diabetes-revise-project2

November 24, 2024

1 diabetes prediction using svm

[42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

[4]:
```
df=pd.read_csv("diabetes.csv")
df
```

[4]:
```
     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  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
..                        …  …      …
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
```

```
766                       0.349   47         1
767                       0.315   23         0

[768 rows x 9 columns]
```

[15]:
```python
df.columns
```

[15]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

[16]:
```python
df['Pregnancies'].min()
```

[16]: 0

[17]:
```python
df['Pregnancies'].max()
```

[17]: 17

[18]:
```python
df['Glucose'].min()
```

[18]: 0

[19]:
```python
df['Glucose'].max()
```

[19]: 199

[20]:
```python
df['BloodPressure'].min()
```

[20]: 0

[21]:
```python
df['BloodPressure'].max()
```

[21]: 122

[22]:
```python
df['SkinThickness'].min()
```

[22]: 0

[25]:
```python
df['SkinThickness'].max()
```

[25]: 99

[23]:
```python
df['Insulin'].min()
```

[23]: 0

[24]:
```python
df['Insulin'].max()
```

[24]: 846

[26]:
```python
df['BMI'].min()
```

[26]: 0.0

[27]:
```python
df['BMI'].max()
```

[27]: 67.1

[28]:
```python
df['DiabetesPedigreeFunction'].min()
```

[28]: 0.078

[29]:
```python
df['DiabetesPedigreeFunction'].max()
```

[29]: 2.42

[30]:
```python
df['Age'].min()
```

[30]: 21

[31]:
```python
df['Age'].max()
```

[31]: 81

[32]:
```python
df['Outcome'].value_counts()
```

[32]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

1.1 column analysis:

1. pregnancies: minimum value 0 and maximum value 17
2. Glucose: minimum value 0 and maximum value 199
3. Bloodpressure: minimum value 0 and maximum value 122
4. Skin Thickness: minimum value 0 and maximum value 99
5. Insulin: minimum value 0 and maximum value 846
6. BMI: minimum value 0 and maximum value 67.1
7. DiabetesPedigreeFunction: minimum value 0.078 and maximum value 2.42
8. Age: minimum value 21 and maximum value 81 9.Outcome: 500 for 'O' and 268 for '1' : 0 for non diabetes and 1 for diabetes patient.

1.2 Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Pregnancies: Number of times pregnant Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test BloodPressure: Diastolic blood pressure (mm Hg) SkinThickness: Triceps skin fold thickness (mm) Insulin: 2-Hour serum insulin (mu U/ml) BMI: Body mass index (weight in kg/(height in m)^2) DiabetesPedigreeFunction: Diabetes pedigree function Age: Age (years) Outcome: Class variable (0 or 1)

[5]:
```
# there are altogether 768 rows and 9 columns.
```

[6]:
```
df.shape
```

[6]:
```
(768, 9)
```

[7]:
```
df.size
```

[7]:
```
6912
```

[8]:
```
df.head()
```

[8]:
```
   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
```

2 Two-hour postprandial glucose

Normal values are as follows [1] :

0-50 years - < 140 mg/dL or < 7.8 mmol/L (SI units) 50-60 years - < 150 mg/dL 60 years and older - < 160 mg/dL

[9]:
```
df.tail()
```

[9]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

[10]:
```
df.sample()
```

[10]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
292            2      128             78             37      182  43.3

     DiabetesPedigreeFunction  Age  Outcome
292                     1.224   31        1
```

[11]:
```
df['Outcome'].value_counts()
```

[11]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

[12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[13]:
```
df.describe()
```

[13]:

```
       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
```

[14]:

```
df.groupby('Outcome').mean()
```

[14]:

```
         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
```

Key Observations: Glucose Levels: The second person has higher glucose levels (141.26 vs. 109.98), which could indicate a higher likelihood of diabetes. BMI: The second person has a higher BMI (35.14 vs. 30.30), indicating obesity, which is a significant risk factor for diabetes. Age: The second individual is older (37.07 vs. 31.19), and age is a known risk factor for diabetes. Diabetes Pedigree Function: The second individual has a higher family history of diabetes (0.5505 vs. 0.4297), indicating a greater genetic predisposition. Outcome: Given the data above, the Outcome value for both individuals (not shown in your data snippet) would likely be:

Person 1: Given lower glucose, BMI, and a younger age, this person might not have diabetes (Outcome = 0). Person 2: With higher glucose, BMI, and age, this person is at a higher risk of diabetes (Outcome = 1).

[34]:

```
X=df.drop(columns='Outcome',axis=1)
y=df['Outcome']
```

[35]:

```
# use standard scaler to reduce every columns value in one uniform data.
```

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

[56]:
```
labels=scaler.fit(X)
```

[57]:
```
stand=labels.transform(X)
```

[58]:
```
print(stand)
```

```
[[ 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]]
```

[59]:
```
# Now
X=stand
y=df['Outcome']
```

[60]:
```
# use train test split:
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪2,stratify=y,random_state=3)
```

[61]:
```
model=SVC(kernel='linear')
```

[63]:
```
model.fit(X_train,y_train)
```

[63]:
```
SVC(kernel='linear')
```

[64]:
```
model.predict(X_train)
```

[64]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
```

```
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
      dtype=int64)
```

[65]:
```python
# for training data:
y_pred_train=model.predict(X_train)
print(y_pred_train)
```

```
[0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1
 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0
 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1
 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0
 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1
 0 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0
 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1
 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1
 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 1 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0]
```

[66]:
```python
# check the accuracy of the training dataset.
print("Accuracy score of training dataset",accuracy_score(y_pred_train,y_train))
```

```
Accuracy score of training dataset 0.7833876221498371
```

[67]:
```
# for test data:
y_pred_test=model.predict(X_test)
print(y_pred_test)
```

```
[0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0
 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0 1
 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0
 1 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0]
```

[68]:
```
# check the accuracy of the test dataset.
print("Accuracy score of test dataset",accuracy_score(y_pred_test,y_test))
```

```
Accuracy score of test dataset 0.7337662337662337
```

[75]:
```
y_pred=model.predict([[6,148,72,35,0,33.6,0.627,50]])
y_pred
```

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

[76]:
```
# input data:
input_=(6,148,72,35,0,33.6,0.627,50)
input_ar=np.asarray(input_)
input_resh=input_ar.reshape(1,-1)
data=scaler.transform(input_resh)
```

[77]:
```
result=model.predict(input_resh)
print(result)
```

```
[1]
```

[78]:
```
if (result[0]==0):
    print("Congrats, you are non diabetic")
else:
    print("You are suffering from diabeties")
```

```
You are suffering from diabeties
```

Key Takeaway: Always use fit_transform during training to compute scaling parameters, and transform during testing or prediction to ensure consistent scaling. This ensures your model sees the data in the same format it was trained on.

[ ]:
```

```