MSAI 349- MACHINE LEARNING

GROUP 7 - HW2

By *Clarissa Cheam* *Tianchang Li* *Ayushi Mishra* *Geethanjali Vasudevan*

---

1. Implement functions to compute Euclidean distance and Cosine Similarity between two input vectors ***a*** and ***b***. These functions should return a scalar float value. To ensure that your functions are implemented correctly, you may want to construct test cases and compare against results packages like numpy or sklearn.

   Ans: The code for the Euclidean and Cosine Similarity distance function is as follows:

```python
# returns Euclidean distance between vectors a dn b
def euclidean(a, b):
    # print('a:', a)
    # print('b:', b)
    # a = list(map(int, a))
    # b = list(map(int, b))
    summ = sum((u - v) ** 2 for u, v in zip(a, b))
    dist = math.sqrt(float(summ))
    return dist


# returns Cosine Similarity between vectors a dn b
def cosim(a, b):
    # print('a:', a)
    # print('b:', b)
    # a = list(map(int, a))
    # b = list(map(int, b))
    dot = sum(u * v for u, v in zip(a, b))
    mag_a = math.sqrt(sum(x ** 2 for x in a))
    mag_b = math.sqrt(sum(x ** 2 for x in b))
    dist = float(dot) / (mag_a * mag_b)
    return dist
```

   The code for test case comparison of the results between our created distance function and that of the packages (sklearn or numpy) is as follows:

```
def unit_test(a, b):
    euc = euclidean(a, b)
    cos = cosim(a, b)
    from sklearn.metrics.pairwise import euclidean_distances
    from sklearn.metrics.pairwise import cosine_similarity
    sk_euc = euclidean_distances([a], [b])
    print('Euclidean match sk-learn') if sk_euc == euc else 'Euclidean doesnt match'
    sk_cos = cosine_similarity([a], [b])
    print('Cos similarity match sk-learn') if sk_cos == cos else 'Cos similarity doesnt match'
```

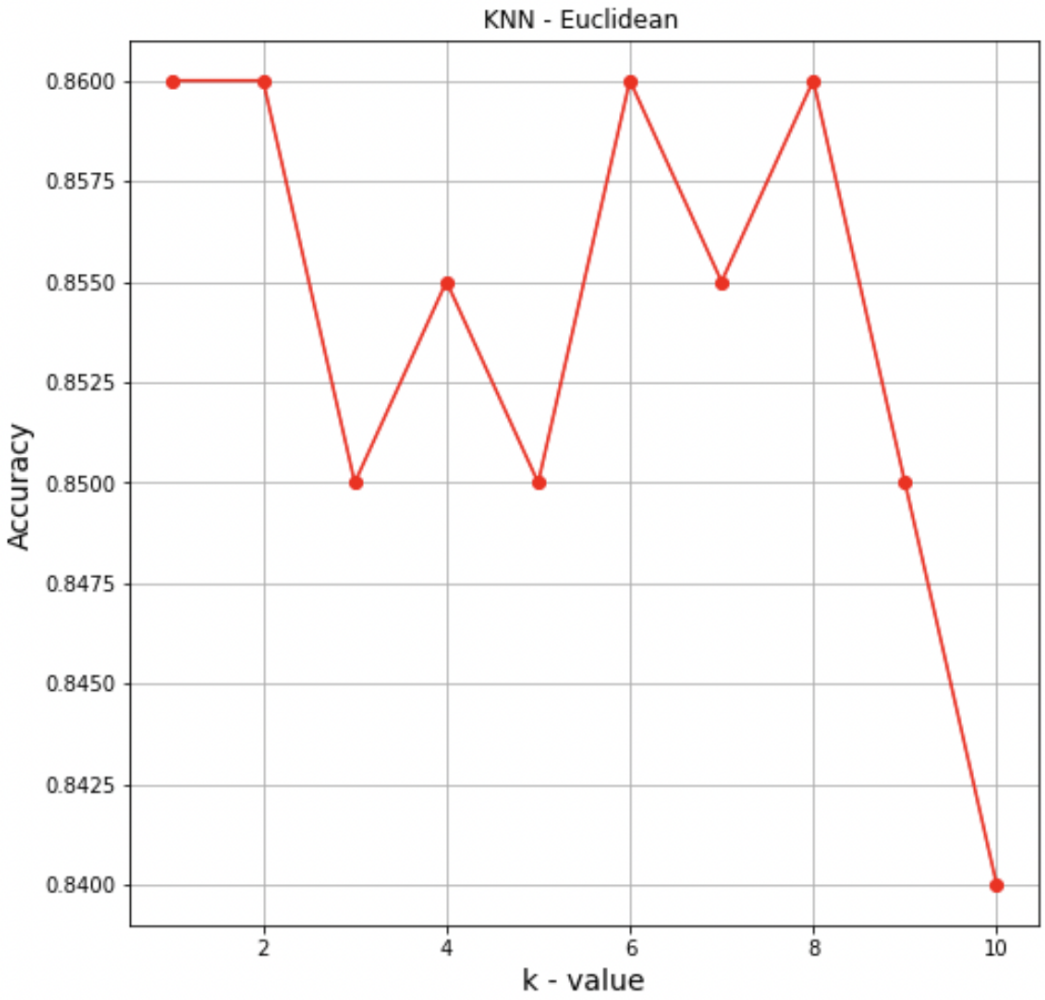
2. Implement a k-nearest neighbors classifier for both Euclidean distance and Cosine Similarity using the signature provided in starter.py. This algorithm may be computationally intensive. To address this, you must transform your data in some manner (*e.g.,* dimensionality reduction, mapping grayscale to binary, dimension scaling, etc.) -- the exact method is up to you. This is an opportunity to be creative with feature construction. Similarly, you are free to select your own hyper-parameters (e.g., K, the number of observations to use, default labels, etc.). Please describe all of your design choices and hyper-parameter selections in a paragraph. Once you are satisfied with performance on the validation set, run your classifier on the test set and summarize results in a 10x10 confusion matrix. Analyze your results in another paragraph.

   Ans: We discuss our design approach, design choices, hyper parameters and results for k-nearest neighbors as follows-

   KNN design approach: The KNN classifier was implemented by calculating the distance between query points and each training point. The distances are stored in a tuple with the labels of the training points. For each query point, the tuple is sorted and the top k elements as the k nearest neighbors to the query point are selected. A majority vote on their labels determines the prediction of the query point.

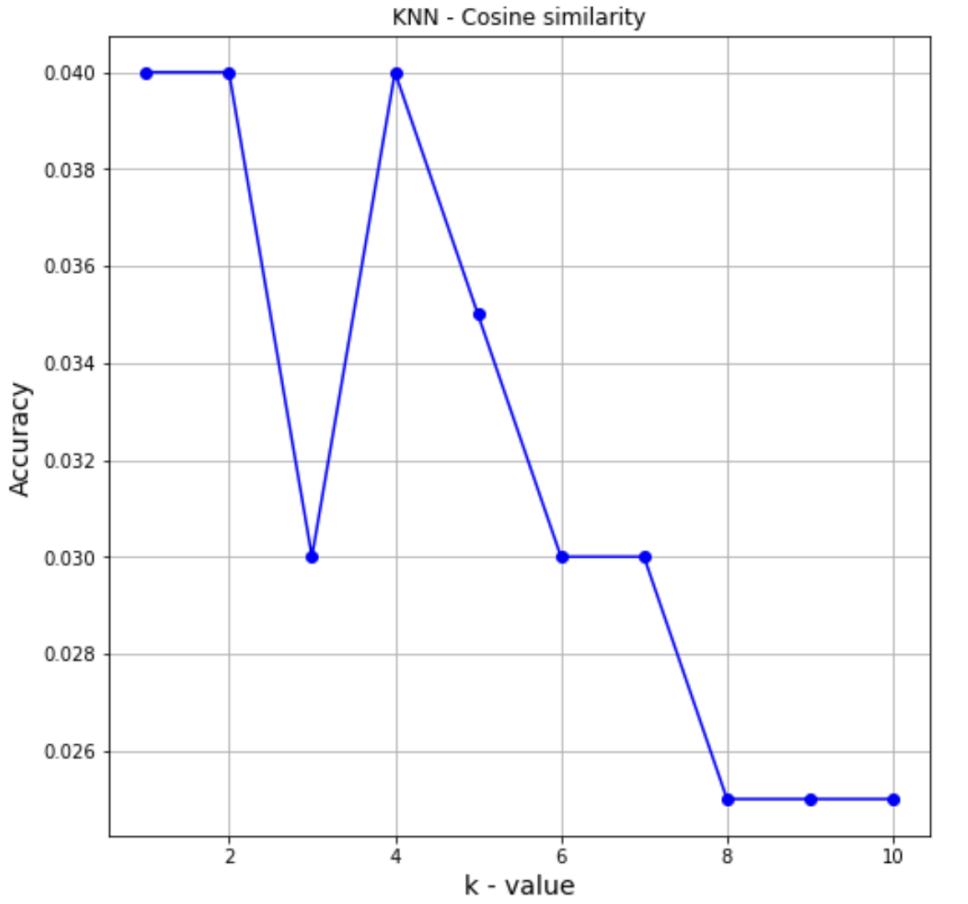
   Design choice w.r.t. Dimension reduction: We tried using PCA for dimensionality reduction. PCA was not able to give enough information to identify the digits. So, we decided to use the entire dataset without any feature engineering.

Hyper parameters: The hyper parameters used for k-nearest neighbors are the k value and the distance metric function (Euclidean or Cosine Similarity). We have used k values through 1 to 10 with an increment of 1 for both distance function and following are the results we found:



As we can see for the Euclidean distance metric after k=8 the accuracy value for our k-nearest neighbors model starts dipping consistently so we chose k=8 as the optimum value for our model.

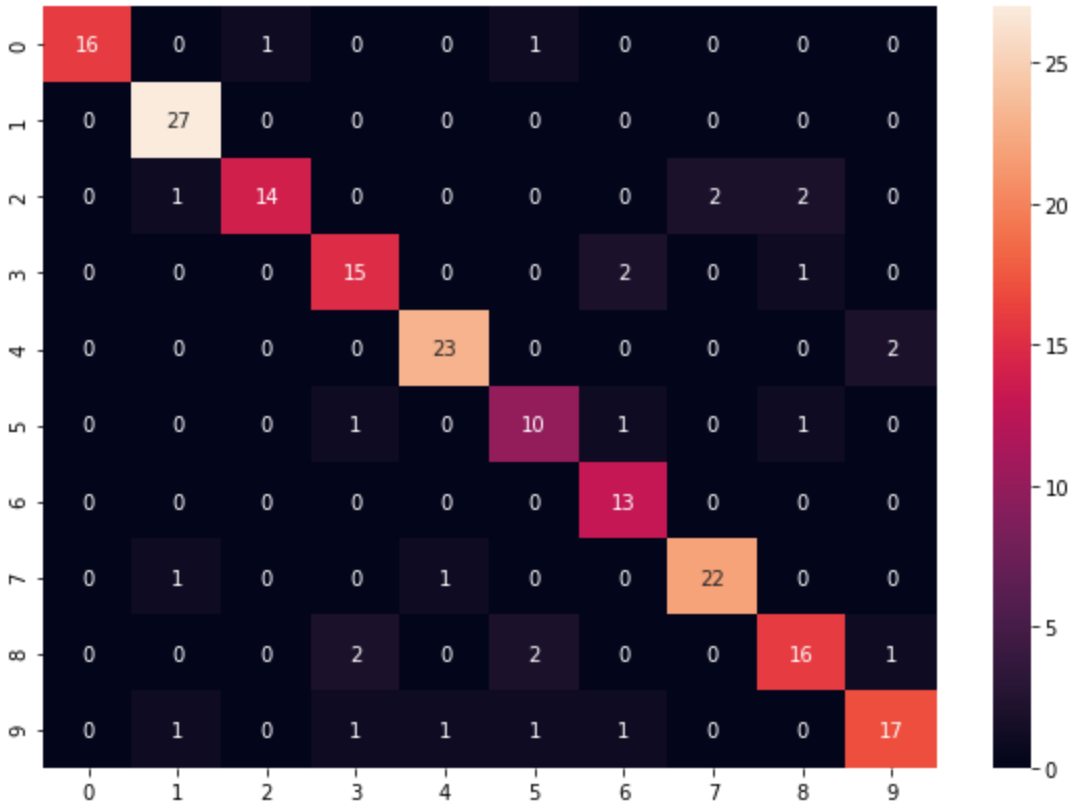
Now lets evaluate the accuracy of our model for various values of k using the Cosine Similarity distance metric.



As we can see for the Cosine Similarity distance metric after k=7 the accuracy value for our k-nearest neighbors model starts dipping consistently and the highest accuracy can be seen at k=4, so we chose k=4 as the optimum value for our model.

However, when comparing both distance metrics we find that overall the accuracy of the Euclidean distance metric is almost twice the accuracy of Cosine Similarity distance metric. So using the Euclidean distance as our metric hyper parameter would be better. And as we have concluded earlier k=8 is a good choice of k hyper parameter for Euclidean distance metric knn model choice in terms of accuracy.

Results using Confusion matrix: Now that we are satisfied with our results and have selected our hyper parameters we plot the confusion matrix on our test set.



Confusion matrix of Knn - Euclidean for k=8

As we can see from the confusion matrix, the labels are mostly correctly classified as themselves (the coloured diagonal of the confusion matrix) with a few misclassifications here and there (the blackish area with 0,1,2 numbers filled representing the count of misclassification). Please note that the y axis represents the actual expected label value and the x axis is the model predicted label value.

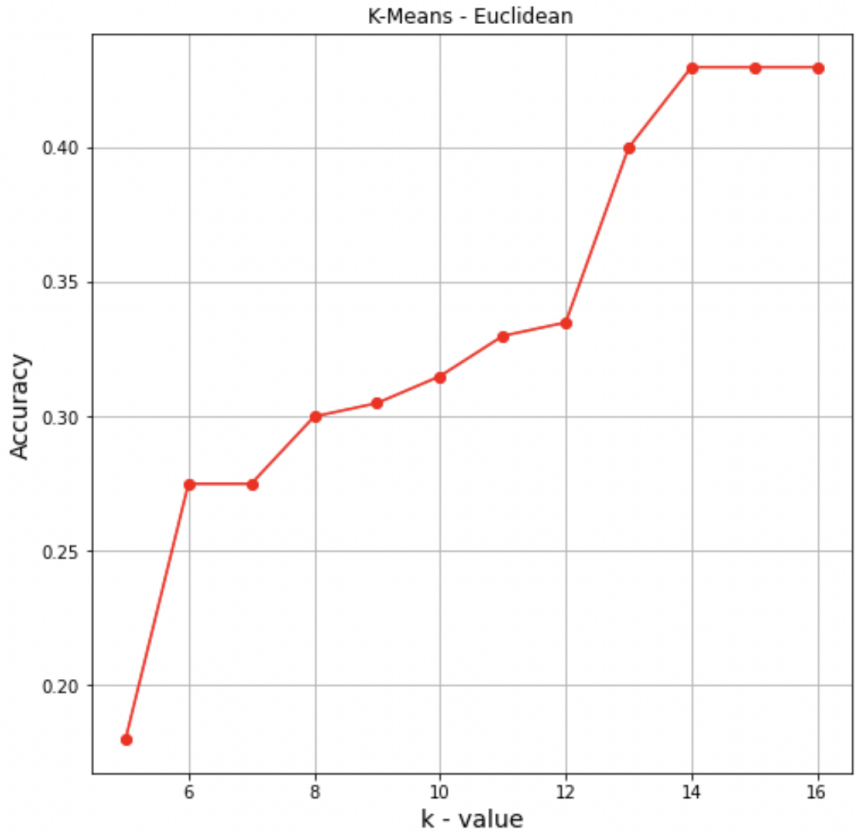
3. Implement a k-means classifier in the same manner as described above for the k-nearest neighbors classifier. The labels should be ignored when training your k-means classifier. Describe your design choices and analyze your results in about one paragraph each.

   Ans: We discuss our design approach, design choices and results for k-means classifier as follows-

   Design approach: Our k-means classifier was implemented by iteratively updating the current centroids by averaging the training points that are clustered around the centroids.

The initial k centroids are randomly selected. Each training point is assigned to the cluster whose centroid is the closest to it. The way our algorithm keeps track of the assignments is through a dictionary with the key being the index of centroids and the values being the stacked data points. When all points are assigned, new centroids are calculated as the average point and assigned as current centroids. Two types of convergence checks are implemented, including the maximum number of iterations (=20) and the tolerance value (when updates on all centroids are smaller than this).

Hyper parameters: The hyper parameters used for k-means are the k value, tolerance value (discussed above), maximum number of iterations (discussed above) and the distance metric function (Euclidean or Cosine Similarity). We have used values of k from 5 to 16 with an increment of 1 for both distance function and following are the results we found:



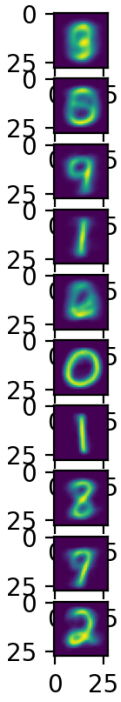
So we can clearly see that from and beyond k=14 the k-means classifier using the Euclidean distance metric gives the best accuracy before plateauing. So using k=14

would be the most optimal choice for our k-means classifier model if we were to use the Euclidean distance metric.

Our model performed very poorly for Cosine Singularity distance metric so we chose to forgo that distance metric entirely since varying k values didn't change anything and computation wasn't being done correctly.

Design choices: So we have used the tolerance value as 0.001 which is a standard practice and maximum number of iterations as 20 for reducing computational time and quick convergence of label values. We tried using PCA for dimensionality reduction. PCA was not able to give enough information to identify the digits. So, we decided to use the entire dataset without any feature engineering.

Results: Our k-means model at this stage gives us the output as centroids predicted for our data points. It looked something as follows-



centroids - k-means for k=10

We visualized these centroids and manually input the label predictions for each cluster. The accuracy was then using code calculated comparing these predictions with the true labels. We also noticed that it is especially hard for the algorithm to distinguish between ‘0’ ,‘8’, ‘3’. This showcases the limitation of our k -means classifier model for the given dataset.

4. Collaborative filters are essentially how recommendation algorithms work on sites like Amazon ("people who bought blank also bought blank") and Netflix ("you watched blank, so you might also like blank"). They work by comparing distances between users. If two users are similar, then items that one user has seen and liked but the other hasn't seen are recommended to the other user. What distance metric should you use to compare users 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.

   Ans: We would use cosine similarity as the distance metric for our k-nearest neighbor to implement the collaborative filter. First, we would calculate the similarity between our target user and all other users who have watched the movies that the user hasn’t watched to find the k nearest neighbors. We can then use either mode or mean of the user ratings of the k nearest neighbors to predict how our target user might rate a movie.

5. Implement a soft k-means classifier in the same manner as described above for the k-nearest neighbors classifier. Describe your design choices and analyze your results in about one paragraph each.

   Ans: The design approach, design choices, hyper parameters and results for the soft k-means classifier is discussed as follows-

   Design approach: For soft k-means, the initial centers were chosen at random. This is followed by cluster identifications, using the softmax formula, by inputting the centers calculated, the beta value and the data points and then calculating the matrix norm difference between centers and training data points, calculating the exponential term and

finding the total sum and then calculating the final resultant that is the code equivalent of the mathematical softmax function . After this the center points are iteratively updated and the cost function of this update is then calculated. If the difference between the previous cost function and the cost function after updated center points is greater than the set difference values (1e-5) we break out of the loop of updating the center points since we have obtained our most optimum classification results.

Hyper parameters: The hyper parameters used for soft k-means classifiers are k value, beta value, maximum number of iterations and cost difference value for break condition for cluster label updates.

Design choices: So we have used the beta value as 1. which is a standard practice and maximum number of iterations as 20 for reducing computational time and quick convergence of label values. As a starting choice k=3 was chosen at random and then various k values were used to run the model. The cost difference was set to (1e-5) used as a standard practice when implementing a soft k-means classifier. We tried using PCA for dimensionality reduction. PCA was not able to give enough information to identify the digits. So, we decided to use the entire dataset without any feature engineering.

Results: The output we receive at this stage is the classification probabilities of each training data point for different classes that we are trying the data point to classify into. Here we pick the maximum probability value as the classified class for a particular training data point. This is then made to compare with our expected results and then accuracy is calculated. For the same training data the soft k-means classifier is able to classify the uncertain data points (that lies on the borderline of two different classes) with greater confidence.