CSE 575 Homework 1

Sunayana Gupta
1222389090

- [4pts] implement your own version of the method of least-squares, compute and report θ_0 and θ_1 that minimize the residual sum of squares,

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

Solution:

```
def least_squares(X,y,m,c):
  X_mean = np.mean(X)
  Y_mean = np.mean(y)

  num = 0
  den = 0
  for i in range(len(X)):
      num += (X[i] - X_mean)*(y[i] - Y_mean)
      den += (X[i] - X_mean)**2
  m = num / den
  c = Y_mean - m*X_mean

  return [c, m]
```

Theta = [array([152.91886183]), array([938.23786125])]

- [4pts] implement your own version of the gradient descent algorithm, compute and report θ_0 and θ_1 that minimize the mean squared error

$$\sum_{i=1N} 1N(y_{(i)} - h\theta(x_{(i)}))$$

Solution:

```
def gradient_descent(X, y, theta, alpha, n_iter):
  n_samples = len(y)
  X = np.hstack((np.ones((n_samples, 1)), X))
  y = y[:, np.newaxis]
  for i in range(n_iter):
    theta = theta - ((alpha/n_samples) *X.T @ (X @
theta - y))
  return theta
```

Theta = [[152.918864] [938.23328304]]

- [2pts] derive the analytical expression of the gradient if the loss is defined as

$$\sum_{i=1}^{N} \frac{1}{2}\left(y^{(i)} - h_\theta(x^{(i)})\right)^2 + \frac{\lambda}{2}\|\theta\|_2^2, \text{ where } \theta = [\theta_0, \theta_1]^T$$

Q- Derive the analytical expression of the gradient if loss is defined as

$$\sum_{i=1}^{N} \frac{1}{2} \left(y^{(i)} - h_\theta x^{(i)}\right)^2 + \frac{\lambda}{2} \|\theta\|_2^2$$

where $\theta = [\theta_0, \theta_1]^T$, $h_\theta(x) = \theta_0 + \theta_1 x$

Solution- We know that,

$$\theta_1 \text{new} = \theta_1 - \eta \frac{dL}{d\theta_1}$$

$$\theta_0 \text{new} = \theta_0 - \eta \frac{dL}{d\theta_0}$$

Given that,

$$L = \sum_{i=1}^{N} \frac{1}{2} \left(\theta_1 x^{(i)} + \theta_0 - y^{(i)}\right)^2 + \frac{\lambda}{2}\left(\theta_1^2 + \theta_0^2\right)$$

$$\frac{\partial L}{\partial \theta_1} = \frac{1 \times 2}{2} \sum_{i=1}^{N} \left(h(x^{(i)}) - y^{(i)}\right) x + \lambda \theta_1$$

$$\frac{\partial L}{\partial \theta_0} = \frac{1 \times 2}{2} \sum_{i=1}^{N} \left(h(x^{(i)}) - y^{(i)}\right) + \lambda \theta_0$$

Therefore,

$$\theta_1 \text{new} = \theta_1 - \eta \left(\sum_{i=1}^{N} \left(h(x)^{(i)} - y^{(i)}\right) x + \lambda \theta_1 \right)$$

$$\theta_0 \text{new} = \theta_0 - \eta \left(\sum_{i=1}^{N} \left(h(x)^{(i)} - y^{(i)}\right) + \lambda \theta_0 \right)$$

Complete code –

```
# -*- coding: utf-8 -*-
"""CSE575_HW1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1kguZUDVX6OxyfhpDofzPyphXLyagMQvH

# Linear regression [10 pts]

In this homework, you will implement solution algorithms for linear regression.


## Import libraries
Let's begin by importing some libraries.
"""

# Commented out IPython magic to ensure Python compatibility.
print(__doc__)
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
# %matplotlib inline

"""## Load dataset

Now, we are importing a dataset of diabetes. You can check the details on this dataset
here: https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset.

The dataset consists of 442 observations with 10 attributes ($X$) that may affect the
progression of diabetes ($y$). Ten baseline variables, age, sex, body mass index,
average blood pressure, and six blood serum measurements were obtained for each of $n$
= 442 diabetes patients, as well as the response of interest, a quantitative measure
of disease progression one year after baseline.
"""

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
print('The shape of the input features:',diabetes_X.shape)
print('The shape of the output varaible:',diabetes_y.shape)

"""We will choose just one attribute from the ten attributes as an input variable."""

# Use only one feature
diabetes_X_one = diabetes_X[:, np.newaxis, 2]
print(diabetes_X_one.shape)
```

```
"""## Dataset split

Now, we split the dataset into two parts: training set and test set.

- training set: 422 samples
- test set: 20 samples
"""

# Split the data into training/testing sets
diabetes_X_train = diabetes_X_one[:-20]
diabetes_X_test = diabetes_X_one[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

print('Training input variable shape:', diabetes_X_train.shape)
print('Test input variable shape:', diabetes_X_test.shape)

from matplotlib import pyplot as plt
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True

plt.grid()
plt.plot(diabetes_X_one, diabetes_y, marker="o", markersize=5, markeredgecolor="red",
markerfacecolor="green")
plt.show()

"""## Linear regression

Assume that we have a hypothesis $$ h_{\theta}(x) = \theta_0 + \theta_1 x. $$

Your tasks:

- [4pts] implement your own version of the method of least-squares, compute and report
$\theta_0$ and $\theta_1$ that minimize the residual sum of squares,
$$ \sum_{i=1}^{N} \frac{1}{2}( y^{(i)} - h_{\theta}(x^{(i)})^2$$

- [4pts] implement your own version of the gradient descent algorithm, compute and
report $\theta_0$ and $\theta_1$ that minimize the mean squared error $$
\sum_{i=1}^{N} \frac{1}{N}( y^{(i)} - h_{\theta}(x^{(i)})^2$$

- [2pts] derive the analytical expression of the gradient if the loss is defined as
$$ \sum_{i=1}^{N} \frac{1}{2}( y^{(i)} - h_{\theta}(x^{(i)})^2 + \frac{\lambda}{2} \|
\theta \|_2^2, $$
where $\theta = [\theta_0, \theta_1]^{\intercal}$
```

```
To check whether your computation is correct, consider using an API such as Scikit
learn linearregression.
"""

def least_squares(X,y,m,c):
  X_mean = np.mean(X)
  Y_mean = np.mean(y)

  num = 0
  den = 0
  for i in range(len(X)):
      num += (X[i] - X_mean)*(y[i] - Y_mean)
      den += (X[i] - X_mean)**2
  m = num / den
  c = Y_mean - m*X_mean

  return [c, m]

def gradient_descent(X, y, theta, alpha, n_iter):
  n_samples = len(y)
  X = np.hstack((np.ones((n_samples, 1)), X))
  y = y[:, np.newaxis]
  for i in range(n_iter):
    theta = theta - ((alpha/n_samples) *X.T @ (X @ theta - y))
  return theta

def predict(X,theta):
  n_samples = len(X)
  X = np.hstack((np.ones((n_samples, 1)), X))
  y_pred = X @ theta
  return y_pred

def score(X, y,theta):
  n_samples = np.size(X, 0)
  X = np.hstack((np.ones((n_samples, 1)), X))
  y = y[:, np.newaxis]
  y_pred = X @ theta
  score = 1 - (((y - y_pred)**2).sum() / ((y - y.mean())**2).sum())
  return score

n_iters = 60000
n_features = np.size(diabetes_X_train, 1)
learning_rate = 0.09
m=0
c=0
(optimal_params) = least_squares(diabetes_X_train,diabetes_y_train , m,c)
print("Optimal parameters are: \n", optimal_params, "\n")
accuracy= score(diabetes_X_test,diabetes_y_test, optimal_params)
```

```python
print("Score: ", accuracy, "\n")

n_iters = 60000
n_features = np.size(diabetes_X_train, 1)
learning_rate = 0.09
theta = np.zeros((n_features + 1, 1))
(optimal_params) = gradient_descent(diabetes_X_train,diabetes_y_train , theta,
learning_rate, n_iters)
print("Optimal parameters are: \n", optimal_params, "\n")
accuracy= score(diabetes_X_test,diabetes_y_test, optimal_params)
print("Score: ", accuracy, "\n")

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression as lr

our_parameters = gradient_descent(diabetes_X_train,diabetes_y_train , theta,
learning_rate, n_iters)
sklearn_regressor = lr().fit(diabetes_X_train, diabetes_y_train)
our_train_accuracy = score(diabetes_X_train,diabetes_y_train, optimal_params)
sklearn_train_accuracy = sklearn_regressor.score(diabetes_X_train, diabetes_y_train)
our_test_accuracy = score(diabetes_X_test,diabetes_y_test, optimal_params)
sklearn_test_accuracy = sklearn_regressor.score(diabetes_X_test, diabetes_y_test)

print("sklearn_train_accuracy: " ,sklearn_train_accuracy)
print("our_train_accuracy: ", our_train_accuracy)
print("sklearn_test_accuracy: ",sklearn_test_accuracy)
print("our_test_accuracy: " ,our_test_accuracy)
```