CSE 575 Homework 1

Sunayana Gupta

1222389090

* [4pts] implement your own version of the method of least-squares, compute and report 𝜃0 and 𝜃1 that minimize the residual sum of squares,

∑𝑖=1𝑁12(𝑦(𝑖)−ℎ𝜃(𝑥(𝑖))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 𝜃0 and 𝜃1 that minimize the mean squared error

∑𝑖=1𝑁1𝑁(𝑦(𝑖)−ℎ𝜃(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

∑𝑖=1𝑁12(𝑦(𝑖)−ℎ𝜃(𝑥(𝑖))2+𝜆2‖𝜃‖22, where 𝜃=[𝜃0,𝜃1]⊺

Text, letter

Description automatically generated

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)

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)