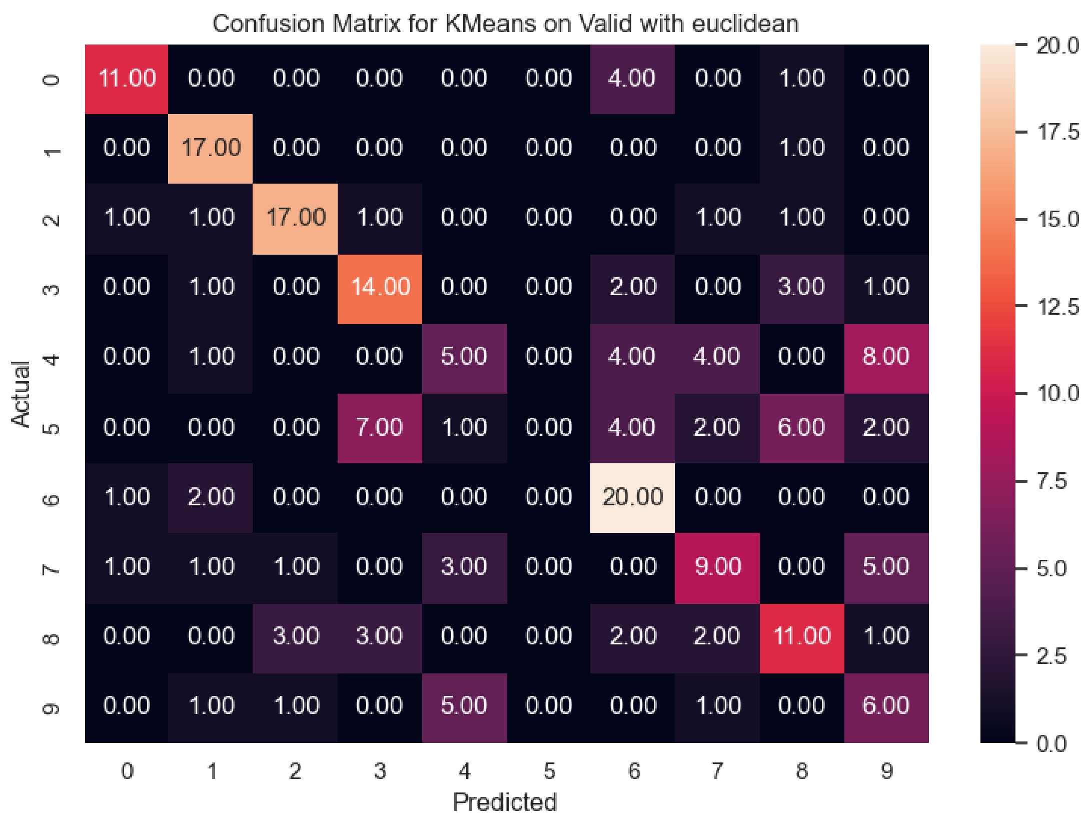
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	15.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00
1	0.00	26.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
2	2.00	3.00	13.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	2.00	1.00	1.00	12.00	0.00	0.00	0.00	0.00	2.00	0.00
4	0.00	0.00	0.00	0.00	13.00	0.00	0.00	7.00	0.00	5.00
5	0.00	1.00	0.00	5.00	1.00	0.00	2.00	0.00	3.00	1.00
6	1.00	0.00	0.00	0.00	0.00	0.00	12.00	0.00	0.00	0.00
7	0.00	1.00	0.00	0.00	2.00	0.00	0.00	19.00	0.00	2.00
8	1.00	2.00	1.00	6.00	1.00	0.00	0.00	1.00	7.00	2.00
9	0.00	1.00	0.00	0.00	8.00	0.00	1.00	5.00	0.00	7.00



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	15.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00	0.00
1	0.00	25.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
2	1.00	2.00	14.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00
3	0.00	1.00	1.00	13.00	0.00	0.00	1.00	0.00	2.00	0.00
4	0.00	0.00	0.00	0.00	12.00	0.00	1.00	6.00	0.00	6.00
5	0.00	0.00	0.00	4.00	1.00	0.00	1.00	1.00	5.00	1.00
6	1.00	0.00	0.00	0.00	0.00	0.00	12.00	0.00	0.00	0.00
7	0.00	1.00	0.00	0.00	2.00	0.00	0.00	18.00	0.00	3.00
8	1.00	1.00	1.00	1.00	1.00	0.00	2.00	1.00	13.00	0.00
9	0.00	1.00	0.00	0.00	9.00	0.00	2.00	3.00	0.00	7.00



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	12.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	3.00	0.00
1	0.00	17.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
2	2.00	1.00	18.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	0.00	1.00	0.00	17.00	0.00	0.00	0.00	0.00	2.00	1.00
4	0.00	4.00	0.00	0.00	7.00	0.00	1.00	3.00	0.00	7.00
5	1.00	1.00	0.00	6.00	2.00	0.00	0.00	2.00	9.00	1.00
6	3.00	4.00	0.00	0.00	0.00	0.00	15.00	0.00	1.00	0.00
7	1.00	2.00	2.00	0.00	3.00	0.00	0.00	11.00	0.00	1.00
8	1.00	4.00	1.00	7.00	1.00	0.00	0.00	1.00	6.00	1.00
9	0.00	1.00	1.00	0.00	4.00	0.00	0.00	2.00	0.00	6.00



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	11.00	0.00	0.00	0.00	0.00	0.00	4.00	0.00	1.00	0.00
1	0.00	17.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	1.00	1.00	17.00	1.00	0.00	0.00	0.00	1.00	1.00	0.00
3	0.00	1.00	0.00	14.00	0.00	0.00	2.00	0.00	3.00	1.00
4	0.00	1.00	0.00	0.00	5.00	0.00	4.00	4.00	0.00	8.00
5	0.00	0.00	0.00	7.00	1.00	0.00	4.00	2.00	6.00	2.00
6	1.00	2.00	0.00	0.00	0.00	0.00	20.00	0.00	0.00	0.00
7	1.00	1.00	1.00	0.00	3.00	0.00	0.00	9.00	0.00	5.00
8	0.00	0.00	3.00	3.00	0.00	0.00	2.00	2.00	11.00	1.00
9	0.00	1.00	1.00	0.00	5.00	0.00	0.00	1.00	0.00	6.00

KMeans Results Analysis

1. After generating the confusion matrices on test and valid with both matrices, certain clusters seem to have high true positive counts i.e. they have been correctly classified.
2. The general precision across clusters seems to be fairly high as evident by the classification reports.
3. For the confusion matrix generated on test set with euclidean, we have notable correct predictions (with relatively high values) for classes 0 (15 correct predictions), 1 (25 correct predictions), 4 (12 correct predictions), 7 (18 correct predictions), and 8 (13 correct predictions).
4. However, there are certain misclassifications made, which might be due to the random initialization of the means when the initial clustering happens. Since this is unsupervised learning, the model doesn't necessarily learn the most meaningful clusters, but the ones that emerge empirically.
5. Cluster 5 in particular does not capture any true positives. Clusters 5 seems to have been classified majorly into 8 and 3.
6. Overall accuracy : while precision and recall is higher for most classes, the overall accuracy is brought down by the misclassifications.
7. Distance metric: Euclidean gave a slightly better accuracy than the transformed cosine distance metric for Kmeans.
8. Consistency : We see a consistency in classifications across the test and validation sets and with both distance metrics.
9. Kmeans: while the majority of predictions lie on the main diagonal, which suggests that the classifier does have a reasonable performance for several classes, but there's still room for improvement.

Part 4 - Collaborative Filter

Step 1: Data representation We assume the data for the problem has rows of users and columns of movies. The values in a cell would represent the rating that the user gave the movie. This would be a very sparse matrix.

If users rated a movie on a scale of 1-4, based off domain knowledge, we might say they loved the movie if they rated it a 4, liked it if they rated it a 3, were indifferent to it if they rated it a 2, and didn't like it if they rated it a 1.

Step 2: Distance Metric For this problem, we care about directionality (liked or disliked the movie) and magnitude (love or liked movie). If we just cared about directionality, cosine similarity would be a good choice, as it would measure the angles between users' movie rating vectors. But since we care about the magnitude as well, the Pearson correlation coefficient should be used. This similarity metric captures both directionality and magnitude.

Step 3: K-nearest neighbors Using the pearson correlation coefficient as the distance metric, the similarities of a user with each other user is calculated. Then we sort the most similar users in descending order. The top k users will then be selected. These are the users that have the most similar preferences to the target user. The determination of the value of k is the most important step. A higher K will capture more diverse preferences but may also introduce noise, while a lower K may lead to more personalized recommendations but could be sensitive to outliers.

Pseudocode

- Movie recommender that determines the best n movie recommendations from k nearest neighbors.
- Since this data is sparse, we make sure to use a weighted rating (neighbor movie ranking weighted by similarity) to rank potential movie rankings, instead of voting for example.

```
def get_recommendations(user, k_neighbors, n_recs):

    potential_recs_ratings = {}
    potential_recs_sims = {} # similarities


    # get top k users sorted by pearson
    neighbors = get_other_users_sorted_by_pearson(user)[0:k_neighbors]

    for neighbor in neighbors:

        # make sure user is actually similar to top k most similar users (pearson_threshold should be at least 0)
        if pearson(user, neighbor) > pearson_threshold:

            for movie in movie_list(neighbor):
                if user has not seen movie:
                    # keep track of the neighbor ratings of this movie and the similarity between neighbor and user
                    potential_recs_ratings[movie] += [neighbor_rating(movie)]
                    potential_recs_sims[movie] += [pearson(user, neighbor)]



    # get average rating across neighbors (not weighted by similarity)
    ratings = get_average(potential_recs_ratings)

    # filter for movies with average ranking above 3 ("like"), since don't want to recommend movies user won't like
    filter(ratings) where rating >= 3

    # weight the ratings by the similarity scores
    recs = weight(ratings, potential_recs_sims)

    # sort recommendations by weighted ratings
    sort(recs)

    # return top n recommendations
    return recs[0:n_recs]
```

Part 5 - SoftMax KMeans

Design Choices

1. Similar to the KMeans in step 3, we assume an MNIST dataset, we have taken a default of 10 clusters.
2. Hyperparameters: a. max_iter = ensures that the algorithm converges. b. Beta = A second hyperparameter of beta (temperature)is added for which we tune the model to a range of values and test the classification accuracies. A value of beta generally lower than 1 gives better performance when assessed by accuracies.
3. Tolerance = We have set a value of tolerance = 1e-4, that determines the stopping condition for the algorithm based on how much the cluster centroids changes between iterations.
4. Label assignment = Though soft kmeans uses a probabilistic distribution to fit the labels, the prediction part assigns hard labels and this is common practise which we have taken into consideration while assigning the clusters and testing it out.
5. Dimensionality reduction = No dimensionality reduction was performed for this data as it dropped the performance of the algorithm. The scaling is done in the same way as Kmeans using a minmax scaling technique.
6. Perform validation = The classification accuracies are obtained in the same way as step 3.

Softmax Analysis

1. Distance metric - cosine similarity works as a much more effective metric for the soft kmeans

2. While the precision is higher for certain clusters, the low values of precision for certain clusters is bringing down the overall accuracy.
3. Beta - values of beta much lesser than 1 are effective in clustering.
4. Higher recall values for soft k means is observed. When we say "higher recall", it means that a larger proportion of the actual positive cases were identified correctly.
5. Inconsistency is observed in the classifications which signifies room for improvement since no definitive clusters seem to be underperforming.