Medical Diagnosis with Support Vector Machines

Task 1: Import Libraries

In [1]:

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

Task 1: Get Data

In [2]:

```
column_names = ["pregnancies", "glucose", "bpressure", "skinfold", "insulin", "bmi", "pe
digree", "age", "class"]
df = pd.read_csv("data.csv", names=column_names)
print(df.shape)
df.head()
```

```
(768, 9)
```

Out[2]:

	pregnancies	glucose	bpressure	skinfold	insulin	bmi	pedigree	age	class
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

Task 1: Extract Features

In [3]:

```
X = df.iloc[:,:8] #all rows all columns upto 8
X.head()
```

Out[3]:

	pregnancies	glucose	bpressure	skinfold	insulin	bmi	pedigree	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Task 1: Extract Class Labels

In [4]:

```
y = df['class']
y.head()
```

Out[4]:

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

Task 2: Split Dataset

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
X_test.head()
```

```
(576, 8)
(576,)
(192, 8)
(192,)
```

Out[5]:

	pregnancies	glucose	bpressure	skinfold	insulin	bmi	pedigree	age
661	1	199	76	43	0	42.9	1.394	22
122	2	107	74	30	100	33.6	0.404	23
113	4	76	62	0	0	34.0	0.391	25
14	5	166	72	19	175	25.8	0.587	51
529	0	111	65	0	0	24.6	0.660	31

Task 2: Normalize Features

In [6]:

```
scaler = StandardScaler() #normalize means we neewd to give a range in which it is expect
ing
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_train[:5,:]
```

Out[6]:

```
array([[ 1.50755225, -1.01521454, -0.40451932, -1.31921491, -0.71823753,
        -1.22070104, -0.98325882, -0.04863985],
       [-0.82986389, -0.09964691, -0.61509602,  0.9287299 ,  0.08374747,
         0.13719053, -0.62493647, -0.88246592],
       [-1.12204091, -0.95207195,  0.54307587, -1.31921491, -0.71823753,
         0.0240329 ,  0.39884168, -0.5489355 ],
       [ 2.38408331,  0.59492164,  0.64836422,  1.36583027,  2.05458297,
         0.87900167,  0.17903049,  2.03592532],
       [ 1.50755225,  0.75277813,  0.54307587,  1.55315901,  0.39089067,
         0.71555175,  0.50724171,  0.53503839]])
```

Task 3: Training a Support Vector Machine

In [7]:

```
clf = svm.SVC(kernel='sigmoid') #scikit learn support vector classifier
clf.fit(X_train, y_train)
```

Out[7]:

```
SVC(kernel='sigmoid')
```

If we are buiding model from scratch by hand, by looking at data we can notice that people above BMI 25 are prone to diabetes so it has a simple decision boundary (above 2 class 1 , below 25 class 0) but real data is messy, we will have people with higher bmi but donot have diabetes and vice versa so SVM looks through all the features not only bmi and finds the best decision boundary

Task 3: Decision Boundary

In [8]:

```
y_pred = clf.predict(X_train)
print(y_pred)
print(accuracy_score(y_train, y_pred))
```

```
[0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0
 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0
 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0
 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0
 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 0 1 1 0 0 0 0 1 1
 0 0 1 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1
 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1
 0 0 1 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 1
 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0
 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 1 1 0 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0
 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0
 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1]
0.6510416666666666
```

Task 3: SVM Kernels

Functions used in SVM are called kernels and different kernels work better on diiferent datasets, so we should comapre them to know which gives best decision boundary

In [9]:

```
for k in ('linear', 'poly','rbf','sigmoid'):
    clf = svm.SVC(kernel=k)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_train)
    print(k)
    print(accuracy_score(y_train, y_pred)) #this process of trying different parmeters fo
r our svm is known as Hyperparameter optimisation
```

```
linear
0.7638888888888888
poly
0.7934027777777778
rbf
0.8246527777777778
sigmoid
0.6510416666666666
```

Task 4: Instantiating the Best Model

In [10]:

```
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[10]:

```
SVC()
```

Task 4: Making a single prediction

In [11]:

```
# "pregnancies", "glucose", "bpressure",
# "skinfold", "insulin", "bmi",
# "pedigree", "age", "class"
patient = np.array([[1., 50., 75., 40.,0.,45.,1.5,20]])
patient = scaler.transform(patient)
clf.predict(patient)
```

Out[11]:

```
array([0])
```

Task 4: Testing Set Prediction

In [12]:

```
patient = np.array([X_test.iloc[0]])
patient = scaler.transform(patient) #normalizing these features
print(clf.predict(patient))
print(y_test.iloc[0])
```

```
[1]
1
```

Task 5: Accuracy on Testing Set

In [13]:

```
X_test = scaler.transform(X_test)
y_pred = clf.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

```
0.7760416666666666
```

Task 5: Comparison to All-Zero Prediction

In [14]:

```
y_zero = np.zeros(y_test.shape)
print(accuracy_score(y_test, y_zero))
```

```
0.6770833333333334
```

Task 5: Precision and Recall

In [15]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       130
           1       0.71      0.52      0.60        62
```

```
    accuracy                           0.78       192
   macro avg       0.75      0.71      0.72       192
weighted avg       0.77      0.78      0.77       192
```

Class 1: PRECISION ranges from O to 1. If it is close to 1, our model avoids predicting that people have diabetescif they actually dont. That would be false positive. But precision may be high because we are underestimating how many people are sick.

RECALL: It also ranges from 0 to 1, if its close to 1 our model has correctly predicted that all the people with diabetes do have disease. This is avoiding false negatives.

We need high presision and high recall

F1 SCORE: It combines the two values but focuses on worst of the two. Its kind of pessimistic(worst) average of precision and recall

SUPPORT: It tells how many samples there were of each class.

In this project

1. **I have loaded a dataset and extracted its features and labels**
2. **Split it into training and test subsets and normalized**
3. **Set up a SVM using best kernel for job and used it to make medical diagnosis**

```
In [ ]:
```