Project 2

SiCheng Yi

The purpose of this project is to perform multiple linear regression analysis on the data. There are two types, one is to analyze the data normally, and the other is to analyze the data after Standardization preprocessing.

There are 9 groups of data, divided into three types: Continuous, Multivariate Categorical, String. These 9 types of data are:

1. MPG (Continuous)
2. Cylinder s(Multivariate Categorical)
3. Displacement (Continuous)
4. Horsepower (Continuous)
5. Weight (Continuous)
6. Acceleration (Continuous)
7. Model Year (Multivariate Categorical)
8. Origin (Multivariate Categorical)
9. Car (String)

The data itself is arranged in order, each group of data occupies one row, and when dealing with Standardization, I directly use the code on the website:

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler().fit(X)

standardizedX = scaler.transform(X)

featureBoxPlot(standardizedX, names)
```

And another code attempt, another standardization I tried:

First, we first take the three columns of data to be used as the training data set:

```
trainData = cReader[['mpg','displacement','acceleration']]

trainData.insert(0,'ones',1)

cols = trainData.shape[1]

X = trainData.iloc[:,0:cols-1]

Y = trainData.iloc[:,cols-1:cols]

X = np.mat(X.values)

Y = np.mat(Y.values)

for i in range(1,3):

        X[:,i] = (X[:,i]-min(X[:,i])) / (max(X[:,i])-min(X[:,I]))

Y[:,0] = (Y[:,0]-min(Y[:,0])) / (max(Y[:,0])-min(Y[:,0]))
```

formula:

Regression formula:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{21} & \cdots & X_{k1} \\ 1 & X_{22} & \cdots & X_{k2} \\ \vdots & \vdots & & \vdots \\ 1 & X_{2n} & \cdots & X_{kn} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\underset{n\times 1}{\boldsymbol{Y}} \qquad \underset{n\times k}{\boldsymbol{X}} \qquad \underset{k\times 1}{\boldsymbol{\beta}} \qquad \underset{n\times 1}{\boldsymbol{u}}$$

Cost function formula:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(\theta^T X - y^{(i)})^2$$

Gradient descent formula:

$$\theta_i = \theta_i - \alpha\frac{\partial}{\partial\theta}J(\theta) = \theta_i - \alpha(h_\theta(x) - y)x^{(i)}$$

First draw a picture to check the situation.

Then,

In the first code, I took out the'mpg' and'weight' columns separately as the X and Y variables to construct the regression analysis:

data = cars[['mpg','acceleration']]#Select the'mpg','weight' column in the table

data.insert(0,'Ones', 1) #Insert a column of all 1s between columns 1-2 of data

```
# set X (training data) and y (target variable)

cols = data.shape[1] #Calculate the number of columns of data, where cols is 3

X = data.iloc[:,0:cols-1]

y = data.iloc[:,cols-1:cols]
```

Then use the least square method to solve the regression and calculate the loss function.

In the second code, I first use the least squares method to calculate the regression coefficients and give the code representation of the cost function of gradient descent:

```
theta_n = (X.T*X).I*X.T*Y

print(theta_n)
```

Then define the cost function. I defined a new module linearRegrassion.py, in which I wrote linear regression related functions and observed the formula. h(x) is the predicted value and y is the actual value. The difference between them is used to reflect the difference The reliability of the fitted curve. The smaller the value, the closer the curve fitted by the corresponding coefficient θ is to the actual situation. What we have to do next is to use the gradient descent algorithm to take a series of θ values to approximate the optimal solution. Finally Get the regression curve.

At the end of the gradient descent iteration, the slope is getting closer and closer to 0, and the cost function is getting closer and closer to the optimal solution. At this time, we get the coefficient θ with the highest degree of fit, and finally perform regression.

The result of the least squares estimation: [[17.74518558] [-0.63629322] [-

5.95952521]]

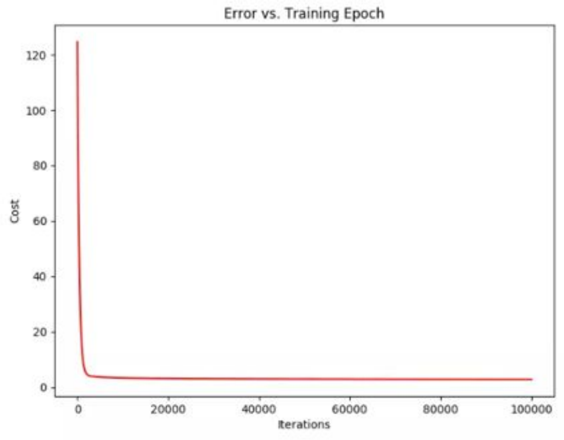
The final θ: [[15.47017446 2.99065096 -3.31870705]]

The value of the cost function: [124.67279846 124.37076615 124.06949161 ...,

2.77449658 2.77449481 2.77449304]

In the first code: lr_cost = 3.12288717494

In the second code the result of gradient descent is as follows:


Error vs. Training Epoch
Cost
Iterations