Machine Learning Weekly Assignments 1: Matthew Flynn Student Number 17327199

Q1 a.

(i) To read the input data the code given in the question was sufficient and ran with no issues.

(ii) I normalised the data so everything was between 0 and 1. To normalise the data I took the data array, x, and used this formula for every element in the array. This was done for both x and y arrays.

$$norm(x_i) = \frac{x_i - \min(x)}{\max(x) - \min(x)} \; for\ every\ element\ in\ x$$

(iii) To use gradient descent to train a linear regression model I constructed the following functions. First hsum which will perform the $H(\theta) = \theta_0 + \theta_1 x_i$ function. Then I had a calsum function which can perform the sum component of the following functions:

$$\delta_0 = -\frac{2}{m}\sum_{i=1}^{m}\left(h_\theta(x^i) - y^i\right)$$

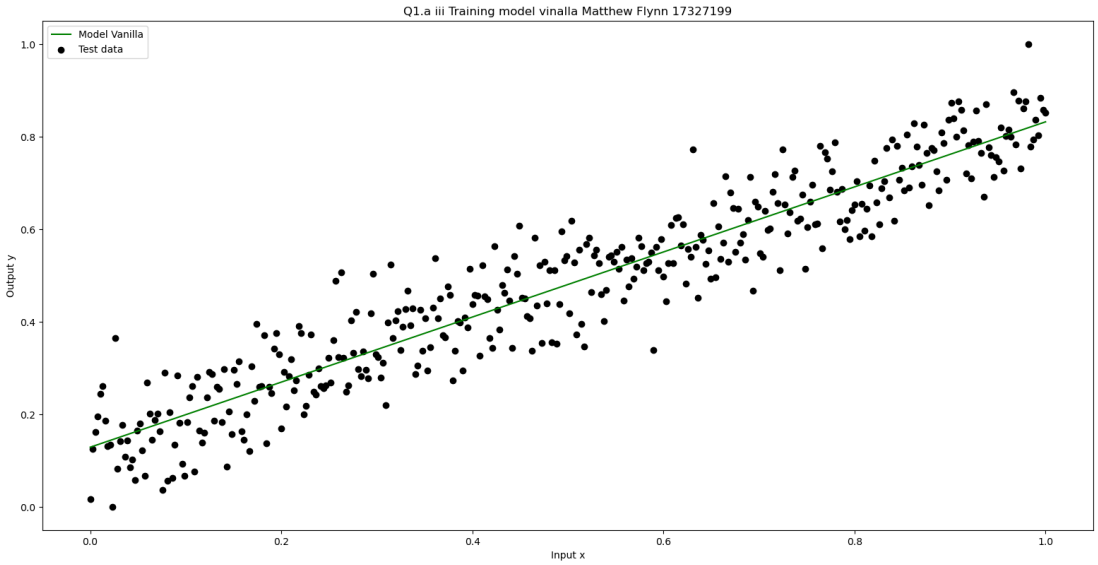
$$\delta_1 = -\frac{2}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x^i$$

$$J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

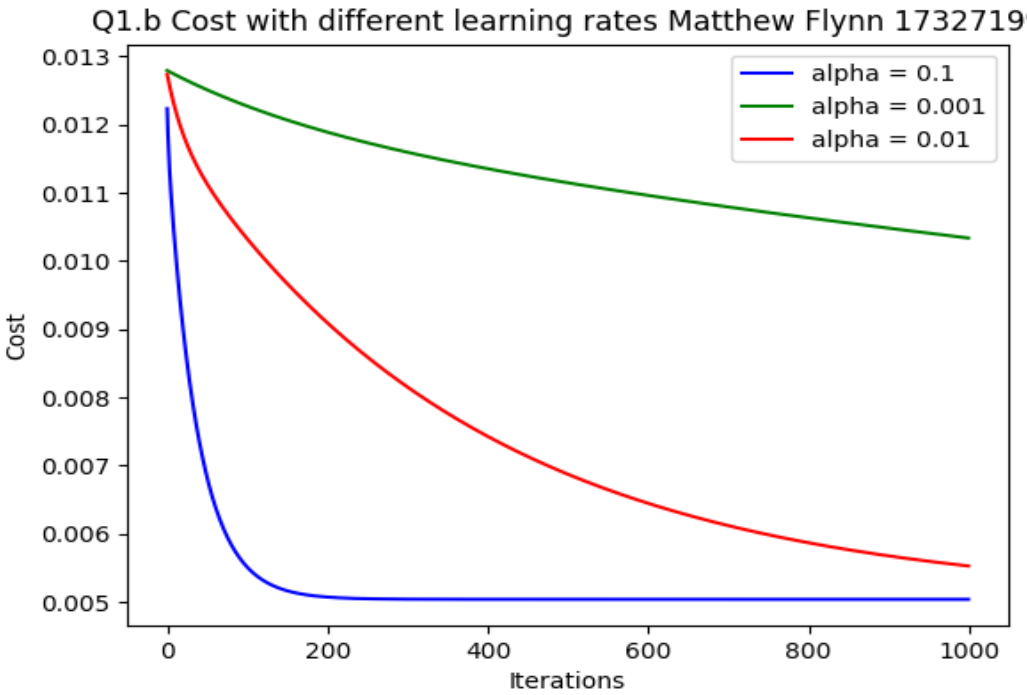
In the overall gradient_descent function I have set the iteration to 1000. $\delta_0\ and\ \delta_1$ are calculated first using the hsum and calsum function. Then the multiplication aspect of the program is done. Finally they are added to the previous version of theta0 and theta1 to get the new values of theta0 and theta1. Once these are calculated the cost is then calculated and appended to the list of costs. The following results were generated.

$$Cost = 0.00503947, \theta_0 = 0.12941161, \quad \theta_1 = 0.70291929.$$

The learning rate to produce these results was 0.1 and the number of iterations were 1000.



b. (i)To test out the different learning rates I used the following values, 0.1, 0.01 and 0.001. The following graph was created. As can be seen. When the learning rate is smaller the cost remains high over many iterations. To the point where it is impractical.



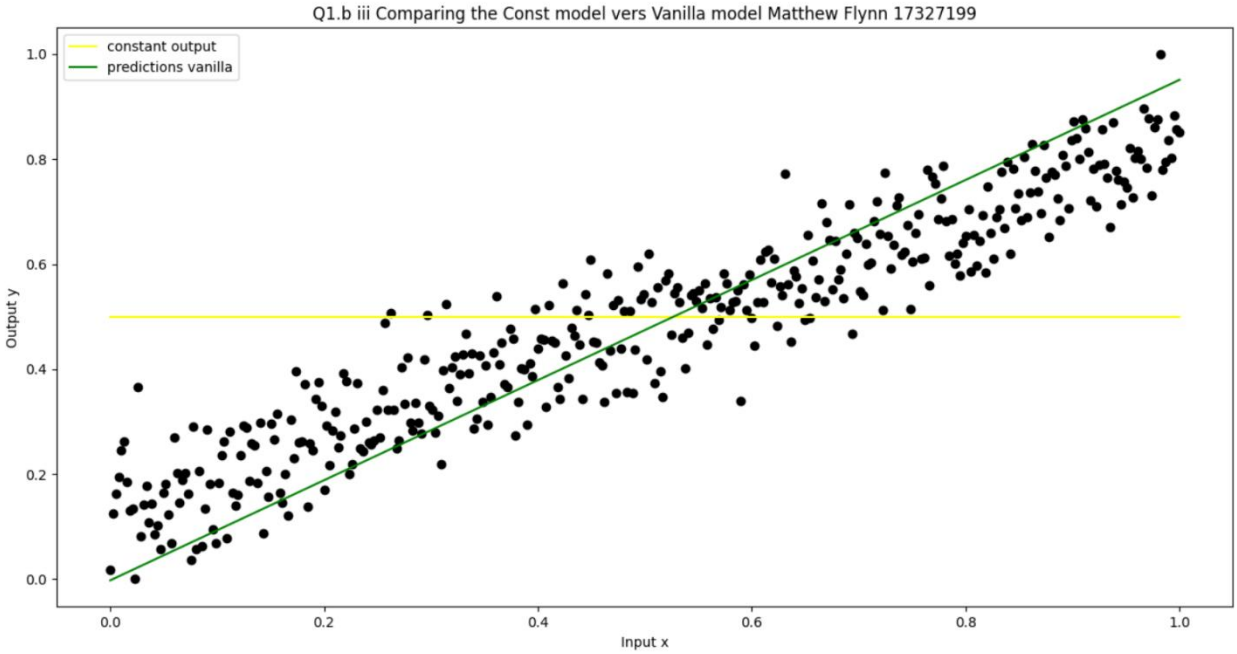
When $learning\ rate = 0.1$: $Cost = 0.00503947$, $\theta_0 = 0.12941161$, $\theta_1 = 0.70291929$.

When $learning\ rate = 0.001$: $Cost = 0.01033705$, $\theta_0 = -0.000234287$, $\theta_1 = 0.95338849$.

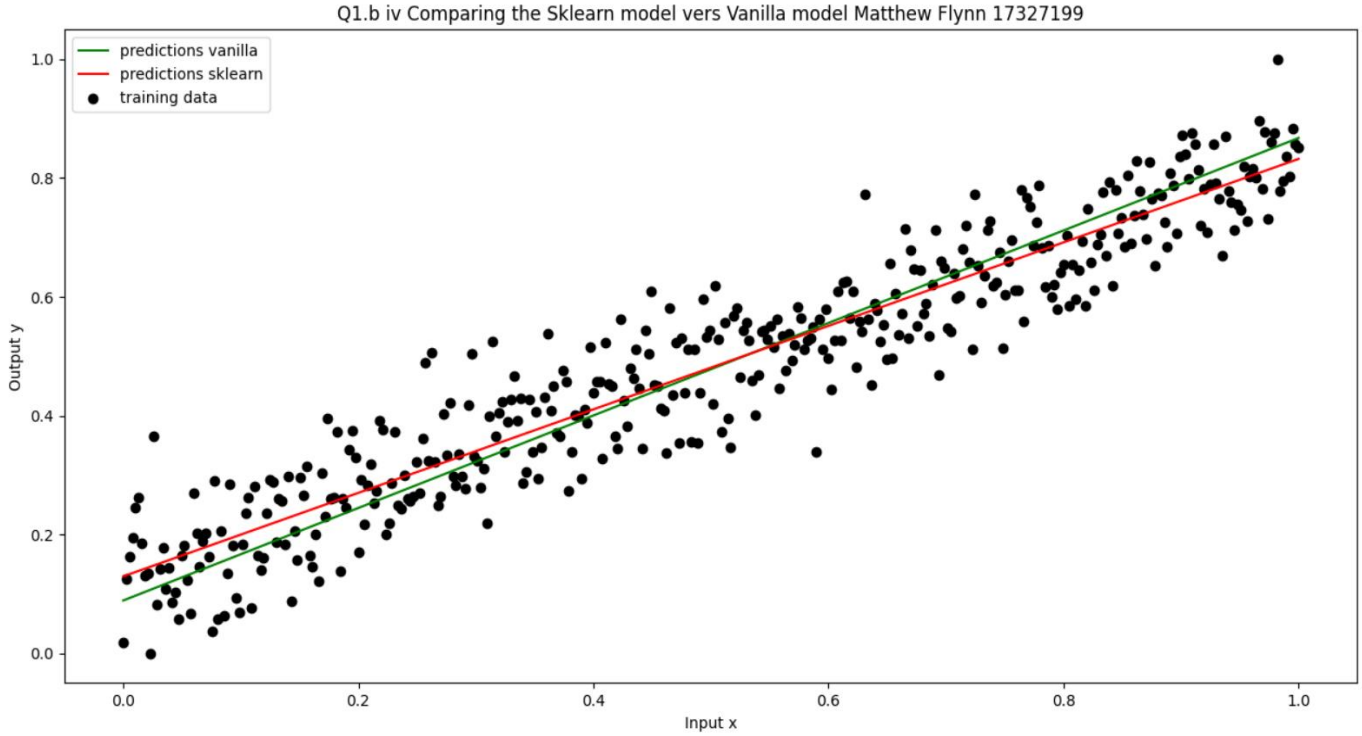
When $learning\ rate = 0.01$: $Cost = 0.005529$, $\theta_0 = 0.0887867$, $\theta_1 = 0.77880048$.

(ii) The parameter values of the linear regression model after being trained on my data are $\theta_0 = -0.12941161$, $and\ \theta_1 = 0.70291929$

(iii)The cost function of a trained model is 0.00503947, when there is a constant function for a baseline model the cost is 0.04679386. This baseline model is h(x) = the average of the array of the inputs x normalised. The results are as follows. The cost function for the baseline is much greater then the trained model because the baseline function is far less accurate then the trained model. The cost of both functions reflect this.



(iv) When sklearn is used to train your linear regression model the results are very similar to each other especially when compared to the 0.1 learning rate value. The resulting model seems more accurate then my model however the difference is minimal within he data range. The cost of the sklearn is 0.00501945 which is very close to 0.00503947. The coefficients are the following $\theta_0 = 0.12941187$, $\theta_1 = 0.70291902$. compared to $\theta_0 = 0.12941161$, $\theta_1 = 0.70291929$.



Code Appendix:

```
import numpy as np
import pandas as pd
from array import array
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression


# a part(ii) normalize the data

def normalise(x):
    # calculate scaling factor
    scaling_factor = max(x)-min(x)
    # now normalise the data
    count = 0
    norm = np.array(x, dtype=float)
    for i in x:
        norm[count] = (i-min(x))/scaling_factor
        count = count+1
    return norm


def calsum(x, y, h_sum, mode):
    count = 0
    rt = 0
    tmp = 0
    while count < len(x):
        tmp = (h_sum[count] - y[count])
        if mode == 1:  # cal for theta1
            tmp = (tmp)*x[count]
        if mode == 2:  # cal for J(0)
            tmp = (tmp)**2
        rt = rt + tmp
        count = count+1
    return rt

# a part(iii) gradient descent


def gradient_descent(x, y, t0, t1, step_size):
    # create a gradient descent model.
    m = len(x)  # get m value
    cost = 100  # generic value for the cost function to start with
    cost_list = []  # list for graphing the costs
    i = 0  # index

    while i < 1000:
        # cal h(0) = t0 - t1(x(i)) for all elems of x
        h_theta = hsum(x, t0, t1)

        # get the new values for theta0 and theta1
        sumh0 = calsum(x, y, h_theta, 0)
```

```
        sumh1 = calsum(x, y, h_theta, 1)
        dif0 = (step_size*(-2))/m * sumh0
        dif1 = (step_size*(-2))/m * sumh1

        # cal new theta values
        t0 = t0 + dif0
        t1 = t1 + dif1

        # cal sum for cost funtion and add to the list
        h_theta = hsum(x, t0, t1)
        sumhth = calsum(x, y, h_theta, 2)
        cost = 1/m * (sumhth)
        cost_list.append(cost)
        i = i+1

    return t0, t1, cost, cost_list


def hsum(x, t0, t1):
    y_axis = []
    i = 0
    while i < len(x):
        y[i] = t0+(t1*x[i])
        i = i+1
    return y


# Q1 a (i)
# read in the data
df = pd.read_csv("week1.csv", comment='#')
x = np.array(df.iloc[:, 0], dtype=float)
x = x.reshape(-1, 1)
y = np.array(df.iloc[:, 1])
y = y.reshape(-1, 1)

# Q1 a(ii)
# Normalise the data
norm_x = normalise(x)
norm_y = normalise(y)

# Q1 a(iii)
theta0, theta1, cost, cost_list = gradient_descent(
    norm_x, norm_y, 0, 1, 0.1)  # call the function
plt.scatter(norm_x, norm_y, color='black', label='test data')
guess_plot_y = hsum(norm_x, theta0, theta1)
plt.plot(norm_x, guess_plot_y, color='Green', label='guess data')
plt.title(
    "Q1.a iii Training model vinalla Matthew Flynn 17327199")
plt.xlabel("Input x")
plt.ylabel("Output y")
plt.legend(["Model Vanilla", "Test data"])
plt.show()
```

```
# # Q1 b (i)
plt.plot(list(range(1000)), cost_list, '-b')  # plot the cost function.

theta0, theta1, cost, cost_list = gradient_descent(
    norm_x, norm_y, 0, 1, 0.001)  # call the function
plt.plot(list(range(1000)), cost_list, '-g')  # plot the cost function.

# Calculate the gradient descent and linear regression with different learn-
ing values

theta0, theta1, cost, cost_list = gradient_descent(
    norm_x, norm_y, 0, 1, 0.01)  # call the function
plt.plot(list(range(1000)), cost_list, '-r')  # plot the cost function.


theta0, theta1, cost, cost_list = gradient_descent(
    norm_x, norm_y, 0, 1, 0.1)  # call the function
plt.plot(list(range(1000)), cost_list, '-b')  # plot the cost function.

plt.title("Q1.b Cost with different learning rates Matthew Flynn 17327199")
plt.ylabel("Cost")
plt.xlabel("Iterations")
plt.legend(["alpha = 0.001", "alpha = 0.01", "alpha = 0.1", ])
plt.show()

# Q1 b(iii)
# cost of baseline vs trained
mean = sum(norm_x)/len(norm_x)
constant_output = [mean]
i = 0
while i < (len(norm_x)-1):
    constant_output.append(mean)
    i = i+1
const_cost = (1/len(constant_output)) * \
    calsum(norm_x, norm_y, constant_output, 2)
print(const_cost)
plt.plot(norm_x, constant_output, color='yellow', label='Const data')

# print scatter plot and vanilla for comparison
plt.scatter(norm_x, norm_y, color='black')
guess_plot_y = hsum(norm_x, theta0, theta1)
plt.plot(norm_x, guess_plot_y, color='Green', label='guess data')
plt.title(
    "Q1.b iii Comparing the Const model vers Vanilla model Mat-
thew Flynn 17327199")
plt.xlabel("Input x")
plt.ylabel("Output y")
plt.legend(["constant output", "predictions vanilla"])
plt.show()


# Q1 b(iv)

# using sklearn
```

```
model = LinearRegression().fit(norm_x, norm_y)
y_pre = model.predict(norm_x)
sklearn_cost = (1/len(norm_x)) * calsum(norm_x, norm_y, y_pre, 2)
print("The Sklearn is = ", sklearn_cost)
guess_plot_y = hsum(norm_x, theta0, theta1)
print(model.intercept_, model.coef_)
plt.plot(norm_x, y_pre, color='red', label="sklearn")
plt.plot(norm_x, guess_plot_y, color='Green', label='guess data')
plt.scatter(norm_x, norm_y, color='black')
plt.title(
    "Q1.b iv Comparing the Sklearn model vers Vanilla model Mat-
thew Flynn 17327199")
plt.xlabel("Input x")
plt.ylabel("Output y")
plt.legend(["predictions vanilla", "predictions sklearn", "training data"])
plt.show()
```