Assignment 3: PCA, k-Means and Kernel Methods

Due 14 Aug at 23:59 **Points** 35 **Questions** 6
Available 18 Jul at 0:00 - 17 Aug at 23:59 **Time limit** None
Allowed attempts Unlimited

Instructions

Weighting

This assessment is worth 35% of your overall grade.

Course Learning Outcomes

- CLO 1: Apply basic concepts of machine learning, and classic algorithms such as Support Vector Machines and Neural Networks, Deep Learning.
- CLO 2: Demonstrate an understanding of the basic principles and theory of machine learning necessary to develop algorithms.
- CLO 3: Devise algorithms.

Task Description

Purpose:

The purpose of this assignment is to examine your ability to demonstrate an understanding of the basic principles and theory of machine learning necessary to develop algorithms. This is to demonstrate the outcomes of your learning and receive feedback on your progress.

Instructions:

This test comprises multiple-choice questions. You are welcome to use your textbook, readings and notes to assist you in answering the questions. You must complete the online test by the deadline.

Academic Integrity

It is your responsibility to ensure that any work you submit is your own.
You can use the **Turnitin Originality Checker** before you submit your work.

Instructions and submission guidelines:

Please read the following instruction carefully before you complete assignment 3

This is not a group assignment. Everyone is asked to complete the assignment individually.

- The assignment consists of five sub-tasks. You need to provide your solutions in the corresponding sections in this quiz.
- You need to also upload your code. We will check the code to ensure that your results can be generated from your code. If your result does not match your code, you will get a 0 mark for the relevant sections.
- Use Jupyter notebook for this assignment.

Data:

Please download the data that will be used in this assignment here: **mnist.csv**

The above data is a subsampled version of the MNIST dataset. It contains images for 10 digits (10 classes). The dataset contains 6,000 samples. The images from the data set have the size 28 x 28. They are saved in the csv data files. Every line of these files consists of an image, i.e. 785 numbers between 0 and 1. The first number of each line is the label, i.e. the digit which is depicted in the image. The following 784 numbers are the pixels of the 28 x 28 image.

Third-party Libraries,

You can use any third-party libraries to provide required results.

Please follow the instruction for each question to complete the relevant parts.

This quiz is no longer available as the course has been concluded.

Attempt history

Attempt	Time	Score

	Attempt	Time	Score
LATEST	**Attempt 1**	38,305 minutes	35 out of 35

Correct answers are hidden.

Score for this attempt: **35** out of 35
Submitted 14 Aug at 17:36
This attempt took 38,305 minutes.

Question 1

8 / 8 pts

1. Perform PCA on the dataset to reduce each sample into a 10-dimensional feature vector. (4 pts)

2. Show the covariance matrix of the transformed data. (2 pts)

3. For easy verification, show also the sum of covariance matrix. (1 pt)

4. Please also copy your code snippet here. (1 pt)

Your answer:

```
def PCA(data, n_components):
    # Find mean
    mean = np.mean(data, axis = 0)

    #substract mean
    difference = data - mean

    # Calculate the covariance matrix
    cov = np.cov(difference, rowvar = False)

    # Calculate eigenvectors and eigenvalues of the covariance matrix
    eig_value, eig_vector = np.linalg.eig(np.mat(cov))

    # get index from low to high
    idx = np.argsort(eig_value)

    # reverse it to from high to low
    high_to_low_idx = idx[::-1]

    # P vector projects the high dimensional data onto 10 dimensions
    P = eig_vector[:,high_to_low_idx[:n_components]].real

    # Transform the centralised data with a matrix P to 10 dimensions
    reduced_data = (difference @ P).real

    return reduced_data, mean, P

# convert df to numpy for the convience of matrix calculations
data = np.array(data_df)

reduced_data, mean, P = PCA(data, 10)
C_y = np.cov(reduced_data,rowvar = 0)

print(f'cov of transformed data:\n {C_y} \n')
print(f'sum: {np.sum(C_y)}\n')

# show dimension reduction
print(f'original data shape: {data.shape}, reduced_data shape: {reduced_data.shape}') # reduced from 28x28 features to 10 features
```

Covariance of the transformed data:

```
[[ 5.30558677e+00  7.69113061e-15 -2.33570641e-15 -8.46871239e-16
  -1.13705789e-16  2.76269533e-15 -2.96108825e-16  8.11338180e-16
   2.70051248e-16  1.42724454e-16]
 [ 7.69113061e-15  3.87701635e+00 -1.53976589e-15 -2.00643340e-15
  -1.39763365e-16  9.77159122e-17  5.17598226e-16 -1.44501106e-16
   5.98139826e-16  1.63452071e-16]
 [-2.33570641e-15 -1.53976589e-15  3.28704664e+00  2.84886300e-15
   1.17259095e-16  3.61030685e-16 -3.37564060e-17  7.30278389e-17
  -1.11336918e-15  4.73774120e-16]
```

```
[-8.46871239e-16 -2.00643340e-15  2.84886300e-15  2.91254096e+00
 -5.06938308e-16 -5.39637976e-15 -4.05076872e-16  1.22574248e-15
  9.95517869e-16  2.33037645e-16]
[-1.13705789e-16 -1.39763365e-16  1.17259095e-16 -5.06938308e-16
  2.48633206e+00 -2.90956531e-15 -1.39763365e-15  1.07280227e-15
  1.22648275e-15 -3.04992089e-16]
[ 2.76269533e-15  9.77159122e-17  3.61030685e-16 -5.39637976e-15
 -2.90956531e-15  2.35358942e+00 -2.62441251e-15  1.09116102e-15
  1.74182315e-15  1.64044289e-16]
[-2.96108825e-16  5.17598226e-16 -3.37564060e-17 -4.05076872e-16
 -1.39763365e-15 -2.62441251e-15  1.75556820e+00 -6.63283767e-16
 -5.28021256e-15  5.62606767e-17]
[ 8.11338180e-16 -1.44501106e-16  7.30278389e-17  1.22574248e-15
  1.07280227e-15  1.09116102e-15 -6.63283767e-16  1.54475982e+00
  8.44502368e-16  1.08849604e-15]
[ 2.70051248e-16  5.98139826e-16 -1.11336918e-15  9.95517869e-16
  1.22648275e-15  1.74182315e-15 -5.28021256e-15  8.44502368e-16
  1.45474519e+00 -5.36549190e-16]
[ 1.42724454e-16  1.63452071e-16  4.73774120e-16  2.33037645e-16
 -3.04992089e-16  1.64044289e-16  5.62606767e-17  1.08849604e-15
 -5.36549190e-16  1.24337974e+00]]
```

The sum of the cov above is:

```
sum: 26.2205651617228
```

good work easy to mark

Question 2

8 / 8 pts

1. Perform k-means clustering to cluster the dataset (without applying PCA) into 10 groups. (4 pts)

2. Show your 10 centroid points. (2 pts)

3. For easy verification, show sums of all dimensions for each of these 10 centroid points (i.e. show 10 numbers). (1 pt)

4. Please copy your code snippet here. (1 pt)

Your answer:

(I set a random seed = 0, not sure whether this will give a good initialization, but it can make sure every time I run the code, the results are the same.)

1. Code:

```
def kmeans(data,k = 10): # because next Q need to plot iterations vs loss, so did not set convergence condition here
    loss = []
    random.seed(0) #32, 42, 33, 2: at 8.

    #initially random pick k centroids
    idx = random.sample(range(0,data.shape[0]),k)
    centroids = data[idx, :]

    # get a matrix of squares of distances to all centroids for each sample in each row
    dist_sq_mat = find_dist_sq(data,centroids,k)

    # add new loss (sum up all closest dist_sq for all samples)
    loss.append(np.sum([min(dist_row) for dist_row in dist_sq_mat]))

    # update cluster group (rik) by assigning each sample to closest cluster — E Step
    clusters = np.array([np.argmin(i) for i in dist_sq_mat])


    converge = False
    ite = 1
    while converge != True:
        ite += 1
        # mask cluster k and update the k centroids(uk) (samples in row, so average each col and get a row for each k) — M Ste
        centroids = np.array([np.mean(data[clusters == k_value],axis = 0) for k_value in range(k)]).reshape(k,data.shape[1])

        # get a matrix of squares of distances to all centroids for each sample in each row
        dist_sq_mat = find_dist_sq(data,centroids,k)

        # add new loss (sum up all closest dist_sq for all samples)
        loss.append(np.sum([min(dist_row) for dist_row in dist_sq_mat]))

        # update cluster group (rik) by assigning each sample to closest cluster — E Step
        new_clusters = np.array([np.argmin(i) for i in dist_sq_mat])

        #check convergence
        if np.array_equal(clusters, new_clusters):
            converge = True
        else:
            clusters = np.array(new_clusters.copy())

    return clusters,centroids,loss, ite
```

Helper Function:

```
# a helper function returns a matrix of all samples' squared euclidean distance to each centroids (used in Q2 ,and Q4 validation)
def find_dist_sq(data,centroids,k):
    dist_mat = []

    for sample in data:
        # i.e. dist_lst = [||Xi - u1||^2, ||Xi - u2||^2, ....,||Xi - uk||^2]
        dist_lst = [np.sum(np.square(sample - centroids[i])) for i in range(k)]
        dist_mat.append(dist_lst)

    arr = np.array(dist_mat) # convert back to numpy array for the convinience of numpy operations
    return arr
```

Results:

2. Please see the 10 centroids in the code (too large to be copied to here)

3. Sums

```
sum: 65.89028801782221
sum: 118.51778330667635
sum: 106.05934838581905
sum: 145.93894256247216
sum: 115.76537433155077
sum: 127.12701361246262
sum: 94.18354803662905
sum: 94.48874364560652
sum: 99.2330552584671
sum: 116.8240144478846
```

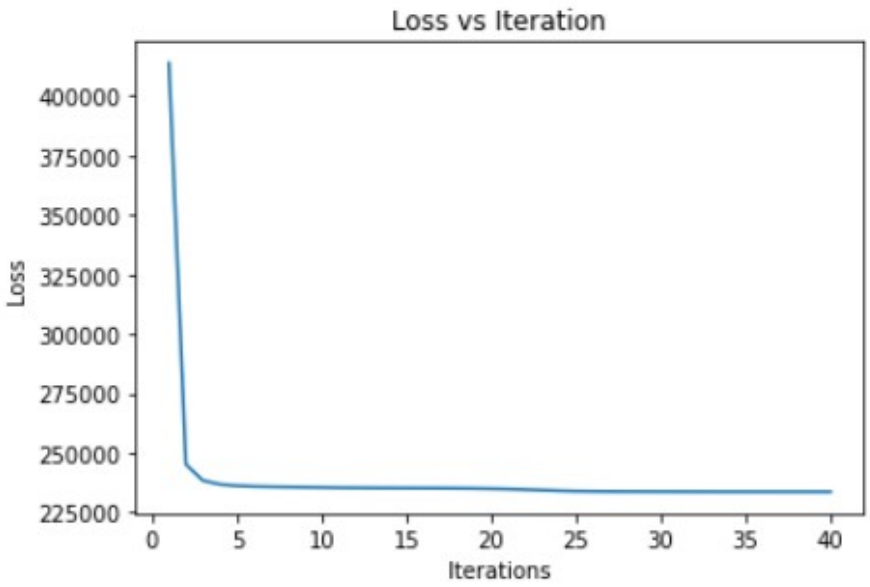
Question 3

4 / 4 pts

Please plot the loss curve, that is, the change of loss value of the k-means algorithm with respect to the number of iterations of k-means clustering from Question 2.

Your answer:

```
clusters, centroids, loss,ite = kmeans(data, k=10)

plt.title('Loss vs Iteration')
plt.plot(range(1,ite+1), loss,)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



Question 4

4 / 4 pts

1. From the original dataset (before PCA) please use the first 4000 samples as the training set and remaining 2000 samples as the validation set, and design a way to choose the best k in k-means algorithm. (2 pts)

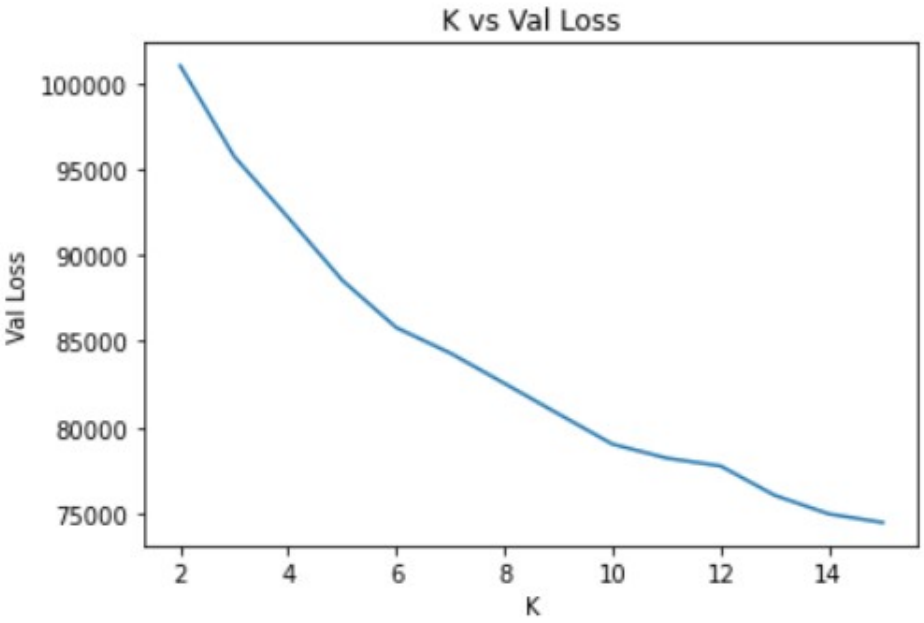
2. Clearly show your optimal k. (1 pt)

3. Please copy your code snippet here. (1 pt)

Your answer:

```
# slice df, convert to numpy array
data_train = np.array(data_df[:4000])
data_val = np.array(data_df[4000:])

# Show dimensions
print(f'train shape: {data_train.shape}, val shape: {data_val.shape}')

# This function picks k by elbow plot
def chose_k(reduced_train,reduced_val):
    # initialise loss lst
    loss_lst = []

    # try k = 2,3,...,15 # start from 2, because there's no point of clustering them into one class, we are doing a classification task.
    k_lst = range(2,16)

    # collect all the val losses for different k
    for k_value in k_lst:
        # call kmeans function defined before
        clusters, centroids, _ ,_ = kmeans(reduced_train, k = k_value)

        # a matrix of all samples' squared euclidean distance to each centroids
        dist_sq_mat = find_dist_sq(reduced_val,centroids,k_value) # on val set

        # add new loss (sum up all closest dist_sq for all samples)
        loss_lst.append(np.sum([min(dist_row) for dist_row in dist_sq_mat]))


    plt.title('K vs Val Loss')
    plt.plot(k_lst, loss_lst)
    plt.xlabel('K')
    plt.ylabel('Val Loss')
    plt.show()
    return loss_lst

loss_lst = chose_k(data_train,data_val)
```

```
train shape: (4000, 784), val shape: (2000, 784)
```



From the elbow plot above, we can see the loss decrease slower just after k=10, i.e. an elbow at k = 10. The optimal k is 10.

For each k setting, I train the kmeans using the training set to get the centroids; and calculate the loss from the validation set.

The logic behind elbow method is, that we want to minimize the loss, but still keep a reasonable amount of clusters. The 'loss vs k' is a strictly decreasing function, but we don't want the loss to be zero (in that case each sample is a class, there is no point in doing that). At the beginning of this "K vs loss plot" is decreasing very fast, until we reached the 'elbow' it will decrease slowly. The elbow point is like a balance point between the 'k vs loss' trade-off.

Question 5

11 / 11 pts

1.
Please implement kernel k-means algorithm with RBF-kernel, that is,

_____ .

Use — .

Please only use the first 500 samples of the original dataset and cluster them into 5 groups. This is for reducing the running time of your code. (7 pts for the above)

2. Show the 5 cluster centroids. (2 pts)

2. For easy verification, show sums of all dimensions for each of these 5 centroid points (i.e. show 5 numbers) (1 pt)

3. Please copy your code snippet here. (1 pt)

TIPS: If you can use matrix operations to replace summations, your code will be more efficient. However, this is just optional.

Your answer:

The function that calculates the RBF kernel matrix required in this question:

```
def get_kernel(X):
    X_norm = np.sum(X ** 2, axis = -1)
    # numerator = -||xi - xj||^2 = - (l2_norm_sq) = -(||xi||^2 + ||xj||^2 - 2 * xi^T * xj)
    l2_norm_sq = X_norm[:,None] + X_norm[None,:] - 2 * np.dot(X, X.T) #[:,None] creat a new axis
    numerator = -l2_norm_sq

    N = X.shape[0]
    #demoninator =  (1/N^2) * sum (|xi - xj||^2) = (1/N^2) * sum (l2_norm_sq)
    demoninator = (1/(N ** 2)) * np.sum(l2_norm_sq)
    kernel_matrix = np.exp(numerator/demoninator)
    return kernel_matrix

small_data = np.array(data_df[:500])

print(get_kernel(small_data).shape)
print(get_kernel(small_data))
```

```
(500, 500)
[[1.         0.42356793 0.31368959 ... 0.35607876 0.44157821 0.415671  ]
 [0.42356793 1.         0.2817014  ... 0.34320126 0.34275148 0.37696311]
 [0.31368959 0.2817014  1.         ... 0.40774871 0.44471621 0.29878051]
 ...
 [0.35607876 0.34320126 0.40774871 ... 1.         0.43982207 0.40343488]
 [0.44157821 0.34275148 0.44471621 ... 0.43982207 1.         0.48823275]
 [0.415671   0.37696311 0.29878051 ... 0.40343488 0.48823275 1.        ]]
```

The function calculates the kernel Kmeans:

```
def RBF_kmeans(data,k = 5):
    kernel_matrix = get_kernel(data)

    random.seed(0)

    R = np.zeros((data.shape[0],k))
    #initially random pick k centroids
    cen_idx = random.sample(range(0,data.shape[0]),k) #cen_idx is idx of centroids in the dataset
    #print('idx: ',idx)
    for i in range(k):
        R[cen_idx[i]][i] = 1

    #assign samples to the closet centroid
    for sample_idx in range(data.shape[0]):
        if sample_idx in cen_idx: # if the sample is the picked centroid, continue
            continue
        else:
            lst = [kernel_matrix[sample_idx,idx] for idx in cen_idx]# a lst of K(i=sample_idx,j= position of centroid in the dataset)
            label = np.argmax(lst) # use max because,closer distance have larger kernel matrix entry (kernel is a similarity measurement)
            R[sample_idx][label] = 1
```

```
#itera = 0
converge = False
while (not converge):
    #itera += 1
    #print(itera)
    new_R = []
    for sample_idx in range(data.shape[0]):
        cost_lst = []
        for clus_idx in range(k):# for all clusters groups

            #cluster population is how many samples in one class k
            cluster_population = R[:,clus_idx].sum() # sum r_jk
            rk = R[:,clus_idx]/cluster_population

            cost = kernel_matrix[sample_idx,sample_idx] -2 * rk.T @ kernel_matrix[:,sample_idx] + rk.T @ kernel_matrix @ r
            cost_lst.append(cost)

        # the class with min cost is the label
        label = np.argmin(cost_lst)

        #create a R matrix row for this sample, with a 1 in its labeled class
        new_R_row = [0]*k
        new_R_row[label] = 1
        #append it to new R
        new_R.append(new_R_row)

    # After classify all the samples, convert new_R to array, for checking convergence
    new_R = np.array(new_R)

    # check convergence
    if np.array_equal(R, new_R):
        converge = True
    else:
        R = np.array(new_R.copy())

return new_R
```

Please see the 5 centroids in the code (too larger to be copied here)

The sums of the centroids:

```
127.62925278219397

88.7140550337782

91.91649526943644

67.12004901960785

129.50529897909576
```

Question 6

0 / 0 pts

Please upload your code for this assignment as .ipynb file

Please comment your code and clearly annotate each section of the code with the question number. These section will be used to verify your solutions to each question.

A3_PCA_Kmeans_Kernel.ipynb