Diabetes Prediction

April 16, 2025

1 Diabetes Prediction System

[1]:
```
# importing the dependencies
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score

# Ignoring warnings
import warnings
warnings.filterwarnings('ignore')
```

Data Collection and analysis

PIMA Diabetes Dataset

[2]:
```
# Loading the diabetes dataset to a pandas dataframe
diabetes_dataset = pd.read_csv('dataset/diabetes.csv')
```

[3]:
```
# printing the first 5 rows
diabetes_dataset.head()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[4]:
```
# No of rows and cols of dataset
diabetes_dataset.shape
```

[4]:
```
(768, 9)
```

[5]:
```
# Getting statistical measures of the df
diabetes_dataset.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]:
```
# Checking for null values
diabetes_dataset.isna().sum()
```

[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[7]:
```
# total number of labels
diabetes_dataset['Outcome'].value_counts()
```

[7]:
```
Outcome
0    500
1    268
```

```
Name: count, dtype: int64
```

Here 1 represents the patient is having diabetes 0 represents the patient is not having diabetes

[8]:
```
diabetes_dataset.groupby('Outcome').mean()
```

[8]:
```
         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
```

seeing this we can say that the diabetic people have more glucose value , this difference is very important for us as this will be used by machine learning model

[9]:
```
# Seprating the data and label
X = diabetes_dataset.drop(columns='Outcome',axis = 1)
y = diabetes_dataset['Outcome']
```

[10]:
```
print(X)
print(y)
```

```
     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]
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
```

Standardizing the data

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

[12]:
```
standardizer.fit(X)
standardized_data = standardizer.transform(X)
```

[13]:
```
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]]
```

Now we can see that all the values are now between a fixed range

[14]:
```
X = standardized_data
y = diabetes_dataset['Outcome']
```

[15]:
```
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
```

Splitting the data

[16]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify =y,random_state=2)
```

[17]:
```
print(X.shape,X_train.shape,X_test.shape)
```

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

Training the model

[18]:
```
classifier = svm.SVC(kernel='linear') # SVC represent support vector classifier'linear sayes we are going to use linear model'
```

[19]:
```
# training the support vector machine classifier
classifier.fit(X_train,y_train)
```

[19]:
```
SVC(kernel='linear')
```

Evaluating the model Accuracy Score

[20]:
```
# accuracy score for training data
X_train_prediction = classifier.predict(X_train)
X_train_accuracy = accuracy_score(X_train_prediction,y_train)
print('Accuracy score of training data is : ',X_train_accuracy)
```

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

[21]:
```
# accuracy score for training data
X_test_prediction = classifier.predict(X_test)
X_test_accuracy = accuracy_score(X_test_prediction,y_test)
print('Accuracy score of testing data is : ',X_test_accuracy)
```

```
Accuracy score of testing data is :  0.7727272727272727
```

Making a predictive system

[22]:
```
input_data = (4,110,92,0,0,37.6,0.191,30)

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

# Reshaping the data
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# Standardizing the data
std_data = standardizer.transform(input_data_reshaped)
print(std_data)

# Prediction model
prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
    print("The patient is not a diabetic pateint")
else:
   print("The patient is a diabetic pateint")
```

```
[[ 0.04601433 -0.34096773  1.18359575 -1.28821221 -0.69289057  0.71168975
  -0.84827977 -0.27575966]]
[0]
The patient is not a diabetic pateint
```