Week 1 Assignment: Linear Regression

Ciarán Donegan

4th October 2020

Dataset id: 24-7080–240.

(a) See code in appendix.

(b) (i)

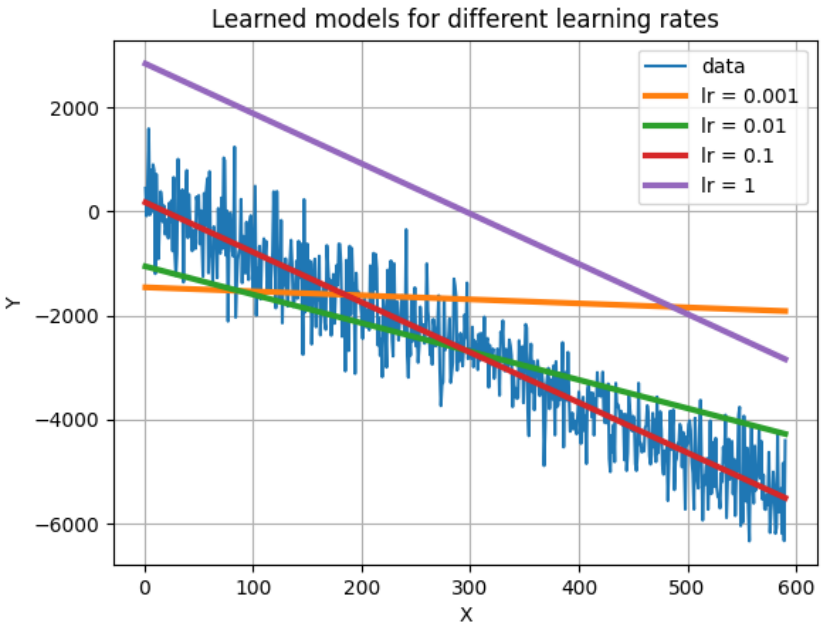

Figure 1: Data along with linear models of different learning rates. The learning rate of 0.1 produces the best fit.

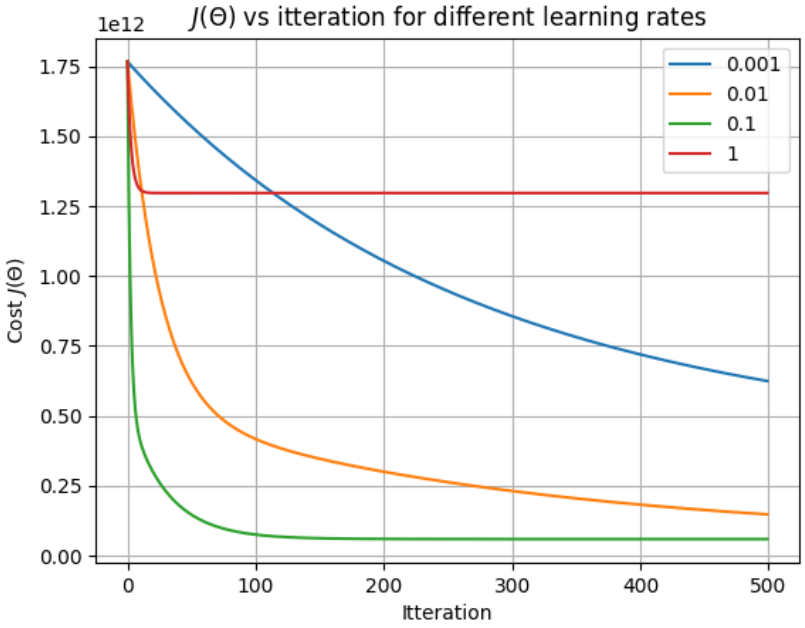

Figure 2: Evolution of cost function at each step in gradient descent process for different learning rates. For a large learning rate ($\alpha = 1$), the model never converges to the correct parameters to minimise the cost function. For a very small learning rate ($\alpha = 0.001$), the model takes a long time to converge and does not converge to correct parameters after 500 update iterations. A learning rate of 0.1 seems to be the optimal choice and the model appears to converge after 100 iterations or so.

(ii) The learned parameters after 500 iterations of gradient descent with a learning rate of 0.1 are:

$$\Theta_0 = -2666.08$$

$$\Theta_1 = -5679.04$$

Note: these Θ values seem very large when you inspect the plotted data, but I believe this is because I am using normalised X values. The normalised X values lie in the range -0.5 to 0.5 while the original X values lie in the range 0 to 590.

(iii) Minimised cost function = $59546907276 \approx 5.9 \times 10^{10}$

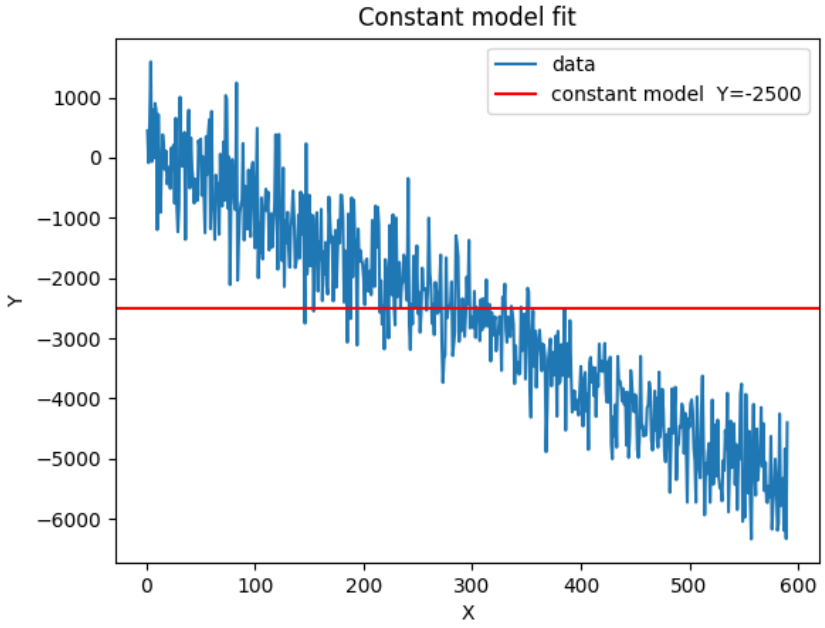

Figure 3: Fitting the data with a constant model that always predicts -2500. i.e. $h_0 = -2500 + (0)X$

Cost function with constant model = 533921345673 $\approx 5.3 \times 10^{11}$.

(iv) Comparing the sklearn linear regression with my own, I noticed that if I don't normalise the X values for the sklearn method I will get different model parameters.

$$\Theta_0 = 183.7.08$$

$$\Theta_1 = -9.64$$

If I do normailise the X values before using sklearn Linear Regression, the parameters agree with those that I got when I implemented gradient descent from scratch.

$$\Theta_0 = -2866.08$$

$$\Theta_1 = -5680.28$$

A Linear regression from scratch

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


def read_csv(filename):
    df = pd.read_csv(filename, comment="#")
    # print(df.head())
    X = np.array(df.iloc[:,0])
    X = X.reshape(-1, 1)
    y = np.array(df.iloc[:,1])
    y = y.reshape(-1,1)
    return X, y

def normalise(values):
    mean = values.mean()
    sigma = values.max() - values.min()
    norm = (values - mean) / sigma
    return norm

def sgd(X, y, lr, epochs):
    m = len(X) # number of features
    assert len(X) == len(y) # ensure labels and fetaures are same size

    # initialise paramaters to 0
    theta_0 = theta_1 = 0

    past_costs = []
    for _ in range(epochs):
        # function to learn
        h = theta_0 + theta_1 * X
        # cost function
        cost = (1/2*m) * ((h - y)**2).sum()
        past_costs.append(cost)
        # update paramaters
        theta_0 -=  ( 2 * lr * (h-y).sum() ) / m
        theta_1 -=  ( 2 * lr * (X * (h-y)).sum() )  / m

    return theta_0, theta_1, past_costs


X, y = read_csv("week1.csv")
X_norm = normalise(X)
epochs = 500

plt.figure(1)
plt.plot(X, y)

learned_values = []

# Perfrom gradient descent for different learning rates and plot the results
```

```
for i, lr in enumerate([0.001, 0.01, 0.1, 1]):
    theta_0, theta_1, past_costs = sgd(X_norm, y, lr, epochs)
    print ("lr = ", lr, theta_0, theta_1)
    print("cost=", past_costs[-1])
    y_hat = theta_0 + theta_1 * X_norm

    plt.figure(1)
    plt.plot(X, y_hat, linewidth=3)
    plt.figure(2)
    plt.plot(range(epochs), past_costs)



plt.figure(1)
plt.legend(["data", "lr = 0.001", "lr = 0.01", "lr = 0.1", "lr = 1"])
plt.title("Learned models for different learning rates")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.savefig( "models.png")
plt.figure(2)
plt.legend(["0.001", "0.01", "0.1", "1"])
plt.title("$J(\Theta)$ vs itteration for different learning rates")
plt.xlabel("Itteration")
plt.ylabel("Cost $J(\Theta)$")
plt.grid()
plt.savefig("cost_function.png")


# Compare with cost function of baseline model that always predicts
# a constant value

m = len(X) # number of features
h = -2500  # always predict a constant value
cost = (1/2*m) * ((h - y)**2).sum()
print("cost for constant model", cost)

plt.figure(3)
plt.plot(X, y)
plt.axhline(y=h, color='r')
plt.legend(["data", "constant model  Y=-2500"])
plt.title("Constant model fit")
plt.xlabel("X")
plt.ylabel("Y")
plt.savefig("constant_model.png")
```

B Linear regression using sklearn

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

def read_csv(filename):
    df = pd.read_csv(filename, comment="#")
    # print(df.head())
    X = np.array(df.iloc[:,0])
    X = X.reshape(-1, 1)
    y = np.array(df.iloc[:,1])
    y = y.reshape(-1,1)
    return X, y


def normalise(values):
    mean = values.mean()
    sigma = values.max() - values.min()
    norm = (values - mean) / sigma
    return norm

X, y = read_csv("week1.csv")
X_norm = normalise(X)

reg = LinearRegression().fit(X_norm, y)
print("y-intercept", reg.intercept_)
print("slope = ", reg.coef_)
```