Assignment: Homework 8

How to Hand It In

1. Put all your solutions in one folder. Compress this folder and name it <firstname>_<lastname>_HW8.zip. For example, "Zhiyao_Duan_HW8.zip".
2. Make sure you have a report to show graphs and explain your answers, in addition to source code.
3. Cite resources (e.g., external code repo) in your answers.
4. Submit to the corresponding entry on Blackboard.

When to Hand It In

It is due at 11:59 PM on the date specified on the course calendar. **Late assignments will receive a 20% deduction of the full grade each day.**

1. (3.5 points) K-means clustering.
    a. (1 point) Implement the K-means clustering algorithm using Euclidean distance. The number of clusters, K, should be a parameter, and it determines the number of clusters to form as well as the number of centroids to use. Initialize the cluster centroids randomly.
    b. (0.5 points) Run K-means on a 2D toy dataset with K=3 (the actual number of clusters). The code for generating the synthetic dataset can be found in the provided skeleton code. Visualize the clustering results after the 25^{th} iteration. Plot the loss value versus the number of iterations.
    c. (1 point) Run K-means on the toy dataset using different K values. Visualize the final clustering results for each K value and discuss the differences. Plot a curve about the final loss versus K. Based on this curve, discuss which value of K seems most suitable.
    d. (1 point) Run K-means on the dataset created in HW1, utilizing only the numerical features. Perform feature normalization as appropriate before clustering. Select a suitable distance or similarity measure for the K-means algorithm. Plot the final loss value versus K. Based on the plot, choose an appropriate value for K. Perform clustering using the chosen K value and analyze the results.

    Note: Implemented KMeans functions, such as sklearn.cluster.KMeans, cannot be used to solve this question. But you can use other functions in sklearn and numpy if needed.

2. (6.5 points) Dimensionality reduction.
    In this section, you will implement Principal Component Analysis (PCA) and an autoencoder to learn compressed representations of human faces. You will be using Labeled Faces in the Wild (LFW) dataset (http://vis-www.cs.umass.edu/lfw/#download). The LFW dataset is designed for face verification and consists of 13,233 images. To

simplify preprocessing, we provide the LFWcrop version (https://conradsanderson.id.au/lfwcrop/), which is a cropped version of the LFW dataset that retains only the central portion of each image (i.e., the face), removing most of the background. The dataset has been split into training and validation subsets, and the loader code is provided in the skeleton.ipynb.

a. (1 point) Implement PCA from scratch. Note that implemented PCA functions, such as sklearn.decomposition.PCA can not be used for this question. The input of your PCA function is the data matrix, the percentage of total variance to preserve, and the output is the PCA transformation matrix.
b. (1 point) Train the PCA on the training dataset. Keep enough components to explain at least 90% of the total variance. After implementing PCA, plot the eigenfaces, which are the eigenvectors corresponding to the largest eigenvalues.
c. (1 point) Project the validation subset onto the learned components to realize dimensionality reduction (i.e., compression). Reconstruct the faces from the compressed representations and report the reconstruction mean squared error (MSE). Visualize the original image vs the reconstructed image for several pictures.
d. (1 point) Implement an autoencoder in PyTorch. The input consists of flattened grayscale images of size (1, 64 x 64). The encoder starts with a linear layer containing 1024 neurons, followed by BatchNorm1d and ReLU activation. The next layer (bottleneck layer) in the encoder has the same number of neurons as the principal components you choose in b), also followed by BatchNorm1d and ReLU activation. The decoder has a linear layer with 1024 neurons with BatchNorm1d and ReLU activation. Finally, the last layer in the decoder is a linear layer with 64x64 neurons followed by BatchNorm1d and Sigmoid activation.
e. (1.5 points) Train the autoencoder using the training subset with MSE loss, Adam optimizer and learning rate of 0.001 . Set an appropriate batch size and training epochs that fit to your computational resources. After training the autoencoder, extract the encoder part of the network, which maps the input to the compressed representation. Use this encoder to map images in the validation subset to the compressed representation, and then use the decoder to reconstruct the original images. Calculate the reconstruction MSE for the validation data and compare it with the PCA reconstruction MSE. Visualize the original image vs the reconstructed image for several pictures.
f. (1 point) Finally, visualize the weights of the first layer of the encoder, which can be interpreted as the most important patterns in the data. Compare the eigenfaces in PCA with the weights learned in autoencoder and describe your findings.