ECE4880J: Computer Vision July 1, 2022

Homework 4: Machine Learning

Instructor: Siheng Chen **YOUR NAME**

Instruction

- This homework is due at 11:59:59 p.m. on ** July 11th, 2022.
- The write-up must be a soft copy *.pdf* file edited by LaTeX.
- The overall submission should be a *.zip* file named by xxx(student id)-xxx(name)-Assignment4.zip

Python Environment. We are using Python 3.7 for this course. We will use the following packages in this course: Numpy, SciPy, Matplotlib, Pytorch.

Q1. Principal Component Analysis [50 points]

Part A task: basic calculation.

Given the following dataset for a 2-class problem with 2-D features:

$$c_1 = \left\{ \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right\}, \; c_2 = \left\{ \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \begin{bmatrix} 6 \\ 4 \end{bmatrix} \right\}$$

Using global PCA, find the best direction onto which the data will be projected on. You need to fill the function in *PCA_basic.py*. Do not otherwise change the function name in the provided code.

1. Express the equation of the line along this direction passing through the samples mean in the following form: $\vec{w}^T \vec{x} + w_0 = 0$, where

   $$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ and } \vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

   and w_0 is a scalar known as the bias. Plot the $\vec{w}^T \vec{x} + w_0 = 0$ line along with all the sample points as well the line along $\vec{w}$ which recall starts from the mean.

2. Project and reconstruct all the sample points. Plot the reconstructed points.

3. Find the total mean square error MSE for all sample points (between the original points and the reconstructed points).

4. Find the Fisher Ratio for this projection defined by

   $$FR = \frac{(m_1 - m_2)^2}{\sigma_1^2 + \sigma_2^2}$$

   where m_i is the mean of the projected samples of class i, and σ_i^2 is the equivalent variance. You can compute the FR on the projected 1-D points (rather than the reconstructed points which are 2-D vectors).

Part B task: MNIST implementation. Download the MNIST (http://yann.lecun.com/exdb/mnist/) dataset. You are going to implement PCA from scratch. You are allowed to use SVD and/or eigenvalue decomposition. Please fill the *PCA_MNIST.py*.

1. Plot the mean image of digit 1 and plot the first 5 global PCA vectors corresponding to the dataset corresponding to two cases: using Gram Matrix Trick and without Gram Matrix trick. Do not forget to remove the mean before you run PCA. Measure the time taken for the PCA vector computation in each case.

2. Comment on the time taken in each case. Does it make sense to use the Gram trick for this particular dataset?

3. Pick any random image from the dataset which can be any image from any class and project onto the eigen space (the global PCA eigen space that you obtained in part 1 of this problem) and reconstruct it using the first n eigen vectors (not just the nth vector but all of them up until n) (for the two cases, Gram trick and without it) where $n = \{1, 2, 5, 10, 20\}$. Do not forget to remove the mean before projecting the image, and also to add it back once it has been reconstructed using only the first n eigen vectors.

   For each of the 5 reconstructions, compute the mean square error (MSE) of the reconstructed image. Note that the MSE between two vectors $\vec{a}$ and $\vec{b}$ is given by

   $$MSE(\vec{a}, \vec{b}) = ||\vec{a} - \vec{b}||_2^2$$

   Display all reconstructed images and the original in one figure, with the corresponding MSE value as the title of each sub-figure.

Submission format. Please submit your filled *PCA_basic.py* and *PCA_MNIST.py*. Report your experimental results in the write-up.

Q2. K-means [20 points]

Use K-means to cluster the 2D data points provided in *kmeans_array1.npy* and *kmeans_array2.npy*. Figure 1 is a data visualization of one of the datasets. You are to implement K-means from scratch and fill the *kmeans.py*.

1. Use *kmeans_array1.npy*. Set the number of clusters K as 5. Vary the iteration number as $1, 2, 4, 8, 16$. Plot the objective value as a function of the iteration number. Show the scatter plot of K-means result for each case.

2. Use *kmeans_array1.npy*. Set the iteration number as 16. Vary the number of clusters K as $2, 3, 4, 5$. Show the scatter plot of K-means result for each case. Describe the phenomenon you observe.

3. Use another data *kmeans_array2.npy* and repeat the above tasks.

Here we provide a sample scatter plot of K-means result for referring, see Figure 2. Different color represents the different clustering classes and stars present clustering centroids.

Submission format. Please submit your filled *kmeans.py*. Report your experimental results in the write-up.

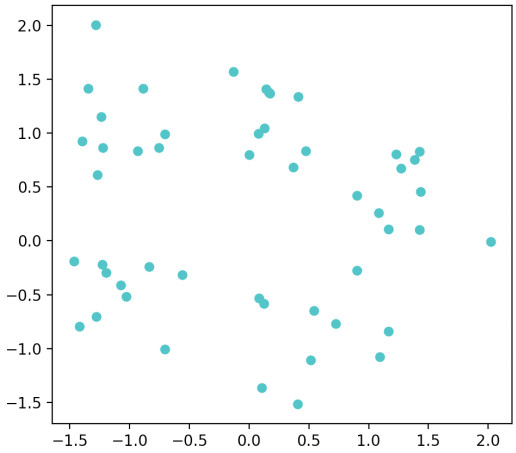

Figure 1: Visualization of K-means data.

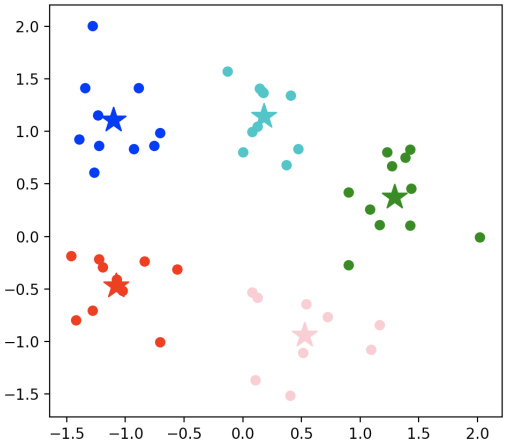

Figure 2: A visualization sample of K-means result.

Q3. Logistic Regression [30 points]

Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This sigmoid function is represented by the following formulas:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

This is a really important function to understand well!

Part A task: visualization.

1. Plot $\sigma(x)$ and $1 - \sigma(x)$ on the same plot. Restrict the domain to $x \in [-20, 20]$. Make sure to label which curve is $\sigma(x)$ and which curve is $1 - \sigma(x)$.

2. What relationship between $\sigma(x)$ and $1-\sigma(x)$ do you notice? (Beyond the fact that the second is one minus the first).

3. Use this insight to prove a relationship between $\sigma(x)$ and $\sigma(-x)$. Show your work and write your final answer as $\sigma(-x)$ in terms of $\sigma(x)$.

Part B task: implementation. Complete the functions marked with the identifier "ADD YOUR CODE HERE" in *log_regression.py*. Extensive function descriptions are also provided in the code. Do not otherwise change the functionality of any of the provided code (though you are welcome to replace it if you can guarantee that the functionality won't change).

1. Run gradient descent for 100 epochs with a learning rate of 0.1 (no explicit regularization): a) plot train loss and test loss on one plot; b) plot train accuracy and test accuracy on another separate plot.

2. If you did this correctly, you might wonder if training longer could squeeze out a bit more test accuracy. Rerun for 3000 epochs and produce new versions of the a) loss plot; b) accuracy plot from the previous part. Did your test accuracy improve?

Submission format. Please submit your filled *log_regression.py*, report your plots, and analyze your experimental results in the write-up.

Q4: Linear Regression [20 points]

Here, we'll be solving some toy (i.e., small synthetic dataset) linear regression problems, in both a closed-form and via iterative optimization (gradient descent, specifically).

Consider $X \in \mathbb{R}^{n \times d}$ (for small values of n and d to be defined below), $y \in \mathbb{R}^n$, and $w \in \mathbb{R}^d$. For simplicity, we will not use a bias term in this problem, but please note that it's typically a very good idea to have a bias term in practice. The objective function we are trying to minimize is:

$$f(w) \;=\; \|Xw - y\|_2^2 + \lambda \|w\|_2^2 .$$

This is just a vectorized way of computing the square loss over an n-item dataset.

Requirements. Some of the following questions require only writing, and some require writing + code. We are providing you with *lin_regression.py* as a template, which includes a skeleton gradient descent implementation and incomplete functions. The only package you need/are allowed to use is numpy.

Consider the following design matrix $X = [[1,1],[2,3],[3,3]]$. In the notation above, $n = 3, d = 2$. Consider $y = [[1],[3],[3]]$. In what follows, we will try various ways to solve this linear regression problem.

1. **Writing only.** Show that a closed-form solution for w is $w = (X^T X + \lambda I)^{-1} X^T y$ (assuming the inversion is valid).

2. **Writing + code.** Attempt to use the closed-form solution in (1) to find w, with $\lambda = 0$. Report w or explain what happened if you ran into a problem.

3. **Writing + code.** Now try (2) again with ℓ_2 regularization at three different values of λ : $1, 10^{-3}, 10^{-5}$. In each case, What values of w do you find for (2)?

4. **Writing + code.** Derive the update step based on the gradient descent algorithm.

5. **Writing + code.** Now use gradient descent to find w (for $\lambda = 0$). What values of w do you find using a learning rate of 0.01 and 10 iterations, 100 iterations, 1000 iterations, 10000 iterations?

6. **Writing only.** In one sentence, compare the effect of ℓ_2 regularization in part (3) with the effect of early stopping in gradient descent (i.e., running gradient descent for a limited number of iterations) in part (5).

Submission format. Please submit your filled *lin_regression.py* and report your experimental results in the write-up.