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

School of Computer Science and Statistics

CS7CS4 Machine Learning
Week 1 Assignment

Boris Flesch
20300025

October 8, 2020

MSc Computer Science, Intelligent Systems

1 Downloaded Dataset

id:2-1018.8–10

2 Questions

(a) All sections of the Python code have been commented in accordance with each question of the assignment (see appendix A.1 or attached ZIP file). Figure 1 shows a visual representation of the data read from the downloaded CSV file:

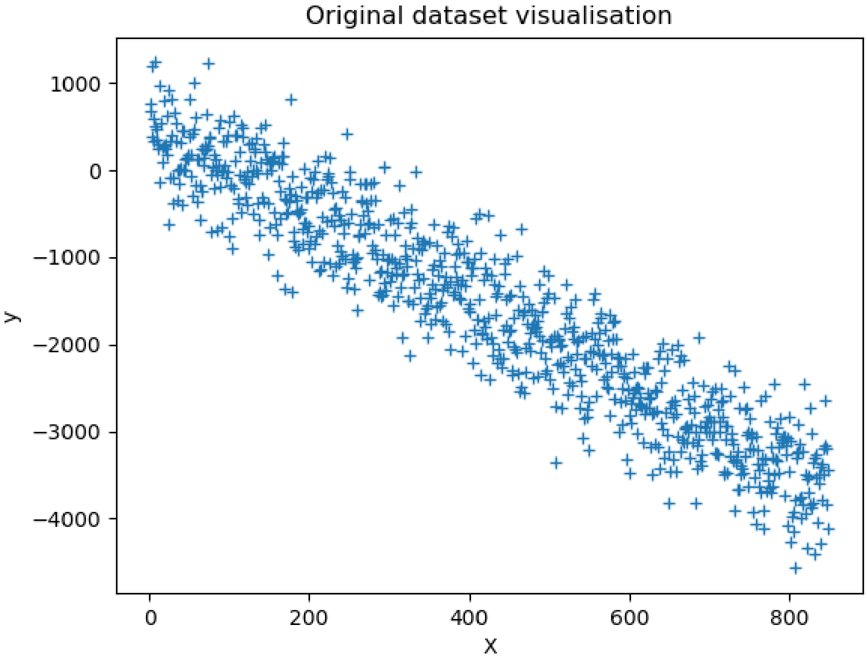

Figure 1

(b) (i) The learning rate values 0.001, 0.01, 0.1 and 1 all seem to be converging but at a different speed (Figure 2). More iterations would be required to observe the evolution of the cost function for $\alpha = 0.01$ and $\alpha = 0.001$.
Note about θ initial value: after having executed several tests with different initial values for θ_0 and θ_1, that seems to have an insignificant impact on the values of the trained model parameters. The only observation that I have had is that a big initial value (i.e. $\theta_0 = \theta_1 = 1000$) will prevent the gradient descent from converging with a learning rate $\alpha = 1$. Therefore, I decided to keep initial values $\theta_0 = \theta_1 = 0$.

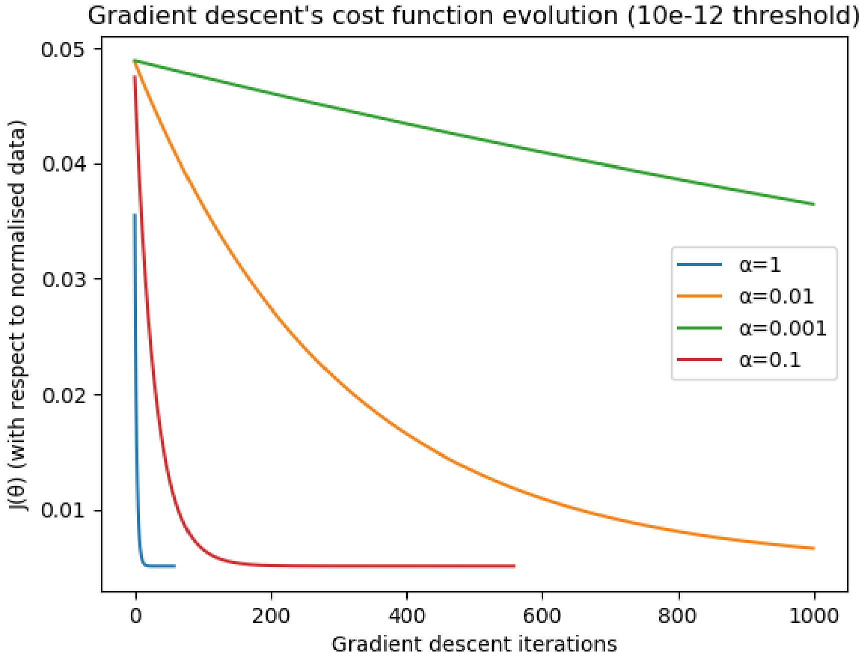

Figure 2

(ii) After having been trained on my downloaded data, the parameter values of my linear regression model are the following: $\theta_0 = 518.027317$, $\theta_1 = -4.964503$

(iii) The value of the cost function of my linear regression model is $J(\theta) = 172100.778305$. As a baseline model, I chose $\theta_0 =$ "mean of output y values" and $\theta_1 = 0$. This baseline model gives $J(\theta_{baseline}) = 1652762.172120$.

As we can see on Figure 3, $J(\theta_{baseline}) >> J(\theta)$. This confirms that the linear regression model is much more accurate than the baseline model which does actually not follow the trend/path of the training data (see green dashed line on Figure 3).

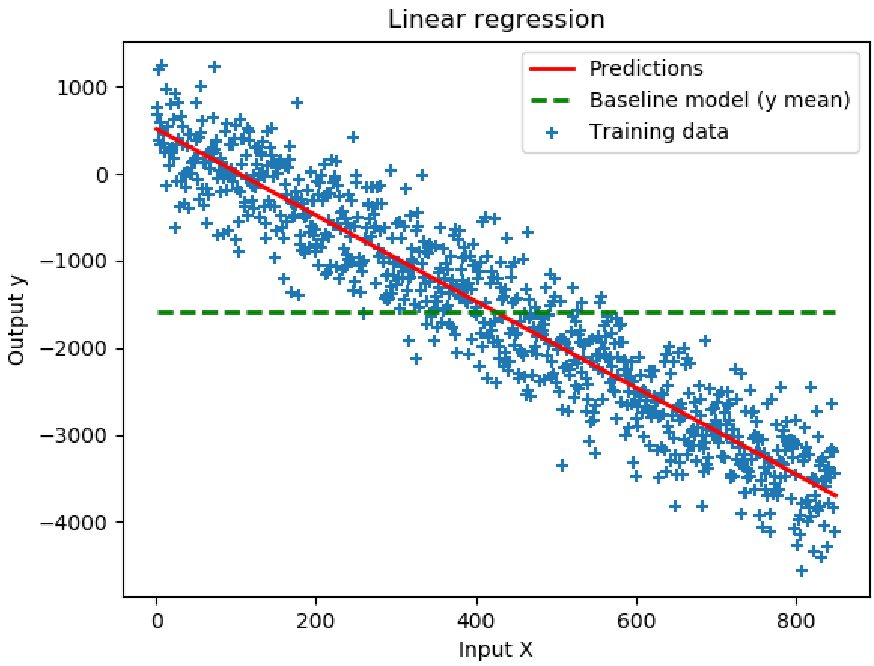

Figure 3

(iv) The linear regression model trained using sklearn (Figure 4) has the following parameter and cost function values: $\theta_0 = 518.198839$, $\theta_1 = -4.964907$ and $J(\theta) = 172100.768521$. These values are very close to the ones obtained using the gradient descent algorithm in vanilla Python, thereby confirming the accuracy of the linear regression model using gradient descent.

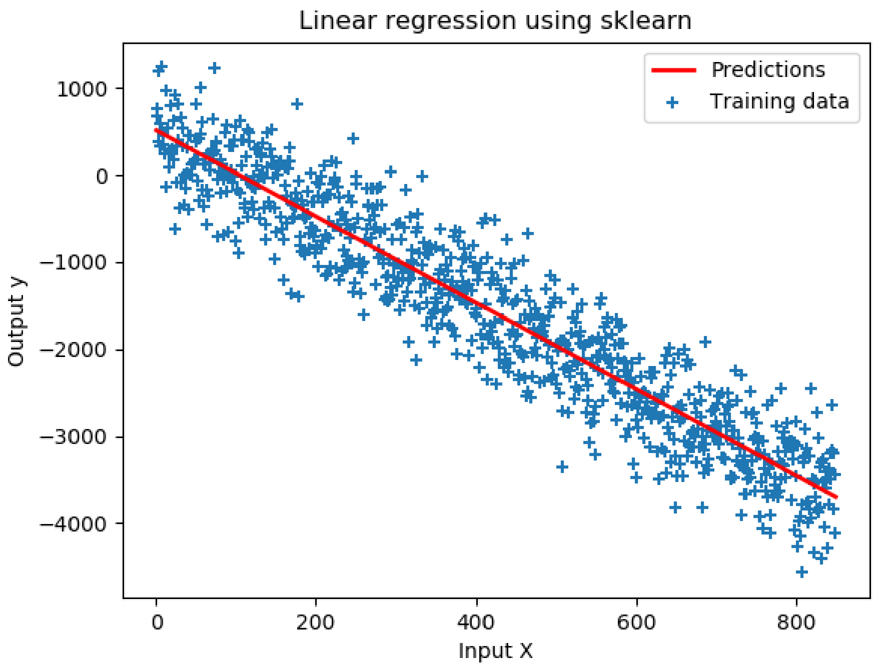

Figure 4

A Appendix

A.1 Python Code

```python
#!/usr/bin/env python
# coding: utf-8

# CS7CS4/CSU44061 Machine Learning
# Week 1 Assignment
# Boris FLesch (20300025)
#
# Downloaded dataset
# id:2-1018.8--10

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

# (a)(i) Read data
def readData(file):
    """
    Read data from a file and outputs reshaped arrays

    :param file: path to the source file
    :return: 2 numpy arrays X and y
    """
    df = pd.read_csv(file, comment="#")
    X = np.array(df.iloc[:, 0]).reshape(-1,1)
    y = np.array(df.iloc[:, 1]).reshape(-1,1)

    return X, y

X, y = readData("week1.csv")

# Visualise data
plt.plot(X, y, '+')
plt.title("Original dataset visualisation")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
#plt.savefig("original-dataset.png")


# (a)(ii) Normalise data
def normaliseData(X):
    """
    Normalise data using average as shift and max-min as scaling factor

```

```python
    :param X: Data array to normalise
    :return: Normalised data, shift value, scaling factor value
    """
    shift = np.average(X)
    scalingFactor = np.max(X) - np.min(X)
    X = (X - shift) / scalingFactor
    return X, shift, scalingFactor

# Linear Regression
def linearRegression(X_in, y_in, alpha=[0.1], theta=[0,0], costThreshold=10e-10, maxIterations=1000):
    """
    Process a linear regression on data

    :param X_in: X data (array)
    :param y_in: y data (array)
    :param alpha: Value of the learning rate
    :param theta: Initial values for theta (array)
    :param costThreshold: Cost function limit between two iterations of the gradient descent
    :param maxIterations: Maximum number of iterations for the gradient descent
    :return: theta (mapped to original data, i.e. "de-normalised"), errors (with respect to normalised data), finalError (mapped to original data)
    """

    # Copy of input arrays
    X = np.copy(X_in)
    y = np.copy(y_in)

    # Normalise data
    X, shiftX, scalingFactorX = normaliseData(X)
    y, shiftY, scalingFactorY = normaliseData(y)

    # (a)(iii) Gradient Descent
    costs = []
    m = X.size

    for k in range(maxIterations):
        # Theta calculation
        theta[0] += (-2*alpha / m) * sum((theta[0] + theta[1] * X) - y)
        theta[1] += (-2*alpha / m) * sum(((theta[0] + theta[1] * X) - y) * X)

        # Cost calculation
        cost = 1/m * sum(np.power((theta[0] + theta[1] * X) - y, 2))
        costs.append(cost)
```

```

        if (k > 2 and abs(costs[-1] - costs[-2]) <= costThreshold):
            break

    costs = np.array(costs)

    # Theta calculation (mapping value to original data)
    theta = [
        theta[0] * scalingFactorY + shiftY - theta[1] * shiftX * scalingFactorY / scalingFactorX,
        theta[1] / scalingFactorX * scalingFactorY
    ]

    # Final cost value (mapping value to original data)
    finalCost = 1/m * sum(np.power((theta[0] + theta[1] * X_in) - y_in, 2))

    return theta, costs, finalCost


# (b)(i) Linear regression using different learning rates
alphas = [1, 0.01, 0.001, 0.1]
for alpha in alphas:
    theta, costs, finalCost = linearRegression(X, y, alpha, theta=[0,0], costThreshold=10e-12, maxIterations=1000)
    plt.plot(range(costs.size), costs, '-', label="=" + str(alpha))

plt.legend()
plt.title("Gradient descent's cost function evolution (10e-12 threshold)")
plt.xlabel("Gradient descent iterations")
plt.ylabel("J(theta) (with respect to normalised data)")
plt.show()
#plt.savefig("cost-function-evolution.png")

# (b)(ii)
print("Linear regression model using vanilla Python:")
print("theta0 = %f ; theta1 = %f" % (theta[0], theta[1]))

# (b)(iii)
print("J(theta) = %f" % finalCost)

# Baseline model (theta0 = y mean, theta1 = 0)
bTheta = [np.mean(y), 0]
bCost = 1/X.size * sum(np.power((bTheta[0] + bTheta[1] * X) - y, 2))
print("J(theta_baseline) = %f\n" % bCost)

# Plot predictions, training data and baseline model
```

```
plt.title("Linear regression")
plt.xlabel("Input X")
plt.ylabel("Output y")
plt.scatter(X, y, marker='+', label="Training data")
plt.plot(X, theta[0] + theta[1] * X, 'r-', linewidth=2,
    label="Predictions")
plt.plot(X, bTheta[0] + bTheta[1] * X, color='g', linestyle='dashed',
    linewidth=2, label="Baseline model (y mean)")
plt.legend()
plt.show()
#plt.savefig("linear-regression.png")


# (b)(iv) Linear Regression model with sklearn
from sklearn.linear_model import LinearRegression

X, y = readData("week1.csv")

model = LinearRegression().fit(X, y)
cost = 1/X.size * sum(np.power((model.intercept_ + model.coef_ * X) - y,
    2))

print("Linear regression model using sklearn:")
print("theta0 = %f ; theta1 = %f" % (model.intercept_, model.coef_))
print("J(theta) = %f" % cost)

plt.title("Linear regression using sklearn")
plt.scatter(X, y, marker='+', label="Training data")
plt.plot(X, model.intercept_ + model.coef_ * X, 'r-', linewidth=2,
    label="Predictions")
plt.legend()
plt.xlabel("Input X")
plt.ylabel("Output y")
plt.show()
#plt.savefig("linear-regression-sklearn.png")
```