diabetes-prediction

October 13, 2024

0.1 Importing the libraries

[2]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score, precision_score, r2_score
```

0.1.1 Data Collection and Analysis

[3]:

```
# importing the data

df = pd.read_csv('diabetes.csv')

df.head()
```

[3]:

```
   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

   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
```

[4]:

```
# number of rows and columns in the data
df.shape
```

[4]:

```
(768, 9)
```

[5]:

```
# stastical summary
df.describe()
```

```
[5]:        Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
     count   768.000000  768.000000     768.000000     768.000000  768.000000
     mean      3.845052  120.894531      69.105469      20.536458   79.799479
     std       3.369578   31.972618      19.355807      15.952218  115.244002
     min       0.000000    0.000000       0.000000       0.000000    0.000000
     25%       1.000000   99.000000      62.000000       0.000000    0.000000
     50%       3.000000  117.000000      72.000000      23.000000   30.500000
     75%       6.000000  140.250000      80.000000      32.000000  127.250000
     max      17.000000  199.000000     122.000000      99.000000  846.000000

                   BMI  DiabetesPedigreeFunction         Age     Outcome
     count  768.000000                768.000000  768.000000  768.000000
     mean    31.992578                  0.471876   33.240885    0.348958
     std      7.884160                  0.331329   11.760232    0.476951
     min      0.000000                  0.078000   21.000000    0.000000
     25%     27.300000                  0.243750   24.000000    0.000000
     50%     32.000000                  0.372500   29.000000    0.000000
     75%     36.600000                  0.626250   41.000000    1.000000
     max     67.100000                  2.420000   81.000000    1.000000
```

```
[6]: # value count of the output feature
     df['Outcome'].value_counts()
```

```
[6]: Outcome
     0    500
     1    268
     Name: count, dtype: int64
```

- 0 —-> Non-diabetic
- 1 —-> Diabetic

```
[7]: df.groupby('Outcome').mean()
```

```
[7]:          Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
     Outcome
     0           3.298000  109.980000      68.184000      19.664000   68.792000
     1           4.865672  141.257463      70.824627      22.164179  100.335821

                    BMI  DiabetesPedigreeFunction        Age
     Outcome
     0        30.304200                  0.429734  31.190000
     1        35.142537                  0.550500  37.067164
```

```
[35]: # splitting the data into dependent and independent features
      x = df.drop(columns= 'Outcome', axis=1)
      y = df['Outcome']
```

```
[9]: x
```

```
[9]:      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
     ..            …       …            …            …      …   …
     763           10      101             76             48      180  32.9
     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

          DiabetesPedigreeFunction  Age
     0                       0.627   50
     1                       0.351   31
     2                       0.672   32
     3                       0.167   21
     4                       2.288   33
     ..                        …  …
     763                     0.171   63
     764                     0.340   27
     765                     0.245   30
     766                     0.349   47
     767                     0.315   23

     [768 rows x 8 columns]
```

```
[10]: y
```

```
[10]: 0      1
      1      0
      2      1
      3      0
      4      1
            ..
      763    0
      764    0
      765    0
      766    1
      767    0
      Name: Outcome, Length: 768, dtype: int64
```

0.2 Standard the data into the same level

[11]:
```
scaler = StandardScaler()

standatdized_data = scaler.fit_transform(x)
standatdized_data
```

[11]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, …,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, …, -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, …, -1.10325546,
         0.60439732, -0.10558415],
       …,
       [ 0.3429808 ,  0.00330087,  0.14964075, …, -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, …, -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, …, -0.20212881,
        -0.47378505, -0.87137393]])
```

[12]:
```
x = standatdized_data
y
```

[12]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

[13]:
```
## SPlit the data into train and test

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.
 ↪2,stratify=y, random_state=19)
```

[14]:
```
x_train.shape
```

[14]:
```
(614, 8)
```

[15]:
```
x_test.shape
```

```
[15]: (154, 8)
```

```
[16]: y_train.shape, y_test.shape
```

```
[16]: ((614,), (154,))
```

0.3 Model Training

```
[17]: clf = svm.SVC(kernel='linear')
```

```
[18]: # fit the training data to the

      clf.fit(x_train, y_train)
```

```
[18]: SVC(kernel='linear')
```

0.4 Model Evavulation

```
[19]: ## Accuracy score on the training , precision, recall, r2_score

      x_train_prediction = clf.predict(x_train)
```

```
[20]: training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

```
[21]: print('Accuracy score : ', training_data_accuracy)
```

```
Accuracy score :  0.7703583061889251
```

```
[22]: # accuracy on the test data

      x_test_prediction = clf.predict(x_test)
      test_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

```
[23]: print('Accuracy score : ', test_data_accuracy)
```

```
Accuracy score :  0.7727272727272727
```

0.4.1 Predicting system

```
[24]: # add all the fearure data as input
      # input_data = (4, 100, 92, 0, 0, 37.6, 0.191, 30)

      input_data = (5, 166, 72, 19, 175, 25.8, 0.587, 51)

      # change the sample/input data to np.asarray
      input_data_nparray = np.asarray(input_data)
```

```python
# reshape the array we are predicting
input_data_reshape = input_data_nparray.reshape(1, -1)

# now we need to standardise the data as we standardise the training data
std_data = scaler.transform(input_data_reshape)
print(std_data)




# prediction
prediction = clf.predict(std_data)
print(prediction)


if prediction[0] == 0:
  print("The person is non-diabetic")
else:
  print("The person has diabeties")
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person has diabeties
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

[24]:

```python

```

0.4.2 Saving the model

[25]:

```python
import pickle
```

[27]:

```python
filename = 'diabetes_model.sav'

pickle.dump(clf, open(filename, 'wb'))
```

[28]:

```python
## loading the model

loaded_model = pickle.load(open('diabetes_model.sav', 'rb'))
```

[29]:

```python
input_data = (5, 166, 72, 19, 175, 25.8, 0.587, 51)

# change the sample/input data to np.asarray
input_data_nparray = np.asarray(input_data)
```

```python
# reshape the array we are predicting
input_data_reshape = input_data_nparray.reshape(1, -1)

prediction = loaded_model.predict(input_data_reshape)
print(prediction)
```

```
[1]
```

[31]:

```python
# prediction
prediction = loaded_model.predict(std_data)
print(prediction)


if prediction[0] == 0:
  print("The person is non-diabetic")
else:
  print("The person has diabeties")
```

```
[1]
The person has diabeties
```

[36]:

```
```

```
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
```

[ ]:

```
```