Cmpsc 448

Problem 1)

original OLS objective: $f_{OLS}(w) = \frac{1}{2}\sum_{i=1}^{n}(w \cdot x_i - y_i)^2$

to get w take the derivative $\frac{\partial f_{OLS}}{\partial w} = \sum_{i=1}^{n}(w \cdot x_i - y_i)\, x_i$

$$w = \frac{\sum x_i y_i}{\sum x_i^2}$$

since the means are 0, we can rewrite it as $\frac{\sum x_i y_i}{n-1} \cdot \frac{n-1}{\sum x_i^2} = \frac{Cov(X,Y)}{Var(X)}$

$0 = \sum w x_i^2 - x_i y_i \rightarrow \sum w x_i^2 = \sum x_i y_i$

mistake OLS objective: $f_{OLS}(\hat{w}) = \frac{1}{2}\sum_{i=1}^{n}(w y_i - x_i)^2$

to get $\hat{w}$ take the derivative $\frac{\partial f_{OLS}(\hat{w})}{\partial \hat{w}} = \sum_{i=1}^{n}(w y_i - x_i)\, y_i$

$$\hat{w} = \frac{\sum x_i y_i}{\sum y_i^2}$$

Since the means are 0, we can rewrite as $\frac{\sum x_i y_i}{n-1} \cdot \frac{n-1}{\sum y_i^2} = \frac{Cov(X,Y)}{Var(Y)}$

$0 = \sum \hat{w} y_i^2 - x_i y_i \rightarrow \sum \hat{w} y_i^2 = \sum x_i y_i$

relating w to $\hat{w}$

since $w = \frac{Cov(X,Y)}{Var(X)}$ and $\hat{w} = \frac{Cov(X,Y)}{Var(Y)}$

$$\frac{w}{\hat{w}} = \frac{Cov(X,Y)}{Var(X)} \cdot \frac{Var(Y)}{Cov(X,Y)} = \frac{Var(Y)}{Var(X)} \rightarrow w = \hat{w}\frac{Var(Y)}{Var(X)}$$

to get $w = \hat{w}$, we need the variances of x and y to be equal

Problem 2) $\hat{y} = w_1 x_{i1} + w_2 x_{i2}$ $\quad \vec{x}_i = [x_{i1}, x_{i2}]^T$

objective for OLS is $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ $\quad \vec{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$ $\quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

$$f(w_1, w_2) = \frac{1}{2}\sum(w_1 x_{i1} + w_2 x_{i2} - y_i)^2$$

to find the optimal solution, take gradient of function

use given formula for multidimensional OLS solution: $w^* = \left(\sum \vec{x}_i \vec{x}_i^T\right)^{-1} \sum y_i \vec{x}_i$

$\vec{x}_i \vec{x}_i^T = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}[x_{i1}\ x_{i2}] = \begin{bmatrix} x_{i1}^2 & x_{i1}x_{i2} \\ x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}$ $\quad y_i \vec{x}_i = \begin{bmatrix} x_{i1} y_i \\ x_{i2} y_i \end{bmatrix}$ $\quad \sum y_i \vec{x}_i = \begin{bmatrix} \sum x_{i1} y_i \\ \sum x_{i2} y_i \end{bmatrix}$

$\sum \vec{x}_i \vec{x}_i^T = \begin{bmatrix} \sum x_{i1}^2 & \sum x_{i1}x_{i2} \\ \sum x_{i1}x_{i2} & \sum x_{i2}^2 \end{bmatrix}$ $\quad \left(\sum \vec{x}_i \vec{x}_i^T\right)^{-1} = \frac{1}{\sum x_{i1}^2 \sum x_{i2}^2 - \sum x_{i1}x_{i2} \sum x_{i1}x_{i2}} \begin{bmatrix} \sum x_{i2}^2 & -\sum x_{i1}x_{i2} \\ -\sum x_{i1}x_{i2} & \sum x_{i1}^2 \end{bmatrix}$

$$\vec{w}^* = \begin{bmatrix} \frac{Var(x_2)}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} & \frac{-Cov(x_1,x_2)}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} \\ \frac{-Cov(x_1,x_2)}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} & \frac{Var(x_1)}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} \end{bmatrix} \begin{bmatrix} \sum x_{i1} y_i \\ \sum x_{i2} y_i \end{bmatrix}$$

$\sum x_{i1}^2 = Var(x_1)$

$\sum x_{i2}^2 = Var(x_2)$

$\sum x_{i1}x_{i2} = Cov(x_1,x_2)$

$$\vec{w}^* = \begin{bmatrix} \frac{Var(x_2)\sum x_{i1} y_i - Cov(x_1,x_2)\sum x_{i2} y_i}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} \\ \frac{Var(x_1)\sum x_{i2} y_i - Cov(x_1,x_2)\sum x_{i1} y_i}{Var(x_1)Var(x_2) - Cov^2(x_1,x_2)} \end{bmatrix} = \begin{bmatrix} w_1^* \\ w_2^* \end{bmatrix}$$

now solve for each feature seperately

$f_{ols}(\hat{y}) = \frac{1}{2}\sum(\hat{y}_i - y_i)^2 = \frac{1}{2}\sum(w_1 x_{i1} + w_2 x_{i2} - y_i)^2$

Since we are solving for just w_1, we set $w_2 = 0$

$f_{ols}(w_1) = \frac{1}{2}\sum(w_1 x_{i1} - y_i)^2$

$\frac{\partial f}{\partial w_1} = \sum(w_1 x_{i1} - y_i)x_{i1} = 0 \rightarrow \sum w_1 x_{i1}^2 = \sum x_{i1} y_i \rightarrow w_1 = \frac{\sum x_{i1} y_i}{\sum x_{i1}^2} = \frac{Cov(x_1, y)}{Var(x_1)}$

$\frac{\partial f}{\partial w_2} = \sum(w_2 x_{i2} - y_i)x_{i2} = 0 \rightarrow \sum w_2 x_{i2}^2 = \sum x_{i2} y_i \rightarrow w_2 = \frac{\sum x_{i2} y_i}{\sum x_{i2}^2} = \frac{Cov(x_2 y)}{Var(x_2)}$

$w_1'^* = \frac{Cov(x_1 y)}{Var(x_1)} \qquad w_2'^* = \frac{Cov(x_2 y)}{Var(x_2)}$

The two values for the 2-dimensional and the 1-dimensional solutions are different. The weights are different to eachother. The scenario where these weights would be the same is if the two variables were uncorrelated.

$w_1^* = \frac{Var(x_2)\sum x_{i1} y_i - Cov(x_1 x_2)\sum x_{i2} y_i}{Var(x_1)Var(x_2) - Cov^2(x_1 x_2)} \overset{\text{if uncorrelated } Cov(x_1, x_2) = 0}{\rightarrow} \frac{Var(x_2)\sum x_{i1} y_i - 0}{Var(x_1)Var(x_2) - 0} = \frac{\sum x_{i1} y_i}{Var(x_1)} = w_1'$

Problem 3)

objective function for Ridge: $\frac{1}{2}\sum(w^T x_i - y_i)^2 + \lambda \|w\|_2^2$

find w_1^*: $\frac{1}{2}\sum(w_1 x_{i1} - y_i)^2 + \lambda w_1^2$

take derivative $\frac{\partial f}{\partial w_1} = \sum(w_1 x_i - y_i)x_i + 2\lambda w_1 = 0 \rightarrow \sum(w_1 x_i^2 - x_i y_i) + 2\lambda w_1 = 0$

$w_1 \sum x_i^2 - \sum x_i y_i + w_1(2\lambda) \rightarrow w_1 = \frac{\sum x_i y_i}{\sum x_i^2 + 2\lambda} \qquad w_1^* = \frac{\sum x_i y_i}{\sum x_i^2 + 2\lambda}$

now find $[w_1'^*, w_2'^*]^T \qquad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

$f(w_1, w_2) = \frac{1}{2}\sum(w_1 x_i + w_2 x_i - y_i)^2 + \lambda(w_1^2 + w_2^2)$

take gradient $\frac{\partial f}{\partial w_1} = \sum(w_1 x_i + w_2 x_i - y_i)x_i + 2\lambda w_1 \rightarrow \sum[(w_1 + w_2)x_i^2 - x_i y_i] + 2\lambda w_1 = 0$

$= \sum w_1 x_i^2 + \sum w_2 x_i^2 - \sum x_i y_i + 2\lambda w_1 = 0 \rightarrow \frac{\sum x_i y_i - \sum w_2 x_i^2}{\sum x_i^2 + 2\lambda} = w_1$

$\frac{\partial f}{\partial w_2} = \sum(w_1 x_i + w_2 x_i - y_i)x_i + 2\lambda w_2 \rightarrow \frac{\sum x_i y_i - \sum w_1 x_i^2}{\sum x_i^2 + 2\lambda} = w_2$

the equations for w_1 and w_2 are essentially the same equations

we can simplify the equations further $w_1'^* = \frac{Cov(x\,y) - w_2 Var(x)}{Var(x) + 2\lambda}$

$w_2'^* = \frac{Cov(x\,y) - w_1 Var(x)}{Var(x) + 2\lambda}$

from the first part we know $w_1^* = \frac{Cov(x\,y)}{Var(x) + 2\lambda}$

$w_1' + w_2' = \frac{Cov(x\,y)}{Var(x) + 2\lambda} = 2w_1'$ since $w_1' = w_2'$ so we get $\boxed{w_1^* = 2w_1'}$

CPMSC 448 Homework 2

Problem 4

Name: Kush Lalwani

PSU id: kgl5163

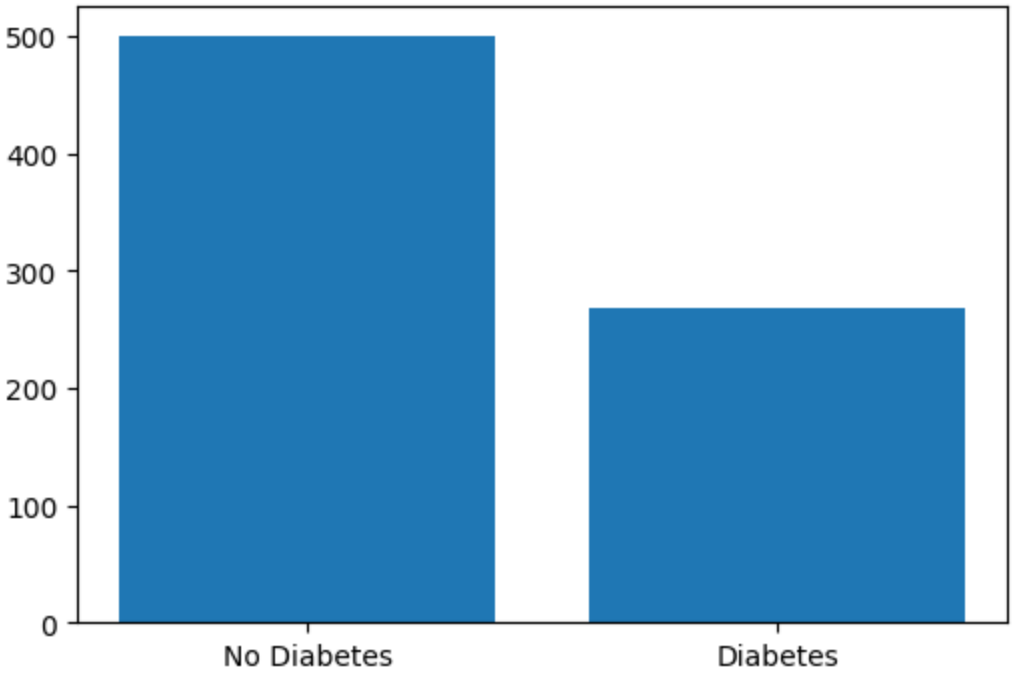
In this problem will use the Pima Indians Diabetes dataset from the UCI repository to experiment with the k-NN algorithm and find the optimal value for the number of neighbors k. You do not need to implement the algorithm and encouraged to use the implementation in \texttt{scikit-learn}.

a)

Download the provided `Pima.csv` data file and load it using `pandas`. As a sanity check, make sure there are 768 rows of data (potential diabetes patients) and 9 columns (8 input features including `Pregnancies`, `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, `BMI`, `DiabetesPedigreeFunction`, `Age`, and 1 target output). Note that the data file has no header and you might want to explicitly create the header. The last value in each row contains the target label for that row, and the remaining values are the features. Report the statics of each feature (min, max, average) and the histogram of the labels (target outputs).

In [40]:
```
import pandas as pd
import matplotlib.pyplot as plt

varNames = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulir

data = pd.read_csv("Pima.csv", names = varNames)

print(f"Dataset shape: {data.shape}\n\nColumn Names: {data.columns}\n")

print(data.drop(columns=["Target"]).agg(["min", "max", "mean"]))

plt.figure(figsize=(6, 4))
data["Target"].hist(bins=2, rwidth=0.8)
plt.xticks([0.25, 0.75], ["No Diabetes", "Diabetes"])
plt.grid(visible = False)
plt.show()
```

```
Dataset shape: (768, 9)

Column Names: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickne
ss', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Target'],
      dtype='object')

      Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
min      0.000000    0.000000       0.000000       0.000000    0.000000
max     17.000000  199.000000     122.000000      99.000000  846.000000
mean     3.845052  120.894531      69.105469      20.536458   79.799479

            BMI  DiabetesPedigreeFunction        Age
min    0.000000                  0.078000  21.000000
max   67.100000                  2.420000  81.000000
mean  31.992578                  0.471876  33.240885
```



b)

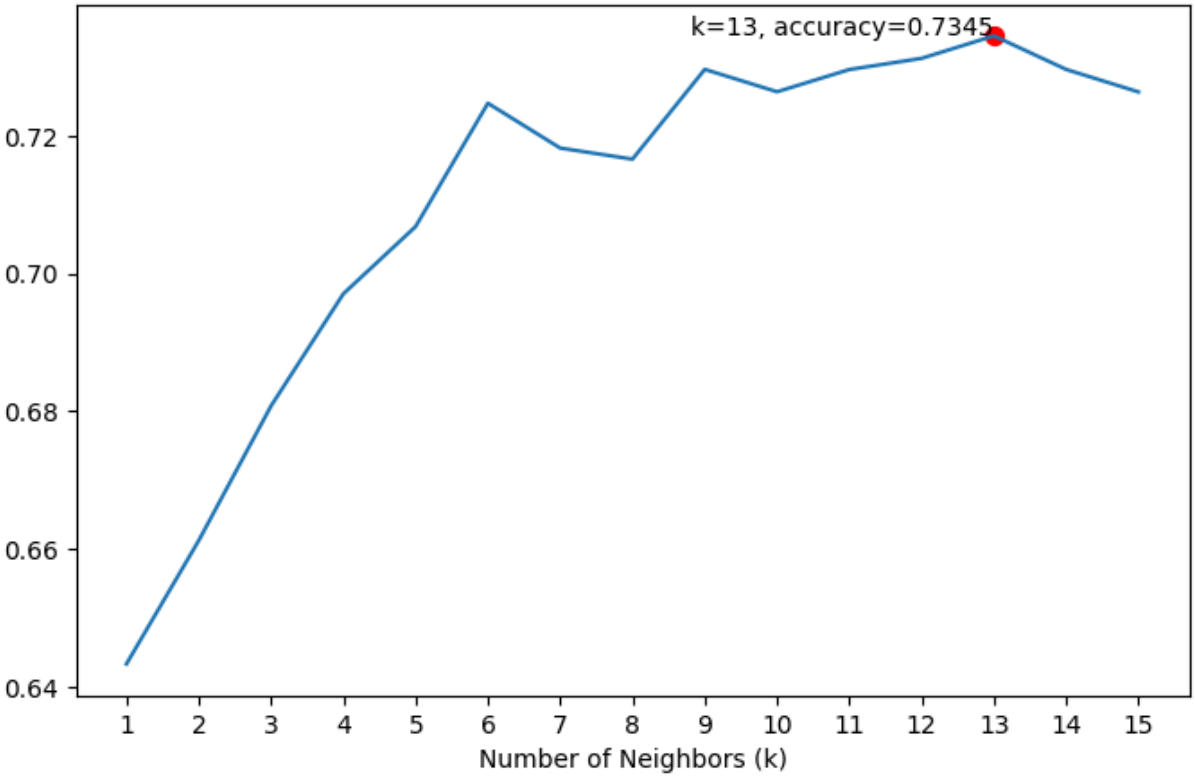
Split the data into training and test data with 80% training and 20% test data sizes.

Use 5-fold cross-validation on training data to decide the best number of neighbours k. To this end, you can use the built in functionality in `scikit-learn` such as `cross_val_score`. For $k = 1, 2, 3, \ldots, 15$ compute the 5-fold cross validation error and plot the results (with values of k on the x-axis and accuracy on the y-axis). Include the plot in your report and justify your decision for picking a particular number of neighbors k.

In [85]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
```

```
np.random.seed(448)

X = data.drop(columns=["Target"])
y = data["Target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

accuracies = []
klist = range(1,16)

for k in klist:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5, scoring="accuracy"
    accuracies.append(scores.mean())

best_accuracy = max(accuracies)
best_k = accuracies.index(best_accuracy) + 1

print(f"Best k: {best_k} with accuracy: {best_accuracy:.4f}")


plt.figure(figsize=(8, 5))
plt.plot(klist, cv_scores)
plt.xlabel("Number of Neighbors (k)")
plt.xticks(klist)
plt.scatter(best_k, best_accuracy, color="red", s=50)
plt.text(best_k, best_accuracy, f"k={best_k}, accuracy={best_accuracy:.4f}",

plt.show()
```

```
Best k: 13 with accuracy: 0.7345
```



c)

Evaluate the k-NN algorithm on test data with the optimal number of neighbours you obtained in previous step and report the test error.

In [94]:
```
from sklearn.metrics import accuracy_score

#train model with ideal k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)

y_hat = knn_best.predict(X_test)

test_accuracy = accuracy_score(y_test, y_hat)

test_error = 1 - test_accuracy

print(f"Test Error: {test_error:.4f}")
```

```
Test Error: 0.2338
```

d)

Process the input data by subtracting the mean (a.k.a. centralization) and dividing by the standard deviation (a.k.a. standardization) over each dimension (feature), repeat the previous part and report the accuracy. Do centralization and standardization affect the accuracy? Why?

In [100…
```
X_mean = X_train.mean(axis=0) #find the means and standard deviations by the
X_std = X_train.std(axis=0)

X_train_scaled = (X_train - X_mean) / X_std
X_test_scaled = (X_test - X_mean) / X_std

knn_standard = KNeighborsClassifier(n_neighbors=best_k)
knn_standard.fit(X_train_scaled, y_train)

y_hat_scaled = knn_standard.predict(X_test_scaled)

test_accuracy_scaled = accuracy_score(y_test, y_hat_scaled)

test_error_scaled = 1 - test_accuracy_scaled

print(f"Test Error: {test_error_scaled:.4f}\n\nTest Accuracy: {test_accuracy
```

```
Test Error: 0.2532
```

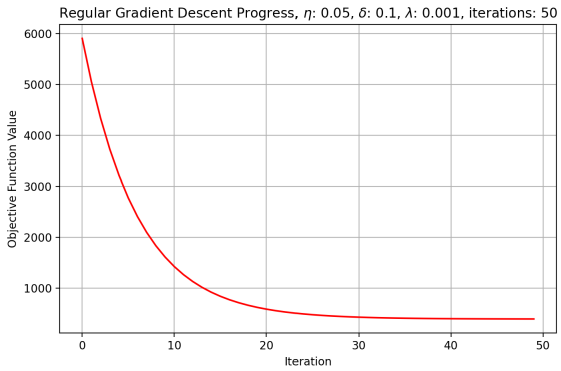
```
Test Accuracy: 0.7468
```

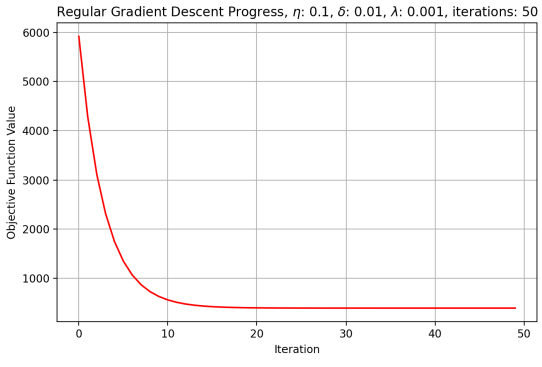
As we can see, the test error and the accuracy slightly increased for the scaled knn regression model. This could be because standardizing the features in the dataset could potentially decrease the euclidean distance between points. Since,

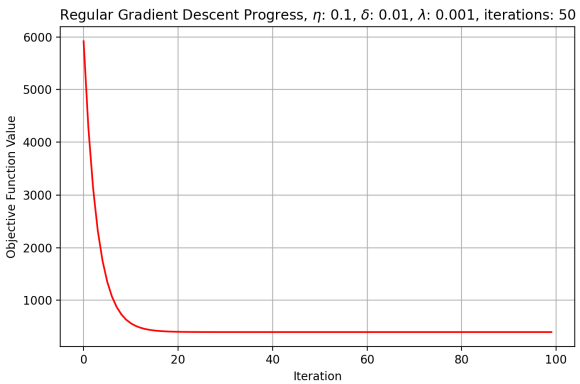
the scaling is different for each feature, some points may have gotten closer to others. This could lead to certain points being classified differently as they may now be closer to other points.

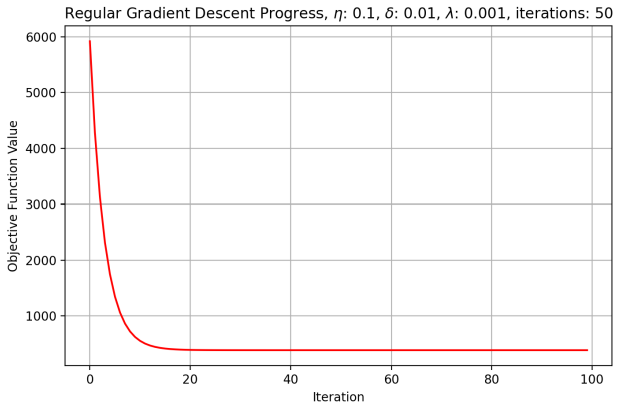
```python
import random
import numpy as np
import matplotlib.pyplot as plt
from Problem5 import bgd_l2, sgd_l2

def plot_loss_history(history_fw, params, type = "Regular"):

    plt.figure(figsize=(8, 5))
    plt.plot(range(len(history_fw)), history_fw,  color="r")
    plt.xlabel("Iteration")
    plt.ylabel("Objective Function Value")
    plt.title(rf"{type} Gradient Descent Progress, $\eta$: {params[0]}, $\delta$: {params[1]}, $\lambda$: {params[2]}, iterations: {params[3]}")
    plt.grid()
    plt.show()

def load_data():
    data = np.load("c:/Users/giris/OneDrive - The Pennsylvania State University/Cmpsc 448/Homework/hw2/data.npy")  # Load dataset (100 × 2)

    X = data[:, 0]  # Extract feature column
    y = data[:, 1]  # Extract target column

    X = np.column_stack((np.ones(X.shape[0]), X))  # Add intercept column

    return X, y

if __name__ == '__main__':
    np.random.seed(448)
    X,y= load_data() #load the features and the target values
    w = np.zeros(X.shape[1]) # initialize the weights


    w1, hist1 = bgd_l2(X,y,w,0.05,0.1,0.001,50)
    plot_loss_history(hist1, [0.05,0.1,0.001,50])

    w2, hist2 = bgd_l2(X,y,w,0.1,0.01,0.001,50)
    plot_loss_history(hist2,[0.1,0.01,0.001,50])

    w3, hist3 = bgd_l2(X,y,w,0.1,0,0.001,100)
    plot_loss_history(hist3,[0.1,0.01,0.001,50])

    w4, hist4 = bgd_l2(X,y,w,0.1,0,0,100)
    plot_loss_history(hist4,[0.1,0.01,0.001,50])
    #Stochastic Gradient Descent
    w5, hist5 = sgd_l2(X,y,w,1,0.1,0.5,800)
    plot_loss_history(hist5, [1,0.1,0.5,800], "Stochastic")

    w6, hist6 = sgd_l2(X,y,w,1,0.01,0.1,800)
    plot_loss_history(hist6,[1,0.01,0.1,800],"Stochastic")

    w7, hist7 = sgd_l2(X,y,w,1,0,0,40)
    plot_loss_history(hist7,[1,0,0,40],"Stochastic")

    w8, hist8 = sgd_l2(X,y,w,1,0,0,800)
    plot_loss_history(hist8,[1,0,0,800],"Stochastic")

```

```python
import math
import random
import numpy as np

def bgd_l2(data, y, w, eta, delta, lam, num_iter):

    n, d = data.shape  # extract size of dataset and number of features
    new_w = np.copy(w)  # create new weight vector
    history_fw = []  # Store function values

    for _ in range(num_iter):
        gradient = np.zeros(d)  # Initialize gradient
        loss = 0  # Initialize loss function value

        for i in range(n):
            x_i = data[i]  # Feature vector
            y_i = y[i]  # Target value
            pred = np.dot(new_w, x_i)  # Compute prediction w * x

            # Follow function definition
            if y_i >= pred + delta:
                gradient += -2 * (y_i - pred - delta) * x_i #derivative of function with respect to x_i
                loss += (y_i - pred - delta) ** 2
            elif y_i <= pred - delta:
                gradient += -2 * (y_i - pred + delta) * x_i #derivative of function with respect to x_i
                loss += (y_i - pred + delta) ** 2

        # Add regularization term to gradient
        gradient = (gradient / n) + (2 * lam * new_w)

        # Update weights using gradient descent
        new_w = new_w - eta * gradient

        # Compute full objective function value and store
        reg_term = lam * np.sum(new_w ** 2)  # regularization term
        fw = (loss / n) + reg_term  # Compute full loss function
        history_fw.append(fw)

    return new_w, history_fw  # Return updated weights and loss history
```









```python
def sgd_l2(data, y, w, eta, delta, lam, num_iter, i=-1):

    n, d = data.shape
    new_w = np.copy(w)
    history_fw = []

    for t in range(1, num_iter + 1):  # Start iterations from 1
        # Select data point
        if i == -1:
            ind = np.random.randint(0, n)  # select random point
        else:
            ind = i  # Use the specific data point

        x_i = data[ind] #feature vector
        y_i = y[ind] #target vector
        pred = np.dot(new_w, x_i) # Compute prediction w * x

        # Compute learning rate
        learn_rate = eta / np.sqrt(t)

        # Follow function definition
        gradient = np.zeros(d)
        loss = 0

        if y_i >= pred + delta:
            gradient = -2 * (y_i - pred - delta) * x_i #derivative of function with respect to x_i
            loss = (y_i - pred - delta) ** 2
        elif y_i <= pred - delta:
            gradient = -2 * (y_i - pred + delta) * x_i #derivative of function with respect to x_i
            loss = (y_i - pred + delta) ** 2

        # Add regularization term
        gradient += 2 * lam * new_w

        # Update weights
        new_w = new_w - learn_rate * gradient

        # Compute objective function value
        reg_term = lam * np.sum(new_w ** 2)
        fw = loss + reg_term
        history_fw.append(fw)
        # dont iterate again if i != -1
        if i != -1:
            break

    return new_w, history_fw
```