Support Vector Machines in Python

Contents

Contents

Introduction to Support Vector Machines

- Support Vector Machines (SVM's) are a relatively new learning method generally used for classification problem.
- Although the first paper dates way back to early 1960's it is only in 1992-1995 that this powerful method was universally adopted as a mainstream machine learning paradigm

The basic idea is to find a hyper plane which separates the d-dimensional data perfectly into its classes. However, since training data is often not linearly separable, SVM's introduce the notion of a "Kernel-induced Feature Space" which casts the data into a higher dimensional space where the data is separable.

What is a Hyper Plane

In two dimensions, a hyper plane is defined by the equation:

$$\boxed{W_1X_1 + W_2X_2 + b = 0}$$

This is nothing but **equation of line.**

The above equation can be easily extended to the p-dimensional setting:

$$\boxed{W_1X_1 + W_2X_2 + \cdots + W_pX_p + b = 0}$$

In short,

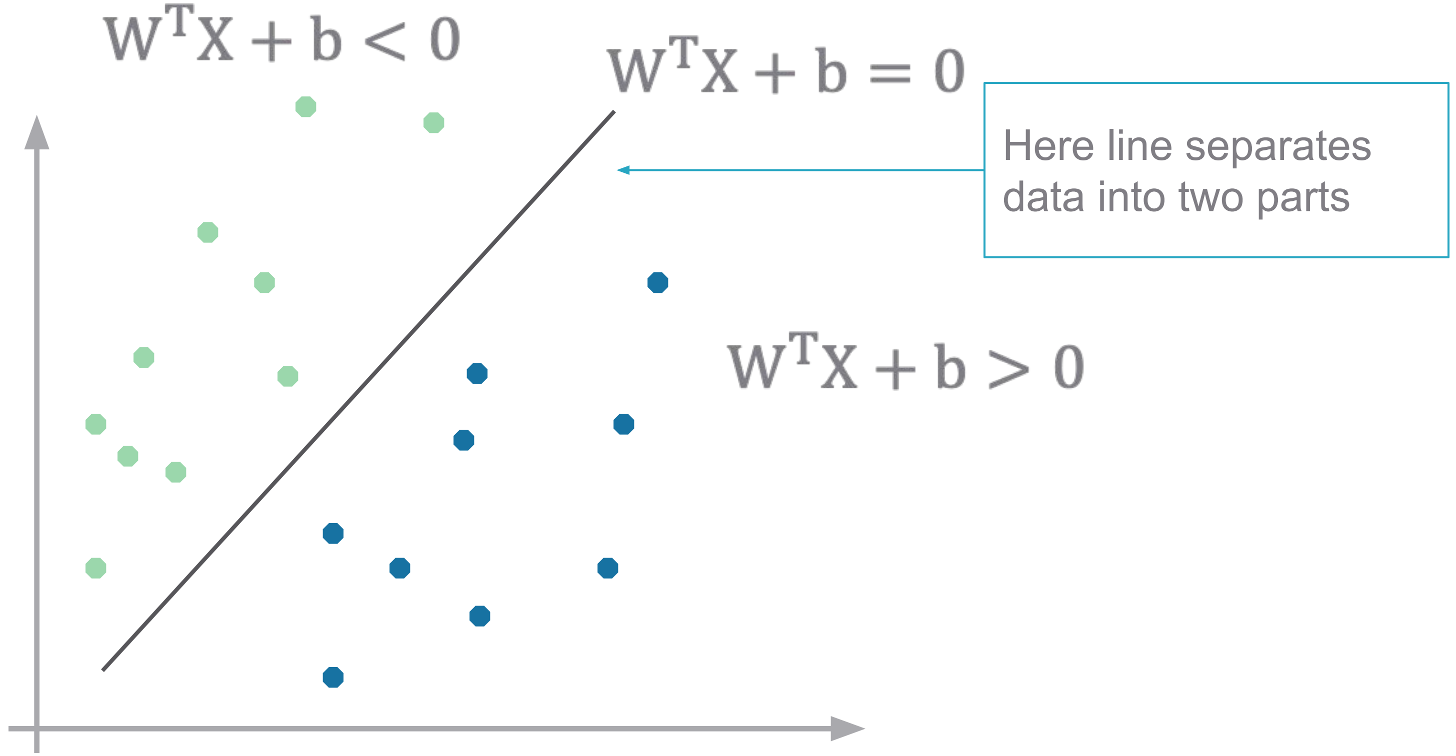
$$\boxed{\mathbf{W^TX + b = 0}}$$

In p > 3 dimensions, it can be hard to visualize a hyper planes.

Separating a Hyper Plane

- Binary classification can be viewed as the task of separating classes in feature space:

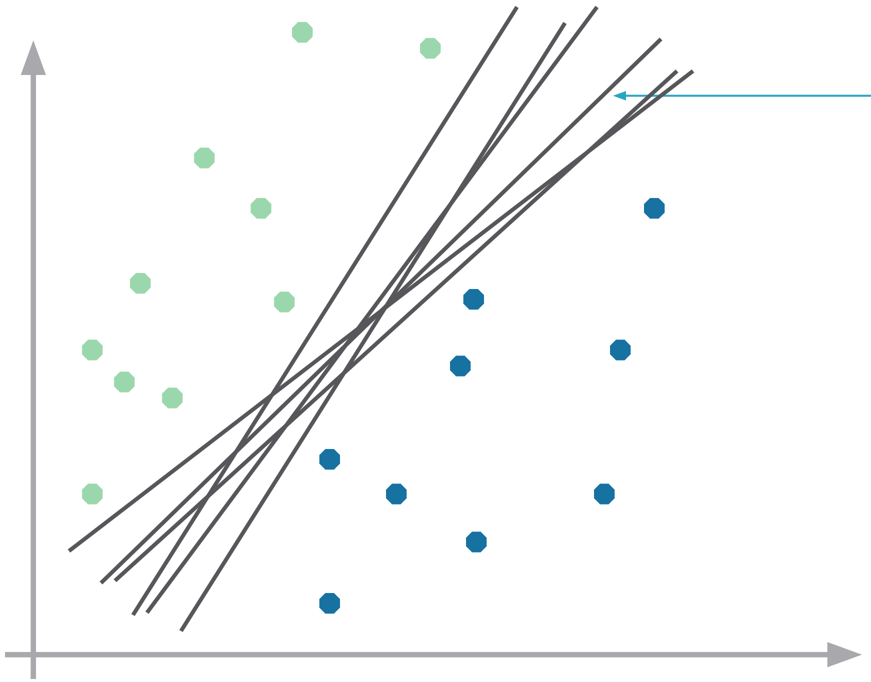
Fig. 01: Binary Classification



Linear Separators

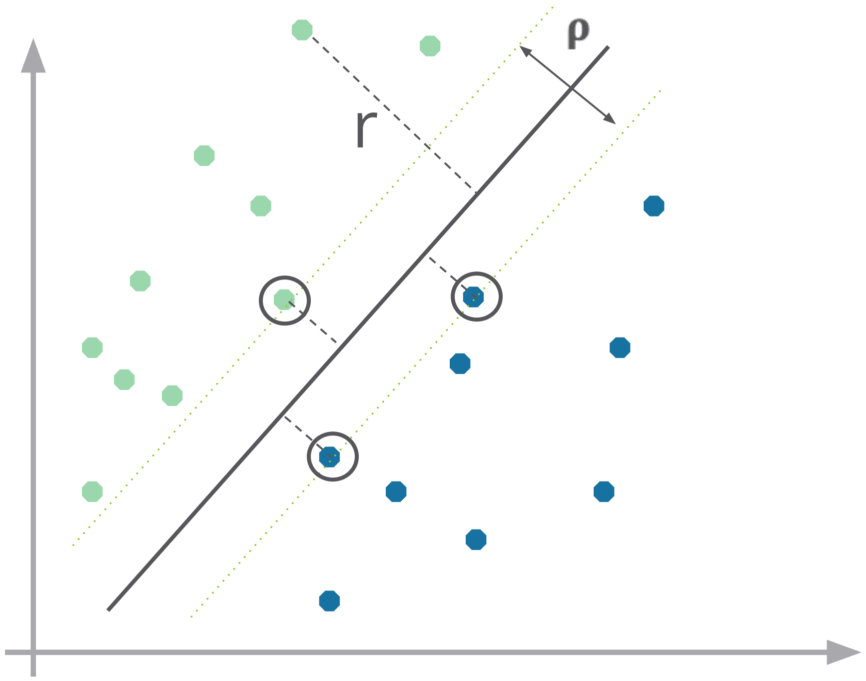
The objective in SVM is to find optimum separator

Fig. 02: Linear Separators



Which of the linear separators is optimal?

Classification Margin



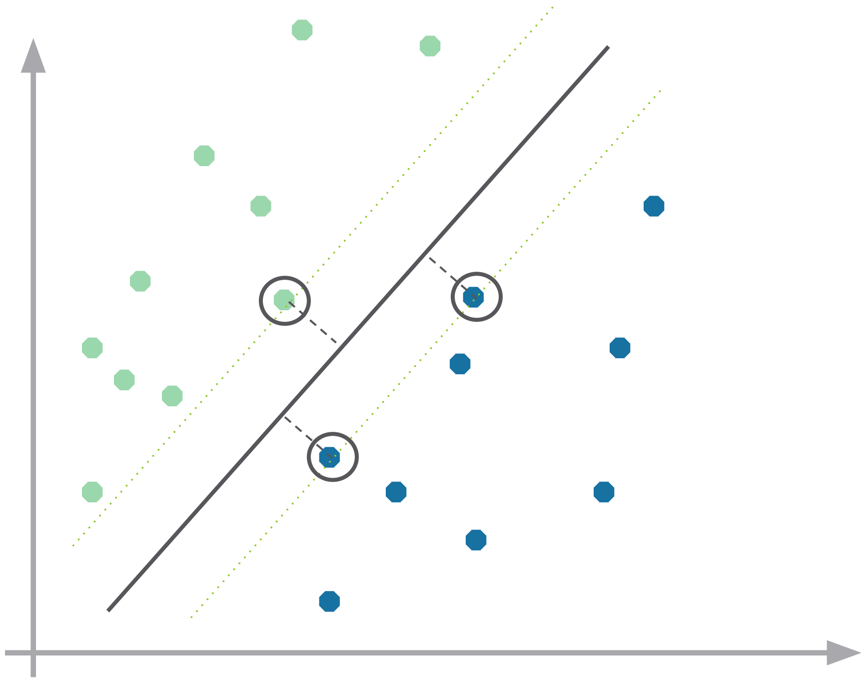
- Distance from case x_i to the separator is

$$r = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$$

Here $\|\mathbf{w}\|$ is length of a vector given by sqrt(sum(W^2))

- Cases closest to the hyper plane are Support Vectors
- Margin ρ of the separator is the distance between support vectors

Maximum Margin Classification



- The objective is now to maximize the margin ρ of the separator
- The focus is on 'Support Vectors'
- Other cases are not considered in the algorithm

Mathematical Approach to Linear SVM

Let training set be separated by a hyper plane with margin ρ. Then for each training observation

$$\begin{matrix} w^T x_i + b \le -\rho/2 & \text{if } y_i = -1 \\ w^T x_i + b \ge \rho/2 & \text{if } y_i = 1 \end{matrix} \leftrightarrow \boxed{y_i(w^T x_i + b) \ge \rho/2}$$

For every support vector $\mathbf{x}_s$ the above inequality is an equality

After rescaling $\mathbf{w}$ and $\mathbf{b}$ by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyper plane is

$$r = \frac{y_i(w^T x_s + b)}{\| w \|} = \frac{1}{\| w \|}$$

Margin can be expressed through (rescaled) $\mathbf{w}$ and $\mathbf{b}$ as:

$$\boxed{\rho = 2r = \frac{2}{\| w \|}}$$

Mathematical Approach to Linear SVM

Quadratic Optimisation problem is:

Find $\mathbf{w}$ and $\mathbf{b}$ such that

$\rho = \frac{2}{\|\mathbf{w}\|}$ is maximised

and

$$\mathrm{y_i}(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + \mathbf{b}) \geq 1$$

which can be reformulated as:

Find $\mathbf{w}$ and $\mathbf{b}$ such that

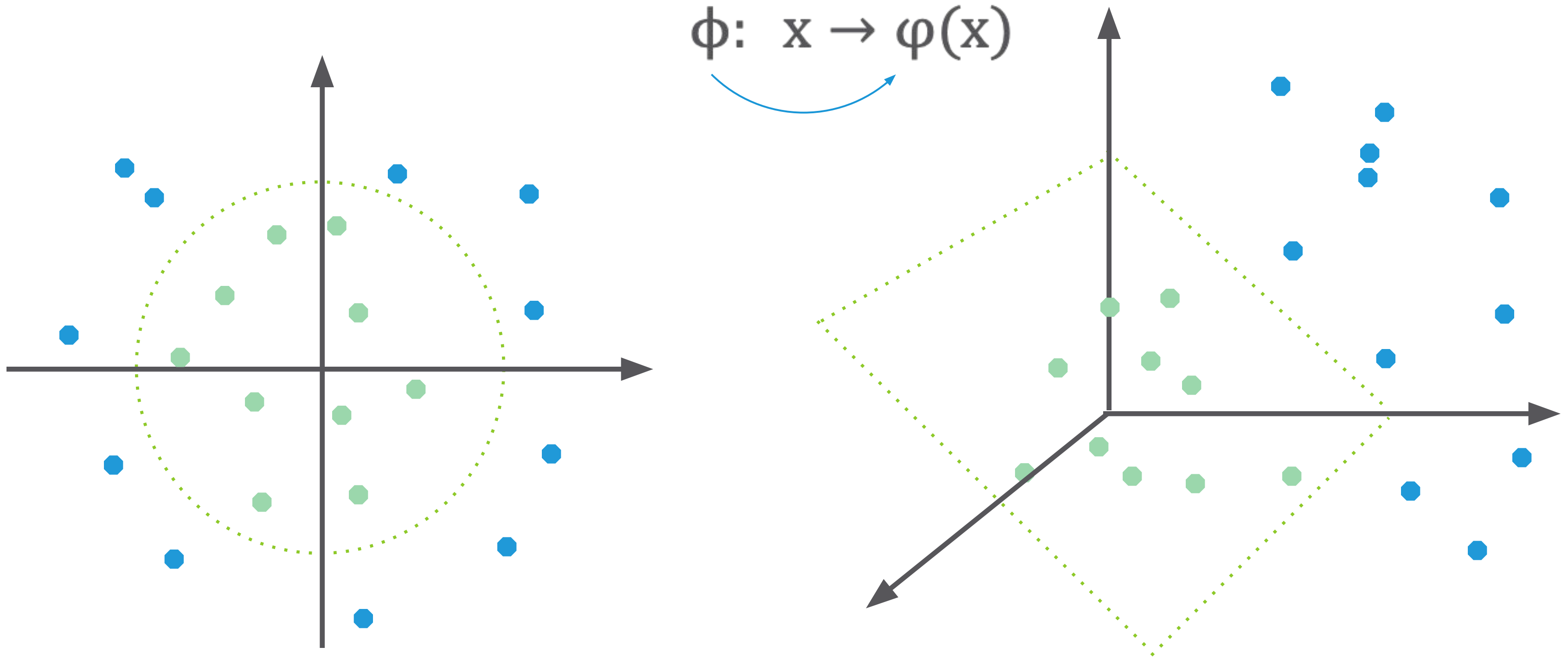
$\phi(w) = w^T w$ is minimised

and

$$\mathrm{y_i}(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + \mathbf{b}) \geq 1$$

Non-Linear SVMs – Feature Spaces

General idea: The original feature space can always be mapped to some higher-dimensional feature space where the training set is separable



The "Kernel Trick"

The linear classifier relies on inner product between vectors

$$K(x_i, x_j) = x_i^T x_j$$

If every data point is mapped into high-dimensional space via some transformation $\phi\colon\ x \rightarrow \varphi(x)$

then the inner product becomes

$$K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$$

A kernel function is a function that is equivalent to an inner product in some feature space

The "Kernel Trick"

Example:

2-dimensional vector $x = [x_1 \quad x_2]$;

Let $K(x_i, x_j) = (1 + x_i^T x_j)^2$

Need to show that $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$:

$K(x_i, x_j) = (1 + x_i^T x_j)^2$

$= 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$

$= [1 \quad x_{i1}^2 \; \sqrt{2}x_{i1}x_{i2} \quad x_{i2}^2 \; \sqrt{2}x_{i1} \quad \sqrt{2}x_{i2}]\ T\ [1$

$x_{j1}^2 \; \sqrt{2}x_{j1}x_{j2} \quad x_{j2}^2 \; \sqrt{2}x_{j1} \quad \sqrt{2}x_{j2}]$

$= \varphi(x_i)^T \varphi(x_j)$ where $\varphi(x) = [1 \; x_1^2 \sqrt{2}x_1x_2 \; x_2^2 \sqrt{2}x_1 \sqrt{2}x_2]$

Thus, a kernel function implicitly maps data to a high-dimensional space (Without the need to compute each $\varphi(x)$ explicitly)

Examples of Kernel Functions

Linear

$$K(x_i, x_j) = x_i^T x_j$$

Mapping ϕ

$x \rightarrow \varphi(x)$ where $\varphi(x)$ is x itself

Polynomial of power ρ

$$K(x_i, x_j) = (1 + x_i^T x_j)^{\rho}$$

Gaussian (Radial basis function)

$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

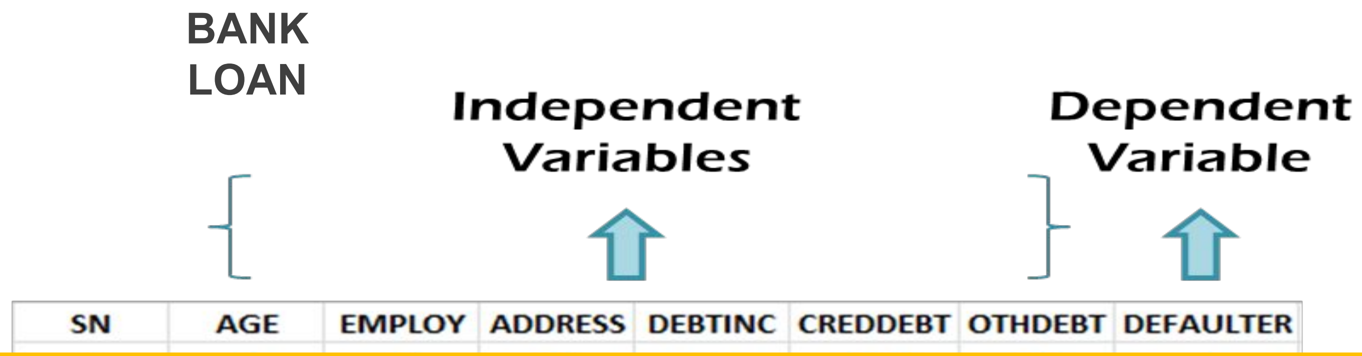
Objective

- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 700
- Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts are the independent variables
- Defaulter (=1 if defaulter, 0 otherwise) is the dependent variable

Data Snapshot



Column	Description	Type	Measurement	Possible Values
SN	Serial Number	Numeric	-	-
AGE	Age Groups	Categorical	1(<28 years), 2(28-40 years), 3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Continuous	-	Positive value
ADDRESS	Number of years customer staying at current address	Continuous	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit to Debit Ratio	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Binary	1(Defaulter), 0(Non-Defaulter)	2

SVM in Python

```
# Importing the Libraries
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix, f1_score,
precision_score, recall_score, accuracy_score,
roc_curve, roc_auc_score,auc
```

```
# Importing and Readying the Data
```

```python
bankloan = pd.read_csv("BANK LOAN.csv")

bankloan['AGE'] = pd.Categorical(bankloan['AGE'])

bankloan.info()
bankloan1 = bankloan.drop(['SN','AGE'], axis = 1)
```

- **pd.Categorical()** changes age from an integer to a factor variable.

- **info()** is used to check if the conversion to category has taken place and if all other variable formats are appropriate, before moving to SVM modeling.

SVM in Python

```
# Output
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 389 entries, 0 to 388
Data columns (total 8 columns):
SN          389 non-null int64
AGE         389 non-null category
EMPLOY      389 non-null int64
ADDRESS     389 non-null int64
DEBTINC     389 non-null float64
CREDDEBT    389 non-null float64
OTHDEBT     389 non-null float64
DEFAULTER   389 non-null int64
dtypes: category(1), float64(3), int64(4)
memory usage: 21.9 KB
```

```
# Creating Train and Test Data Sets
```

```
X = bankloan1.loc[:,bankloan1.columns != 'DEFAULTER']
y = bankloan1.loc[:, 'DEFAULTER']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state = 999)
```

- **train_test_split()** from sklearn.model_selection is used to split dataset into random train and test sets.
- **test_size** represents the proportion of dataset to be included in the test set.
- **random_state** sets the seed for the random number generator.

SVM in Python

```
# Model fitting
```

```
svclassifier = SVC(kernel='linear',probability=True)
svclassifier.fit(X_train, y_train)
```

- **svc()** trains a support vector machine.
- **kernel=** specifies the kernel type to be used in the algorithm‘(linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’).

```
# Output
```

```
SVC(kernel='linear', probability=True)
```

```
# Predicted Probabilities
```

```
predprob_test = svclassifier.predict_proba(X_test)
```

- **predict_proba()** returns predicted probabilities for the test data.

Predictions Based on SVM

```
# Custom Cutoff Value for Prediction Labels
```

```python
cutoff = 0.3
pred_test = np.where(predprob_test[:,1] > cutoff, 1, 0)
pred_test
```

- **np.where** is applied to the probabilities of true event to consider custom cutoff value. The output is an array of binary labels.

```
# Output
```

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

Confusion Matrix and Area Under ROC Curve

```
# Confusion Matrix
confusion_matrix(y_test, pred_test, labels=[0, 1])

array([[118,  36],
       [ 13,  43]])
accuracy_score(y_test, pred_test)
0.7666666666666667
precision_score(y_test, pred_test)
0.5443037974683544
recall_score(y_test, pred_test)
0.7678571428571429
```

- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as ‘Sensitivity’ = true positives / (true positives + false negatives)

```
# Area Under ROC Curve
auc = roc_auc_score(y_test, predprob_test[:,1])
print('AUC: %.3f' % auc)
AUC: 0.847
```

* Note : Output will be slightly different as observations are randomly assigned to train-test data.

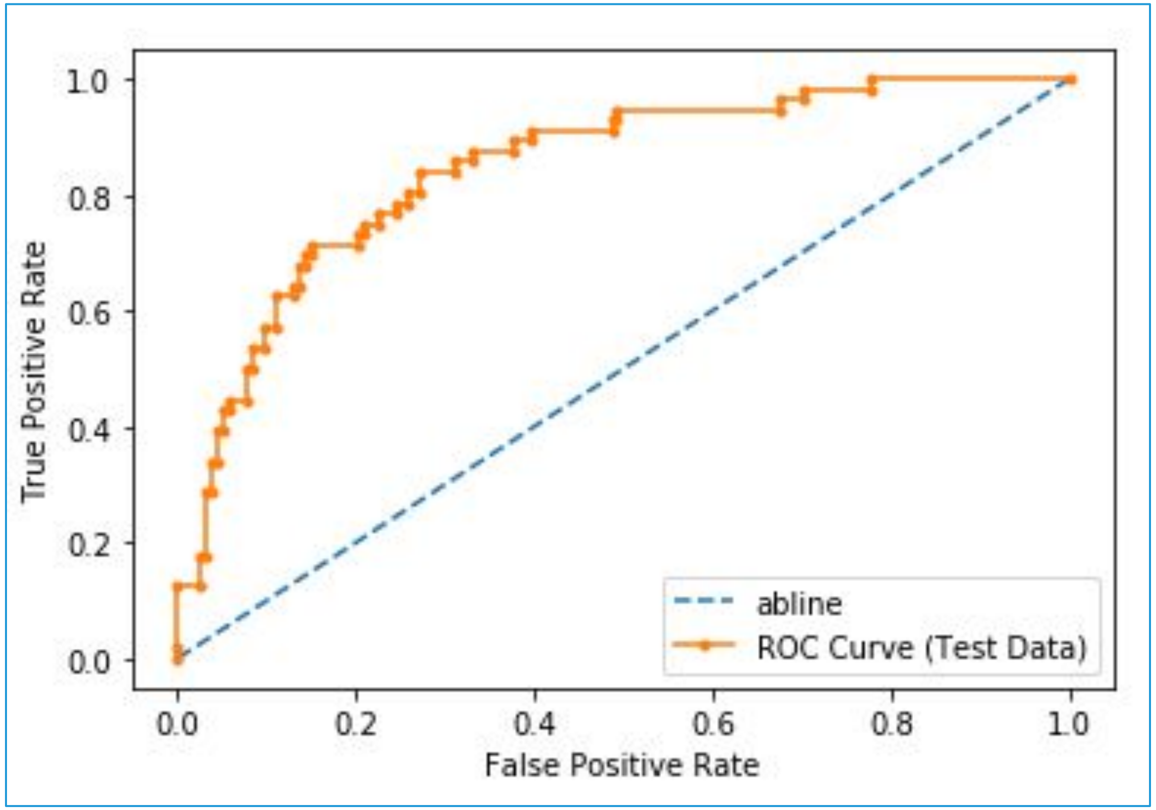
ROC Curve and Area Under ROC Curve

```
# ROC Curve
```

```
fpr, tpr, thresholds = roc_curve(y_test, predprob_test[:,1])

#Compute AUC using ‘auc’ function
roc_auc = auc(fpr, tpr)


#Plot the curve for model

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

ROC Curve and Area Under ROC Curve

```
# Output:
```



Quick Recap

In this session, we learnt about **Support Vector Machines:**

Support Vector Machines	• SVMs find a hyper plane which separates the d-dimensional data perfectly into its classes • Since training data is often not linearly separable, SVM's introduce the notion of a “Kernel-induced Feature Space" which casts the data into a higher dimensional space where the data is separable
SVM in Python	• Library **“`sklearn.svm`"** has **`SVC()`** that trains a support vector machine • The function takes arguments to specify whether **`SVC()`** is to be used for classification or regression; if probabilities are to be returned and which kernel to use for training and predicting