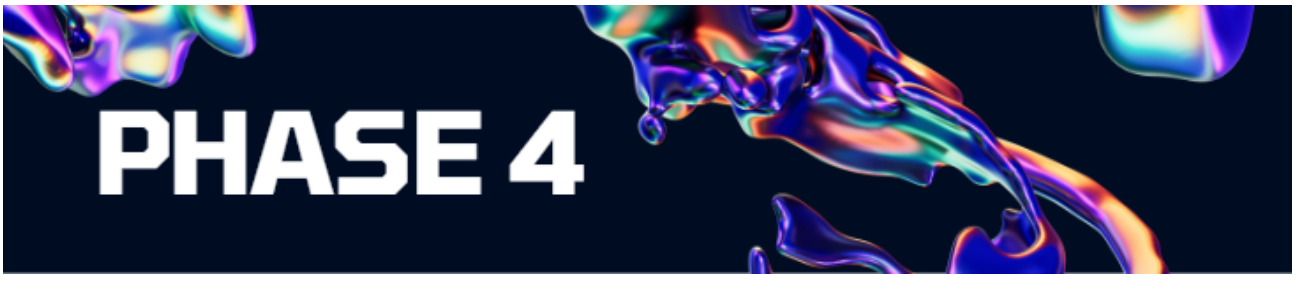


TITLE : "AI-BASED DIABETES PREDICTION SYSTEM"

In this phase, we continue building the prediction system by **selecting a machine learning algorithm, training our prediction system & evaluate its performance.**

Importing the Dependencies:

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

In the last phase, we have seen about **numpy** and **pandas**. StandardScalar is used standardize our data in similar range. To train our model we are using train test split method. SVM is the algorithm which we use to build this project. For evaluation purpose, accuracy score is imported.

Data Standardization:

- It is difficult for a machine learning model to make some predictions when the range of the values in the dataset are in vast difference. So it is required to **standardize our data in a particular range.**

- As we already imported the standard scaler from sklearn, we can use it for fitting and transforming the data in same range.

```
[ ] scaler = StandardScaler()

[ ] scaler.fit(X)

    StandardScaler(copy=True, with_mean=True, with_std=True)

[ ] standardized_data = scaler.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]
```

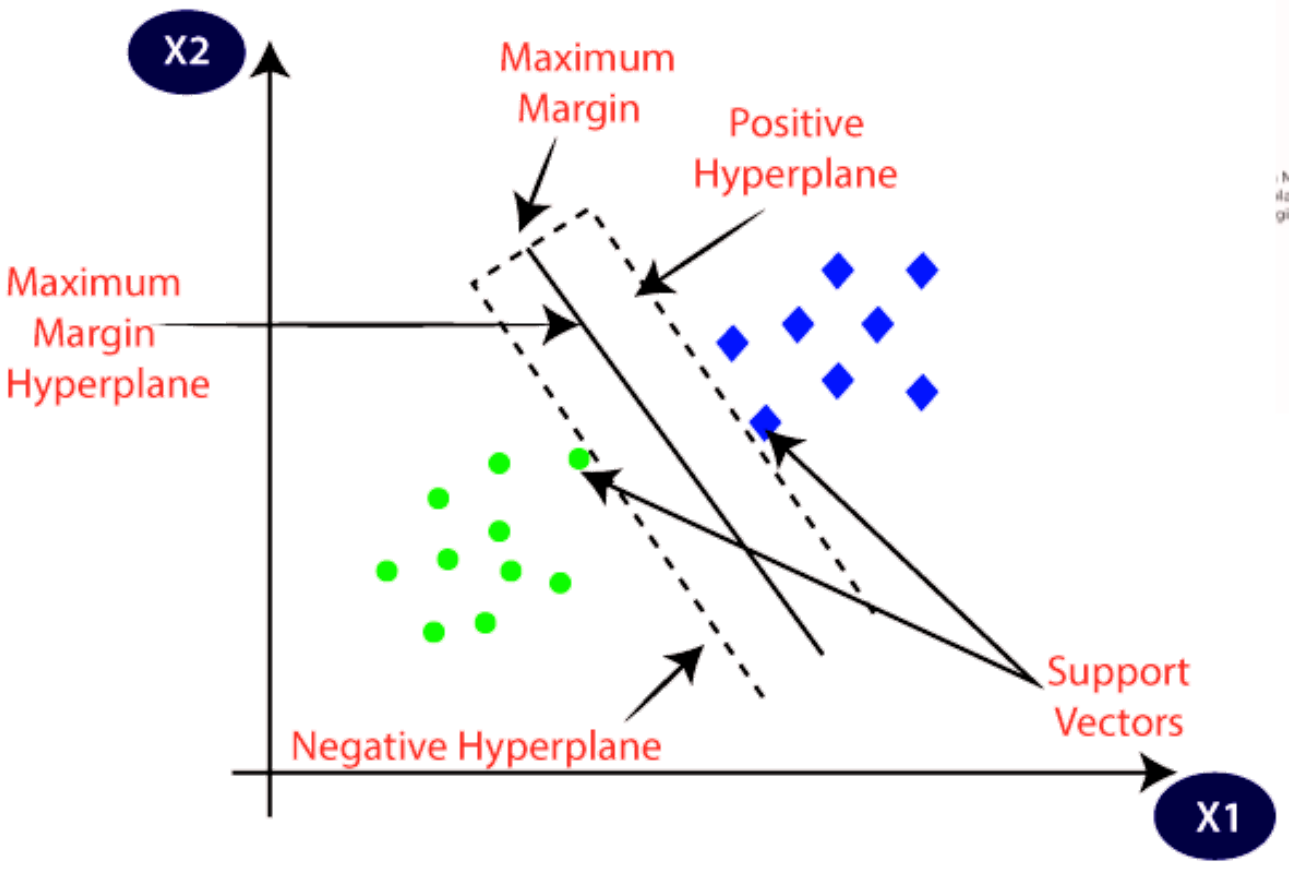
- we can see that values are in the range similar to each other.
- lets assign the variable X to the standardized data .

```
[ ] X = standardized_data
    Y = diabetes_dataset['Outcome']

    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]
     ...
```

Algorithm :

We are using Support Vector Machine Algorithm for our project since they can generate more complex decision boundaries.

Support Vector Machine or **SVM** is one of the most popular **Supervised Learning algorithms**, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
The goal of the SVM algorithm is to create the **best line** or **decision boundary** that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. **This best decision boundary is called a hyperplane.**
SVM chooses the extreme points/vectors that help in creating the hyperplane. **These extreme cases are called as support vectors**, and hence algorithm is termed as Support Vector Machine.



Training the Model:

Train Test Split

```
[ ]  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, X_train.shape, X_test.shape)

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

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- **Train Dataset: Used to fit the machine learning model.**
- **Test Dataset: Used to evaluate the fit machine learning model**

The objective is **to estimate the performance of the machine learning model on new data**: data not used to train the model.

#training the support vector Machine Classifier

```
[ ]  classifier = svm.SVC(kernel='linear')

     #training the support vector Machine Classifier
     classifier.fit(X_train, Y_train)

     SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coefo=0.0,
       decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
       max_iter=-1, probability=False, random_state=None, shrinking=True,
       tol=0.001, verbose=False)
```

Model Evaluation:

accuracy score on the training data

```
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score of the training data : ', training_data_accuracy)

Accuracy score of the training data :  0.7866449511400652
```

- We are comparing the X_train predictions with the original label Y_train in order to get the accuracy of our model.
- We got the accuracy score of our model as **0.78** which is pretty good.

accuracy score on the test data

```
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)

Accuracy score of the test data :  0.7727272727272727
```

- We got the accuracy score as **0.77** which represents that our model has **not been over trained**. Thus our model works good and we can build our predictive system.

Making a Predictive System:

```
input_data = (5,166,72,19,175,25.8,0.587,51)

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

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

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

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

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

- We have trained our model with 768 data points. But here we are using just one data point. If we don't reshape our input data, what our model expects is 768 data points. Thus it will make confusions. So we are re- shaping the arrays and standardizing.
- Now our trained model will predict the predictions for the given values and displays whether the person is diabetic or not by using the if condition.

Result:

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

- The given input is a random data from our PIMA dataset with the outcome 1 which denotes positive for diabetes.
- Our model has also predicted it correctly. Thus we have built a diabetes predictive system successfully.