

# CSE 142 Assignment 3, Fall 2023

4 Questions, 100 pts, due: 23:59 pm, Nov 8th, 2023

## Instruction

- Submit your assignments onto **Gradescope** by the due date. Upload a `zip` file containing:

(1) The saved/latest `.ipynb` file, please **rename this file with your name included**.

(2) Also save your file into a pdf version, if error appears, save an html version instead (easy to grade for written questions).

(3) All other materials to make your `.ipynb` file runnable.

**For assignment related questions, please reach TA or grader through Slack/Piazza/Email.**

- This is an **individual** assignment. All help from others (from the web, books other than text, or people other than the TA or instructor) must be clearly acknowledged.

## Objective

- **Task 1:** Perceptron (math)
- **Task 2:** Support Vector Machine (math)
- **Task 3:** Image Classification (with Scikit-learn)
- **Task 4:** Coding Linear Regression from scratch

## Question 1. (Perceptron, 30 pts)

(a -- 10 pts)

On what kinds of training data does the perceptron algorithm converge?

(Ans.) Perceptron converges data that is linearly separable, since it always converges the given dataset to its best linear boundary. In other words, it is because perceptron can not learn non-linear boundaries.

(b -- 5\*4 pts)

Simulate one pass through the following data with the perceptron algorithm described in lecture and homework. Use the learning rate  $\eta = 1$ .

Start with  $w = (0, 0, 0)$  and show the resulting weight vector after each example.

(Assume that the perceptron algorithm predicts incorrectly when  $w \cdot x = 0$ , and ignore the bias term.)

$x_1$	$x_2$	$x_3$	$y$
1	0	1	+1
0	-1	1	-1
1	1	1	+1
-1	2	0	-1

Initially,  $w = (0, 0, 0)$ .

After the first example,  $w = (-1, 0, -1)$ ;

After the second example,  $w = (-1, 0, -1)$ ;

After the third example,  $w = (-2, -1, -2)$ ;

After the fourth example,  $w = (-3, 1, -2)$ .

**I added the explanation below as a screenshot.**

**Reminder:** if you are unsure about your answers, give as many details as possible so that you won't get 0 points in the wrong answers.

```
In [ ]: from IPython.display import Image
        Image(filename='Question1B.jpg')
```

Out[ ]:

$$\begin{aligned}
 \omega_0 &= (0, 0, 0) \\
 \omega_0 \cdot x &= \text{sign}((0, 0, 0) \times (1, 0, 1)) = \text{sign}(0) \text{ incorrect. } \hat{y}_1 = -1 \\
 &\text{from } \text{sign}(0), y_1 \neq \hat{y}_1 \\
 \omega_1 &= \omega_0 + 1 \cdot (-1) \cdot (1, 0, 1) \\
 &= (-1, 0, -1) \\
 \omega_1 \cdot x &= \text{sign}((-1, 0, -1) \times (0, -1, 1)) = \text{sign}(-1) = -1 \text{ correct. } \hat{y}_2 = -1 \\
 &\text{from } y_2 = \hat{y}_2, \\
 \omega_2 &= \omega_1 = (-1, 0, -1) \\
 \omega_2 \cdot x &= \text{sign}((-1, 0, -1) \times (1, 1, 1)) = \text{sign}(-2) = -1 \text{ incorrect. } \hat{y}_3 = -1 \\
 &\text{from } y_3 \neq \hat{y}_3 \\
 \omega_3 &= \omega_2 + 1 \cdot (-1) \cdot (1, 1, 1) \\
 &= (-2, -1, -2) \\
 \omega_3 \cdot x &= \text{sign}((-2, -1, -2) \times (-1, 2, 0)) = \text{sign}(0) \text{ incorrect } \hat{y}_4 = 1 \\
 &\text{from } y_4 \neq \hat{y}_4 \\
 \omega_4 &= \omega_3 + 1 \cdot 1 \cdot (-1, 2, 0) \\
 &= (-3, 1, -2)
 \end{aligned}$$

## Question 2. (Support Vector Machines, 30 pts)

Suppose that we have the following training set (where the instances have two features):

$x_1$	$x_2$	$y$
1	1	+1
1	2	+1
2	1	+1

$x_1$	$x_2$	$y$
0	0	-1
1	0	-1
0	1	-1

## (a -- 10 pts)

Plot them (in hand or with python) and find the support vectors (by eye).

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

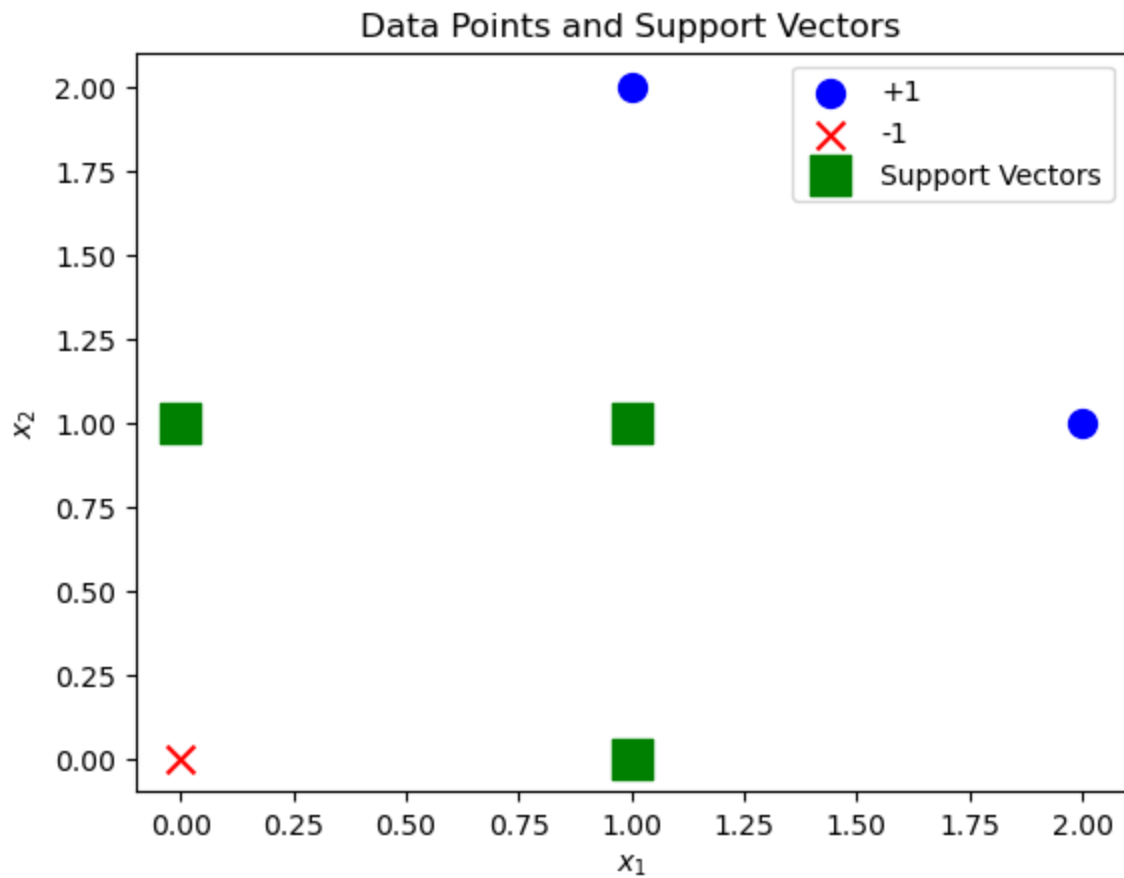
X = np.array([[1, 1], [1, 2], [2, 1], [0, 0], [1, 0], [0, 1]])
y = np.array([1, 1, 1, -1, -1, -1])

clf = SVC(kernel='linear', C=2)
clf.fit(X, y)
support_vectors = clf.support_vectors_

plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], label='+1', c='blue', marker='o', s=100)
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], label='-1', c='red', marker='x', s=100)
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], c='green', marker='s', s=200,

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Data Points and Support Vectors')
plt.legend()

plt.show()
print("Support Vectors:")
print(support_vectors)
```



Support Vectors:

```
[[1.  0.]
 [0.  1.]
 [1.  1.]]
```

## (b -- 10 + 10 pts)

Using the support vectors, find the equation for the maximum margin separating plane, and determine the geometric margin. (Assume a simple linear SVM and no soft-margin).

```
In [ ]: from IPython.display import Image
# Replace the figure name
Image(filename='Question2B.jpg')
```

Out[ ]:

The support vectors:  $(1,0), (0,1), (1,1)$

The equation for maximizing margin separating the plane would be


$$y = wx + b$$

here,  $w = -1$

therefore  $y = -x + b$

$$x + y - b = 0$$

from  $\frac{|x+y-b|}{\sqrt{1+1}}$  and having all margins of the support vectors equal to each other,



$$\frac{|1-b|}{\sqrt{2}} = \frac{|2-b|}{\sqrt{2}}$$

$$|1-b| = |2-b|$$

$$(1-b)^2 = (2-b)^2$$

$$b^2 - 2b + 1 = b^2 - 4b + 4$$

$$2b = 3 \quad \therefore b = \frac{3}{2}$$

the equation would be

$$y = -x + \frac{3}{2}$$

Geometric margin:

$$\frac{|2 - \frac{3}{2}|}{\sqrt{2}} = \frac{\frac{\sqrt{2}}{4}}{4}$$

## Question 3. (Binary Image Classification, 30 pts + bonus pts)

In this question, you will perform a binary image classification task with scikit-learn implemented models. You will see the importance of a powerful feature extractor. Meanwhile, you will learn to perform hyper-parameter tuning and select machine learning models with scikit-learn. Please **do not** use deep learning models for training use in this question.

### Import/Install required packages

```
In [ ]: # Import libraries
import os
from os.path import join
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from PIL import Image
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

## Data preprocessing

```
In [ ]: # A help function which appends the path list for train and test image folder
def get_path(is_train=True):
    if is_train:
        directory = "train"
    else:
        directory = "test"

    # Append the image path to a list for images that contain a dog
    dog_image_dir = f'./cat_vs_dog/{directory}/dogs'
    dog_paths = [join(dog_image_dir, filename) for filename in os.listdir(dog_image_dir)]

    # Append the image path to a list for images that contain a cat
    cat_image_dir = f'./cat_vs_dog/{directory}/cats'
    cat_paths = [join(cat_image_dir, filename) for filename in os.listdir(cat_image_dir)]

    img_paths = dog_paths + cat_paths

    # Return the unshuffled image paths
    return img_paths
```

```
In [ ]: # Show how many figures contained in the train and test dataset
print(f"There are {len(get_path())} train images and {len(get_path(is_train=False))} test images.")

There are 3002 train images and 1000 test images.
```

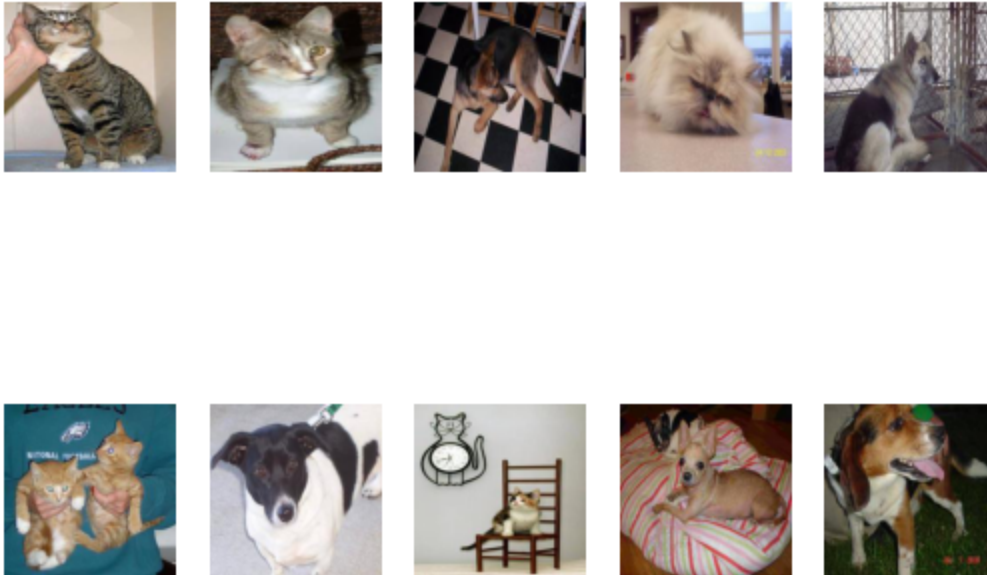
Take a look at a few randomly selected example images we will deal with.

```
In [ ]: import random

fig, ax = plt.subplots(2, 5)
index_list = [random.randrange(1, len(get_path()), 1) for i in range(10)]
print(index_list)
train_paths = get_path()

filenames=[train_paths[idx] for idx in index_list]
for i in range(10):
    with open(filenames[i], 'rb') as f:
        image=Image.open(f)
        ax[i%2][i//2].imshow(image)
        ax[i%2][i//2].axis('off')
fig.show()
```

```
[1890, 2805, 1680, 427, 60, 2286, 2771, 532, 1018, 701]
```



Clearly, the above images are of different shape/size.

```
In [ ]: # See the shape of an image
sample_img = Image.open(train_paths[1])
print(np.array(sample_img).shape)

(200, 200, 3)
```

Standard PCA methods implemented in Scikit-learn do not work for color images (RGB), so we will transform the images into grayscale.

```
In [ ]: import matplotlib

# Transform color images to grayscale
def rgb2gray(rgb_img):
    return np.dot(rgb_img[...,:3], [0.2989, 0.5870, 0.1140])

# Given the image path, return the resized image as a numpy 2d array
def get_image(path):
    img = Image.open(path)
    img = img.resize((200,200))
    img.save(path)
    img = matplotlib.image.imread(path)
    gray = rgb2gray(img)
    return np.array(gray)
```

```
In [ ]: print(get_image(train_paths[1]).shape)

(200, 200)
```

Append image data and labels in to list

```
In [ ]: # Prepare the raw data: grayscale, train images and train labels
h, w = get_image(train_paths[1]).shape
train_data = np.empty((len(train_paths), h, w))
train_label = [1] * np.int_(len(train_paths)/2) + [0] * np.int_(len(train_paths)/2)
count = -1
for pth in train_paths:
```



```

count += 1
image = get_image(pth)
train_data[count] = image
n_samples = train_data.shape[0]
X = np.empty((n_samples, h * w))
for i in range(n_samples):
    X[i] = train_data[i].flatten()
y = train_label
print(f"The shape of the training data is {X.shape}")

```

The shape of the training data is (3002, 40000)

## Prepare the raw test data

```

In [ ]: test_paths = get_path(is_train=False)

# Prepare the raw dataframe: grayscale, test images and test labels
test_data = np.empty((len(test_paths), h, w))
test_label = [1] * np.int_(len(test_paths)/2) + [0] * np.int_(len(test_paths)/2)
count = -1
for pth in test_paths:
    count += 1
    image = get_image(pth)
    test_data[count] = image
n_samples = test_data.shape[0]
X_test = np.empty((n_samples, h * w))
for i in range(n_samples):
    X_test[i] = test_data[i].flatten()
y_test = test_label
print(f"The shape of the test data is {X_test.shape}")

```

The shape of the test data is (1000, 40000)

## Shuffle the training dataset

```

In [ ]: # Shuffle the training dataset
from sklearn.utils import shuffle
X_train, y_train = shuffle(X, y, random_state=0)
X_train
print(f"The shape of the training data is {X.shape}")

```

The shape of the training data is (3002, 40000)

```

In [ ]: # Import additional libraries
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

```

## Question 3.1 Reducing the dimensions (feature extraction using PCA)

Directly training on current training dataset may consume a large amount of time. One wise choice is to firstly perform feature extraction so that we do not have to train model with non-necessary features.

### 3.1.1 SVM on extracted features (15pts)

```
In [ ]: # The list of number of components for PCA
n_list = [10, 20, 50, 100, 150, 200, 498]
for i in range(len(n_list)):
    n_components = n_list[i]
    print(f"##### Number of components is {n_components} #####")

    #
    # your code here for PCA
    pca = PCA(n_components)
    pca.fit(X_train)
    #
    print("Projecting the input data on the eigenimages orthonormal basis")
    X_train_pca = pca.transform(X_train)
    print(X_train_pca.shape)
    X_test_pca = pca.transform(X_test)

    #
    # your code here for training SVM classifier and prediction on X_test (y_pred are pr
    svm = SVC(kernel='rbf')
    svm.fit(X_train_pca, y_train)
    y_pred = svm.predict(X_test_pca)
    #

    print(f"Confusin matrix: {confusion_matrix(y_test, y_pred, labels=range(2))}")

    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

```

##### Number of components is 10 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 10)
Confusin matrix: [[282 218]
 [173 327]]
Accuracy: 0.609
##### Number of components is 20 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 20)
Confusin matrix: [[290 210]
 [162 338]]
Accuracy: 0.628
##### Number of components is 50 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 50)
Confusin matrix: [[286 214]
 [169 331]]
Accuracy: 0.617
##### Number of components is 100 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 100)
Confusin matrix: [[292 208]
 [168 332]]
Accuracy: 0.624
##### Number of components is 150 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 150)
Confusin matrix: [[288 212]
 [162 338]]
Accuracy: 0.626
##### Number of components is 200 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 200)
Confusin matrix: [[284 216]
 [168 332]]
Accuracy: 0.616
##### Number of components is 498 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 498)
Confusin matrix: [[291 209]
 [172 328]]
Accuracy: 0.619

```

### 3.1.2 Logistic regression on extracted features (15pts)

```

In [ ]: # The list of number of components for PCA
n_list = [10, 20, 50, 100, 150, 200, 498]
for i in range(len(n_list)):
    n_components = n_list[i]
    print(f"##### Number of components is {n_components} #####")

    #
    # your code here for PCA
    pca = PCA(n_components=n_components)
    pca.fit(X_train)
    #

    print("Projecting the input data on the eigenimages orthonormal basis")
    X_train_pca = pca.transform(X_train)

```

```

print(X_train_pca.shape)
X_test_pca = pca.transform(X_test)

#
# your code here for training Logistic Regression classifier and prediction on X_test
lr = LogisticRegression()
lr.fit(X_train_pca, y_train)
y_pred = lr.predict(X_test_pca)
#

print(f"Confusin matrix: {confusion_matrix(y_test, y_pred, labels=range(2))}")

print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

```

```

##### Number of components is 10 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 10)
Confusin matrix: [[277 223]
 [226 274]]
Accuracy: 0.551
##### Number of components is 20 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 20)
Confusin matrix: [[279 221]
 [212 288]]
Accuracy: 0.567
##### Number of components is 50 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 50)
Confusin matrix: [[273 227]
 [231 269]]
Accuracy: 0.542
##### Number of components is 100 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 100)
Confusin matrix: [[276 224]
 [231 269]]
Accuracy: 0.545
##### Number of components is 150 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 150)
Confusin matrix: [[275 225]
 [235 265]]
Accuracy: 0.54
##### Number of components is 200 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 200)
Confusin matrix: [[283 217]
 [234 266]]
Accuracy: 0.549
##### Number of components is 498 #####
Projecting the input data on the eigenimages orthonormal basis
(3002, 498)
Confusin matrix: [[257 243]
 [238 262]]
Accuracy: 0.519

```

**Leaderboard Depending on your best achieved accuracy score, the remaining points and bonus points are organized as below:**

Accuracy score > 55%: 30 pts

Accuracy score 50-55%: 15 pts

Accuracy score < 50%: 5 pts

## Question 4. (Coding assignment: Linear Regression, 10 pts)

In this question, you'll be coding up linear regression algorithm from scratch.

### Instructions

- Download the datasets `heights_weights.csv`. **Place these files in the same directory as this notebook.**
- You are **NOT** allowed to use machine learning libraries such as `scikit-learn` to build the models for this question.
- You are required to complete the functions defined in the code blocks following each question. Fill out sections of the code marked `"YOUR CODE HERE"`.
- You're free to add any number of methods within each class.
- You may also add any number of additional code blocks that you deem necessary.
- If not comfortable with Python, you can code in Matlab. Make sure to add the files in the submission zip folder.
- You may use `scikit-learn` package **only** to compare your final error.

```
In [ ]: # Importing Libraries
import time
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

### Data Preparation.

To keep things simple, first we'll use a toy dataset to test our implementation. This dataset contains the heights and weights of a few individuals. Our goal is to predict the weight of an individual given their height using a linear regression model.

```
In [ ]: df = pd.read_csv('./heights_weights.csv')
```

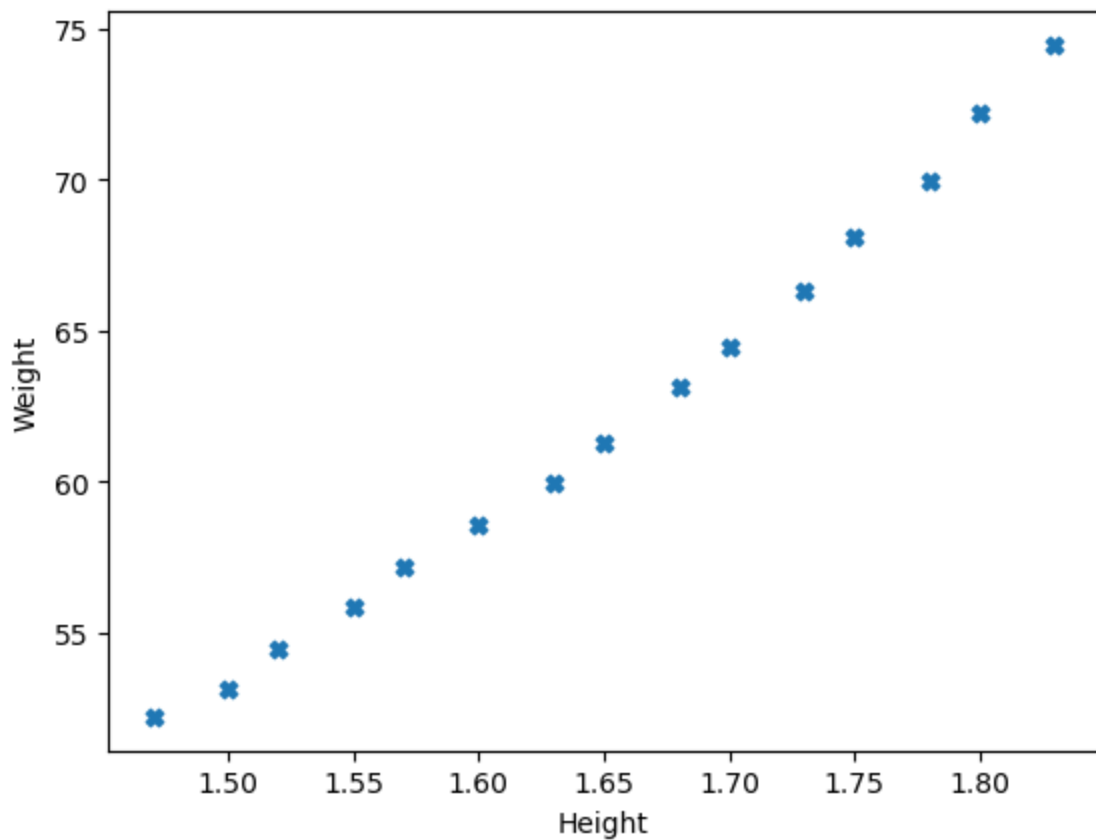
```
In [ ]: df.head()
```

```
Out[ ]:
```

	Height	Weight
0	1.47	52.21
1	1.50	53.12
2	1.52	54.48
3	1.55	55.84
4	1.57	57.20

```
In [ ]: import matplotlib.pyplot as plt

plt.scatter(df['Height'], df['Weight'], marker='X')
plt.xlabel("Height")
plt.ylabel("Weight")
plt.show()
```



Looking at the distribution of the data, it seems like `Weight` and `Height` have a linear relationship. Hence, a linear regression model should be able to capture this relationship.

Let's us convert the dataframe `df` to a Numpy array so that it is easier to perform operations on it.

```
In [ ]: X_train = np.array(df['Height'])
y_train = np.array(df['Weight'])
X_train = np.expand_dims(X_train, -1)
```

## Implement the LinearRegression class

Make sure it works with more than 1 feature.

**NOTE:** Do **NOT** forget to include a bias term in the weights.

```
In [ ]: class LinearRegression:
    def __init__(self, lr=0.001, epochs=30):
        """
        Fits a linear regression model on a given dataset.

        Args:
            lr: learning rate
            epochs: number of iterations over the dataset
        """
        self.lr = lr
        self.epochs = epochs
        #####
        self.weights = None
        #####
        # You may add additional fields

    def train(self, X, y):
        """
        Initialize weights. Iterate through the dataset and update weights once every epoch.

        Args:
            X: features
            y: target
        """
        #####
        X = np.column_stack([np.ones(X.shape[0]), X])
        self.weights = np.zeros(X.shape[1])
        for i in range(self.epochs):
            self.update_weights(X, y)
        #####

    def update_weights(self, X, y):
        """
        Helper function to calculate the gradients and update weights using batch gradient descent.

        Args:
            X: features
            y: target
        """
        #####
        n = X.shape[0]
        grad = X.T.dot(X.dot(self.weights) - y) / (2 * n)
        self.weights -= self.lr * grad
        #####

    def predict(self, X):
```

```

"""
    Predict values using the weights.

    Args:
        X: features

    Returns:
        The predicted value.
    """

    #####
    X = np.column_stack([np.ones(X.shape[0]), X])
    return X.dot(self.weights)
    #####

```

## Build the model and train on the dataset.

```
In [ ]: model = LinearRegression(0.01, 300000)
        model.train(X_train, y_train)
```

## (5 points) Implement the evaluation metric mean squared error.

We use the [mean squared error \(MSE\)](#) as the metric to evaluate our model.

```
In [ ]: def mean_squared_error(y_pred, y_actual):
        """
            Calculates the mean squared error between two vectors.

            Args:
                y_pred: predicted values
                y_actual: actual/true values

            Returns:
                The mean squared error.
            """

            #####
            mse = np.mean(((y_pred - y_actual) ** 2))
            return mse
            #####

```

## Make predictions using the model and evaluate it.

```
In [ ]: y_pred = model.predict(X_train)
        print("Train MSE: {:.4f}".format(mean_squared_error(y_pred, y_train)))
```

Train MSE: 0.5001

## Plot the predicted and the actual values.



```
In [ ]: import matplotlib.pyplot as plt

plt.scatter(X_train, y_train, marker='x', label='actual')
plt.scatter(X_train, y_pred, marker='o', label='predicted')
plt.legend()
plt.show()
```

