# svm-classifier

September 23, 2023

## 1 Support Vector Machine

- Support Vector Machine is a supervised classification model which is used for predicting discrete class labels/categories.
- The target variable is a categorical value.
- SVM classifies cases by finding a separator,that is,
  1. Mapping the data to a higher dimensional space.
  2. Finding a separator.
- **Kernelling** - Kernelling is the process of mapping the data to a higher dimensional space in such a way that can change a linearly inseparable data to a linearly separable data.

SVM Algorithm: * It reads all the datapoints from the data set, finds a centre value and draw a vector along the centre point. * It also draws maximum margin towards the two classes, thus forming the shape of a hyper-plane * when a new data point arrives in, it decides the class based on the direction of the unknown data point.

[ ]:
```
# Importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
# Onboarding data onto colab
from google.colab import files
rawdata=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving diabetes (1).csv to diabetes (1).csv
```

[ ]:
```
# DataFrame

df=pd.read_csv('diabetes (1).csv')
df
```

[ ]:
```
     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]
```

[ ]:
```
# Shallow copy

df_copy=df.copy()
```

## 2 Exploratory Data Analysis

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

[ ]:
```
   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
```

[ ]:
```
df.values
```

[ ]:
```
array([[  6.   , 148.   ,  72.   , …,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , …,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , …,   0.672,  32.   ,   1.   ],
       …,
       [  5.   , 121.   ,  72.   , …,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , …,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , …,   0.315,  23.   ,   0.   ]])
```

[ ]:
```
df.columns
```

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

[ ]:
```
df.value_counts('Pregnancies')
```

[ ]:
```
Pregnancies
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
dtype: int64
```

[ ]:
```
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
```

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

[ ]:

| | 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 |

[ ]:
```
df.isnull().sum()
```

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

[ ]:
```
df.duplicated().sum()
```

[ ]: 0

## 3 Standardization

[ ]:
```
X=df.iloc[:,:-1]
Y=df.iloc[:,[-1]]
```

[ ]:
```
from sklearn import preprocessing

X_sc=preprocessing.scale(X)
X_sc
```

[ ]:
```
array([[ 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]])
```

## 4 Train test split

[ ]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.
↪3,random_state=23)
```

## 5 Model Building

[ ]:
```
from sklearn.svm import SVC

svc=SVC(kernel='linear')
svc.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
```

```
ravel().
  y = column_or_1d(y, warn=True)
```

[ ]:
```
SVC()
```

[ ]:
```
y_predict=svc.predict(x_test)
```

[ ]:
```
y_predict
```

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

[ ]:
```
y_test
```

[ ]:
```
     Outcome
93         1
228        0
424        1
635        1
684        0
..       …
271        0
46         0
476        1
130        1
359        1

[231 rows x 1 columns]
```

## 6 Model Evaluation

[ ]:
```
from sklearn.metrics import␣
  ↪confusion_matrix,accuracy_score,recall_score,f1_score

cm=confusion_matrix(y_test,y_predict)
acs=accuracy_score(y_test,y_predict)
rs=recall_score(y_test,y_predict)
F1=f1_score(y_test,y_predict)
```

```
print("Confusion_Matrix",cm)
print("Accuracy_score",acs)
print("Recall_score",rs)
print("F1_score",F1)
```

```
Confusion_Matrix [[132  16]
 [ 40  43]]
Accuracy_score 0.7575757575757576
Recall_score 0.5180722891566265
F1_score 0.6056338028169013
```

F1 score reveals that the model's accuracy rate is 0.60 which means that the model predicted 60% of the data correctly.

- The same diabetes data set was used in KNN model too which showed an accuracy rate of 0.5; Which shows that SVM gives better results than KNN.