

diabetes-prediction

June 14, 2023

Importing the Dependencies

```
[26]: import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

```
[27]: # loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('/content/diabetes.csv')
```

```
[28]: # printing the first 5 rows of the dataset
diabetes_dataset.head()
```

```
[28]: 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
```

```
[29]: # number of rows and Columns in this dataset
diabetes_dataset.shape
```

```
[29]: (768, 9)
```

```
[30]: # getting the statistical measures of the data
diabetes_dataset.describe()
```

```
[30]:
```

	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

```
[31]: diabetes_dataset['Outcome'].value_counts()
```

```
[31]: 0    500
      1    268
      Name: Outcome, dtype: int64
```

0 -> Non-Diabetic

1 -> Diabetic

```
[32]: diabetes_dataset.groupby('Outcome').mean()
```

```
[32]:
```

	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

```
[33]: # separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']
```

```
[34]: print(X)
```

	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]

```
[35]: print(Y)
```

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

Data Standardization

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

```
[37]: scaler.fit(X)
```

```
[37]: StandardScaler()
```

```
[38]: standardized_data = scaler.transform(X)
print(standardized_data)
```

```
[[ 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]]
```

```
[39]: X = standardized_data
      Y = diabetes_dataset['Outcome']
```

```
[40]: print(X)
      print(Y)
```

```
[[ 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]]
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
```

Train Test Split

```
[41]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2,
↳stratify=Y, random_state=2)
```

```
[42]: print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the Model

```
[43]: classifier = svm.SVC(kernel='linear')
```

```
[44]: #training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

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

Model Evaluation

Accuracy Score

```
[45]: # accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[46]: print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data : 0.7866449511400652
```

```
[47]: # accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
[48]: print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data : 0.7727272727272727
```

Making a Predictive System

```
[49]: input_data = (4,110,92,0,0,37.6,0.191,30)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
```

```

input_data_resaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_resaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
    print('The person is not diabetic')
else:
    print('The person is diabetic')

```

```

[[ 0.04601433 -0.34096773  1.18359575 -1.28821221 -0.69289057  0.71168975
 -0.84827977 -0.27575966]]

```

```
[0]
```

The person is not diabetic

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
warnings.warn(

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

```

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_resaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_resaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
    print('The person is not diabetic')
else:
    print('The person is diabetic')

```

```

[[ 0.3429808  1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
  0.34768723  1.51108316]]

```

```
[1]
```

The person is diabetic

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does

```
not have valid feature names, but StandardScaler was fitted with feature names
warnings.warn(
```