```
#%% md
# AI305 Lab 3 - Linear Regression

Objective: Build Linear regression model using gradient descent and normal equations


#%% md
#### Read the dataset and show the first five records.

Datasets are provided as csv file, and `pandas` library is used to read csv files.

`read_csv` function writes the dataset into the variable called `“data”`.

`head()` function returns the first 5 rows of the dataset.

NOTE: the Jupyter notebook and the csv files should be in the same directory to write the code as above, otherwise you have to copy the full path where your csv file is stored:

`data = pd.read_csv(r"Full path\Filename.csv")`
#%%
import pandas as pd
import numpy as np
#%%
data = pd.read_csv("train.csv")
print(data.head())
#%% md
After the csv file is read, x and y values should be stored as separate variables in order to be able to work with them. This can be done in many ways.

In this example we will convert the data from a pandas Series to a Python list. This is done to to simplify working with the data in certain algorithms or libraries.
```

```
29
30 The type of X and Y variables is pandas series.
   Pandas series is more complex data structure than
   both numpy arrays and python lists. That is why it
   requires a lot more time to do operations on pandas
    series. For this reason, we convert X and Y from
   pandas series to python lists:
31 #%%
32 X1 = data['x']; Y1 = data['y']
33 X = X1.tolist()
34 Y = Y1.tolist()
35 #%% md
36 For visualization, matplotlib library is used.
37 #%%
38 import matplotlib.pyplot as plt
39 plt.scatter(X, Y)
40 plt.grid()
41 plt.xlabel("x values")
42 plt.ylabel("y values")
43 plt.show()
44 #%% md
45 Every dot in the graph represents one sample from
   the dataset. As is seen from the output, Linear
   Regression Algorithm is quite appropriate for this
   dataset.
46 #%% md
47 ## Gradient Descent
48
49 #%% md
50 #### the hypothesis:
51 ![hx.png](attachment:hx.png)
52
53 #### Cost function
54 ![cost.png](attachment:cost.png)
55 It’s used to calculate how well the line fits the
   data
56
57 #### Gradient Descent
58 We start the algorithm with random initial values (
   usually zero) of w and b, thus the cost function
   will return some high value. So we somehow have to
```

```
58 optimize w and b to reduce the return of the cost
   function. In each iteration gradient descent
   algorithm updates the values of w and b and the
   line fits the data better.
59
60 ![w.png](attachment:w.png)
61
62 ![b.png](attachment:b.png)
63 #%% md
64 Inputs:
65 - X: A list or array of input feature values (the
   independent variable).
66 - Y: A list or array of target values (the
   dependent variable).
67 - w: The initial weight (slope) of the regression
   line.
68 - b: The initial bias (intercept) of the regression
    line.
69 - alpha: The learning rate, which controls how big
   the update steps are during gradient descent.
70 #%%
71 #alpha - learning rate
72 def gradient_descent(X, Y, w, b, alpha):
73
74     dl_dw = 0.0 #gradient with respect to the
   weight w
75     dl_db = 0.0 #gradient with respect to the
   weight b
76     N = len(X) #the total number of data points
77
78     #loop Through Data Points to Calculate
   Gradients
79     for i in range(N):
80         dl_dw += -1*X[i] * (Y[i] - (w*X[i] + b))
81         dl_db += -1*(Y[i] - (w*X[i] + b))
82
83     #update w and b
84     #adding a normalization factor 1/float(N) to
   prevent w and b from becoming too large
85     w = w - (1/float(N))* dl_dw * alpha
86     b = b -  (1/float(N))* dl_db * alpha
```

```
 87
 88     return w, b
 89 #%% md
 90 Inputs:
 91 - X: A list or array of input feature values (
    independent variables).
 92 - Y: A list or array of target values (dependent
    variables).
 93 - w: The weight (slope) of the linear regression
    model.
 94 - b: The bias (intercept) of the linear regression
     model.
 95 #%%
 96 #cost function: Mean Squared Error (MSE)
 97 def cost_function (X, Y, w, b):
 98
 99     N = len(X)
100     total_error = 0.0 #total squared error over
    all data points (will be updated)
101
102     #loop Through Data Points
103     for i in range(N):
104         total_error += (Y[i] - (w*X[i] - b))**2
105
106     return total_error / (2*float(N))
107 #%% md
108 #### Function to plot the linear regressor
109
110 #%%
111 def plot_LR (X,Y,w,b):
112             plt.grid()
113             plt.xlabel("x values")
114             plt.ylabel("y values")
115
116             xx=np.linspace(-3,100)
117             h=w*xx+b
118             plt.plot(xx,h,color="red")
119             plt.scatter(X, Y,s=5)
120             plt.pause(0.05)
121             plt.show()
122 #%%
```

```
def train(X, Y, w, b, alpha, n_iter):

    for i in range(n_iter):
        w, b = gradient_descent(X, Y, w, b, alpha)

        if i % 50 == 0:
            print ("iteration:", i, "cost: ",
cost_function(X, Y, w, b))
            plot_LR(X, Y, w, b)

    return w, b
#%%
def predict(x, w, b):
    return x*w + b
#%% md
## Train the model
#%%
 # w, b = train(X, Y, w, b, alpha, num_iter)
 w, b = train(X, Y, 0.0, 0.0, 0.00001, 200)

#%% md
## Test the model
#%% md
Now we can use our trained model to predict new
values using LR:
#%%
x_new = 50.0 #input feature

#call the predict() function to compute the
predicted value y_new based on the input x_new
y_new = predict(x_new, w, b)

print("theta1= ", w)
print("theta0= ",b)
print("the new predicted value",y_new)
#%% md
## sklearn linear regression

#%% md
Use the `LinearRegression` class in the `scikit-
learn` library to fit a linear model to the data:
```

```
#%%
from sklearn.linear_model import LinearRegression
import pandas as pd
#%% md
Converting the data into DataFrames is necessary because `scikit-learn` models often work well with structured data like DataFrames.
#%%
l_reg = LinearRegression() #create an instance of the model

#convert data into dataframes
X2=pd.DataFrame(X1)
Y2=pd.DataFrame(Y1)

#the model is trained (or "fit") to the data
l_reg.fit(X2, Y2)
#%% md
Extract information about the model's performance and the parameters it learned during training:
- R² score: tells you how well the model fits the data. It ranges from 0 to 1.
- - 1 means the model perfectly explains all the variance in the target variable.
- - 0 means the model does not explain any of the variance in the target variable.
- Slope: tells you how much the target variable changes for a unit change in the corresponding feature.
- Intercept (bias): is the point where the regression line crosses the y-axis when all feature values are 0.
#%%
# Coefficient of determination
r_squared = l_reg.score(X2, Y2)
print("Coefficient of determination (R²):", r_squared)

# Slope
slope = l_reg.coef_
print("Slope (Coefficients):", slope)
```

```
189
190 # Intercept
191 intercept = l_reg.intercept_
192 print("Intercept:", intercept)
193 #%% md
194 Now we can read the test data from the CSV file,
    then use the trained LR model (l_reg) to make
    predictions based on the test data.
195 #%%
196 testdata = pd.read_csv("test.csv")
197 x_test = pd.DataFrame(testdata['x']); y_test = pd.
    DataFrame(testdata['y'])
198 y_pred = l_reg.predict(x_test)
199
200 print(y_pred)
201 #%% md
202 ### Calculating R-squared (R2) score.
203 calculating the R² score for the model's
    predictions on the test data
204 #%%
205 r2_score = l_reg.score(x_test,y_test)
206 print(r2_score*100,'%') #multiply the R² score by
    100 to convert it into a percentage
207 #%% md
208 # Activities
209 ## 1. Implement the normal equations method and
    compare the result with the previous results
210 #%%
211 x1 = data['x'].values
212 Y1 = data['y'].values
213
214 def lrne(X, Y):
215     X = np.c_[np.ones(X.shape[0]), X]
216     theta = np.linalg.inv(X.T @ X) @ X.T @ Y
217     return theta
218
219 theta = lrne(x1, Y1)
220 print(theta)
221 plot_LR(x1, Y1, theta[1], theta[0])
222 #%%
223 def predict_NE(x, theta):
```

```
224     return x * theta[1] + theta[0]
225 
226 x_new = 50.0
227 y_new = predict_NE(x_new, theta)
228 print("the new predicted value",y_new)
229 plot_LR(x1, Y1, theta[1], theta[0])
230 
231 #%% md
232 ## 2. Update the gradient descent method for
    multiple input features
233 #%%
234 def plot_LR_New(X, Y, w, b):
235     plt.figure()
236     plt.xlabel("x values")
237     plt.ylabel("y values")
238     xx = np.linspace(X[:, 0].min(), X[:, 0].max
    (), 100)
239     h = w[0] * xx + b
240     plt.plot(xx, h, color="red")
241     plt.scatter(X[:, 0], Y, s=5)
242     plt.show()
243 #%%
244 data = pd.read_csv("train.csv")
245 print(data.columns)
246 #%%
247 
248 X1 = data[['x','y']]
249 Y1 = data['y']
250 X = X1.values
251 Y = Y1.values
252 
253 def gradient_descent_mul(X, Y, w, b, alpha):
254     dl_dw = np.zeros(w.shape)
255     dl_db = 0.0
256     N = len(X)
257     for i in range(N):
258         dl_dw += -1*X[i] * (Y[i] - (np.dot(w,X[i
    ]) + b))
259         dl_db += -1*(Y[i] - (np.dot(w,X[i]) + b))
260 
261     w = w - (1/float(N))* dl_dw * alpha
```

```
    b = b -  (1/float(N))* dl_db * alpha

    return w, b

def cost_function_multi(X, Y, w, b):
    N = len(X)
    total_error = 0.0
    for i in range(N):
        total_error += (Y[i] - (np.dot(w,X[i]) - b
))**2

    return total_error / (2*float(N))

def train_multi(X, Y, w, b, alpha, n_iter):
    for i in range(n_iter):
        w, b = gradient_descent_mul(X, Y, w, b,
alpha)
        if i % 50 == 0:
            print ("iteration:", i, "cost: ",
cost_function_multi(X, Y, w, b))
            plot_LR_New(X, Y, w, b)
    return w, b

w = np.zeros(X.shape[1])
b = 0.0
alpha = 0.00001
num_iter = 200
w, b = train_multi(X, Y, w, b, alpha, num_iter)
print('weight: ',w)
print('bias: ',b)


#%%

```