LINEAR REGRESSION CENTRAL TEST

March 18, 2024

[19]:
```
#importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression,Ridge
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error,r2_score
```

[20]:
```
data=pd.read_csv("C:\\Users\\SAMUEL K\\Desktop\\concrete_data.csv")
data
```

[20]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	\
0	540.0	0.0	0.0	162.0	2.5	
1	540.0	0.0	0.0	162.0	2.5	
2	332.5	142.5	0.0	228.0	0.0	
3	332.5	142.5	0.0	228.0	0.0	
4	198.6	132.4	0.0	192.0	0.0	
…	…	…	…	…	…	
1025	276.4	116.0	90.3	179.6	8.9	
1026	322.2	0.0	115.6	196.0	10.4	
1027	148.5	139.4	108.6	192.7	6.1	
1028	159.1	186.7	0.0	175.6	11.3	
1029	260.9	100.5	78.3	200.6	8.6	

	Coarse Aggregate	Fine Aggregate	Age	Strength
0	1040.0	676.0	28	79.99
1	1055.0	676.0	28	61.89
2	932.0	594.0	270	40.27
3	932.0	594.0	365	41.05
4	978.4	825.5	360	44.30
…	…	…	…	…
1025	870.1	768.3	28	44.28
1026	817.9	813.4	28	31.18
1027	892.4	780.0	28	23.70
1028	989.6	788.9	28	32.77
1029	864.5	761.5	28	32.40

[1030 rows x 9 columns]

[21]:
```
#finding the x indepedent variables
x=data.drop("Strength",axis=1)
x
```

[21]:
```
      Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0      540.0                 0.0      0.0  162.0               2.5
1      540.0                 0.0      0.0  162.0               2.5
2      332.5               142.5      0.0  228.0               0.0
3      332.5               142.5      0.0  228.0               0.0
4      198.6               132.4      0.0  192.0               0.0
…        …                   …        …    …                   …
1025   276.4               116.0     90.3  179.6               8.9
1026   322.2                 0.0    115.6  196.0              10.4
1027   148.5               139.4    108.6  192.7               6.1
1028   159.1               186.7      0.0  175.6              11.3
1029   260.9               100.5     78.3  200.6               8.6

      Coarse Aggregate  Fine Aggregate  Age
0               1040.0           676.0   28
1               1055.0           676.0   28
2                932.0           594.0  270
3                932.0           594.0  365
4                978.4           825.5  360
…                  …               …    …
1025             870.1           768.3   28
1026             817.9           813.4   28
1027             892.4           780.0   28
1028             989.6           788.9   28
1029             864.5           761.5   28

[1030 rows x 8 columns]
```

[22]:
```
#finding the dependent variables
y=data["Strength"]
y.head()
```

[22]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

[23]:
```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,␣
 ↪random_state=40)
```

[24]:
```
#model building on ordinary data
model=LinearRegression()
model.fit(x_train,y_train)
```

[24]:
```
LinearRegression()
```

[25]:
```
#making a prediction for y on ordinary model
y_pred=model.predict(x_test)
print("y_pred", y_pred,("."))
```

```
y_pred [15.12967904 27.74830172 32.80022365 26.61742385 36.35489873 63.41389277
 16.92950452 20.34676613 52.05033513 48.81739398 44.23017182 24.4260516
 54.46028348 21.0804814  28.23683893 43.03089136 43.25379249 21.36354556
 43.19408915 70.94466516 60.31755089 20.78284795 49.7859168  21.60457959
 21.0995023  32.01943787 33.37144904 29.8187493  27.33135605 41.10670261
 11.77548975 22.5842409  36.15654446 28.04712087 67.89270783 52.24155567
 30.76146329 25.21833099 43.73000091 50.17754954 46.95667196 50.49089895
 39.664836   36.15416099 25.8720597  57.8486609  14.96995011 23.64615107
 27.86079551 58.07421726 33.98169587 34.58749214 60.24965616 50.10855883
 24.13843416 68.90176786 40.65812556 21.57764658 38.59447135 29.29202629
 26.75985081 52.31665118 62.88059512 24.71043304 35.52009231 39.04168128
 49.31182522 20.93312264 47.25262541 27.81618572 38.18998174 32.04532184
 48.37546005 27.6592957  19.64827541 36.49158255 30.6572809  57.26921503
 33.33402027 53.19404644 47.59830137 24.09439512 40.20495727 28.49063381
 26.74100121 32.64531792 28.29151495 38.13648986 30.90854662 58.76934234
 23.28981494 37.61912489 39.56864167 18.29405528 32.45628056 24.80905314
 37.60538277 25.68690928 37.18164902 49.76751304 32.65173711 30.89257462
 34.33514685 55.42011592 19.77761257 21.69319057 29.77442016 18.76872281
 30.99316626 28.6393427  35.63011288 31.58933667 38.87881229 34.67139156
 34.29218629 22.67524996 31.28712692 60.69672367 69.71743526 47.46499745
 51.99139349 33.32758964 33.44381896 32.5130526  37.25284982 25.97265134
 32.91398716 52.37131209 36.19135887 33.25452611 50.90089238 33.14295872
 33.30594305 46.65443727 31.47041563 37.46193102 51.99139349 32.64171114
 35.27203182 52.22557171 50.07850243 56.10292472 24.9736571  52.361421
 42.76246391 43.05348687 34.79163619 53.11767475 16.87206621 45.32131167
 25.39728043 37.53744783 28.91965259 40.65982452 51.92869598 50.88571751
 18.42464291 42.41376063 47.60326238 46.37621531 36.0649394  31.94573534
 31.58384888 25.22163576 48.83140484 39.99054457 47.28396613 34.03704214
 36.43295733 39.16878262 21.43674156 31.47903555 28.83335213 39.86894028
 40.54322941 16.96915892 32.22398155 60.30415461 37.58078947 25.61032366
 53.80040914 16.35763835 71.59499495 34.27552866 37.93450208 24.81845518
 21.43797208 27.55045798 25.47069244 31.0027103  48.41892978 25.95680749
 28.59872539 16.10070577 17.50856699 33.46416016 35.68939387 38.16347364
 18.61096323 40.99376114 25.57264264 27.06897846 52.46930893 28.41053636
 36.02105035 36.97146956] .
```

[26]:
```
#calculating for ordinary accuracy
accuracy=r2_score(y_test,y_pred)
accuracy
```

[26]: 0.5179120685190279

[27]:
```
#finding the mean score
mean=mean_squared_error(y_test,y_pred)
mean
```

[27]: 126.14668267505864

[28]:
```
#finding the r2 score
r2=r2_score(y_test,y_pred)
r2
```

[28]: 0.5179120685190279

[29]:
```
#optimizing the model
scaler=StandardScaler()
x_train1=scaler.fit_transform(x_train)
x_test1=scaler.transform(x_test)
```

[30]:
```
# using Ridge
ridge_model=Ridge(alpha=0.7)
ridge_model.fit(x_train1,y_train)
```

[30]: Ridge(alpha=0.7)

[31]:
```
# predicting y for optimized model
y_pred_ridge= ridge_model.predict(x_test1)
# finding the accuracy
accuracy_opt=r2_score(y_test, y_pred_ridge)
accuracy_opt
```

[31]: 0.5188075401063381

[32]:
```
if accuracy_opt> accuracy:
    print("The optimized model better than ordinary model")

elif accuracy_opt<accuracy:
    print("The ordinary model is greater than optimized model")
else:
    print("Both models have the same accuracy")
```

```
The optimized model better than ordinary model
```

[ ]: