|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decorative | | | | |
|  |  |  | |  |
| Machine Learning  Assignment 1 | | |
| Laman Khudadatzada | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decorative | | | | |
|  |  |  | |  | |
|  | Introduction  The report presents the application and analysis of linear regression to predict car prices using a dataset from turbo.az. It outlines the steps taken to install, visualize, and implement linear regression both from scratch and using a library. | |  | |

|  |
| --- |
| **Importing libraries:**  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D  from sklearn.model\_selection import KFold  from sklearn.linear\_model import LinearRegression  **Data Loading and Visualization:**  Using the pandas library in Python the data is loaded and three relevant columns – Yurush, model Buraxilish ili and Qiymet are extracted from the dataset.  data = pd.read\_csv("turboaz.csv")[["Yurush", "Buraxilish ili", "Qiymet"]]  data.head() # displaying few rows   * With the given path three columns are read. * Few lines from the dataset are displayed.   def yurush\_string\_to\_int(data):  data['Yurush'] = data['Yurush'].str.replace(' km', '').str.replace(' ', '').astype(int)  yurush\_string\_to\_int(data) #calling the convert function  data.head() #displaying   * This function converts the string values of “Yurush” into integers. Simply, it removes the word “km” from the data. * Calling the function   def qiymet\_dollar\_to\_azn(data):  converted\_qiymet = [float(price.replace(' $', '').strip()) \* 1.7 if '$' in price else float(price.replace(' AZN', '').strip()) for price in data['Qiymet']]  data['Qiymet'] = converted\_qiymet  qiymet\_dollar\_to\_azn(data)  data.head()   * This function converts the string values of “Qiymet” into integers. Simply, it removes the word “AZN” or “$” sign from the data. * Calling the function and displaying the data |
|  |
|  |

**Visualization:**

Visualizations are created using matplotlib, including scatter plots and a 3D plot.

Scatter plots using the pandas dataframe:

data.plot.scatter(x = "Yurush", y = "Qiymet")

The output:

A graph with blue dots

Description automatically generated

data.plot.scatter(x = "Buraxilish ili", y = "Qiymet")

The output:

A graph with blue dots

Description automatically generated

def three\_d\_plot(data):

x1 = data['Yurush'].to\_numpy()

x2 = data['Buraxilish ili'].to\_numpy()

y = data['Qiymet'].to\_numpy()

fig = plt.figure(figsize = (10, 8))

ax = plt.axes(projection ='3d')

ax.scatter(x1, x2, y, c = 'red')

plt.show()

* Extracting “Yurush”, “Buraxilish ili” and “Qiymet” columns.
* Creating a new figure with size of 10x8 inches.
* Then creating a scatter plot in 3D space.

three\_d\_plot(data)

* Calling the function

The output:

A graph with red dots

Description automatically generated

**Linear Regression:**

Here, we apply Z-score normalization in order to ensure convergence during gradient descent. Then using the gradient descent algorithm, the cost function is calculated.

x = data.drop(columns=['Qiymet']).values # the values of all columns in data except the 'Qiymet'

y = data['Qiymet'].values # the values of 'Qiymet' column

m, n = x.shape

theta = np.zeros((n+1))

# for matrix multipication

x\_o = np.hstack((np.ones((m, 1)), x))

* Extracting all data except “Qiymet” column to the x variable.
* “Qiymet” column to the y variable
* Unpacking the shape of the array x into two variables m (rows) and n (columns).
* Creating a column vector of ones using np.ones((m, 1)), then it horizontally stacks using np.hstack().
* (m, n+1) is used for matrix multiplication during the training.

def multiply\_matrix(x, theta):

return np.dot(x, theta)

* The function is used for matrix multiplication between a matrix (x) and a vector (theta)

Cost Function:

def calc\_cost(x, y, theta):

m = len(y)

hyp\_values = multiply\_matrix(x, theta)

squared\_errors = (hyp\_values - y) \*\* 2

total\_cost = np.sum(squared\_errors) / (2 \* m)

return total\_cost

* m represents the number of samples.
* Calculating the predicted values (hyp\_values) for the given matrix and vector by calling the multiply\_matrix function.
* Calculating the squared errors
* Then, by summing up all the squared errors and dividing by twice the number of samples, it computes the total cost.

calc\_cost(x\_o, y, theta)

* Calling the function

The output:

A number on a black background

Description automatically generated

Normalization:

def normalize(x):

epsilon = 1e-10 # prevent division by zero

sigma = x.std(0)

mean = x.mean(0)

sigma[sigma == 0] += epsilon

normalized\_x = (x - mean) / sigma

return normalized\_x, mean, sigma

* Normalization is a common technique used to standardize features by subtracting the mean and dividing by the standard deviation.
* Checking for any feature with zero standard deviation and adds a very small value epsilon.

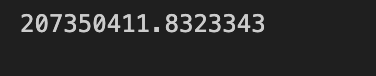
x\_normalized, mean, sigma = normalize(x)

x\_no = np.hstack((np.ones((m, 1)), x\_normalized))

calc\_cost(x\_no, y, theta)

* It returns the normalized feature matrix x\_normalized, as well as the mean and standard deviation (stored in sigma) used for normalization.
* The next line creates a column vector of ones with the same number of rows then horizontally stacks it with x\_normalized using np.hstack().
* calculates the cost function using the normalized feature matrix.

The output:



Gradient Descent:

learning\_rate = 0.001

iterations = 10000

def gradient\_desc(x, y, theta):

n = len(y)

cost\_values =[]

for i in range(iterations):

hypo\_cost = multiply\_matrix(x, theta) - y

theta = theta - (learning\_rate / n) \* np.matmul(x.transpose(), hypo\_cost)

cost\_values.append(calc\_cost(x, y, theta))

return theta, cost\_values

alt, cost\_values = gradient\_desc(x\_no, y, theta)

* Initializing learning rate and the number of iterations,
* gradient descent to optimize the parameters of the linear regression model.
* Calculating the number of samples in the dataset.
* Initializing an empty list (cost\_values) to store the cost values.
* Finding the difference between the predicted values and the target values.
* It returns the optimized parameter vector and the list of cost function values for each iteration.
* Then, calling the gradient descent function.

def cost\_function\_graph(vals):

plt.plot(vals)

plt.show()

cost\_function\_graph(cost\_values)

* Displaying the cost function graph

The output:

A line graph with numbers

Description automatically generated

Line of predictions for Qiymet and Buraxilish ili:

def year\_prediction(x, y, z):

plt.scatter(x[:, 2], y, color = 'green')

plt.xlabel('Buraxilish ili')

plt.ylabel('Qiymet')

#min and max values of 'Buraxilish ili'

min\_index = x[:,2].argmin()

max\_index = x[:,2].argmax()

x\_val = [x[:,2][min\_index], x[:,2][max\_index]]

y\_val = [np.matmul(x[min\_index], z), np.matmul(x[max\_index], z)]

# subtracting a constant value from y values

offset = np.mean(y) - np.mean(y\_val)

y\_values\_adjusted = [val + offset for val in y\_val]

plt.plot(x\_val, y\_values\_adjusted, color = "red")

year\_prediction(x\_no, y, alt)

The output:

A graph with a red line

Description automatically generated

Line of predictions for Qiymet and Yurush:

def yurush\_prediction(x, y):

plt.scatter(x[:, 1], y, color = 'orange')

plt.xlabel('Yurush')

plt.ylabel('Qiymet')

plt.title('Predicted Line')

#min and max values of 'Yurush'

min\_index = x[:,1].argmin()

max\_index = x[:,1].argmax()

x\_value = [x[:,1][ min\_index], x[:,1][max\_index]]

y\_value = 10000

plt.plot(x\_value, [y\_value, y\_value], color = "purple")

* Setting labels for x and y axes
* Finding the index of the maximum and minimum values in the third column of the matrix
* The offset is used to get the predicted prices.
* Taking the predicted “Qiymet”s by adding the offset (already calculated)

The output:

A graph with orange dots

Description automatically generated

Line of predictions for Qiymet, Buraxilis ili and Yurush:

def scatter\_predictions\_3D(x, y, z):

# Creating a figure

fig = plt.figure(figsize = (12, 8))

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(x[:, 1], x[:, 2], y, color="blue")

ax.scatter(x[:, 1], x[:, 2], np.matmul(x, theta), color='red')

ax.set\_xlabel('Buraxilish ili')

ax.set\_ylabel('Yurush')

ax.set\_zlabel('Qiymet')

plt.show()

scatter\_predictions\_3D(x\_no, y, alt)

* It creates a new figure with a specified size (12x8).
* It adds a 3D subplot.
* Plotting the predicted data points using models on all axes.
* Setting labels for the x, y, and z axes
* Display the 3D scatter plot.

The output:

A graph with red and blue dots

Description automatically generated

**Two New Cars**

car\_1 = np.array([240000, 2000])

car\_2 = np.array([415558, 1996])

normal\_car\_1 = (car\_1 - mean) / sigma

normal\_car\_2 = (car\_2 - mean) / sigma

normal\_car\_1 = np.concatenate(([1], normal\_car\_1))

normal\_car\_2 = np.concatenate(([1], normal\_car\_2))

* First, NumPy arrays represent the features of two new cars with their mileage and model years.
* It standardizes new cars using Z-score normalization.

price\_car\_1 = np.matmul(normal\_car\_1, alt)

price\_car\_2 = np.matmul(normal\_car\_2, alt)

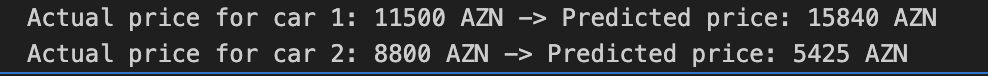
* The predicted price for the first new car is calculated by multiplying its normalized feature vector with the parameter vector (alt).
* The same process happened to the car 2 as well.

print(" Actual price for car 1: 11500 AZN -> " + "Predicted price: " + str(int(price\_car\_1)) + " AZN")

print(" Actual price for car 2: 8800 AZN -> "+ "Predicted price: " + str(int(price\_car\_2)) + " AZN")

* These lines print the actual and predicted prices for the cars with their actual and predicted prices.

The output:



**Linear Regression Using Library**

Linear regression is performed using a library (scikit-learn) to fit the model to the data. At the end, the results are compared with those obtained from the application.

features = data[['Buraxilish ili', 'Yurush']].to\_numpy()

target = data['Qiymet'].to\_numpy()

* Extracting inputs and labels

kf = KFold(n\_splits=10)

regressor = LinearRegression()

* Wth KFold, we create 10 folds.
* Then, created an instance of the LinearRegression class.

for train\_indices, test\_indices in kf.split(features):

train\_x, test\_x = features[train\_indices], features[test\_indices]

train\_y, test\_y = target[train\_indices], target[test\_indices]

regressor.fit(train\_x, train\_y)

r\_squared = regressor.score(test\_x, test\_y)

print(r\_squared)

* First, we obtain indices for training and test data.
* Splitting the features and target variables into training and testing sets.
* Then, I calculated the r-squared values on the testing data.
* Lastly, printing the r-squared values/

car\_1\_price = regressor.predict([[2000, 240000]])

car\_2\_price = regressor.predict([[1996, 415558]])

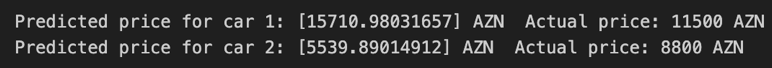
* This part of the code predicts the price of cars by calling the predict function of regressor.

print("Predicted price for car 1: " + str(car\_1\_price) + " AZN" + " Actual price: 11500 AZN")

print("Predicted price for car 2: " + str(car\_2\_price) + " AZN" + " Actual price: 8800 AZN")

* Showing predicted and actual prices for cars.

The output:



To summrize, in the report, I showed the comparison of custom application and library. In additional, I added visualizations of graph