**ADA University, Machine Learning**

**Programming Assignment – 3**

**Linear Regression**

**Emin Aliyev**

**Loading data**

1. # 1. Loading data
2. df **=** pd.read\_csv("turboaz.csv", usecols**=**["Yurush", "Buraxilish ili", "Qiymet"])
3. df["Yurush"] **=** df["Yurush"].map(
4. **lambda** yurush: int(utils.remove\_unit(yurush, "km")))
5. df["Qiymet"] **=** df["Qiymet"].map(**lambda** qiymet: utils.convert\_price(qiymet))

I have read the data with the ***pandas*** library by extracting 3 columns (Yurush, Buraxilish ili, Qiymet) in the 2nd line of the code snippet. The next 2 lines of the code remove ***“km”*** unit from the **df[“Yurush”]** series and cast the values into integer. The last line of the code removes units from the **df[“Qiymet”]** series, converts the prices with dollars, and cast the final values into integer. The functions used in the snippet nest in the ***utils*** module of the code.

**Visualization**

1. # 2. Visualization
2. **def** plot(df):
3. # Qiymet (Y) vs Yurush (X1)
4. df.plot(kind**=**"scatter", x**=**"Yurush", y**=**"Qiymet")
5. plt.show()
7. # Qiymet (Y) vs Buraxilish ili (X2)
8. df.plot(kind**=**"scatter", x**=**"Buraxilish ili", y**=**"Qiymet")
9. plt.show()
11. # 3D plot of all three values (Y, X1, X2)
12. ax **=** plt.axes(projection**=**"3d")
13. ax.set\_xlabel("Yurush")
14. ax.set\_ylabel("Buraxilish ili")
15. ax.set\_zlabel("Qiymet")
16. ax.scatter3D(df["Yurush"], df["Buraxilish ili"], df["Qiymet"])
17. plt.show()

The code for this part of the assignment nests in the ***visualization*** module. I have used the ***matplotlib*** library to visualize the data. The ***plot*** function takes a data frame as a parameter, plots, and shows. Every comment within the ***plot*** function is followed by a corresponding visualization task, such as, having scatter plots for the first two relationships. The last relationship is described by a 3D scatter plot.

**Implementation of Linear Regression from scratch**

**Calculate cost function.**

1. # Calculate cost function
2. **def** cost\_function(x, y):
3. cost **=** np.sum(((x **-** y) **\*\*** 2) **/** (2 **\*** len(y)))
5. **return** cost

The ***cost\_function*** takes **x** (predicted) and **y** arraysas parameters. I have used ***numpy*** library to calculate the cost function (MSE). The function nests in the ***utils*** module of the code.

**Normalize data using Z score normalization.**

1. # Normalize data using Z score normalization
2. x1 **=** df["Yurush"]
3. x2 **=** df["Buraxilish ili"]
4. y **=** df["Qiymet"]
6. x1 **=** np.array((x1 **-** x1.mean()) **/** x1.std())
7. x2 **=** np.array((x2 **-** x2.mean()) **/** x2.std())
8. x **=** np.c\_[np.ones(x1.shape[0]), x1, x2]
10. y **=** np.array((y **-** y.mean()) **/** y.std())

Again, I have used ***numpy*** library to normalize each column of the data by Z score normalization.

**Implement gradient descent algorithm to minimize cost function.**

**Function:**

1. # Implement gradient descent algorithm to minimize cost function
2. **def** gradient\_descent(x, y, w, alpha**=**0.001, iterations**=**10000):
3. n **=** len(y)
4. weights **=** [w]
5. costs **=** []
7. **for** \_ **in** range(iterations):
8. predict **=** x.dot(w)
9. gradient **=** x.T.dot(predict **-** y)
10. w **=** w **-** (1 **/** n) **\*** alpha **\*** gradient
11. cost **=** cost\_function(predict, y)
13. weights.append(w)
14. costs.append(cost)
16. **return** weights, costs

**Usage:**

1. w **=** np.random.rand(3)
2. weights, costs **=** utils.gradient\_descent(x, y, w)

The ***gradient\_descent*** function takes 5 parameters (3 required and 2 optional). **X** is an array of independent variables, **y** is the dependent variable, and **w** are the initial weights for the algorithm. I have initialized the function with random weights, default learning rate and iterations. The function saves all the weights and costs in different lists. For each iteration, it finds the prediction by calculating the dot product of **x** and **w** and uses that prediction to find the gradient by calculating the dot product of the transpose matrix of **x** and error **(predict – y)**. Next, it updates **w** according to the formula and calculates the cost function. Finally, it appends the **w** and **cost** to their corresponding lists and return them. The function nests in the ***utils*** module of the code.

**Plot graph of cost function.**

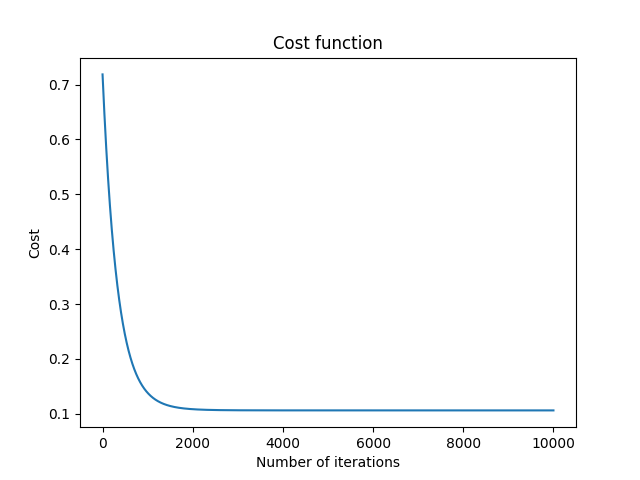
**Function:**

1. # Plot graph of cost function
2. **def** plot\_cost\_function(costs):
3. plt.title("Cost function")
4. plt.xlabel("Number of iterations")
5. plt.ylabel("Cost")
6. plt.plot(costs)
7. plt.show()

**Usage:**

1. # Plot graph of cost function
2. visualization.plot\_cost\_function(costs)

**Result:**

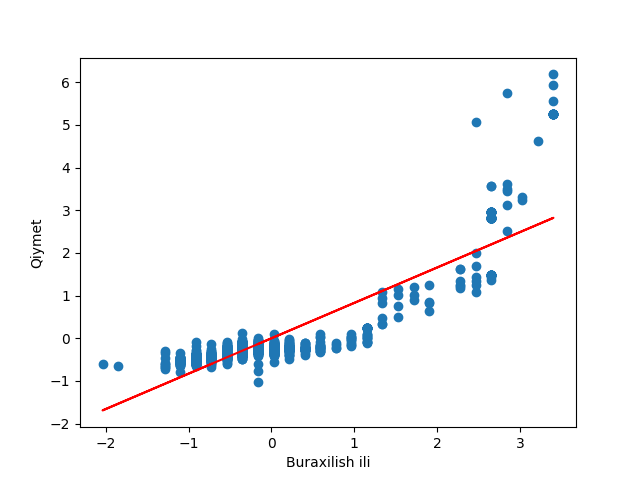
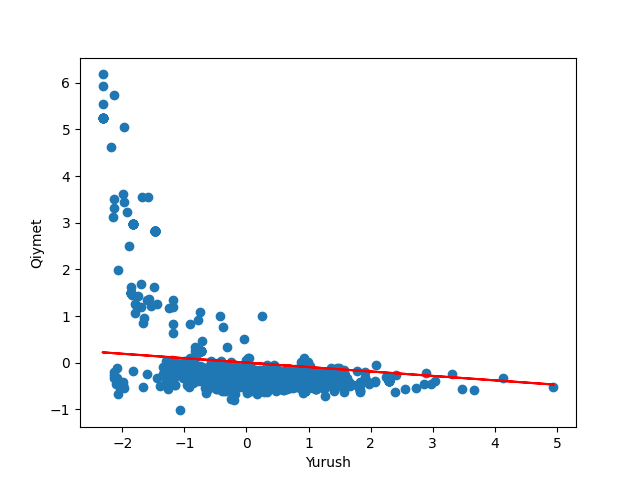
****

As the number of iterations grows, the gradient descent algorithm minimizes the cost function. Therefore, we see that the cost function starts to converge after ≈ 2000 iterations.

**Plot points of Y (Qiymet) vs X1 (Yurush), Y (Qiymet) vs X2 (Buraxilish ili) and draw a line of predictions.**

1. # Plot points of Y (Qiymet) vs X1 (Yurush) and draw a line of predictions.
2. predicts **=** weights[**-**1][0] **+** x1.dot(weights[**-**1][1])
3. plt.scatter(x1, y)
4. plt.plot(x1, predicts, color**=**"red")
5. plt.xlabel("Yurush")
6. plt.ylabel("Qiymet")
7. plt.show()
9. # Plot points of Y (Qiymet) vs X2 (Buraxilish ili) and draw a line of predictions.
10. predicts **=** weights[**-**1][0] **+** x2.dot(weights[**-**1][2])
11. plt.scatter(x2, y)
12. plt.plot(x2, predicts, color**=**"red")
13. plt.xlabel("Buraxilish ili")
14. plt.ylabel("Qiymet")
15. plt.show()
17. # Plot 3D graph of points of Y (Qiymet), X1, X2, and predicted Y (Qiymet).
18. predicts **=** x.dot(weights[**-**1])
19. ax **=** plt.axes(projection**=**"3d")
20. ax.set\_xlabel("Yurush")
21. ax.set\_ylabel("Buraxilish ili")
22. ax.set\_zlabel("Qiymet")
23. ax.scatter3D(x1, x2, y)
24. ax.scatter3D(x1, x2, predicts, color**=**"red")
25. plt.show()

In this code snippet, I have calculated the predictions based on each **X** feature. I plotted normalized **X1** vs **Y**, and **X2** vs **Y** relationships. I have used **polyfit** function to find the best fit for the scatter plot of features and predictions. Finally, I have used **m** and **b** values to plot the lines.

****

**Plot 3D graph of points of Y (Qiymet), X1, X2, and predicted Y (Qiymet).**

1. # Plot 3D graph of points of Y (Qiymet), X1, X2, and predicted Y (Qiymet).
2. predicts **=** x.dot(weights[**-**1])
3. ax **=** plt.axes(projection**=**"3d")
4. ax.set\_xlabel("Yurush")
5. ax.set\_ylabel("Buraxilish ili")
6. ax.set\_zlabel("Qiymet")
7. ax.scatter3D(x1, x2, y)
8. ax.scatter3D(x1, x2, predicts, color**=**"red")
9. plt.show()

In this code snippet, I have calculated the predictions by taking dot product of **x** and **weights**. Next, I have plotted the 3D graph of the normalized data. Finally, I have plotted the next 3D graph of the predicted data with red color.

**Testing**

1. # Testing
2. test\_data **=** np.array([[240000, 2000], [415558, 1996]])
4. # Normalize test data
5. test\_data\_x1 **=** np.array(
6. (test\_data[:, 0] **-** df["Yurush"].mean()) **/** df["Yurush"].std())
7. test\_data\_x2 **=** np.array(
8. (test\_data[:, 1] **-** df["Buraxilish ili"].mean()) **/** df["Buraxilish ili"].std())
9. test\_data\_x **=** np.c\_[np.ones(test\_data\_x1.shape[0]), test\_data\_x1, test\_data\_x2]
11. results **=** []
13. # Denormalize and print the results
14. **for** price **in** np.nditer(test\_data\_x.dot(weights[**-**1])):
15. results.append(price **\*** df["Qiymet"].std() **+** df["Qiymet"].mean())
17. print(results)
18. # Example output: [15823.421869751804, 5450.465380251384]

Firstly, I have initialized **test\_data** with a ***numpy*** array. I have normalized each column of the data and combined them in a single variable called **test\_data\_x**. Next, I have calculated the predicted price by taking dot product of **test\_data** and **weights**. Finally, I have denormalized the prices with **df[“Qiymet”]** and appended the results into a list.

|  |  |  |
| --- | --- | --- |
| **Data** | **Actual price** | **Predicted price** |
| **Car 1** | 11500 | ≈ 15823 |
| **Car 2** | 8800 | ≈ 5450 |

**Linear Regression using library**

1. # 4. Linear Regression using library
2. reg **=** linear\_model.LinearRegression()
3. reg.fit(df[["Yurush", "Buraxilish ili"]], df["Qiymet"])
4. pred **=** reg.predict(test\_data)
6. print(pred)
7. # Output: [15820.54127243  5453.69414862]

I have used ***scikit-learn*** library to model this problem. I have used the same features as input and output to the model. Finally, I have used **predict** method of the regression to predict the prices. We can see only a slight difference between the predictions of the library model and my model.

**Extra tasks**

**Solve linear regression by normal equation**

1. # Solve linear regression by normal equation
2. W **=** np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y)
3. print(W)

In this code snippet, I have found the value of **W** by simply applying the normal equation formula. We can use that value to test our model later.