Homework 3: Advanced Regression, Logistics Regression, and Neural Network CPE 490/590 ST

Instructor: Rahul Bhadani

Due: March 10, 2025, 11:59 PM
170 points

You are allowed to use a generative model-based AI tool for your assignment. However, you must submit an accompanying reflection report detailing how you used the AI tool, the specific query you made, and how it improved your understanding of the subject. You are also required to submit screenshots of your conversation with any large language model (LLM) or equivalent conversational AI, clearly showing the prompts and your login avatar. Some conversational AIs provide a way to share a conversation link, and such a link is desirable for authenticity. Failure to do so may result in actions taken in compliance with the plagiarism policy.

Additionally, you must include your thoughts on how you would approach the assignment if such a tool were not available. Failure to provide a reflection report for every assignment where an AI tool is used may result in a penalty, and subsequent actions will be taken in line with the plagiarism policy.

Submission instruction:

Submission instruction for this homework supersedes one mentioned in the Syllabus.

This homework requires all answers recorded in a single `.ipynb` Python notebook. You may upload hand-written PDF for the theory portion. You can use a combination of text cell (i.e. markdown formatted cell) and code cell to provide your answer. To add equations you should be able to use Latex syntax in the text cells of your Python notebook. As a part of your submission, you must provide executed notebook with code, text, and outputs. Alternatively, you can also provide a url (whose permission you must change to 'anyone with link can view') of your Python notebook from Google Colab. The naming convention for your notebook should follow the format {firstname_lastname}_CPE 490/590 ST_hw03.ipynb. For example, if your name is Sam Wells, and you are enrolled in CPE 490 your file name should be sam_wells_CPE490_hw03.ipynb.

Please refer to https://github.com/rahulbhadani/CPE490_590_Sp2025/blob/master/Code/

for hands-on.

Theory

1 Neural Network by Hand (20 points)

Let's consider features of two training samples $\mathbf{x} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}] = [(x_1^{(1)}, x_2^{(1)}), (x_1^{(2)}, x_2^{(2)})] = [(2,4),(4,5)]$ and the response variable as $\mathbf{y} = [y^{(1)}, y^{(2)}] = [-1, 1]$. Note that when we don't write superscript, i.e. x_1 or x_2, we are simply considering features of *a* training sample without specifying which training sample.

Consider a classification problem for which we will develop a neural network model. The loss function to be considered is

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right)$$

Note that this is similar to the problem demonstrated in the class except we will calculate the loss by taking both training samples.

Here,

$$\frac{\partial \mathcal{L}}{\hat{y}^{(1)}} = -\frac{y^{(1)}}{\hat{y}^{(1)}} + \frac{1-y^{(1)}}{1-\hat{y}^{(1)}}$$

and

$$\frac{\partial \mathcal{L}}{\hat{y}^{(2)}} = -\frac{y^{(2)}}{\hat{y}^{(2)}} + \frac{1-y^{(2)}}{1-\hat{y}^{(2)}}$$

Individual training sample i will lead to the gradient with respect to a weight W as:

$$\nabla_W \mathcal{L}^{(i)} = \frac{\partial \mathcal{L}}{W^{(i)}} = \frac{\partial \mathcal{L}}{\hat{y}^{(i)}} \times \frac{\hat{y}^{(i)}}{\partial z^{(i)}} \times \frac{\partial z^{(i)}}{\partial W^{(i)}} \tag{1}$$

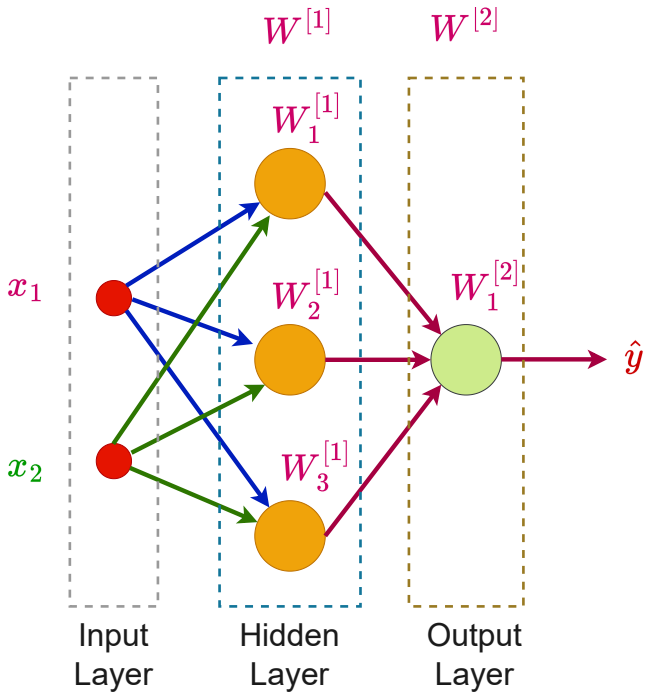

Figure 1: Homework 3: Advanced Regression,Logistics Regression,and Neural Network: Q1

In Equation (1), superscript (i) denotes that gradient is calculated for individual samples. Then we calculate the average gradient as

$$\nabla_W \mathcal{L} = \frac{1}{n} \sum_{i}^{n} \frac{\partial \mathcal{L}}{W^{(i)}} \tag{2}$$

so that the update rule for stochastic gradient descent becomes:

$$W = W - \eta \nabla_W \mathcal{L} \tag{3}$$

What it means is that you need to calculate the gradient for both training samples, take the average, and then update the weights.

Now, consider a neural network with one hidden layer containing three neurons so that the architecture looks as shown in Figure 1. In this problem, we won't consider bias, that is bias $b = 0$ at all neurons of the given Neural Network.

Further, let's consider the following initial values of weights:

$$\begin{aligned} W_1^{[1]} &= [0.1, 0.3] \\ W_2^{[1]} &= [0.2, 0.4] \\ W_3^{[1]} &= [0.1, 0.5] \\ W_1^{[2]} &= [0.1, 0.2, 0.1] \end{aligned} \tag{4}$$

Further, we will also consider the activation functions at all four neurons to be sigmoid, i.e.

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

Note that the derivative of a with respect to z is given

$$\frac{\partial a}{\partial z} = \sigma(z)\Big(1 - \sigma(z)\Big)$$

You are required to complete the following task (only for one epoch) **(5 points each)**:

Use $\eta = 0.05$.

1. Perform the forward propagation on the first hidden layer and write down $z_1^{[1]}$, $z_2^{[1]}$, $z_3^{[1]}$, $a_1^{[1]}$, $a_2^{[1]}$, and $a_3^{[1]}$ for **both training samples**.
2. Perform the forward propagation on the output layer and write down $z_1^{[2]}$, $a_1^{[2]}$ for **both training samples**.
3. Calculate the final average loss value taking into account both training samples.
4. Perform the backward propagation to write down the updated values of $W_1^{[2]}$. Note the vector $W_1^{[2]}$ will contain three values as opposed to the two values you saw in the example done in the classroom. Don't forget to average the gradient using both training samples.

Answer

For the first training sample $(2, 4)$:

$$z_1^{[1]} = 2 \times 0.1 + 4 \times 0.3 = 1.4$$

$$z_2^{[1]} = 2 \times 0.2 + 4 \times 0.4 = 2.0$$

$$z_3^{[1]} = 2 \times 0.1 + 4 \times 0.5 = 2.2$$

$$a_1^{[1]} = \frac{1}{1 + e^{-1.4}} = 0.802$$

$$a_2^{[1]} = \frac{1}{1 + e^{-2.0}} = 0.880$$

$$a_3^{[1]} = \frac{1}{1 + e^{-2.2}} = 0.900$$

$$z_1^{[2]} = 0.802 \times 0.1 + 0.880 \times 0.2 + 0.900 \times 0.1 = 0.346$$

$$a_1^{[2]} = \frac{1}{1 + e^{-0.346}} = 0.586$$

$$\mathcal{L}^{(1)} = -[(-1)\log(0.586) + (1 - (-1))\log(1 - 0.586)] = 1.230$$

$$\frac{\partial \mathcal{L}}{\hat{y}^{(1)}} = -\frac{-1}{0.586} + \frac{1 - (-1)}{1 - 0.586} = 6.536$$

$$\frac{\hat{y}^{(1)}}{\partial z^{(1)}} = 0.586 \times (1 - 0.586) = 0.243$$

$$\frac{\partial z^{(1)}}{\partial W^{(1)}} = [0.802, 0.880, 0.900]$$

$$\frac{\partial \mathcal{L}}{W^{(1)}} = 6.536 \times 0.243 \times [0.802, 0.880, 0.900] = [1.274, 1.398, 1.429]$$

For the second training sample $(4, 5)$:

$$z_1^{[1]} = 4 \times 0.1 + 5 \times 0.3 = 1.9$$

$$z_2^{[1]} = 4 \times 0.2 + 5 \times 0.4 = 2.8$$

$$z_3^{[1]} = 4 \times 0.1 + 5 \times 0.5 = 2.9$$

$$a_1^{[1]} = \frac{1}{1 + e^{-1.9}} = 0.869$$

$$a_2^{[1]} = \frac{1}{1 + e^{-2.8}} = 0.943$$

$$a_3^{[1]} = \frac{1}{1 + e^{-2.9}} = 0.948$$

$$z_1^{[2]} = 0.869 \times 0.1 + 0.943 \times 0.2 + 0.948 \times 0.1 = 0.370$$

$$a_1^{[2]} = \frac{1}{1 + e^{-0.370}} = 0.591$$

$$\mathcal{L}^{(2)} = -[(1)\log(0.591) + (1-1)\log(1-0.591)] = 0.526$$

Final average loss = $(\mathcal{L}^{(1)} + \mathcal{L}^{(2)})/2 = (1.230 + 0.526)/2 = 0.878$

$$\frac{\partial \mathcal{L}}{\hat{y}^{(2)}} = -\frac{1}{0.591} + \frac{1-1}{1-0.591} = -1.692$$

$$\frac{\hat{y}^{(2)}}{\partial z^{(2)}} = 0.591 \times (1 - 0.591) = 0.242$$

$$\frac{\partial z^{(2)}}{\partial W^{(2)}} = [0.869, 0.943, 0.948]$$

$$\frac{\partial \mathcal{L}}{W^{(2)}} = -1.692 \times 0.242 \times [0.869, 0.943, 0.948] = [-0.356, -0.386, -0.388]$$

$$\nabla_W \mathcal{L} = \left[\frac{1.274 - 0.356}{2}, \frac{1.398 - 0.386}{2}, \frac{1.429 - 0.388}{2}\right] = [0.459, 0.506, 0.520]$$

Consider learning rate $\eta = 0.05$,

Updated

$$W_1^{[2]} = [0.1, 0.2, 0.1] - 0.05 \times [0.459, 0.506, 0.520] = [0.0770, 0.175, 0.074]$$

Note: in the above calculation, at every step, the result was rounded off to three places after the decimal. If you didn't do that way, or wrote a computer program to make calculation, your result may slightly vary.

2 Dropout (5 points)

What is dropout in the context of a neural network and how is it beneficial in training a neural network?

Answer

Dropout works by randomly "dropping out" (i.e., setting to zero) a number of output features of the neurons during the training process. During each training iteration, each neuron (with dropout applied) has a probability p of being temporarily "dropped," meaning that it's temporarily removed from the network along with all its incoming and outgoing connections. The value p is the dropout rate, and it's typically set between 0.2 and 0.5. At test time, dropout is not applied, and instead, the weights are scaled appropriately to adjust for the fact that more neurons are active than during training.
Benefits of Dropout:

1. Dropout prevents co-adaptation of neurons by forcing them to learn robust features that are useful in conjunction with many different random subsets of the other neurons.
2. It effectively trains a large number of different "thinned" networks with various combinations of neurons. During testing, it's like combining the predictions from a large number of different models.
3. Dropout introduces noise into the training process, fostering the learning of noise-resistant features within the network. This noise forces the network not to rely too heavily on any single input feature, as that feature could be "dropped" at any time during training.

3 Activation Functions (5 points)

What is the role of activation functions in neural networks, and how do they impact the network's ability to model complex functions?

Answer

Activation functions play a critical role in neural networks by introducing non-linear properties to the system, which enables the network to learn complex patterns in the data. Without activation functions, a neural network would essentially be a linear regression model, which is limited to linear decision boundaries. Activation functions allow the network to create non-linear mappings from inputs to outputs. This non-linearity is what lets neural networks model complex and nonlinear relationships between the input features and the target outputs.

4 Softmax Functions (5 points)

Explain the difference between a softmax function and a sigmoid function when used in the output layer of a fully connected neural network. In which scenarios would you choose one over the other?

Answer

The softmax function is used primarily for multiclass classification where the classes are mutually exclusive. In other words, each instance is expected to belong to only one class. When applied in the output layer, the softmax function takes an unnormalized vector (logits) and normalizes it into a probability distribution consisting of probabilities proportional to the exponentials of the input numbers. The sum of the output probabilities will be 1, which makes the output directly interpretable as class probabilities.
The sigmoid function, also known as the logistic function, is used for binary classification or when each instance can belong to multiple classes (multi-label classification). It takes a real-valued number and squashes it into a range between 0 and 1. This output can be interpreted as the probability that the instance belongs to one class, with 1 indicating certainty for the class and 0 certainty against the class.

5 Logistics Regression by Hand (20 Points)

The following dataset consists of 4 training examples, where $x_k^{(i)}$ denotes the k-th dimension of the i-th training example $\mathbf{x}^{(i)}$, and $y^{(i)}$ is the corresponding label ($k \in \{1,2,3\}$ and $i \in \{1,2,3,4\}$).

i	x_1	x_2	x_3	y
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0

A binary logistic regression model is trained on this dataset, and the parameter vector $\boldsymbol{\theta}$ after training is

$$\boldsymbol{\theta} = \begin{bmatrix} 1.5 & 2 & 1 \end{bmatrix}^T.$$

Note: There is **no intercept term** used in this problem.

Use the data above to answer the following questions. For all numerical answers, please use one number rounded to the fourth decimal place; e.g., 0.1234. Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur.

1. Calculate $J(\boldsymbol{\theta})$, $\frac{1}{N}$ times the negative log-likelihood over the given data and parameter $\boldsymbol{\theta}$. (Note here we are using natural log, i.e., the base is e).
2. Calculate the gradients $\dfrac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ with respect to θ_j for all $j \in \{1,2,3\}$.
3. Update the parameters following the parameter update step $\theta_j \leftarrow \theta_j - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ and write the updated (numerical) value of the vector $\boldsymbol{\theta}$. Use learning rate $\eta = 1$.

Answer

Part 1:
We are asked to calculate the logistic regression cost function $J(\boldsymbol{\theta})$, which is defined as:

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right],$$

where:

- N is the number of training examples ($N = 4$),
- $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)})$ is the sigmoid function,
- $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic (sigmoid) function.

The parameter vector is given as:

$$\boldsymbol{\theta} = \begin{bmatrix} 1.5 \\ 2 \\ 1 \end{bmatrix}.$$

For each training example $\mathbf{x}^{(i)}$, compute:

$$z^{(i)} = \boldsymbol{\theta}^T\mathbf{x}^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)},$$

and then apply the sigmoid function:

$$h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}.$$

Training Sample 1:

$$z^{(1)} = 1.5(0) + 2(0) + 1(1) = 1, \quad h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) = \frac{1}{1+e^{-1}} \approx 0.7311.$$

Training Sample 2:

$$z^{(2)} = 1.5(0) + 2(1) + 1(0) = 2, \quad h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) = \frac{1}{1+e^{-2}} \approx 0.8808.$$

Training Sample 3:

$$z^{(3)} = 1.5(0) + 2(1) + 1(1) = 3, \quad h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) = \frac{1}{1+e^{-3}} \approx 0.9526.$$

Training Sample 4:

$$z^{(4)} = 1.5(1) + 2(0) + 1(0) = 1.5, \quad h_{\boldsymbol{\theta}}(\mathbf{x}^{(4)}) = \frac{1}{1+e^{-1.5}} \approx 0.8176.$$

$$J(\boldsymbol{\theta})_i = -\left[y^{(i)} \log(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right],$$

Training Sample 1 ($y^{(1)} = 0$):

$$J(\boldsymbol{\theta})_1 = -\left[0 \cdot \log(0.7311) + (1-0) \cdot \log(1 - 0.7311)\right] = -\log(0.2689) \approx 1.3133.$$

Training Sample 2 ($y^{(2)} = 1$):

$$J(\boldsymbol{\theta})_2 = -\left[1 \cdot \log(0.8808) + (1-1) \cdot \log(1 - 0.8808)\right] = -\log(0.8808) \approx 0.1278.$$

Training Sample 3 ($y^{(3)} = 1$):

$$J(\boldsymbol{\theta})_3 = -\left[1 \cdot \log(0.9526) + (1-1) \cdot \log(1 - 0.9526)\right] = -\log(0.9526) \approx 0.0488.$$

Training Sample 4 ($y^{(4)} = 0$):

$$J(\boldsymbol{\theta})_4 = -\left[0 \cdot \log(0.8176) + (1-0) \cdot \log(1 - 0.8176)\right] = -\log(0.1824) \approx 1.7022.$$

Total cost is :

$$J(\boldsymbol{\theta}) = \frac{1}{4}(1.3133 + 0.1278 + 0.0488 + 1.7022) = 0.7980$$

Part 2:
The gradient $J(\theta)$ with respect to θ_j is

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{N} \sum_{i=1}^{N} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

We already calculated
$h_{\boldsymbol{\theta}}(\mathbf{x}) = [0.7311, 0.8808, 0.9526, 0.8176]^\top$
Our design matrix X is

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5}$$

The error term $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}$ is

$$\begin{aligned} h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} &= [0.7311 - 0, 0.8808 - 1, 0.9526 - 1, 0.8176 - 0] \\ &= [0.7311, -0.1192, -0.0474, 0.8176] \end{aligned} \tag{6}$$

We multiply error term with features from each column of the design matrix to compute the gradient:
Gradient for θ_1:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{1}{4}\left[(0.7311-0)(0)+(0.8808-1)(0)+(0.9526-1)(0)+(0.8176-0)(1)\right].$$

Simplify:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{1}{4}\cdot(0.8176) = 0.2044.$$

Gradient for θ_2:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} = \frac{1}{4}\left[(0.7311-0)(0)+(0.8808-1)(1)+(0.9526-1)(1)+(0.8176-0)(0)\right].$$

Simplify:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} = \frac{1}{4}\cdot(-0.1192-0.0474) = \frac{1}{4}\cdot(-0.1666) = -0.0417.$$

Gradient for θ_3:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_3} = \frac{1}{4}\left[(0.7311-0)(1)+(0.8808-1)(0)+(0.9526-1)(1)+(0.8176-0)(0)\right].$$

Simplify:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_3} = \frac{1}{4}\cdot(0.7311-0.0474) = \frac{1}{4}\cdot(0.6837) = 0.1709.$$

Thus, the gradients are:

$$\begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_3} \end{bmatrix} = \begin{bmatrix} 0.2044 \\ -0.0417 \\ 0.1709 \end{bmatrix} \tag{7}$$

Part 3:
Weights are updated as:

$$\begin{aligned} \theta_j &\leftarrow \theta_j - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i} \\ \theta_1 &\leftarrow 1.5 - 0.2044 = 1.2956 \\ \theta_2 &\leftarrow 2 + 0.0417 = 2.0417 \\ \theta_3 &\leftarrow 1 - 0.1709 = 0.8291 \end{aligned} \tag{8}$$

6 Locally-weighted Regression (10 Points)

Let the training set be denoted by

$\{(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(N)}, y^{(N)})\}$. Cosnider positive weights $a^{(1)}, \cdots, a^{(N)}$. We define the weighted least squares problem as follows:

$$\hat{\mathbf{w}} = \arg\min \frac{1}{2} \sum_{i=1}^{N} a^{(i)} \left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} \right)^2 + \frac{\lambda}{2} ||\mathbf{w}||^2 \tag{9}$$

Show that the closed-form solution to the weighted least squares problem is given by the formula below.

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{A} \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{A} \mathbf{y} \tag{10}$$

where $\mathbf{X}$ is $N \times D$ design matrix, $\mathbf{A}$ is a diagonal matrix where $\mathbf{A}_{ii} = a^{(i)}$.

In this case, we define the loss function is defined to be the total loss (not the average loss) over all the training examples.

Answer

In the matrix form, the cost function can be written as

$$J(\mathbf{w}) = \frac{1}{2} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top \mathbf{A} (\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}$$

$\mathbf{X}$ is the design matrix, the weight matrix $\mathbf{A} \in N \times N$ is a diagonal matrix with entries as $A_{ii} = a^{(i)}$.
Expanding in the quadratic form,

$$J(\mathbf{w}) = \frac{1}{2} [\mathbf{y}^\top \mathbf{A} \mathbf{y} - \mathbf{y}^\top \mathbf{A} \mathbf{X} \mathbf{w} - \mathbf{w}^\top \mathbf{X}^\top \mathbf{A} \mathbf{y} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{A} \mathbf{X} \mathbf{w}] + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}$$

Taking the derivative with respect to $\mathbf{w}$ to the gradient,

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -\mathbf{X}^\top \mathbf{A} \mathbf{y} + \mathbf{X}^\top \mathbf{A} \mathbf{X} \mathbf{w} + \lambda \mathbf{w}$$

Setting it to zero:

$$-\mathbf{X}^\top \mathbf{A} \mathbf{y} + \mathbf{X}^\top \mathbf{A} \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = 0$$

We rearrange as follows:

$$(\mathbf{X}^\top \mathbf{A}\mathbf{X} + \lambda \mathbf{I})\mathbf{w} = \mathbf{X}^\top \mathbf{A}\mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{A}\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^\top \mathbf{A}\mathbf{y}$$

Since this is the optimal value, we can rewrite it as

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{A}\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^\top \mathbf{A}\mathbf{y}$$

$\hat{\mathbf{w}}$ minimizes the given cost function for the weight least squares problem.

Practice

See notebook for rest of the solution.

7 Classification with Neural Network (25 Points)

For this task, you will be working with the diabetes dataset (see https://github.com/rahulbhadani/CPE490_590_Sp2025/tree/master/Data/Diabetes). Your goal is to implement a Neural Network model that can classify whether a given training sample indicates the presence of diabetes (Outcome = 1) or not (Outcome = 0).

1. **Data Standardization (2 points)** Standardize the dataset using the StandardScaler from the sklearn library.
2. **Neural Network Implementation (5 points)** Implement a Neural Network with the following specifications:
   - The network should have two hidden layers.
   - The first hidden layer should contain 4 neurons.
   - The second hidden layer should contain 3 neurons.
3. **Training the Model (5 points)** Train the neural network using the Adam optimizer with a learning rate of 0.01. The training should be carried out for 1000 epochs. Choose the appropriate loss function for the classification problem.
4. **Loss Plot (3 points)** After training the model, plot the loss against the number of epochs.

5. **Model Saving (5 points)** Save the trained model for future use or deployment in ONNX format.

6. **Model Prediction (5 points)** Load the trained model you saved in the previous step, predict the outcome of diabetes (either 0 or 1) for the following data:

Feature	Value
Pregnancies	7
Glucose	149
BloodPressure	73
SkinThickness	94
Insulin	94
BMI	32
DiabetesPedigreeFunction	0.672
Age	45

Table 1: Dataset

8 Logistics Regression to Predict Equipment Failure (35 Points)

In this problem, we will use Logistics Regression to predict equipment faults during the wafer fabrication process of the semiconductor industries. Dataset has been added to the course Github repo at https://github.com/rahulbhadani/CPE490_590_Sp2025/tree/master/Data/secom

First, read the description of the dataset from the original source : https://archive.ics.uci.edu/dataset/179/secom. The dataset contains features and labels in two separate files. Label -1 means failure and + 1 means success. Based on your observation, answer the following question or perform the given task:

1. For a given sample, some features may contain NaN (not a number). NaN represents a missing value. Remove all the features where more than 20% samples contain NaN. **(5 Points)**

2. For the remaining samples, replace the NaN with the mean of the given feature by taking mean from the entire sample that doesn't contain NaN. **(5 Points)**

3. What is the percentage of remaining sample attributed to failure? Create an appropriate visualization to convey your message. Is there class imbalance in the dataset? **(5 Points)**

4. Scale the dataset using StandardScaler. **(5 Points)**

5. Use Logistics Regression with maximum iteration of 250 to create a prediction model. One-third of the sample must be test dataset that you should divide randomly. **(5 Points)**

6. After training, compute the accuracy, F1-score, precision, and recall on the training and the test dataset. **(5 Points)**

7. Repeat the step 5 and 6 five times, and gather all sets of accuracy, F1-score, precision, and recall and plot a box plot showing the variation of these metrics. What conclusion do you make from the boxplot? **(5 Points)**

9 Multiple Regression with PyTorch (45 Points)

We will build a multiple linear regression model for predicting the net hourly electrical energy output (PE) in MegaWatt. Downlaod the dataset from https://github.com/rahulbhadani/CPE490_590_Sp2025/tree/master/Data/CombinedCyclePowerPlant. The dataset contains following features:

- $\mathbf{x}_1$: Ambient Temperature (AT) in °C,
- $\mathbf{x}_2$: Exhaust Vacuum (V) in cm Hg x_2
- $\mathbf{x}_3$: Ambient Pressure (AP) in milibar x_3
- $\mathbf{x}_4$: Relative Humidity (RH) in percent x_4

Perform the following task and answer the following questions:

1. Create a scatterplot matrix figure that shows correlation among features as well as features and the target variable. **(5 Points)**

2. We are interested in creating a multiple linear regression model

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$$

Perform multiple linear regression using above model using Statsmodel pakcage. However, for this purpose use the Cauchy Loss function:

$$\ell(\hat{y}, y) = \frac{c^2}{2} \log\left(1 + \left(\frac{y - \hat{y}}{c}\right)^2\right) \tag{11}$$

where c is the Cauchy loss function constant. In this question, we will use $c = 1$.

(10 Points)

3. What are the coefficient values for the fitted model? **(5 Points)**

4. What's the final loss value? **(5 Points)**
5. What does the 95% confidence interval for the coefficients tell you? **(5 Points)**
6. Create the residual plot $\hat{y} - y$ against x_1, x_2, x_3, and x_4. What do you conclude? **(5 Points)**
7. Provide the alternative implementation using PyTorch. Use appropriate values for learning rate and number of epochs. **(10 Points)**