Project 1

In case of any question, contact the below TAs:
Ehsan Forootan
Kiana Hooshanfar

Instructions: You may use Python, Matlab or Julia to solve these questions. In case you are using notebook formats (Such as Jupiter or CoLab), write your explanations in the same notebook as well. If not, you need to turn in an additional report. Make sure the results are visualized.

Question 1.

Least squares regression

A - Generate a synthetic (simulated) data that contains feature n-vectors $x_1, ..., x_N$, and the associated 1-D values, $y_1, ..., y_N$, ranging between $[-1, +1]$. consider $n = 30$, $N = 200$. Generate the x's using weibull's distribution, while generating the $y's$ from a poisson distribution. Plot histogram of both y and x. Also plot the Cross-correlation of y with first dimension of x .

B - Repeat the process above for $N = 20$ and name the data set as test. you may change the random distribution parameters by a small difference. Plot the Cross-correlation and histograms for test data.

C- For the train data, Find β, ν such that

$$\min \sum_{i=0}^{N} (\beta^T x_i + \nu - y_i)^2$$

Report the MSE on the train and test data.

D- Now, repeat the last part for the objective below:

$$\min \sum_{i=0}^{N} (\beta^T x_i + \nu - y_i)^2 + \lambda^T \beta$$

Repeat for multiple σ's, where $\lambda_{n\times 1} = \sigma[1, 1, 1, \ldots, 1]^T$ and σ varies from $\exp(-4)$ to $\exp(4)$

Draw the loss function over σ for both train and test.

E- Repeat part D for

$$\min \sum_{i=0}^{N} (\beta^T x_i + \nu - y_i)^2 + \sigma^2 \|\beta\|_2^2$$

F- Now, imagine you have two models, as in β_1 is valid for $0.5 \geq y$ and β_2 is valid for $y \geq -0.5$. Repeat part E for these two models. Did the performance improve?

Question 2.

Fast Inverse

A- Generate an invertible, positive-definite matrix A with size 2000×2000 using a random distribution. generate an non-negative, non-zero vector a with size 2000. (Hint: use Wishart distribution for the matrix generation)

B- Form the matrix $A + a\,a^T$. Form its LU (or Cholesky decomposition if the formed matrix is PD) using Numpy or any other libraries. From this decomposition, suggest a method to calculate $(A + a\,a^T)^{-1}$. Time the whole section from matrix formation to the inverse result.

C- Lookup the Sherman–Morrison formula. Use the formula to calculate $(A + a\,a^T)^{-1}$ directly. Time this section as well and compare the results.

D- Repeat Section A to now generate an A matrix sized 2000×2000 and another invertible, positive-definite matrix B sized 2000×50000.

E- Repeat Part B to calculate $(A + B\,B^T)^{-1}$.

F- Lookup the Woodbury matrix identity. Use the formula to calculate $(A + B\,B^T)^{-1}$ directly. Compare results with Part E.

Question 3.

Weight Update in a Single-Layer Perceptron

Consider a simplified one-layer Multi-Layer Perceptron (MLP) with the following characteristics: some input neurons and one output neuron.

The sigmoid function $\sigma(z)$ for the output neuron is defined as:

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

The loss function which we want to minimize in this MLP is defined as:

$$L = \sum_{i=0}^{N} (y_i - \hat{y}_i),$$

where $\hat{y}_i$ is defined as:

$$\hat{y}_i = \sigma\big(W^T x_i + b\big).$$

The task here is now defined as finding W and b such that L is minimized over sampled data. You may use the same data as Question 1 with $n = 2$ and $N = 100$. Put the y's in between $[0,1]$.

A- Randomly select 20 percent of the data as the test set. You will only evaluate the loss using the test data, while the other 80 percent will be used to train and update W and b.

B- Analytically find the gradient of L with respect to W and b, for one fixed data point (x_i, y_i).

C- Analytically find the Hessian of L with respect to W and b, for one fixed data point (x_i, y_i).

D- Newton update: Analytically find the formula for updating W and b using the formula below:

$$\alpha^+ = \alpha^- - \big(\nabla^2_\alpha L\big)^{-1} \nabla_\alpha L.$$

The $\nabla^2_\alpha L$ is the Hessian matrix for the parameter you want to update, and $\nabla_\alpha L$ is the gradient vector.

E- Use Part D to update W and b on the whole data (i.e., average over the whole data after finding all Hessian matrices and gradient vectors) and update until

$$\lambda = 0.5\left(\nabla f(x)^T H(x)^{-1} \nabla f(x)\right)$$

is smaller than a chosen ϵ (smaller than 0.01), or the number of iterations exceeds 1000.

F- Report the loss L on the train and test sets. Also plot L for train and test during the updates (i.e., update W and b, then calculate L, and continue the algorithm until the stopping criteria).

G- (Bonus) If another network is defined as

$$\hat{y}_i = W_2\, \sigma\big(W_1^T x_i + b\big) + b_2,$$

repeat parts B through F. Did it improve compared to the network used in the previous part (i.e., did the loss on the test set get smaller)?

Question 4.

Vector Orthogonalization & a Recommender Extension

In many recommendation schemes, each "item" (e.g., a movie or product) is represented by a numerical feature-vector. A user's "preference" can also be represented as a vector in the same feature space. One common approach is to compare (normalize) these vectors (e.g., via cosine similarity) and recommend the item whose feature-vector is most "similar" to the user's. Orthogonalizing item-feature vectors via Gram–Schmidt reveals which latent directions (features) are independent; projecting a new item or user-vector onto that orthonormal basis can help approximate missing ratings and produce better predictions.

4A. Gram–Schmidt Orthonormalization

Consider the following three item-feature vectors in $\mathbb{R}^3$:

$$a_1 = \begin{bmatrix}1\\2\\2\end{bmatrix}, \quad a_2 = \begin{bmatrix}2\\0\\1\end{bmatrix}, \quad a_3 = \begin{bmatrix}3\\2\\1\end{bmatrix}.$$

(a) Use the classical Gram–Schmidt process to produce an orthonormal set $\{q_1, q_2, q_3\}$ that spans the same column space as $\{a_1, a_2, a_3\}$. (Hint: after finding q_1, q_2 from a_1, a_2, subtract projections of a_3 onto both q_1 and q_2, then normalize.)

(b) Verify that

$$q_i^\top q_j = 0 \quad (i \neq j), \qquad \|q_i\| = 1, \quad i, j \in \{1,2,3\}.$$

4B. Projection and Approximation Error

A new user's "raw preference vector" $u \in \mathbb{R}^3$ is defined by the user's ratings on the first two items:

$$u = 4\,a_1 + 5\,a_2 \quad (\text{user rated item 1 as 4, item 2 as 5}).$$

(a) Project u onto the subspace spanned by $\{q_1, q_2\}$. That is, compute

$$\widehat{u} = \big(q_1 q_1^\top + q_2 q_2^\top\big)\, u.$$

(b) Compute the reconstruction error $\|u - \widehat{u}\|$. What does this error represent in the recommender context?

4C. Rating Prediction for a Third Item

We now wish to predict how this same user would rate item 3 (feature-vector a_3). Use two different approaches:

(i) **Cosine-Similarity Baseline:**

$$\cos\_\text{sim}(u, a_3) = \frac{u^\top a_3}{\|u\| \, \|a_3\|}.$$

Compute $\cos\_\text{sim}(u, a_3)$ and call this value $r_3^{(\cos)}$. Interpret it as the user's "predicted rating" (on a scale proportional to cosine similarity).

(ii) **Projection-Based Prediction:**

- First, use only $\{q_1, q_2\}$ (the basis from part 4A corresponding to items 1 and 2) to approximate the user: $\widehat{u} = (q_1 q_1^\top + q_2 q_2^\top)\, u$.
- Then compute

$$r_3^{(\text{proj})} = \frac{\widehat{u}^\top a_3}{\|a_3\|}.$$

In other words, measure how large the projection of $\widehat{u}$ is in the direction of a_3.

(i) Calculate $r_3^{(\text{proj})}$.

(ii) Compare $r_3^{(\cos)}$ vs. $r_3^{(\text{proj})}$. Which prediction is more "faithful" to the subspace spanned by the first two items? Explain briefly.

Instructions:

- Implement all steps in Python or Matlab, showing each Gram–Schmidt orthonormal vector q_1, q_2, q_3 in $\mathbb{R}^3$.
- Print the norms $\|q_i\|$ and dot-products $q_i^\top q_j$ to confirm orthonormality.
- Compute and print $\widehat{u}$, $\|u - \widehat{u}\|$, and both predictions $r_3^{(\cos)}$ and $r_3^{(\text{proj})}$.
- In part 4C(ii), briefly explain why projecting onto $\{q_1, q_2\}$ before predicting a_3 enforces that our estimate uses only the "latent directions" learned from items 1 and 2.

Question 5.

Manual "CNN-Style" Convolutions (No Training Required)

Convolutional Neural Networks often apply small filters (kernels) to 2D images or 3D data (e.g., RGB images). In this exercise, you will *not* train any weights—only perform the convolution sums (i.e., multiply-and-add) that a CNN "layer" would do. We will do two parts: a 2D convolution on a grayscale image and then apply the same 2D filter across each channel of a small "color" (3D) image. Assume *valid* convolution (no zero-padding), so that an $m \times n$ input with a $k \times k$ filter produces an $(m-k+1) \times (n-k+1)$ output per channel.

5A. 2D Convolution on a Small Grayscale Image

1. Let the 2D filter (kernel) be

$$H = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

and let the input (grayscale) image be the 4×4 matrix

$$X = \begin{bmatrix} 3 & 1 & 2 & 0 \\ 1 & 5 & 1 & 2 \\ 4 & 0 & 3 & 1 \\ 2 & 2 & 0 & 4 \end{bmatrix}.$$

2. Perform a *valid* 2D convolution of X with kernel H. That is, slide H over X without padding: at each position, compute

$$Y_{i,j} = \sum_{r=1}^{2}\sum_{c=1}^{2} H_{r,c}\, X_{i+r-1,\, j+c-1}.$$

3. Write down the resulting 3×3 output matrix Y.

5B. Applying the Same Filter to a 3D "Color" Image

1. Now suppose we have a tiny "RGB image" $Z \in \mathbb{R}^{3\times4\times4}$. That is, Z_c (the c-th channel) is a 4×4 matrix. Specifically, let

$$Z_1 = \begin{bmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 1 & 0 \\ 2 & 2 & 0 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}, \quad Z_2 = \begin{bmatrix} 0 & 2 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 1 & 0 & 2 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \quad Z_3 = \begin{bmatrix} 2 & 1 & 0 & 2 \\ 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 0 \\ 2 & 2 & 0 & 1 \end{bmatrix}.$$

2. Use the *same* 2×2 filter H from part 5A, i.e.

$$H = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

3. For each channel $c = 1, 2, 3$, perform a valid convolution of Z_c with H. That produces, for each c, a 3×3 result.

4. Stack those three 3×3 outputs to form a $3 \times 3 \times 3$ tensor. Write down all nine values for each channel output (i.e., show the three 3×3 result matrices).

Instructions:

- Implement both parts (5A and 5B) in Python or Matlab—*do not* use any built-in "conv2" function without showing the element-wise multiply and sum.
- Clearly print out the 3×3 matrix from 5A and, in 5B, the three 3×3 matrices (one per channel).