KAGORO FINAL CENTRAL TEST logistic regression

March 20, 2024

[69]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,␣
 ↪accuracy_score
from sklearn.model_selection import GridSearchCV
```

[70]:
```
data = pd.read_csv("C:\\Users\\HP 840\\Desktop\\diabetes 2.csv")
data
```

[70]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …              …              …      …    …
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4
768            0      123             77              0        1  36.3

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        …   …      …
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
768                     0.252   55        1

[769 rows x 9 columns]
```

[71]:

```
x = data.drop(['Outcome'], axis = 1)
x
```

[71]:

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …            …             …        …    …
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4
768            0      123             77              0        1  36.3

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        …    …
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23
768                     0.252   55

[769 rows x 8 columns]
```

[72]:

```
y = data['Outcome']
y
```

[72]:

```
0      1
1      0
2      1
3      0
4      1
      ..
764    0
765    0
766    1
767    0
768    1
Name: Outcome, Length: 769, dtype: int64
```

[73]:
```
x_train,x_test,y_train, y_test = train_test_split(x,y, test_size = 0.25,␣
 ↪random_state = 42)
```

[74]:
```
model = LogisticRegression(max_iter=1000).fit(x_train, y_train)
model
```

[74]:
```
LogisticRegression(max_iter=1000)
```

[75]:
```
y_pred = model.predict(x_test)
y_pred
```

[75]:
```
array([1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], dtype=int64)
```

[76]:
```
mae = mean_absolute_error(y_test,y_pred )
mae
```

[76]:
```
0.26424870466321243
```

[77]:
```
mse = mean_squared_error(y_test,y_pred )
mse
```

[77]:
```
0.26424870466321243
```

[78]:
```
r2_sc = r2_score(y_test,y_pred )
r2_sc
```

[78]:
```
-0.1432055749128922
```

[79]:
```
accuracy = accuracy_score(y_test,y_pred )
accuracy
```

[79]:
```
0.7357512953367875
```

[80]:
```
model = LogisticRegression()
model
```

[80]:
```
LogisticRegression()
```

[81]:
```
param_grid = {
     "penalty":[None, 'l2'],
     "C":[1.0, 1.5],
     "solver":['newton-cg', 'newton-cholesky']
 }
```

[82]:
```
grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)
```

[82]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1.0, 1.5], 'penalty': [None, 'l2'],
                         'solver': ['newton-cg', 'newton-cholesky']})
```

[83]:
```
best_params = grid_search.best_params_
print("Best Parameters :", best_params)
```

```
Best Parameters : {'C': 1.0, 'penalty': None, 'solver': 'newton-cg'}
```

[84]:
```
best_model = LogisticRegression(**best_params)
best_model
```

[84]:
```
LogisticRegression(penalty=None, solver='newton-cg')
```

[85]:
```
best_model.fit(x_train, y_train)
```

[85]:
```
LogisticRegression(penalty=None, solver='newton-cg')
```

[86]:
```
best_model
```

[86]:
```
LogisticRegression(penalty=None, solver='newton-cg')
```

[87]:
```
y_predi = best_model.predict(x_test)
y_predi
```

[87]:
```
array([1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], dtype=int64)
```

[88]:
```
maei = mean_absolute_error(y_test, y_predi)
msei = mean_squared_error(y_test, y_predi)
r2_sci = r2_score(y_test, y_predi)
accuracyi = accuracy_score(y_test, y_predi)
```

[89]:

```
print(f"Mean Absolute Error = ", {maei})
print(f"Mean Squared Error = ", {msei})
print(f"R2 Score = ", {r2_sci})
print("Best Parameters :", best_params)
print(f"Accuracyi = ", {accuracyi})
```

```
Mean Absolute Error =  {0.26424870466321243}
Mean Squared Error =  {0.26424870466321243}
R2 Score =  {-0.1432055749128922}
Best Parameters : {'C': 1.0, 'penalty': None, 'solver': 'newton-cg'}
Accuracyi =  {0.7357512953367875}
```