HW 2 - Linear Model, Logistic Regression and Classification

Complete the following questions and resubmit this entire notebook to canvas.

- For questions that ask you to derive or find a quantity use a **text cell** to show your calculations.
- Use markdown to write math expressions (as was done to create these problems) and make sure to show your work.
- It doesnt have to be perfect looking but it needs to be readible.
- You may also submit a legible picture of your derivation
- For questions that ask you compute something or write code use a **code cell** to write your code.
- You can create additional code cells as needed.
- Just make sure your code is commented, the functions are named appropriately, and its easy to see your final answer.
- The total points on this homework is 100. Out of these 5 points are reserved for clarity of presentation, punctuation and commenting with respect to the code.

SUBMISSION

When you submit you will submit a pdf file **and** the notebook file. The TA will use the pdf file to grade more quickly. The notebook file is there to confirm your work.

To generate a pdf file

1. Click File
2. Click print
3. Set the destinationas "save as pdf"
4. Hit print

Title the pdf file `LASTNAME-FIRSTNAME-HW2.pdf` Title your notebook file as `LASTNAME-FIRSTNAME-HW2.ipynb`

Submit both files.

Do not actually print your notebook out (what year is this?)

In [1]:
```
# libraries and functions you may find useful
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from sklearn.model_selection import train_test_split
```

```
from tqdm.notebook import trange
```

Q1 - Linear Model (25 points)

Let

- $X_{n\times p}$ be a data matrix with n observations and p features
- $Y_{n\times 1}$ be a response matrix with n observations and 1 outcomes
- $\beta_{p\times 1}$ be an unknown p dimensional **slope** vector

We assume a linear model to predict Y using X

- $Y = X\beta + \epsilon$
- $\epsilon \sim N(0, \sigma^2 I_n)$

part 1 - Gradient Descent (5 points)

We will use the following loss function:

$$L(\beta) = (Y - X\beta)^T(Y - X\beta)$$

Show that the gradient of $\ell(\beta)$ with respect to β is

$$\nabla\ell(\beta) = -2X^T(Y - X\beta)$$

Gradient Descent (Answer)

To find the gradient of the loss function:

$$L(\beta) = (Y - X\beta)^T(Y - X\beta)$$

First expand the equation:

$$L(\beta) = Y^TY - 2Y^TX\beta + \beta^TX^TX\beta$$

Differentiating each term:

$$\frac{d}{d\beta}Y^TY = 0$$

$$\frac{d}{d\beta}(-2Y^TX\beta) = -2X^TY$$

$$\frac{d}{d\beta}(\beta^TX^TX\beta) = 2X^TX\beta$$

Combine the term:

$$\nabla\ell(\beta) = -2X^TY + 2X^TX\beta$$

Once simplfied the gradient of the loss function is:

$$\nabla \ell(\beta) = -2X^T(Y - X\beta)$$

This equation is the same as the gradient equation shown above.

part 2 - Exact solution (5 points)

We want to minimize $L(\beta)$ so we find the point $\hat{\beta}$ where $\nabla L(\hat{\beta}) = 0$.

Set

$$\nabla L(\beta) = 0$$

to show that

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Exact solution (Answer)

If we set $\nabla L(\beta) = 0$. Then we get the following equation:

$-2X^T(Y - X(\hat{\beta})) = 0$, where $\hat{\beta}$ is the miminize value.

To solve for $\hat{\beta}$ we first expand the equation.

$$-2X^T(Y - X(\hat{\beta})) = 0$$

$$X^T(Y - X(\hat{\beta})) = 0$$

$$X^T Y - X^T X\hat{\beta} = 0$$

Rearrange so $\hat{\beta}$ is on the left, by multiplying both sides by $(X^T X)^{-1}$:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

This solution for $\hat{\beta}$, is the same as the solution shown above:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

part 3 - Application (10 points)

Now, we're going to estimate β in two ways. One with gradient descent and one with the exact method.

We will use the `diabetes` dataset from sklearn to test our method. The code below will load this dataset and store the features in a matrix called `x` and the targets in a vector called `y`. Then we will split the data further into training and testing.

First, lets write two functions in python

1. Write a function called `loss_grad()` that takes a covariate matrix X, target vector Y, and a parameter vector β and computes $\nabla \ell(\beta)$. `Use the gradient that you derived in previous question`

2. Write a function called `exact_beta_hat()` that takes a covariate matrix X and target vector Y and computes the exact solution.

Now we will fit a linear model in two ways.

1. Gradient descent. Write a gradient descent loop with your `loss_grad()` function to estimate β with `x_train` and `y_train` (don't forget to choose an appropriate γ). Store this in a variable called `beta_gd`

2. The exact method. Compute the exact estimate $\hat{\beta}$ with your function `exact_beta_hat()` on `x_train` and `y_train` and store the output in a variabled called `beta_exact`

Compare `beta_gd` and `beta_exact` . Compare the MSE of each of your fitted models on the test data.

In [2]:
```
from sklearn import datasets
x, y = datasets.load_diabetes(return_X_y = True)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, ra
```

In [3]:
```
# helper functions

# loss_grad()
def loss_grad(X, Y, beta):
    return -2 * X.T @ (Y - X @ beta)

# exact_beta_hat()
def exact_beta_hat(X, Y):
    return np.linalg.inv(X.T @ X) @ (X.T @ Y)  # Compute (X^T X)^(-1) * X^T
```

In [4]:
```
# gradient descent and exact soln

beta = np.zeros((x_train.shape[1], 1))  # Initialize beta for p features (co

# Set parameters
lr = 0.001  # Learning rate
num_iters = 1000  # Number of iterations
tol = 1e-6  # Convergence tolerance

# Gradient Descent Loop
for i in range(num_iters):
    grad = loss_grad(x_train, y_train, beta)  # Compute the gradient
    beta_new = beta - lr * grad  # Update beta by moving against the gradien

    # Check for convergence (if the change in beta is small)
```