MACHINE LEARNING LAB
EXPERIMENT – 3

ANKITH MOTHA
21BAI1284

Linear Regression

Dataset: Abalone Dataset

Input Variables: Length, Height, Viscera weight, Shell weight, Sex

Target Variable: Rings

Regression is a type of Machine learning which helps in finding the relationship between independent and dependent variables. Linear Regression is a type of Regression which assumes a linear relationship between the dependent and independent variables.

The objective of Linear Regression is to find the most efficient coefficients which describe the linear relationship the best,

$$Y = Bo + \mathrm{B}_1\mathrm{X}_1 + \mathrm{B}_2\mathrm{X}_2 + \ldots\ldots + \mathrm{B}_\mathrm{D}\mathrm{X}_\mathrm{D}$$

Linear Regression can be solved through different methods such as Gradient Descent, Genetic Algorithm, Closed-Form Solution, etc. However, gradient descent is an iterative algorithm, whereas closed-form solution gives us the direct answer.

Closed-Form Solution

An optimization problem is closed-form solvable if it is differentiable with respect to the weights w and the derivative can be solved, but that is only true in the case of a minimum/maximum optimization problem.

The optimization problem, aims to reduce the loss function used to calculate the deviation of the predicted value from the true value.

Given a set of N input vectors of D dimension and a set of output vectors of S dimension, we need to find a linear relation between the different input feature and the target.

$$\text{Input:}\quad \mathrm{X} = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{R}^D$$

$$\text{Target:}\quad \mathrm{Y} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_n\}, \mathrm{y}_i \in \mathbb{R}^\mathrm{S}$$

We use the linear relationship given below and find the optimum values of the coefficients, so as to get a linear model with least prediction error

$$Y = Bo + \mathrm{B_1X_1} + \mathrm{B_2X_2} + \ldots\ldots + \mathrm{B_DX_D}$$

Code:

Import all the required packages

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Load the dataset

```
df = pd.read_csv("/kaggle/input/abalone-dataset/abalone.csv")
df
```

Find the outliers in the dataset (Only numeric columns)

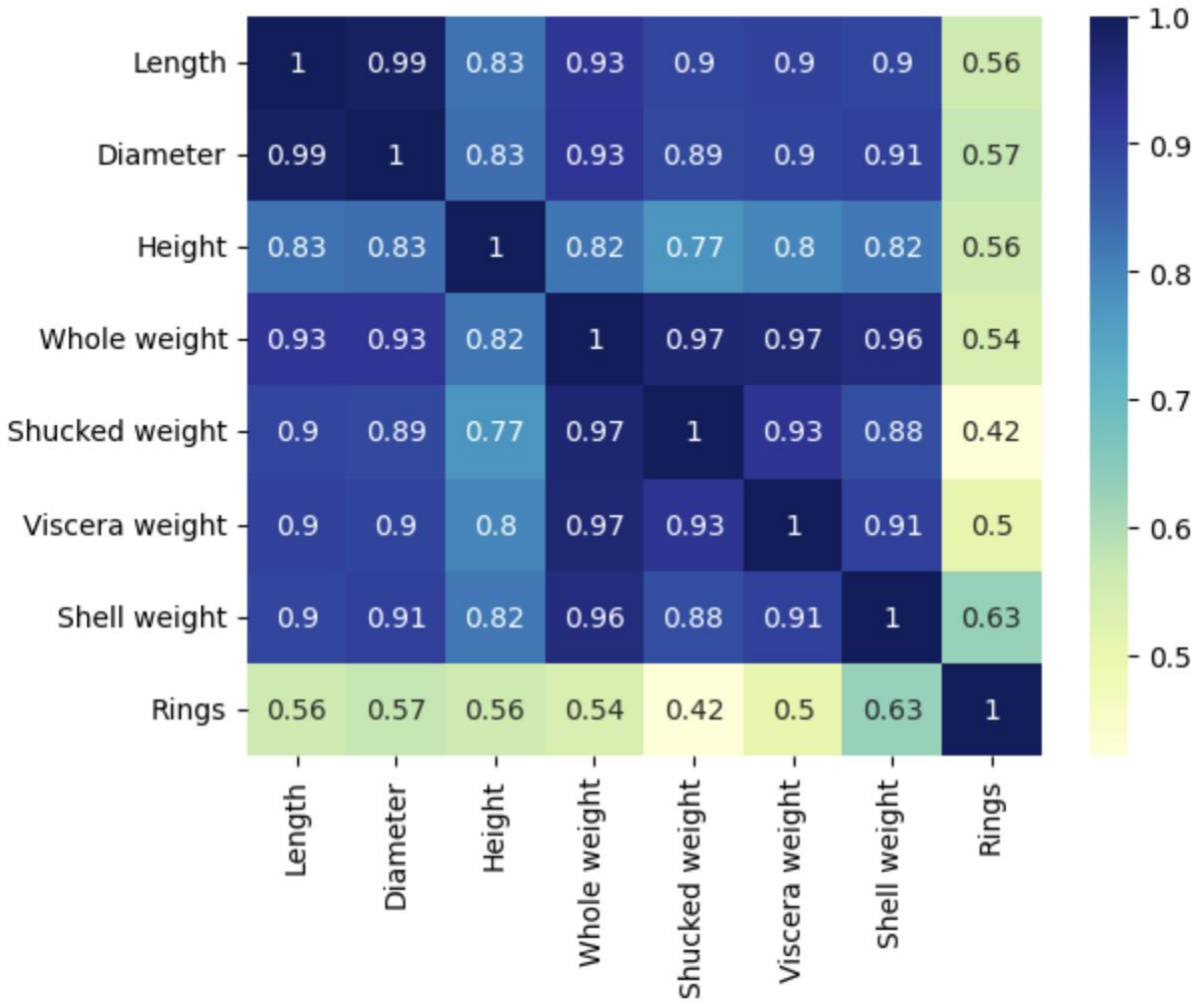
```
print("No. of Outliers in each column:")
for i in df.columns:
    if df[i].dtype!="object" and i!="Rings":
        q1 = df[i].quantile(0.25)
        q3 = df[i].quantile(0.75)
        iqr = q3-q1
        low = q1-1.5*iqr
        high=q3+1.5*iqr
        count = df[(df[i]>high) | (df[i]<low)][i].count()
        print(i,": ",count)
```

```
No. of Outliers in each column:
Length :  49
Diameter :  59
Height :  29
Whole weight :  30
Shucked weight :  48
Viscera weight :  26
Shell weight :  35
```

Display the correlation matrix and remove the extra columns with correlation more than 0.96, since these columns are highly related/similar and do not convey any meaningful features.

```
dataplot = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

```
/tmp/ipykernel_93/456078719.py:1: FutureWarning: The default value of numeric_only in DataFr
version, it will default to False. Select only valid columns or specify the value of numeric
  dataplot = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```



It can be seen above that, (whole weight, shucked weight and Viscera weight) are highly correlated and similarly, (diameter and Length) are also highly correlated, hence some columns can be forpped.

```
df.drop(['Diameter',"Whole weight","Shucked weight"],axis=1, inplace=True)
```

Next, we encode the columns with categorical data, since the ML model can't process text data. We use One-Hot encoding for this.

```
df = pd.get_dummies(df,"Sex")
cols = df.columns
df
```

	Length	Height	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.455	0.095	0.1010	0.1500	15	0	0	1
1	0.350	0.090	0.0485	0.0700	7	0	0	1
2	0.530	0.135	0.1415	0.2100	9	1	0	0
3	0.440	0.125	0.1140	0.1550	10	0	0	1
4	0.330	0.080	0.0395	0.0550	7	0	1	0
...	...	...	...	...	...	...	...	...
4172	0.565	0.165	0.2390	0.2490	11	1	0	0
4173	0.590	0.135	0.2145	0.2605	10	0	0	1
4174	0.600	0.205	0.2875	0.3080	9	0	0	1
4175	0.625	0.150	0.2610	0.2960	10	1	0	0
4176	0.710	0.195	0.3765	0.4950	12	0	0	1

We then perform Normalization to bring the values of all the features within the range of 0 to 1. This will bring uniformity in data and prevents a biased prediction. We use MinMaxScaler to get the values within 0 to 1.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df = scaler.fit_transform(df)

df = pd.DataFrame(df, columns = [cols])
df
```

	Length	Height	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.513514	0.084071	0.132324	0.147982	0.500000	0.0	0.0	1.0
1	0.371622	0.079646	0.063199	0.068261	0.214286	0.0	0.0	1.0
2	0.614865	0.119469	0.185648	0.207773	0.285714	1.0	0.0	0.0
3	0.493243	0.110619	0.149440	0.152965	0.321429	0.0	0.0	1.0
4	0.344595	0.070796	0.051350	0.053313	0.214286	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...
4172	0.662162	0.146018	0.314022	0.246637	0.357143	1.0	0.0	0.0
4173	0.695946	0.119469	0.281764	0.258097	0.321429	0.0	0.0	1.0
4174	0.709459	0.181416	0.377880	0.305431	0.285714	0.0	0.0	1.0
4175	0.743243	0.132743	0.342989	0.293473	0.321429	1.0	0.0	0.0
4176	0.858108	0.172566	0.495063	0.491779	0.392857	0.0	0.0	1.0

4177 rows × 8 columns

We finally split the data into the **Input Vectors and Target Vectors**

```
X = df.drop("Rings",axis=1)
y = df["Rings"]
X,y
```

Now, split the X and y vectors for training and testing.

```
from sklearn.model_selection import train_test_split
X = X.to_numpy()
y = y.to_numpy()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,shuffle=True)
```

Derivation of Closed Form Solution:

We need to find a mapping that maps the multi-dimensional input to multi-dimensional output,

f: D ->S, where D is the dimension of our input and S that of the output.

One way to think of this is to find the weight vector m and bias b for each dimension of the output. It's as if we're stacking a bunch of these functions over N data points:

$$f_i : y = m_i^T x + b_i$$ where i = 1, 2, …, S.

If we formally stack these mappings together, the bias b is simply the stacked biases bi and the weight matrix W can be formulated by stacking all the m vectors so,

$$W = [m_1, m_2, ..., m_S]^T \in R^{D \times S}$$

We rarely have just one sample in a dataset. We usually, as mentioned above, have a set of N data samples. We can add all these data samples into two matrixes, one for input and output.

$$X = [x_1, x_2, ..., x_N]^T$$

$$Y = [y_1, y_2, ..., y_N]^T$$

We want to pass all these through the function at the same time, which means we need to extend the function to have the form,

$$f : N \times D => N \times S$$

The function at this step is:

$$Y \approx XW$$

We just need to absorb the bias. We'll change the matrix X by adding a column of 1s to the end creating:

$$X = [x_1, x_2, ..., x_D, 1]$$

and the matrix *W* by adding the bias vector as the last row, so W becomes

$$W = [w_1, w_2, ..., w_D, b]^T$$

Our new dimensions are:

$$X \in R^{N \times D+1}$$

$$W \in R^{D+1 \times S}$$

Now we need to formulate the optimization. Since we want to get XW as close as possible to Y, the optimization is the minimization of the distance between f(X) = XW and Y or:

$$W = argmin_W \frac{1}{2}(XW - Y)^T(WX - Y)$$

This is essentially the least-squares loss function, which classically looks like:

$$W = argmin_W \frac{1}{2}(xw - y)^2$$

Now we find the derivative or gradient with respect to the weights w,

$$\nabla_W E_{LS}(W) = \nabla_W \frac{1}{2}(XW - Y)^T(WX - Y)$$
$$\nabla_W E_{LS}(W) = \nabla_W \frac{1}{2}(W^T X^T XW - 2W^T X^T Y - Y^T Y)$$
$$\nabla_W E_{LS}(W) = X^T XW - X^T Y$$

Since we need to find the values of W such that loss we least, we find the derivative and equate to 0

$$X^T XW - X^T Y = 0 => W* = (X^T X)^{-1} X^T Y$$

Hence to get the weights of the Linear regression line, we can use this closed form solution

Code:

We now implement this closed form solution programmatically.

```
m = X_train.shape[0]
X_train = np.append(X_train, np.ones((m,1)), axis=1)
y_train = y_train.reshape(m,1)
```

```
a = np.linalg.pinv(np.matmul(X_train.T,X_train))
print(a.shape)
b = np.matmul(X_train.T,y_train)
w = np.matmul(a,b)
w.shape
```

```
(8, 8)
(8, 1)
```

We have now found the weights which contain the coefficients and the bias (intercept)

```
w
```

```
array([[ 0.06772569],
       [ 0.43018065],
       [-0.44146103],
       [ 0.70431617],
       [ 0.05546464],
       [ 0.01420744],
       [ 0.0514753 ],
       [ 0.12114737]])
```

Now, we move on to testing our approach on the test data.

```
X_test = np.append(X_test, np.ones((X_test.shape[0],1)), axis=1)
```

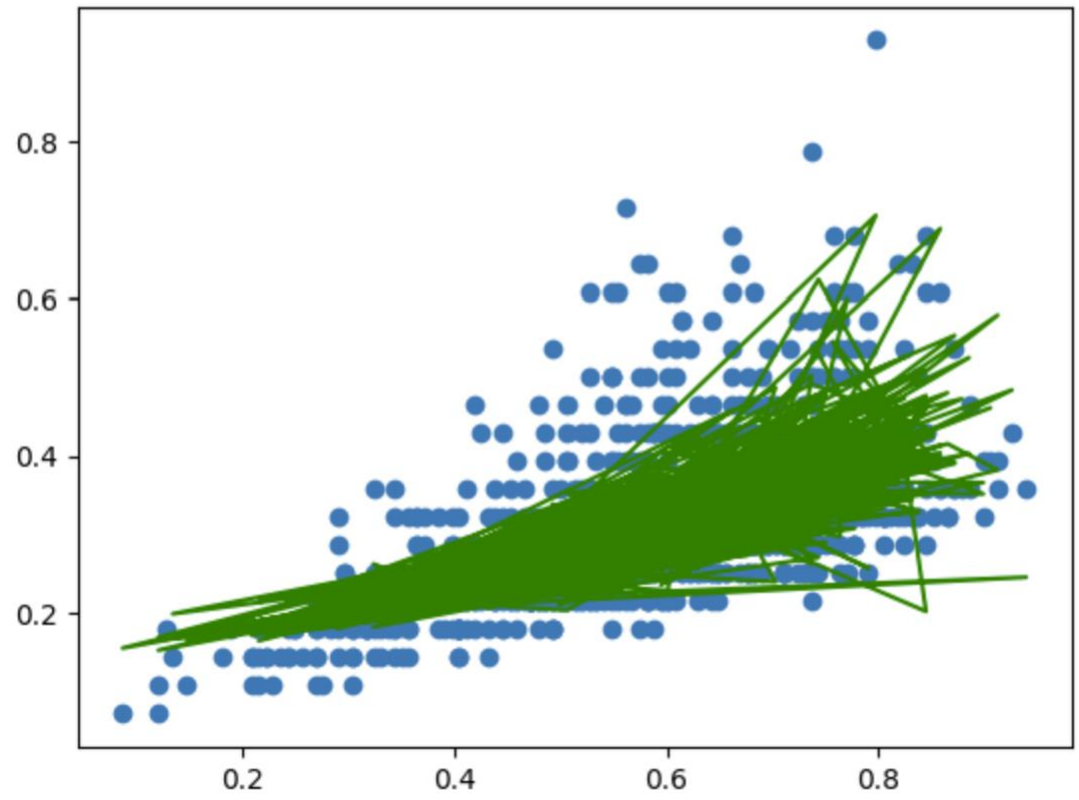
```
y_pred = np.matmul(X_test,w)
y_pred[:5,:]
```

```
array([[0.31998533],
       [0.32488842],
       [0.36433396],
       [0.27702579],
       [0.41188499]])
```

Given below is the plot of the regression line in 2D, however, this doesn't exist as a line since we are only plotting the projection of the line from a higher dimension to two dimensions:

```
plt.plot(X_test[:,0],y_pred, color="green")
plt.scatter(X_test[:,0],y_test)
```

```
<matplotlib.collections.PathCollection at 0x7ac1fc725540>
```



Finally, we must evaluate our model's performance, for which we use 3 metrics:

i) RMSE
ii) R2 (R squared)
iii) Adjusted R2

```
from sklearn.metrics import mean_squared_error, r2_score
print("RMSE:",mean_squared_error(y_test,y_pred,squared=False))

r2 = r2_score(y_test,y_pred)
n = len(X_test)
k = len(X_test[0])
print("R2:",r2)
print("Adjusted R2:",1 - ((1-r2)*(n-1)/(n-k-1)))
```

```
RMSE: 0.08188112019050066
R2: 0.44583840758569593
Adjusted R2: 0.44047771503513433
```

R2 score is a metric that tells the performance of your model.

In contrast, MSE depend on the context, whereas the R2 score is independent of context.

So, with help of R squared we have a baseline model to compare a model, which none of the other metrics provides. So basically R2 squared calculates how must regression line is better than a mean line.

Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.