Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS7CS4/CSU44061 - Machine Learning
Assignment 1:

Student Name:
Hugh Jordan

Student Number:
16321743

Assignment 1

(a) A short python program was written that reads in the data downloaded (# id:15--738—150), normalises it and then uses gradient descent to train a linear regression model. This program is shown in the appendix.

(b) The program was then used to train a linear regression model on the downloaded data.

(i) The learning rates 0.001, 0.01 and 0.1 were used to train a linear regression model. The cost function for each of these models is shown in Figure 1. It is evident from Figure 1 that the cost function decreases very slowly for small learning rates and, conversely, the cost function decreases very quickly for large learning rates. However, if the learning rate is too low, the gradient descent program may fail to converge on the minimum. This is evident in Figure 2, when the model that was trained with a learning rate of 0.001 failed to converge on the minimum. However, the models that were trained with a learning rate of 0.01 and 0.1 did not fail to converge on the minimum. Similarly, if the learning rate is too large, the program may overshoot the minimum. This is not evident in Figure 2, however.

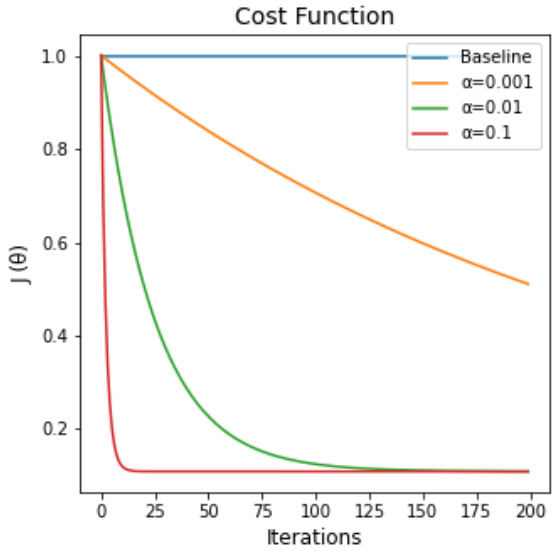

Figure 1. Cost Function with a range of different Learning Rates

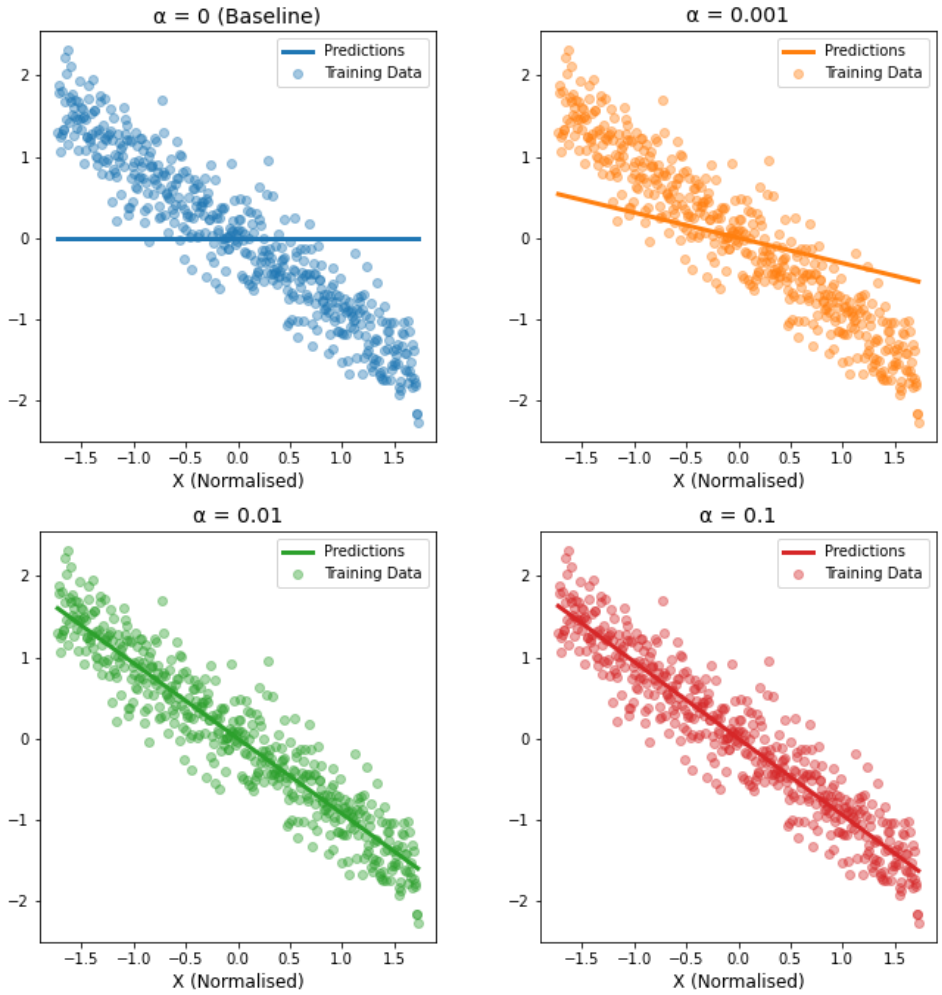

Figure 2. Linear Regression Model with a range of different Learning Rates

(ii) The parameter values for each of the linear regression models after they had been trained on the downloaded data are shown below in Table 1.

α	θ_0	θ_1
Baseline (0.0)	0.0	0.0
0.001	-1.10682917e-16	-0.31141844
0.01	-3.32760016e-16	-0.92723967
0.1	-3.24853062e-16	-0.94383988

Table 1. Parameter Values of Linear Regression Model

(iii) The value of the cost function for each of the trained models and the baseline model is shown in Table 2. It is evident from Table 2 that the value of the cost function for each of the trained models is much lower than that of the baseline model. This is because the gradient descent algorithm attempts to minimise the cost function in order to train the model for the given data. As a result, the value of the cost function for the train models is significantly lower when compared to the baseline model.

The baseline model was chosen by training the model using a learning rate of 0. As a result, the parameter values remained constant after each iteration of the gradient descent algorithm, and thus the cost function always predicted a constant value.

α	$J(\theta)$
0.0 (Baseline)	1.00000000
0.001	0.51072780
0.01	0.10945322
0.1	0.10916629

Table 2. Cost Function of Linear Regression Model

(iv) A linear regression model was trained using sklearn. This model is shown in Figure 3, and the corresponding parameter values for the model are shown in Table 3. It is evident from Figure 3 that the model is very similar to each of the trained models which had converged on the minimum. Additionally, it can be deduced from Table 3 that the difference between the two sets of parameters values is almost negligible. Therefore, the model trained using gradient descent is equal to the model trained using sklearn.

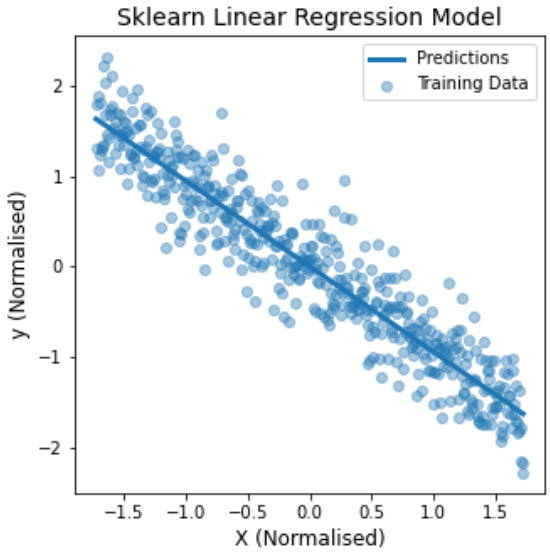

Figure 3. Linear Regression Model using sklearn

Model	θ_0	θ_1
Gradient Descent (α=0.1)	-3.24853062e-16	-0.94383988
Sklearn	-3.97885432e-16	-0.94383988

Table 3. Parameter Values of Sklearn Linear Regression Model

Appendix:

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

df = pd.read_csv("week1.csv",comment='#')
x=np.array(df.iloc[:,0]); x=x.reshape(-1, 1)
y=np.array(df.iloc[:,1]); y=y.reshape(-1, 1)

mean_x = np.mean(x)
std_x = np.std(x)
x = (x - mean_x)/std_x
mean_y = np.mean(y)
std_y = np.std(y)
y = (y - mean_y)/std_y

m = np.size(x)
learning_rates = [0, 0.001, 0.01, 0.1]
iterations = 200
cost_funcs = []
epsilons = []
for learning_rate in learning_rates:
   cost_func = []
   epsilon = [0,0]
   for i in range(iterations):
      h = epsilon[0] + epsilon[1] * x
      cost_func.append(1/m * np.sum((h - y) ** 2))
      delta = [0]*2
      delta[0] = -(2 * learning_rate)/m * np.sum(h - y)
      delta[1] = -(2 * learning_rate)/m * np.sum((h - y) * x)
      epsilon[0] = epsilon[0] + delta[0]
      epsilon[1] = epsilon[1] + delta[1]
   cost_funcs.append(cost_func)
   epsilons.append(epsilon)

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(5, 5))
ax.set_title('Cost Function', fontsize=14)
ax.set_xlabel('Iterations', fontsize=12)
ax.set_ylabel('J (θ)', fontsize=12)
for i in range(len(learning_rates)):
   ax.plot(range(iterations), cost_funcs[i])
labels = ['Baseline']
for learning_rate in learning_rates[1:]:
   labels.append('α='+str(learning_rate))
ax.legend(labels, loc='upper right')
```

```
plt.savefig('cost_functions')
plt.show()

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
ax = ax.flatten()
fig.tight_layout(pad=4.0)
for itr, learning_rate in enumerate(learning_rates):
   if itr == 0:
      ax[itr].set_title('α = ' + str(learning_rate) + ' (Baseline)', fontsize=14)
   else:
      ax[itr].set_title('α = ' + str(learning_rate), fontsize=14)
   ax[itr].set_xlabel('X (Normalised)', fontsize=12)
   ax[itr].set_ylabel('Y (Normalised)', fontsize=12)
   ax[itr].scatter(x, y, linewidth=1, color='C' + str(itr), alpha=0.4)
   ax[itr].plot(x, epsilons[itr][0] + epsilons[itr][1] * x, linewidth=3, color='C' + str(itr))
   ax[itr].legend(['Predictions', 'Training Data'], loc='upper right')
plt.savefig('lr_with_learning_rate')
plt.show()

regr = LinearRegression()
regr.fit(x, y)
y_pred = regr.predict(x)

print('Parameter Values')
print('%-9s %-12s %-12s' % ('α', 'e0', 'e1'))
for itr, learning_rate in enumerate(learning_rates):
   print('%-9f %-12.8f %-12.8f' % (learning_rate, epsilons[itr][0], epsilons[itr][1]))
print()

print('Cost Function')
print('%-9s %-12s' % ('α', 'J(θ)'))
for itr, learning_rate in enumerate(learning_rates):
   print('%-9f %-12.8f' % (learning_rate, cost_funcs[itr][-1]))
print()

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(5, 5))
ax.set_title("Sklearn Linear Regression Model", fontsize=14)
ax.set_xlabel("X (Normalised)", fontsize=12)
ax.set_ylabel("y (Normalised)", fontsize=12)
plt.scatter(x, y, linewidth=1, alpha=0.4)
plt.plot(x, y_pred, linewidth=3)
plt.legend(["Predictions","Training Data"], loc='upper right')
plt.savefig('sklearn_linear_regression_model')
plt.show()

print('Sklearn Parameter Values:')
print('%-24s %-16s %-12s' % ('Model','Intercept', 'Coefficient'))
```

```
print('%-24s %-16.8e %-12.8f' % ('Gradient Descent(α=' + str(learning_rates[-1]) + ')',
epsilons[-1][0], epsilons[-1][1]))
print('%-24s %-16.8e %-12.8f' % ('Sklearn', regr.intercept_, regr.coef_))
```