Cem Daloğlu
21602161 / 10.05.2021

EEE 361 Homework-5 Report

In this homework report, performance of different kernel methods for k-means clustering will be discussed.

Kernel based k-means algorithm can be applied to identify the number of classes since each digit would have similar values at similar pixels. Therefore, it is expected to see the loss function to lessen as number of classifiers increase and see a drop after number of classes reach 10 because there are 10 different digits.

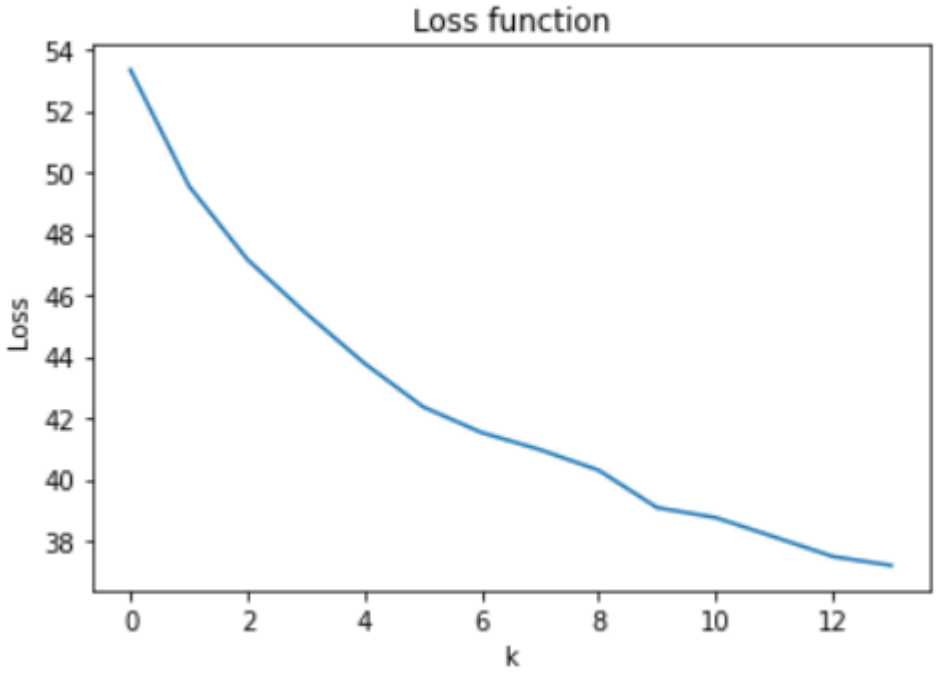

Figure 1: Loss function

The loss function shown in the Figure 1 is calculated by addition of the train images distance to the centroid data. As it can be seen in the Figure 1, the loss graph did not result as expected, we did not see a sudden drop after k=10. The Figure 1 tells that, classification of digits is not complete and convergence is not complete. The reason behind that plot will be discussed in further parts of the report.

Then, for k=10 k-means algorithm implemented on the given dataset. By trial and error, hyper parameters are tuned as the following;

$$\boldsymbol{Polynomial}: c = 1, d = 2$$
$$\boldsymbol{Gaussian}: \sigma = 13$$
$$\boldsymbol{Sigmoid}: c = 0.1, \theta = -8$$

Iteration numbers for 10 thousand training data are found as the following;

$$\boldsymbol{Polynomial}: 36$$
$$\boldsymbol{Gaussian}: 38$$
$$\boldsymbol{Sigmoid}: +100$$

Iterations are interrupted if convergence of the equation (18) in page 253 achieved. For the *Polynomial* and *Gaussian* kernel methods convergence achieved for less than 100 iterations. However, for the *Sigmoid* kernel method convergence could not obtained.

Centroids of the *Polynomial* kernel is found as in Figure 2;

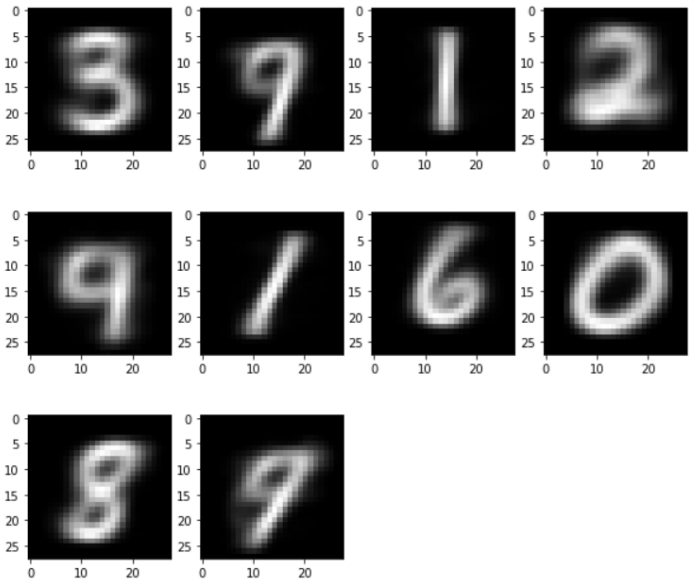

Figure 2: Centroids obtained from Polynomial kernel

It is difficult to see but the second element in the Figure 2 is 4 and the sixth element is 7. The digit 5 is missing and the reason of the result we obtained in Figure 1 is these missing digits. There are unclassified images so that the loss function did not result as expected.

Centroids of the *Gaussian* kernel is found as in Figure 3;

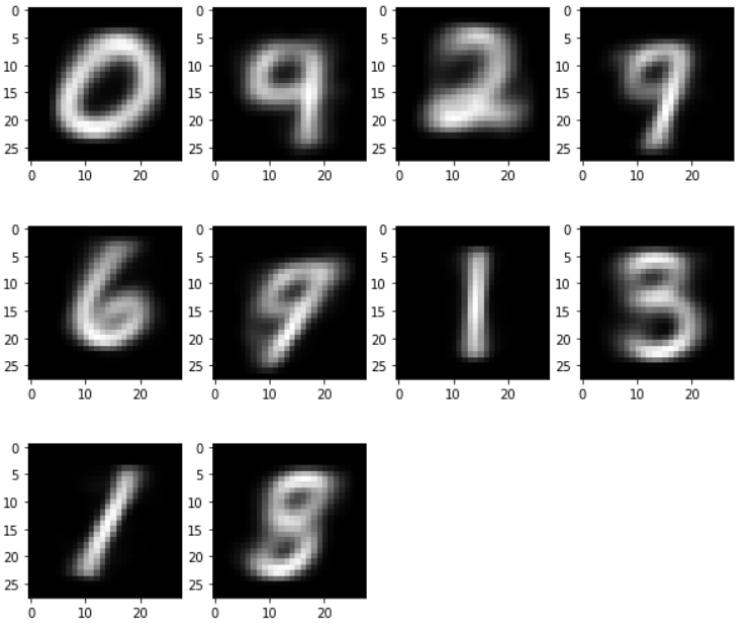

Figure 3: Centroids obtained from Gaussian kernel

Centroids of the *Sigmoid* kernel is found as in Figure 4;

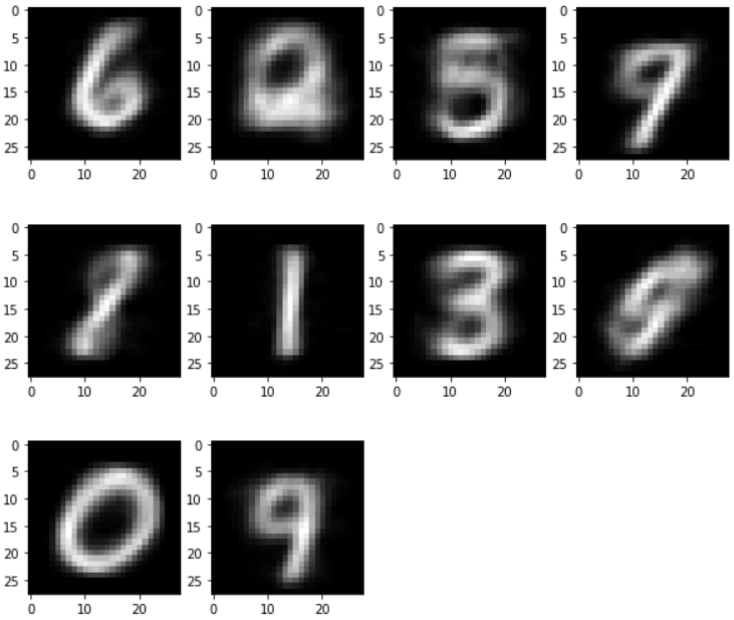

Figure 4: Centroids obtained from Sigmoid kernel

As it is mentioned before, hyper parameters are tuned by trial and error and to compare different parameters, centroid results are checked. In the Figure 5, an example of a non-converged kernel algorithm can be seen;

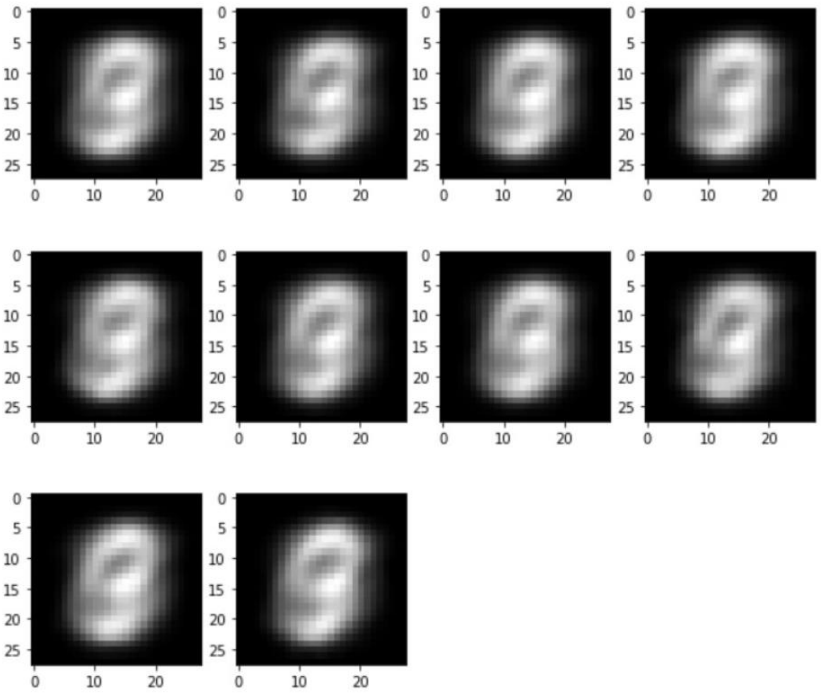

Figure 5: Centroids obtained for non-converged kernel

Then, obtained classifiers are applied to the test data and the following results in Figures 6, 7, 8, 9, 10, and 11 are obtained.

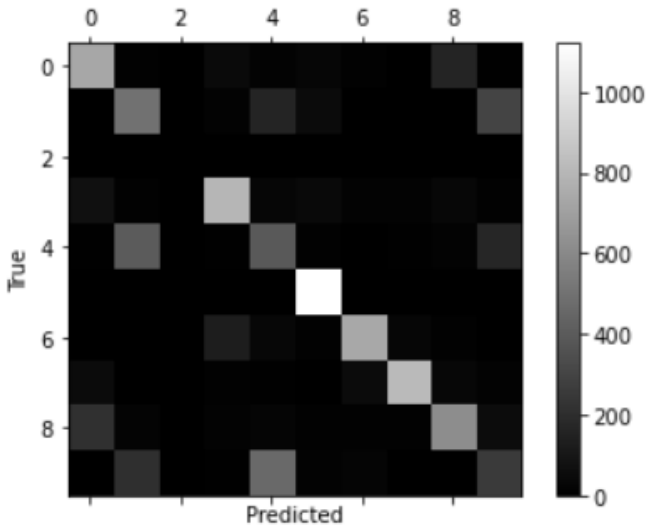

Figure 5: Confusion matrix obtained from *Polynomial* kernel

```
[[ 733   11    0   45   14   29    9    4  158    7]
 [   0  499    0   14  162   49    1    1    2  300]
 [   0    0    0    0    0    0    0    0    0    0]
 [  74   12    0  801   28   43   16   14   33   11]
 [   7  401    0    7  393   10    3    7   14  167]
 [   2    1    0    2    1 1121    4    0    3    1]
 [   3    2    0  130   34   10  736   27   13    3]
 [  50    1    0    9    5    0   49  818   32   16]
 [ 212   19    0   14   25   11   10   10  620   53]
 [   0  207    0    7  467   16   26    2    0  257]]
```

Figure 6: Confusion matrix values in Figure 5

Note that the indexes in the confusion matrix in Figure 5 are referring to the digits;

$$\{3, 7, 5, 2, 9, 1, 6, 0, 8, 4\}$$

The accuracy is resulted as $\%59.78$. After the calculation of the confusion matrix, I realized an error in my code, for instance, row 10 and column 5 is higher than the row 5 and column 5 value. For other methods, this bug is fixed.

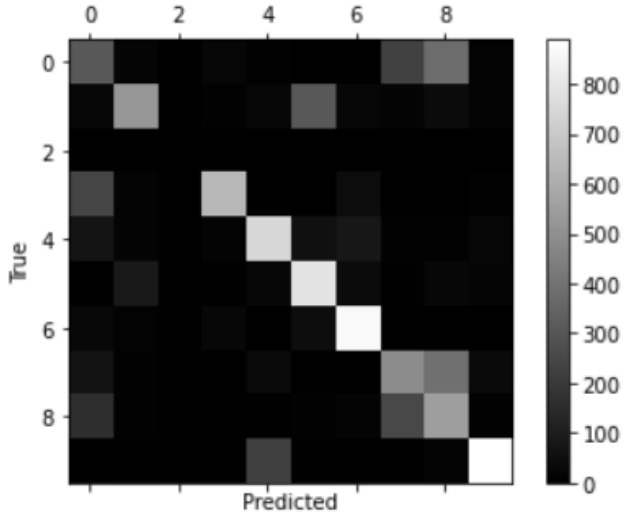

Figure 7: Confusion matrix obtained from *Gaussian* kernel

```
[[307  19   0  24   9   0   2 231 378  12]
 [ 22 521   0  10  23 307  21  16  41  13]
 [  0   0   0   0   0   0   0   0   0   0]
 [243  16   0 637   5   2  43   0   3   9]
 [ 72  16   0  17 750  56  81   9  10  21]
 [  6  87   0   5  25 794  41   6  26  20]
 [ 29  11   0  25   1  47 865   0   2   0]
 [ 63   9   0   0  34   1   3 489 392  37]
 [161  10   0   1   3   8  14 251 552   9]
 [  0   5   0   4 221   4   0   0  12 889]]
```

Figure 8: Confusion matrix values in Figure 7

Note that the indexes in the confusion matrix in Figure 7 are referring to the digits;

$$\{0, 9, 5, 2, 7, 6, 4, 1, 3, 8\}$$

The accuracy is resulted as $\%59.24$.

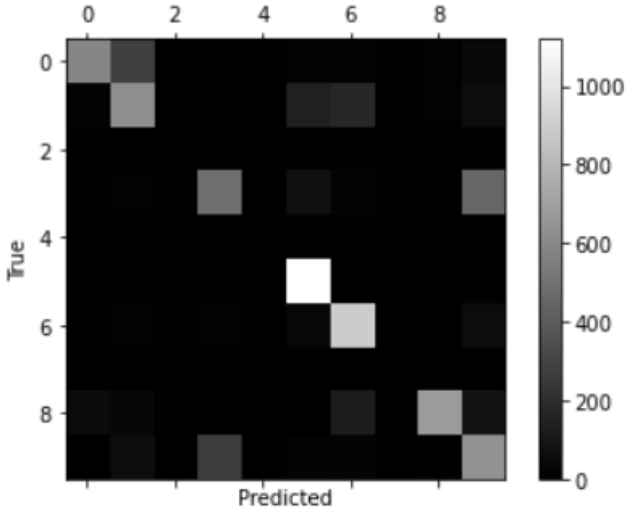

Figure 9: Confusion matrix obtained from *Sigmoid* kernel

```
[[ 589  283    0    0    0   19   14    0   12   41]
 [  18  625    0    5    0  141  173    0   11   59]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    9    0  484    0   72   15    0    1  447]
 [   0    0    0    0    0    0    0    0    0    0]
 [   4    1    0    0    0 1120    3    0    0    7]
 [   5   11    0   10    0   34  896    0    0   54]
 [   0    0    0    0    0    0    0    0    0    0]
 [  52   30    0    0    0    1  130    0  682   85]
 [   3   59    0  263    0   15   20    0    6  643]]
```

Figure 10: Confusion matrix values in Figure 9

Note that the indexes in the confusion matrix in Figure 9 are referring to the digits;

$$\{6, 2, 4, 7, 5, 1, 3, 8, 9, 0\}$$

The accuracy is resulted as $\%50.4$.

From the results obtained, it can be said that the greater accuracy is obtained from the *Polynomial* kernel, *Gaussian* kernel and *Sigmoid* kernel respectively. If the bug in the *Polynomial* kernel is fixed the accuracy would increase. By looking at the results, it can be said that *Gaussian* and *Polynomial* kernel classifies similarly but *Sigmoid* kernel did not classified digits as good as the other two kernel methods.

From the confusion matrixes, it can be said that the digits 4, 7 and 9 are the digits which results with the highest prediction error and the digit 5 did not obtained in any of the converged centroids. 60 thousand data could not be calculated because of the hardware of my computer and after trial and error it is found that 30 thousand and 10 thousand train data results with similar test error. Thus, considering the time complexity of the *Gaussian* kernel 10 thousand train data is used.

Appendix

```
import matplotlib.pyplot as plt
import numpy as np
from keras.datasets import mnist
from sklearn.metrics import confusion_matrix


def random_sample(train_size):
    index = np.random.randint(60000, size=train_size)
    sampled_data = X_train[:, index]
    return sampled_data


def kernel_choice(a, b, c=1, d=2, sigmoid_c=0.1, theta=-10, sigma=15,
kernel_type="polynomial"):
    kernel = a @ b
    if kernel_type == "polynomial":
        K = (kernel + c) ** d
    elif kernel_type == "gaussian":
        K = np.zeros((len(b.T), len(b.T)))
        for i in range(len(a)):
            for j in range(len(b.T)):
                K[i, j] = np.exp(-np.linalg.norm(b[:, i] - b[:, j]) / (2 *
(sigma ** 2)))
    else:
        K = np.tanh(sigmoid_c * kernel + theta)
    np.save(f"K_sigmoid{len(b.T)}.npy", K)
    return K


def distance_kernel(K, j, sum_1, sum_2):
    return sum_1 + K[j, j] - sum_2[:, j]


def W_init():
```

```
    w_current = np.zeros((train_size, k))
    for i in range(train_size):
        w_current[i, np.random.randint(k)] = 1
    pj = np.sum(w_current, axis=0)
    w_current = w_current / pj
    return w_current


def W_update(w_current, K):
    iteration = 0
    check = False
    while not check:
        iteration += 1
        print("Current iteration: ", iteration)
        w_update = np.zeros((train_size, k))
        sum_1 = np.diag(w_current.T @ (K @ w_current))
        sum_2 = 2 * (w_current.T @ K)
        for i in range(train_size):
            w_update[i, np.argmin(distance_kernel(K, i, sum_1, sum_2))] = 1
        partition_sum = np.sum(w_update, axis=0)
        w_update = w_update / partition_sum
        if np.array_equal(w_current, w_update):
            check = True
        if iteration == 200:
            check = True
        w_current = w_update
    np.save("W_30k_sigmoid.npy", w_update)
    return iteration


def kernel_dist_test(index, sum1, sum2):
    return sum1 - sum2[:, index]


def test():
    W_test = np.zeros((test_size, k))
    sum1 = np.diag(w_update.T @ (K @ w_update))
    sum2 = 2 * (w_update.T @ K2)
    for i in range(test_size):
        W_test[i, np.argmin(kernel_dist_test(i, sum1, sum2))] = 1
    return W_test


def results():
    arr = []
    y_pred = []
    y_true = []
    for i in range(test_size):
        if (W_test[i, 0] != 0):
            arr.append(i)
            pred_label = 0
        elif (W_test[i, 1] != 0):
            arr.append(i)
            pred_label = 9
        elif (W_test[i, 2] != 0):
            arr.append(i)
            pred_label = 6
        elif (W_test[i, 3] != 0):
            arr.append(i)
            pred_label = 5
        elif (W_test[i, 4] != 0):
```

```
            arr.append(i)
            pred_label = 1
        elif (W_test[i, 5] != 0):
            arr.append(i)
            pred_label = 6
        elif (W_test[i, 6] != 0):
            arr.append(i)
            pred_label = 0
        elif (W_test[i, 7] != 0):
            arr.append(i)
            pred_label = 7
        elif (W_test[i, 8] != 0):
            arr.append(i)
            pred_label = 8
        else:
            arr.append(i)
            pred_label = 3
        y_pred.append(pred_label)
        y_true.append(y_test[arr[i]])
    check_all = np.zeros(test_size)
    for i in range(test_size):
        check_all[i] = (y_pred[i] == y_true[i])
    return y_pred, y_true, check_all


def show_clusters(index):
    arr = []
    for i in range(test_size):
        if W_test[i, index] != 0:
            arr.append(i)
    plt.gray()  # B/W Images
    plt.figure(figsize=(10, 9))  # Adjusting figure size
    temp = X_test.reshape(28, 28, -1)
    # print(len(arr))
    for i in range(0, len(arr)):
        plt.subplot(1, len(arr), i + 1)
        plt.imshow(temp[:, :, arr[i]])


def show_centroids():
    temp_keeper = []
    for x in range(k):
        arr = []
        for i in range(len(w_update)):
            if w_update[i, x] != 0:
                arr.append(i)
        temp_add = np.zeros([len(w_update), 784])
        for i in range(len(arr)):
            temp_add[i, :] = sampled_data[:, arr[i]]
        centroid_matrix = np.zeros(784)
        for i in range(784):
            centroid_matrix[i] = np.mean(temp_add[:, i])
        temp_keeper.append(centroid_matrix)
    plt.gray()  # B/W Images
    plt.figure(figsize=(10, 9))  # Adjusting figure size
    temp_keeper = np.array(temp_keeper)
    temp = temp_keeper.reshape(k, 28, 28)
    for i in range(10):
        plt.subplot(3, 4, i + 1)
        plt.imshow(temp[i])
```

```
if __name__ == "__main__":
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    # Normalization
    x_train = x_train / 255.0
    x_test = x_test / 255.0

    X_train = x_train.reshape(len(x_train), -1).T
    X_test = x_test.reshape(len(x_test), -1).T

    train_size = 30000
    k = 10
    test_size = 10000
    sigma = 15
    kernel_type = "gaussian"

    X_test = X_test[:, :test_size]

    sampled_data = random_sample(train_size)
    K = kernel_choice(sampled_data.T, sampled_data,
kernel_type=kernel_type)
    # K = np.load("K_sigmoid30000.npy")
    iter_no = W_update(W_init(), K)

    if kernel_type == "gaussian":
        K_gaussian = np.zeros((train_size, test_size))
        for i in range(train_size):
            for j in range(test_size):
                K_gaussian[i, j] = np.exp(-np.linalg.norm(sampled_data[:,
i] - X_test[:, j]) / (2 * (sigma ** 2)))
        K2 = K_gaussian
    else:
        K2 = kernel_choice(sampled_data.T, X_test, sigma=sigma,
kernel_type=kernel_type)

    w_update = np.load("W_30k_sigmoid.npy")
    W_test = test()
    show_clusters(0)
    y_pred, y_true, check_all = results()
    print(confusion_matrix(y_true, y_pred))
    print(sum(check_all) / test_size)
```