**Experiment 12**

**Dhruv Singhal || 500075346 || R177219074 || AIML || Sem5**

**Support Vector Machines**

# Importing required libraries

In

[1]:

**import**

numpy

**as**

np

**import**

matplotlib

.

pyplot

**as**

plt

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

**from**

sklearn

**import**

svm

**from**

sklearn

.

datasets

**import**

make\_classification

**from**

sklearn

.

model\_selection

**import**

GridSearchCV

**from**

sklearn

.

model\_selection

**import**

KFold

**from**

sklearn

.

model\_selection

**import**

cross\_val\_score

# Creating Data

Creating Dataset with 5000 sample points and 2 features and 2 classes and number of informative features are 2 and number of redundant features are 0 and number of repeated features are 0 and random\_state is 50 and difference between 2 class is set as 3.5

In [2]: X, y**=** make\_classification(n\_samples**=**5000,n\_features**=**2,n\_classes**=**2

,n\_informative**=**2,n\_redundant**=**0,n\_repeated**=**0,random\_state**=**0,class\_sep**=**2.5)

# Preprocessing & Visualization

since we have created dataset using make\_classfiocation , it by default make dataset's mean tending to 0 and Standard Deviation to 1 which doesn't require any preprocessing.

# Model Building

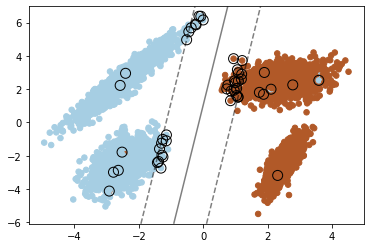
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

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. In this case we have called the in built function svm.SVC

# Compile & Train

Firstly splitting the input dataset into training and test parts following which training set is passed through svm.SVC and is fitted to it. and predicting over the test test to check the model performance.





# Result

since training is giving 99.67% accuracy and in testing it is giving 99.60% accuracy so to check if model is overfitting or not or to improve the validation accuracy more and make more generalized model . So to overcome this we are using k fold cross validation and hyperparameter tuning to improve the overall score and generate generalizesd model and knowing which hyperparameter combination gives best overall score

In

[4]:

*#Cross Validation without Hyper parameter Tuning*

clf

**=**

svm

.

SVC

(

kernel

**=**

'linear'

,

C

**=**

2

)

kf

**=**

KFold

(

n\_splits

**=**

70

)

score

**=**

cross\_val\_score

(

clf

,

X

,

y

,

cv

**=**

kf

)

print

(

"Average Cross Validation score ( Accuracy) :{}"

.

format

(

score

.

mean

()))

Average Cross Validation score ( Accuracy) :0.9965906550413592

In

[5]:

Best parameters set found on development set:

*# Hyper parameter tuning on SVM*

tuned\_parameters

**=**

[{

'kernel'

:[

"linear"

]

,

'C'

:[

100

,

10

,

20

,

40

,

50

]}

,

{

'kernel'

:[

"rbf"

]

,

'C'

:[

90

,

40

,

50

,

100

,

90

]}

,

]

clf

**=**

GridSearchCV

(

svm

.

SVC

()

,

tuned\_parameters

,

scoring

**=**

(

'accuracy'

))

clf

.

fit

(

X

,

y

)

print

(

"Best parameters set found on development set:"

)

print

()

print

(

clf

.

best\_params\_

)

print

()

print

(

"Best Score:"

,

clf

.

best\_score\_

)

{'C': 100, 'kernel': 'linear'}

In

[6]:

Best Score: 0.9966000000000002

Best parameters set found on development set:

*#%% kfold cross validation with hyperparameter tuning*

k

**=**

5

kf

**=**

KFold

(

n\_splits

**=**

k

,

random\_state

**=**

**None**

)

model

**=**

GridSearchCV

(

svm

.

SVC

()

,

tuned\_parameters

,

scoring

**=**

(

'accuracy'

))

**for**

train\_index

,

test\_index

**in**

kf

.

split

(

X

):

X\_train

,

X\_test

**=**

X

[

train\_index

,:],

X

[

test\_index

,:]

y\_train

,

y\_test

**=**

y

[

train\_index

]

,

y

[

test\_index

]

model

.

fit

(

X\_train

,

y\_train

)

pred\_values

**=**

model

.

predict

(

X\_test

)

print

(

"Best parameters set found on development set:"

)

print

()

print

(

model

.

best\_params\_

)

print

()

print

(

"Best Score:"

,

model

.

best\_score\_

)

{'C': 100, 'kernel': 'linear'}

Best Score: 0.9964999999999999

Now the overall score is 99.649% accuracy

In [ ]: