Assignment Week 1 Machine Learning

Saul O’Driscoll (17333932)

a)

- Python code written used the code from the worksheet to read in the data.
- Data was normalised to values between 0 and 1. X-Axis values were not normalised.
- Applied the vanilla python linear regression with gradient descent to the dataset.
  - Notes:
    - m and b were initalised to be 0
    - At first my coefficient and intercept parameters were exploding and quickly overflowing either becoming too large or too small. This lead to them becoming NaNs and python broke down.
    - From my past experience with machine learning algorithms I thought this may have been because of the exploding gradient problem but that mainly applies to neural nets.
    - I realised eventually that my learning rate was too large and because the normalised values at certain points were very very small the algorithm was not able to converge. After adjusting this it worked.
    - The problem I was having now is that my learning rate was so small that my y-intercept value was not converging even after 10,000 iterations because the learning rate was miniscule (0.000001) so I changed the learning rate for the b update rule to 0.01. This fixed the issue.
    - 2 different learning rates, interesting.

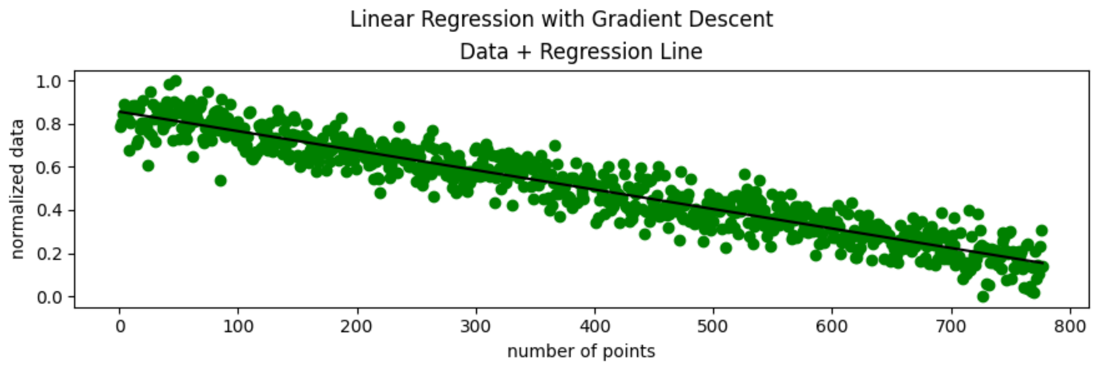
b)

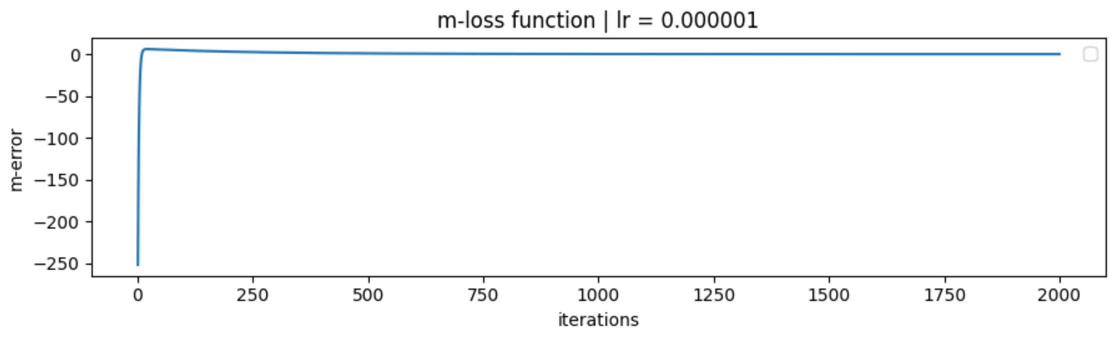
- The final value of my linear regression were this:
  - m = -0.0009027382385940276 b = 0.8555944156459061
- The last few value of my loss function / error were very low, here are the last 4 values of each parameter:
    - m-loss:
      - 0.002160889356988141
      - 0.0021522162228682835
      - 0.002143577900013179
      - 0.002134974248659208
    - b-loss:
      - -0.00011067034850721 69
      - -0.00011022615233732285
      - -0.00010978373903190615
      - -0.00010934310143519913

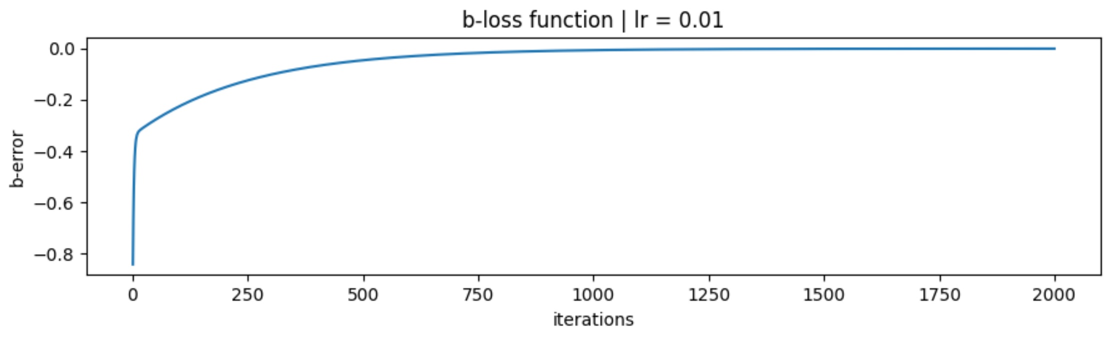
- b iii)
- As a baseline model I used the line of y = -0.0005*x+1
- To get the cost function of these I let the simulation run 3 iterations of gradient descent to get the gradient errors. This would give me an approximation of how far they are off initially.
- Here they are:
    - m loss [229.31068267786, 148.87198629520637, 96.52681685618208]
    - b loss [0.5024306947932468, 0.3451509348147833, 0.24272595686162352]
- As you can see, they are quite far off compared to the last values of my own line.

Below is the output of my code. In the first plot you can see the data and the final line being plotted in black, the points in green.

The M-loss and B-loss after that.



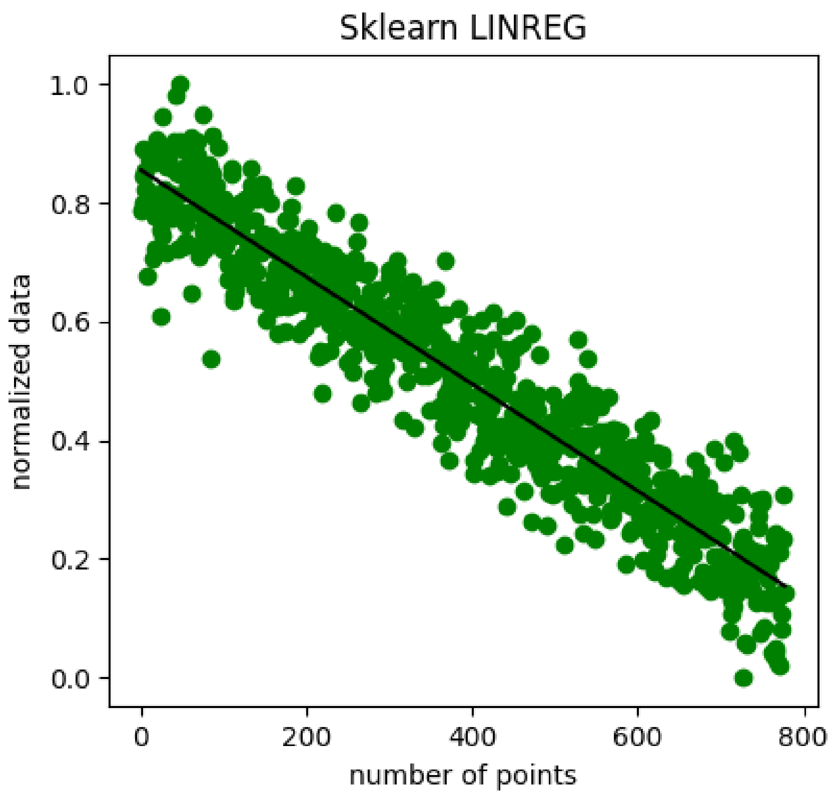




b iv)

- Here is my sklearn graph. Final line ends up being very similar:
    - m = -0.0009032659006308524
    - b = 0.855864658735033

This means that the vanilla python implementation was quite accurate



CODE:

VANILLA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("data2.csv",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)

norm = []

#normalising the data
ymin, ymax = min(y), max(y)
for i, val in enumerate(y):
norm.append((val-ymin) / (ymax-ymin))

iter = 2000
class LinearRegGD:
def __init__(self, learning_rate=0.000001, iterations=1000):
self.learning_rate, self.iterations = learning_rate, iterations
self.m_losses = []
self.b_losses = []
def fit(self, X, y):
m = 0
b = 0
n = 931
print(n)
for _ in range(self.iterations):
m_gradient = (-2/n) * np.sum(X*(y - (m*X + b)))
self.m_losses.append(m_gradient)
b_gradient = (-2/n) * np.sum(y - (m*X + b))
self.b_losses.append(b_gradient)
b = b - (self.learning_rate*10000 * b_gradient)
m = m - (self.learning_rate * m_gradient)
self.m, self.b = m, b
def losses(self):
return self.m_losses, self.b_losses

def predict(self, X):
print("m = ", self.m, "b = ", self.b)
return self.m*X + self.b
```

```
clf = LinearRegGD(iterations=iter)
clf.fit(X, norm)
m_l, b_l = clf.losses()

print("m loss", m_l[-5:-1])
print("b loss", b_l[-5:-1])

fig, (ax1, ax2, ax3) = plt.subplots(3)
fig.tight_layout(pad=2.0)
fig.suptitle("Linear Regression with Gradient Descent")
ax1.scatter(X, norm, color='green')
ax1.plot(X, clf.predict(X), color='black')
ax1.plot(X, -0.0005*X+1, color = "black")
ax1.set_ylabel('normalized data')
ax1.set_xlabel('number of points')
ax1.set_title('Data + Regression Line')
ax2.plot(range(1, iter+1), m_l)
ax2.set_ylabel('m-error')
ax2.set_xlabel('iterations')
ax2.set_title('m-loss function | lr = 0.000001')
ax3.plot(range(1, iter+1), b_l)
ax3.set_ylabel('b-error')
ax3.set_xlabel('iterations')
ax3.set_title('b-loss function | lr = 0.01')

plt.show()
```

SKLEARN:

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

df = pd.read_csv("data2.csv",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)

norm = []

#normalising the data
ymin, ymax = min(y), max(y)
for i, val in enumerate(y):
norm.append((val-ymin) / (ymax-ymin))

reg = LinearRegression().fit(X, norm)
m = reg.coef_[0][0]
b = reg.intercept_[0]
print(m,b)
plt.scatter(X, norm, color='green')
plt.plot(X, m*X+b, color = "black")
plt.ylabel('normalized data')
plt.xlabel('number of points')
plt.gca().set_title("Sklearn LINREG")

plt.show()
```