Diabetes Prediction using SVM

We have the pre-collected data from popular data scouce community, kaggle. Hear we are implementing suport vector machine to predict if a person is diabetic or not.

Steps Followed

1. Collect sonar data
2. Data Preprocessing (Standardization)
3. Train Test Split
4. Support Vector Machine Model (Binary Classification)
5. Predictions on old data (train and test)
6. Model Performance Results
7. Creating predictive system

In [1]:

```
import numpy as np
import pandas as pd
```

1. Collect/Load PIMA diabetes Data

In [2]:

```
#data collection and analysis

diabetes_df = pd.read_csv("C:/Users/Mohankumar MC/Desktop/ML Projects/Diabetes Predictio
```

2. Data Preprocessing

In [3]:

```
diabetes_df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [4]:

```
diabetes_df.shape
```

Out[4]:

```
(768, 9)
```

In [5]:

```
diabetes_df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [6]:

```
diabetes_df['Outcome'].value_counts()
```

Out[6]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

0 -> Non-diabetes Patents

1 -> diabetes Patents

In [7]:

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

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	D
Outcome							
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	

In [8]:

```
#seperating data and lables

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

In [9]:

```
X.head()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [10]:

```
y.head()
```

Out[10]:

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

data standardization

In [11]:

```
# Data preprocessing - data standardization
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
scalar.fit(X)
```

Out[11]:

```
StandardScaler()
```

or

In [12]:

```
Standardized_data = scalar.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]]
```

In [13]:

```
X = Standardized_data
y = diabetes_df['Outcome']
```

In [14]:

```
print(X)
```

```
[[ 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]]
```

In [15]:

```
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
```

In [16]:

```
print(type(Standardized_data))
print(type(y))
```

```
<class 'numpy.ndarray'>
<class 'pandas.core.series.Series'>
```

3. Train Test Split

In [17]:

```
from sklearn.model_selection import train_test_split
```

In [18]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.2,
                                                 stratify=y,random_state=2)
print(X.shape)
print( X_train.shape)
print(X_test.shape)
```

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

4. Logistic Regression Model (Binary Classification)

In [19]:

```
from sklearn import svm

classifier = svm.SVC(kernel = 'linear')
classifier.fit(X_train,Y_train)
```

Out[19]:

```
SVC(kernel='linear')
```

5. Predictions on old data (train and test)

In [20]:

```
train_data_prediction = classifier.predict(X_train)
train_data_prediction
```

Out[20]:

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

In [21]:

```
test_data_prediction = classifier.predict(X_test)
test_data_prediction
```

Out[21]:

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

6. Model Performance Results

In [22]:

```
from sklearn.metrics import accuracy_score

train_data_accuracy = accuracy_score(train_data_prediction,Y_train)
train_data_accuracy
```

Out[22]:

```
0.7866449511400652
```

In [23]:

```
test_data_accuracy = accuracy_score(test_data_prediction,Y_test)
test_data_accuracy
```

Out[23]:

```
0.7727272727272727
```

7. Creating predictive system

In [24]:

```
input_data = (5,116,74,0,0,25.6,0.201,30)
print(type(input_data))

#change input data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

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

# as we have standardized the data befor we need to make it again

std_input_data = scalar.transform(input_data_reshape)
print(std_input_data)

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

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

```
<class 'tuple'>
[[ 0.3429808  -0.15318486  0.25303625 -1.28821221 -0.69289057 -0.81134119
  -0.81807858 -0.27575966]]
[0]
The person is Non-diabetic
```

In [25]:

```
input_data = (2,197,70,45,543,30.5,0.158,53)
print(type(input_data))

#change input data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

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

# as we have standardized the data befor we need to make it again

std_input_data = scalar.transform(input_data_reshape)
print(std_input_data)

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

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

```
<class 'tuple'>
[[-0.54791859  2.38188392  0.04624525  1.53455054  4.02192191 -0.18943689
  -0.94794368  1.68125866]]
[1]
The person is diabetic
```