Lecture 11

May 30, 2023

1 Ridge and Lasso Regression

1.1 Import the required Libraries

[1]:
```
import numpy as np
import pandas as pd
```

1.2 Import the dataset

[2]:
```
df=pd.read_csv('50_Startups.csv')
```

[3]:
```
df
```

[3]:
```
     R&D Spend  Administration  Marketing Spend       State     Profit
0    165349.20       136897.80        471784.10    New York  192261.83
1    162597.70       151377.59        443898.53  California  191792.06
2    153441.51       101145.55        407934.54     Florida  191050.39
3    144372.41       118671.85        383199.62    New York  182901.99
4    142107.34        91391.77        366168.42     Florida  166187.94
..         …             …                …           …         …
103  119943.24       156547.42        256512.92     Florida  132602.65
104  114523.61       122616.84        261776.23    New York  129917.04
105   78013.11       121597.55        264346.06  California  126992.93
106   94657.16       145077.58        282574.31    New York  125370.37
107   91749.16       114175.79        294919.57     Florida  124266.90

[108 rows x 5 columns]
```

[4]:
```
df.head()
```

[4]:
```
   R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
3  144372.41       118671.85        383199.62    New York  182901.99
4  142107.34        91391.77        366168.42     Florida  166187.94
```

[5]:
```
df.shape
```

[5]: (108, 5)

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        108 non-null    float64
 1   Administration   108 non-null    float64
 2   Marketing Spend  108 non-null    float64
 3   State            108 non-null    object
 4   Profit           108 non-null    float64
dtypes: float64(4), object(1)
memory usage: 4.3+ KB
```

1.3 Label Encoding

[7]:
```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
```

[8]:
```
le=LabelEncoder()
```

[9]:
```
df['State']=le.fit_transform(df['State'])
```

[10]:
```
df.head()
```

[10]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	2	192261.83
1	162597.70	151377.59	443898.53	0	191792.06
2	153441.51	101145.55	407934.54	1	191050.39
3	144372.41	118671.85	383199.62	2	182901.99
4	142107.34	91391.77	366168.42	1	166187.94

1.4 Split data into dependent and independent data

[11]:
```
x=df.drop(columns=['Profit'])
```

[12]:
```
x.head()
```

[12]:

	R&D Spend	Administration	Marketing Spend	State
0	165349.20	136897.80	471784.10	2
1	162597.70	151377.59	443898.53	0
2	153441.51	101145.55	407934.54	1
3	144372.41	118671.85	383199.62	2
4	142107.34	91391.77	366168.42	1

[13]:
```
y=df.Profit
```

[14]:
```
y.head()
```

[14]:
```
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
Name: Profit, dtype: float64
```

1.5 Normalization using MinMaxScaler

[15]:
```
from sklearn.preprocessing import MinMaxScaler
```

[16]:
```
scale=MinMaxScaler()
```

[17]:
```
scaled_x=pd.DataFrame(scale.fit_transform(x))
```

[18]:
```
scaled_x
```

[18]:

	0	1	2	3
0	1.000000	0.651744	1.000000	1.0
1	0.983359	0.761972	0.940893	0.0
2	0.927985	0.379579	0.864664	0.5
3	0.873136	0.512998	0.812235	1.0
4	0.859438	0.305328	0.776136	0.5
..	…	…	…	…
103	0.725394	0.801327	0.543708	0.5
104	0.692617	0.543030	0.554864	1.0
105	0.471808	0.535270	0.560312	0.0
106	0.572468	0.714013	0.598948	1.0
107	0.554881	0.478772	0.625116	0.5

[108 rows x 4 columns]

1.6 Split data into training and test data

[19]:
```
#Train and Test Split
from sklearn.model_selection import train_test_split
```

[20]:
```
x_train,x_test,y_train,y_test=train_test_split(scaled_x,y,test_size=0.
↪2,random_state=0)
```

[21]:
```
x_train.shape
```

[21]:
```
(86, 4)
```

[22]:
```
x_test.shape
```

[22]:
```
(22, 4)
```

1.7 Import Ridge and Lasso

[23]:
```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

[24]:
```
r=Ridge()
l=Lasso()
```

[25]:
```
r.fit(x_train,y_train)
```

[25]:
```
Ridge()
```

[26]:
```
l.fit(x_train,y_train)
```

[26]:
```
Lasso()
```

[27]:
```
pred1=r.predict(x_test)
pred2=l.predict(x_test)
```

[28]:
```
pred1
```

[28]:
```
array([ 54556.32416702, 130017.92166782,  84687.15947095, 173295.2223158 ,
       108917.94957822, 128735.89224253, 128736.35934265, 155951.19177423,
       117814.48562718,  52712.59507338, 102790.3781561 , 119096.2726001 ,
        54556.32416702, 124206.72612243,  88379.01243395, 126261.35613731,
       126261.35613731,  98802.1865801 ,  74278.88209886, 141546.67661999,
       145564.21281487, 150251.73759042])
```

[29]:
```
pred2
```

[29]:
```
array([ 48384.86814735, 134845.52354938,  76486.64641608, 181551.13594979,
       112961.07382208, 134236.64101991, 129218.98004997, 160017.16104325,
       116754.23112994,  46273.04713164, 102272.49339834, 115567.13437352,
        48384.86814735, 119116.48630482,  88593.22703248, 127104.80005829,
       127104.80005829,  90948.41312188,  58678.78647171, 146299.80323437,
       149413.8490298 , 152502.10158276])
```

1.8 Metrics to find model accuracy

[30]:
```
from sklearn import metrics
```

1.8.1 MSE (Mean Square Error)

[31]:
```
print(metrics.mean_squared_error(y_test,pred1))
print(metrics.mean_squared_error(y_test,pred2))
```

```
117186385.76630396
96005734.13154325
```

1.8.2 RMSE(Root Mean Square Error)

[32]:
```
print(np.sqrt(metrics.mean_squared_error(y_test,pred1)))
print(np.sqrt(metrics.mean_squared_error(y_test,pred2)))
```

```
10825.266082933202
9798.251585438253
```

1.8.3 R Squared

[33]:
```
print(metrics.r2_score(y_test,pred1))
print(metrics.r2_score(y_test,pred2))
```

```
0.9095565216441845
0.9259035724996549
```

As r2 Score is higher for lasso regression we will use Lasso Regression ML Model for Deployment.

2 Logistic regression

2.1 Import the required Libraries

[34]:
```
import numpy as np
import pandas as pd
```

2.2 Import the dataset

[35]:
```
df=pd.read_csv('Social_Network_Ads.csv')
```

[36]:
```
df
```

[36]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
..	…	…	…	…	…
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1

```
398  15755018    Male   36            33000          0
399  15594041  Female   49            36000          1

[400 rows x 5 columns]
```

[37]:
```
df.head()
```

[37]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

[38]:
```
df.shape
```

[38]: (400, 5)

[39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

2.3 Label Encoding

[40]:
```
from sklearn.preprocessing import LabelEncoder
```

[41]:
```
le=LabelEncoder()
```

[42]:
```
df['Gender']=le.fit_transform(df['Gender'])
```

[43]:
```
df.head()
```

[43]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	1	19	19000	0
1	15810944	1	35	20000	0
2	15668575	0	26	43000	0
3	15603246	0	27	57000	0

```
4  15804002       1   19            76000           0
```

2.4 Split data into dependent and independent data

[44]:
```python
x=df.drop(columns=['Purchased'])
```

[45]:
```python
x.head()
```

[45]:

	User ID	Gender	Age	EstimatedSalary
0	15624510	1	19	19000
1	15810944	1	35	20000
2	15668575	0	26	43000
3	15603246	0	27	57000
4	15804002	1	19	76000

[46]:
```python
y=df.Purchased
```

[47]:
```python
y.head()
```

[47]:
```
0    0
1    0
2    0
3    0
4    0
Name: Purchased, dtype: int64
```

2.5 Normalization using MinMaxScaler

[48]:
```python
from sklearn.preprocessing import MinMaxScaler
```

[49]:
```python
scale=MinMaxScaler()
```

[50]:
```python
scaled_x=pd.DataFrame(scale.fit_transform(x))
```

[51]:
```python
scaled_x
```

[51]:

	0	1	2	3
0	0.232636	1.0	0.023810	0.029630
1	0.982732	1.0	0.404762	0.037037
2	0.409926	0.0	0.190476	0.207407
3	0.147083	0.0	0.214286	0.311111
4	0.954801	1.0	0.023810	0.451852
..	…	…	…	…
395	0.503623	0.0	0.666667	0.192593
396	0.560787	1.0	0.785714	0.059259
397	0.352477	0.0	0.761905	0.037037
398	0.757720	1.0	0.428571	0.133333

```
399  0.110048  0.0  0.738095  0.155556

[400 rows x 4 columns]
```

2.6 Split data into training and test data

[52]:
```
#Train and Test Split
from sklearn.model_selection import train_test_split
```

[53]:
```
x_train,x_test,y_train,y_test=train_test_split(scaled_x,y,test_size=0.
↪2,random_state=0)
```

[54]:
```
x_train.shape
```

[54]:
```
(320, 4)
```

[55]:
```
x_test.shape
```

[55]:
```
(80, 4)
```

2.7 Import Ridge and Lasso

[56]:
```
from sklearn.linear_model import LogisticRegression
```

[57]:
```
logreg = LogisticRegression()
```

[58]:
```
logreg.fit(x_train, y_train)
```

[58]:
```
LogisticRegression()
```

[59]:
```
pred=logreg.predict(x_test)
```

[60]:
```
pred
```

[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

2.8 Metrics to find model accuracy

[65]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
```

2.8.1 Acccuracy score

[66]:
```
print(accuracy_score(y_test,pred))
```

```
0.9375
```

2.8.2 Confusion matrix

[68]:
```
print(confusion_matrix(y_test,pred))
```

```
[[58  0]
 [ 5 17]]
```

2.8.3 Classification report

[70]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        58
           1       1.00      0.77      0.87        22

    accuracy                           0.94        80
   macro avg       0.96      0.89      0.92        80
weighted avg       0.94      0.94      0.93        80
```