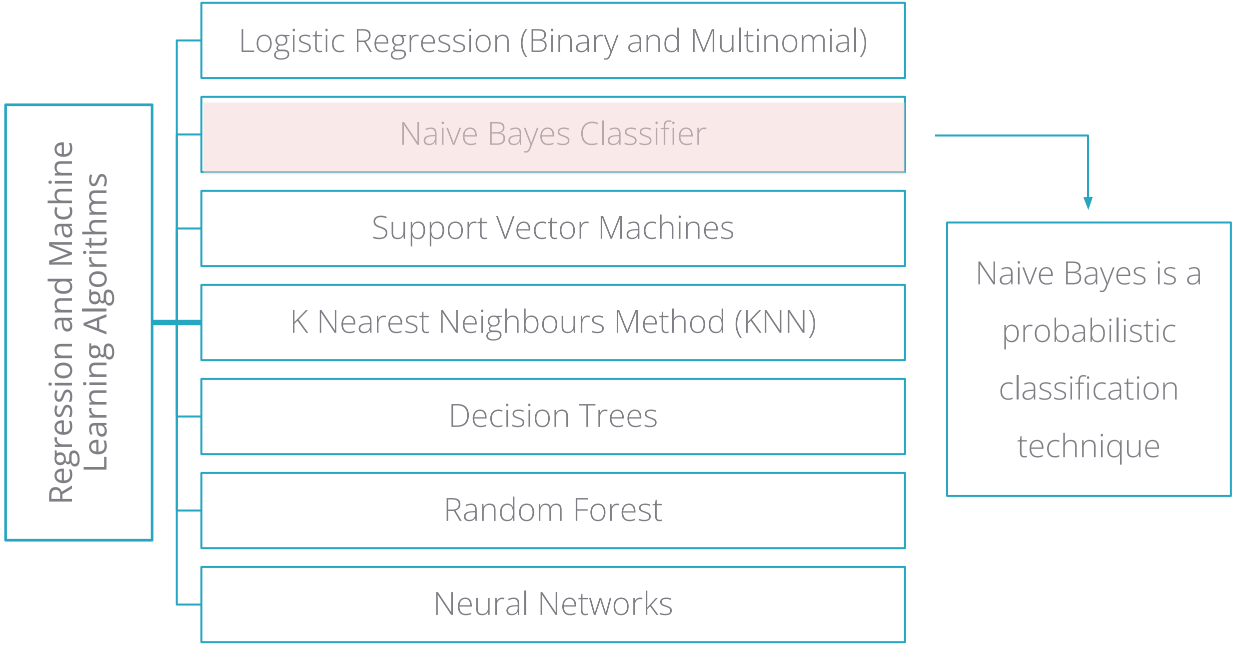
Naive Bayes Classifier - I

Classifier Based on Bayes' Theorem

Contents

Classification Methods

Apart from logistic regression, several types of machine learning algorithms are effective in classification and prediction.



About Naive Bayes Classifier

- Simple **probabilistic classifier based on Bayes Theorem.**
- It can be used as an **alternative method to logistic regression (Binary or Multinomial).**
- It **assumes conditional independence among the predictors.**
- It is particularly **suited when the dimensionality of the inputs is high.**

Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods.

Conditional Probability

The conditional probability of an event B is the probability that event B will occur given the knowledge that an event A has already occurred. This probability is written as $P(B|A)$.

- If A and B are independent events then

$$P(B|A) = P(B)$$

- An unbiased die, with numbers 1-6 is tossed

  A: Getting a number greater than 1

  B: Getting an even number

$$P(A) = 5/6$$

$$P(B) = 3/6$$

$$P(B|A) = 3/5$$

Here the sample space has 5 points given A has occurred.

Bayes Theorem

$$\underbrace{P(B|A)}_{\text{Posterior Probability}} = \frac{\overbrace{P(A|B)}^{\text{Conditional Probability}}\ \overbrace{P(B)}^{\text{Prior Probability}}}{\underbrace{P(A)}_{\text{Prior Probability}}}$$

where

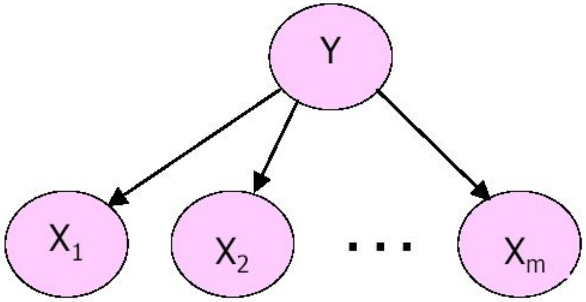
P(A) : Prior probability or marginal probability of A

P(A|B) : Conditional probability of A given B

P(B|A) : Conditional probability of B given A

P(B) : Prior or marginal probability of B

Naive Bayes Framework



Y : Categorical Dependent Variable

X_i : Categorical/Continuous Independent Variable

Objective: To estimate Y given the values of X_i's or

To estimate $P(Y|X_1, X_2, \ldots, X_m)$ using the Naïve Bayes Classifier

Assumption: All X_i's are conditionally independent of each other

Naive Bayes Framework - Example

Consider a simple example where Y is binary (response to a certain question) with 2 independent categorical variables X_1 and X_2

We classify	$Y = 1$ "Buyer" $Y = 0$ "Non-Buyer"
Let $\mathbf{X_1}$ denote **age of the individual**	$X_1 = 0$ for age group 25-30 years $X_1 = 1$ for age group 31-40 years
Let $\mathbf{X_2}$ denote **gender**	$X_2 = 0$ if Gender=female $X_2 = 1$ if Gender=male

Classification Rule

For the given values of X_1 and X_2 we want to know if the individual will be a potential buyer or not. Using Naive Bayes classifier we estimate:

$$P(Y = 0|X_1 = a_1, X_2 = a_2)$$

&

$$P(Y = 1|X_1 = a_1, X_2 = a_2)$$

where a_1 and a_2 are values of X_1 and X_2 for a particular respondent

We classify Y = 0 if $P(Y = 0|X_1 = a_1, X_2 = a_2) > 0.5$ OR

Y = 1 if $P(Y = 1|X_1 = a_1, X_2 = a_2) > 0.5$

In the general case i.e. when Y has more than 2 categories we compare $P(Y= y_k | X)$ for all values of y_k and classify $Y = y_k$ for which $P(Y= y_k | X)$ is the maximum

Expected Output

Once the classification rule is applied the output can be shown as follows:

Case#	X1	X2	$P(Y=1/X_1,X_2)$	$P(Y=0/X_1,X_2)$	Y classified as
1	1	0	0.44	0.56	0
2	1	1	0.7	0.3	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
240	0	0	0.2	0.8	0

Advantages of Naive Bayes Method

- Classification rule is simple to understand.
- The method requires a small amount of training data to estimate the parameters necessary for classification.
- The evaluation of the classifier is quick and easy.
- The method can be a good alternative to logistic regression.

Limitations of Naive Bayes Method

- Assumption of conditional independence of the independent variables is highly impractical.
- In case of continuous independent variables the density function must be known or assumed to be normal.
- In case of categorical independent variables the probabilities cannot be calculated if the count in any conditional category is zero. For instance: If there are no respondents in the age group 25-30 yrs. then $P(X_1=0 \mid Y=1) = 0$

?

How to deal with such cases?
If a category has zero entries we replace 0 by 0.5/n (n = sample size) so that the probability expression does not reduce to zero.

Quick Recap

In this session, we learnt **Naive Bayes Classification** technique:

Conditional Probability and Bayes' Theorem	• The conditional probability of an event B is the probability that event B will occur given the knowledge that an event A has already occurred. • $P(B\|A) = P(A\|B)\ P(B) / P(A)$
Naive Bayes Classifier	• To estimate Y given the values of X_i's or $P(Y\|X_1, X_2, \dots, X_m)$ using the Naïve Bayes Classifier. • **Assumption:** All X_i's are conditionally independent of each other. • **Advantages:** Simple classification rule, requires a small amount of training data to estimate the parameters necessary for classification, Evaluation of the classifier is quick and easy, Good alternative to logistic regression. • **Major drawback:** Assumption of conditional independence of the independent variables is highly impractical.

Naive Bayes Classifier - II

Classifier Based on Bayes' Theorem

Contents

Case Study – Modeling Loan Defaults

Background

- A bank possesses demographic and transactional data of its loan customers. If the bank has a model to predict defaulters it can help in loan disbursal decision making.

Objective

- To predict whether the customer applying for the loan will be a defaulter or not.

Available Information

- Sample size is 700
- Independent Variables: Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts. The information on predictors was collected at the time of loan application process.
- Dependent Variable: Defaulter (=1 if defaulter ,0 otherwise). The status is observed after loan is disbursed.

Data Snapshot

Independent Variables

Dependent Variable

SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTE
1	3	17	12	9.3	11.36	5.01	1
2	1	10	6	17.3	1.36	4	0
3	2	15	14	5.5	0.86	2.17	0
4	3	15	14	2.9	2.66	0.82	0

Column	Description	Type	Measurement	Possible Values
SN	Serial Number	numeric	-	-
AGE	Age Groups	Integer	1(<28 years), 2(28-40 years), 3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Integer	-	Positive value
ADDRESS	Number of years customer staying at current address	Integer	-	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	Integer	1(Defaulter), 0(Non-Defaulter)	2

Naive Bayes Method in Python

```
# Importing required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB

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

- Naive Bayes methods differ based on the type of predictors- continuous or categorical
- Python's sklearn has various methods available and the two methods explored hereon are Gaussian Naive Bayes and Multinomial Naive Bayes.

Naive Bayes Method in Python
For Continuous Predictors

```
# Importing and Readying the Data for Modeling
bankloan = pd.read_csv("BANK LOAN.csv")
bankloan1 = bankloan.drop(['SN','AGE'], axis = 1)
bankloan1.head()
```

- **drop()** is used to remove unwanted variables. AGE is removed because it is a categorical variable.
- **axis = 1** drops columns.

```
# Output
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER
0	17	12	9.3	11.36	5.01	1
1	10	6	17.3	1.36	4.00	0
2	15	14	5.5	0.86	2.17	0
3	15	14	2.9	2.66	0.82	0
4	2	0	17.3	1.79	3.06	1

Naive Bayes Method in Python
For Continuous Predictors

```
# 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.

Naive Bayes Method in Python For Continuous Predictors

```
# Model Fitting
NBmodel = GaussianNB()

NBmodel.fit(X_train, y_train)
```

- **GaussianNB()** fits a Gaussian Naive Bayes algorithm for classification.
- This model is suitable for continuous predictors and assumes the likelihood of predictors to be normal.

```
# Predicted Probabilities
predprob_test = NBmodel.predict_proba(X_test)
predprob_test
```

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

```
# Output
array([[0.96499091, 0.03500909],
       [0.86941828, 0.13058172],
       [0.90585744, 0.09414256],
       [0.97398393, 0.02601607],
       [0.99549445, 0.00450555],
       [0.52978724, 0.47021276],
```

Naive Bayes Method in Python
For Continuous Predictors

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

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

- The output is an array of binary labels.

```
# Output
```

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

Naive Bayes Method in Python
For Continuous Predictors

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

array([[135, 22],
       [ 26, 27]])
```

```
accuracy_score(y_test, pred_test)
0.7714285714285715
```

```
precision_score(y_test, pred_test)
0.5510204081632653
```

```
recall_score(y_test, pred_test)
0.5094339622641509
```

- **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: 0.816
```

- **roc_auc_score** computes Area Under the ROC curve.

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

Naive Bayes Method in Python
For Continuous Predictors

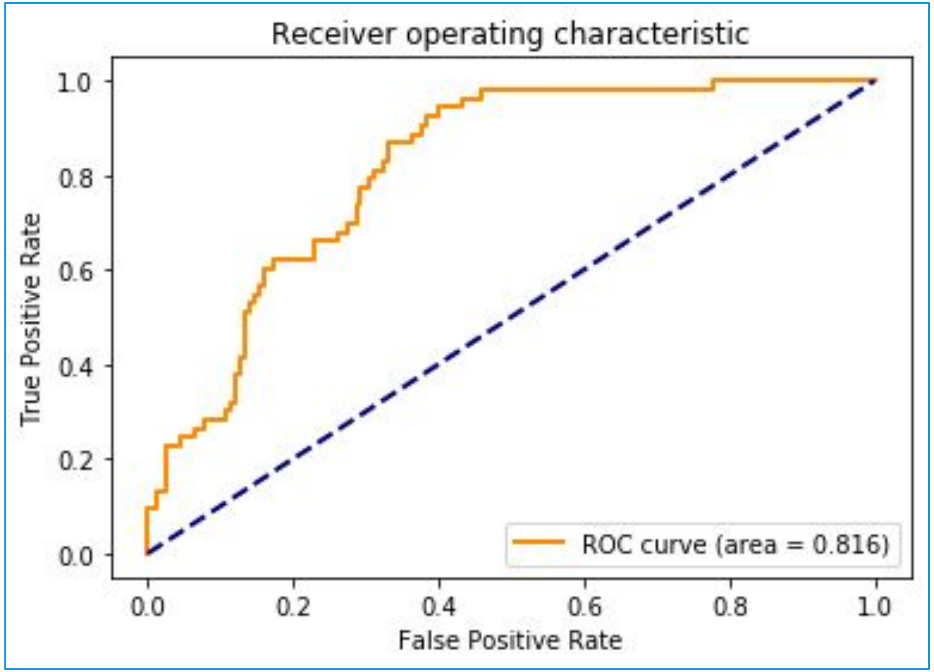
```
# ROC Curve
```

```
NBfpr, NBtpr, thresholds = roc_curve(y_test, predprob_test[:,1])

# plot the roc curve for the model
plt.figure()
lw = 2
plt.plot(NBfpr, NBtpr, color='darkorange',lw=lw, label='ROC curve
(area = %0.3f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.axis('tight')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic’)
plt.legend(loc="lower right")
plt.show()
```

- **roc_curve** is used to Compute Receiver operating characteristic.

Naive Bayes Method in Python
For Continuous Predictors

```
# Output:
```



Case Study – Employee Churn Model

Background

- A company has comprehensive database of its past and present workforce, with information on their demographics, education, experience and hiring background as well as their work profile. The management wishes to see if this data can be used for predictive analysis, to control attrition levels.

Objective

- To develop an Employee Churn model via Naive Bayes

Available Information

- Sample size is 83
- **Gender, Experience Level** (<3, 3-5 and >5 years), **Function** (Marketing, Finance, Client Servicing (CS)) **and Source** (Internal or External) are independent variables
- **Status** is the dependent variable (=1 if employee left within 18 months from joining date)

Data Snapshot

EMPLOYEE CHURN DATA

Dependent Variable

Independent Variables

sn	status	function	exp	gender	source
1	1	CS	<3	M	external

Columns	Description	Type	Measurement	Possible values
sn	Serial Number	Integer	-	-
status	= 1 If the Employee Left Within 18 Months of Joining = 0 Otherwise	Integer	1,0	2
function	Employee Job Profile	Character	CS, FINANCE, MARKETING	3
exp	Experience in Years	Character	<3,3-5,>5	3
gender	Gender of the Employee	Character	M,F	2
source	Whether the Employee was Appointed via Internal or External Links	Character	external, internal	2

Naive Bayes Method in Python For Categorical Predictors

```
# Importing and Readying the Data for Modeling, Model Fitting
empdata = pd.read_csv("EMPLOYEE CHURN DATA.csv")
empdata1 = empdata.loc[:, empdata.columns != 'sn']
empdata1.head()
```

- **loc()** is used to create a subset of the data frame using column name. Removing column with serial numbers.

```
# Output
```

	status	function	exp	gender	source
0	1	CS	<3	M	external
1	1	CS	<3	M	external
2	1	CS	>=3 and <=5	M	internal
3	1	CS	>=3 and <=5	F	internal
4	1	CS	<3	M	internal

Naive Bayes Method in Python
For Categorical Predictors

```
# Creating Dummy Variables
```

```
empdata2 = pd.get_dummies(empdata1)
empdata2.head()
```

- **pd.get_dummies()** converts categorical variables into dummy variables. This step is crucial because the naive Bayes function used for categorical variables requires this format.

```
# Output
```

```
   status  function_CS  ...  source_external  source_internal
0       1            1  ...                1                0
1       1            1  ...                1                0
2       1            1  ...                0                1
3       1            1  ...                0                1
4       1            1  ...                0                1
```

Naive Bayes Method in Python For Categorical Predictors

```
# Creating Data Partitions
```

```
X_emp = empdata2.loc[:,empdata2.columns != 'status']
y_emp = empdata2.loc[:, 'status']
```

```
# Model Fitting
```

```
MNBmodel = MultinomialNB(alpha = 0)

MNBmodel.fit(X_emp, y_emp)
```

- **MultinomialNB()** fits a Multinomial Naive Bayes algorithm for classification. This model is suitable for categorical predictors.
- alpha = 0 ensures the model doesn't apply any smoothing on the data.

Naive Bayes Method in Python
For Categorical Predictors

```
# Predicted Probabilities
```

```
predprob_MNB = MNBmodel.predict_proba(X_emp)
predprob_MNB
```

```
# Output
```

```
array([[0.06419224, 0.93580776],
       [0.06419224, 0.93580776],
       [0.49966736, 0.50033264],
       [0.5358654 , 0.4641346 ],
       [0.1377729 , 0.8622271 ],
       [0.5625865 , 0.4374135 ],
       [0.59789573, 0.40210427],
       [0.07347548, 0.92652452],
```

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

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

Naive Bayes Method in Python
For Categorical Predictors

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

array([[37, 13],
       [ 3, 30]])
accuracy_score(y_emp, pred_test)
0.8072289156626506
precision_score(y_emp, pred_test)
0.6976744186046512
recall_score(y_emp, pred_test)
0.9090909090909091
```

- **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_emp, predprob_MNB[:,1])
print('AUC: %.3f' % auc)
AUC: 0.871
```

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

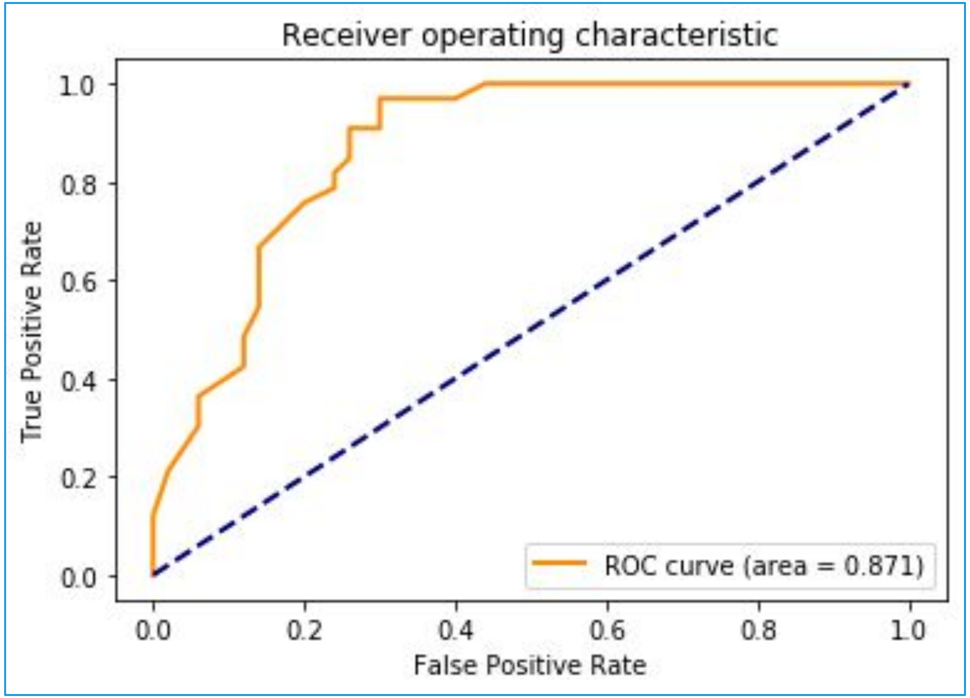
Naive Bayes Method in Python
For Categorical Predictors

```
# ROC Curve
```

```
MNBfpr, MNBtpr, thresholds = roc_curve(y_emp, predprob_MNB[:,1])

# plot the roc curve for the model
plt.figure()
lw = 2
plt.plot(MNBfpr, MNBtpr, color='darkorange',lw=lw, label='ROC curve
(area = %0.3f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.axis('tight')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

Naive Bayes Method in Python For Categorical Predictors

```
# Output
```



Laplace Smoothing

$$P(x = x_i \mid y = y_j) = f_i / N_j$$

This prob will be 0 if numerator count (f_i) is 0

Laplace smoothing will replace this probability with a value obtained by the formula:

$$\boxed{\hat{\theta}_i = \frac{f_i + \alpha}{N_j + \alpha d_i}}$$

where

α : Smoothing Parameter

N_j : Number of observations for $Y = y_j$

d_i : Number of classes of x

Quick Recap

Naive Bayes in Python	• `GaussianNB` for continuous variables, `MultinomialNB` for categorical variables in library `sklearn.naive_bayes`
Laplace Smoothing	• If a given class and feature value never occur together in the training data, then the frequency-based probability estimate will be zero. • A pseudo-count is incorporated, in all probability estimates such that no probability is ever set to be exactly zero. • This way of regularizing naive Bayes is called Laplace Smoothing

K Nearest Neighbours Classifier

Learn how a Simple Lazy Learning Algorithm Works

Contents

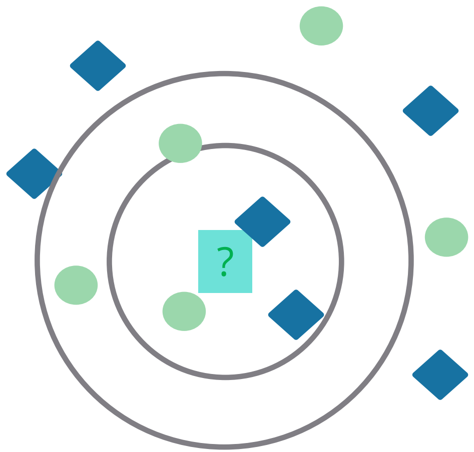
KNN for Classification

- Training dataset has 11 observations belonging to two categories.
- 12th observation is introduced, class of which is not known.
- **Nearest neighbour algorithm classifies new observation to the class of the training observation closest to it.**

When K=1, nearest one case is considered

As we go on increasing K, classification may vary

K	
1	**Blue**
3	**Blue**
5	**Orange**



Three most important components of this method are **Distance** between cases, **Value of K** and **Voting** criteria.

Simple Example To Understand KNN Method

Age	Current Debt	Default
25	40,000	N
35	60,000	N
45	80,000	N
20	20,000	N
35	120,000	N
52	18,000	N
23	95,000	Y
40	62,000	Y
60	100,000	Y
48	220,000	Y
33	150,000	Y
48	**142,000**	**?**

New observation will be classified as “N” or “Y” based on KNN method

Distance Based on Standardised Variables

$$X_s = \frac{X - Min}{Max - Min}$$

Alternatively, $\frac{(X-Mean)}{SD}$ can also be used

Age	Current Debt	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

New observation will be classified as “N”

Selection of K

The second component of KNN model is selecting the appropriate value for K

- If K = 1, the case is classified using the nearest neighbour
- However, K is usually greater than 1. Consider the following when choosing K :
  - Mostly odd numbered K is preferred to avoid tie.
  - For a very large K the classifier may result in misclassification, as group of nearest neighbours may include data points which are actually located far away from it.

Thumb Rule :

K = sqrt (n)

n is the number of observations in training data

Voting Criteria

Most common criteria for classification decision is **Majority Voting.**

Frequency of each class in K instances is measured. Class having the highest frequency is attributed to the new case.

Eg. Suppose for K = 7, 4 cases belong to class A and 3 to class B. New case is given class A

Drawback:

Classification is inappropriate when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.

?

Is there a way to correct this?
One option to remove this drawback is to create training data with equal class frequency. However, this is possible only if data is very large.

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 389
- **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

BANK LOAN KNN

Variables

SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER

Column	Description	Type	Measurement	Possible Values
SN	Serial Number		-	-
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 Card Debt	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Binary	1(Defaulter), 0(Non-Defaulter)	2

KNN Classification in Python

```
# Importing the Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

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

KNN Classification in Python

```
# Importing the Data
bankloan = pd.read_csv("BANK LOAN KNN.csv")

# Preparing data by removing unwanted variables
bankloan1 = bankloan.drop(['SN','AGE'], axis = 1)

bankloan1.head()
```

- **drop()** is used to remove unwanted variables. AGE is removed because it is a categorical variable.
- **axis = 1** drops columns.

```
# Output
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER
0	17	12	9.3	11.36	5.01	1
1	2	0	17.3	1.79	3.06	1
2	12	11	3.6	0.13	1.24	0
3	3	4	24.4	1.36	3.28	1
4	24	14	10.0	3.93	2.47	0

KNN Classification in Python

```
# Creating Train and Test Datasets
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

KNN Classification in Python

```
# Preparing Variables
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train
```

- **StandardScaler()** from sklearn.preprocessing is a generic function used for centering or scaling columns of a numeric matrix. The default method for scaling is (X-Mean)/SD.

```
# Output
```

```
array([[-0.89496854, -0.29977261,  1.11865332, -0.28678534,  0.00538471],
       [-0.23076269, -1.21765047,  1.07813931,  1.32686881, -0.1734379 ],
       [ 0.26739169,  0.00618668,  0.26785916, -0.20818328,  0.80582882],
       ...,
       [-0.06471123,  1.07704418, -0.69097236, -0.3191509 , -0.42889879],
       [-0.56286562,  0.15916632,  0.28136383, -0.18044137, -0.50553705],
       [-1.22707147, -0.14679297,  0.24084982,  0.2356872 , -0.40051425]])
```

- All the continuous predictors are now scaled to mean=0 and sd=1.
- Note: Test data is transformed using the train data parameters

KNN Classification in Python

```
# Building the KNN Classifier (Continuous Predictors)
KNNclassifier = KNeighborsClassifier(n_neighbors =
                                     int(np.sqrt(len(X)).round()))

# using thumb rule k ~= sqrt(n)
KNNclassifier.fit(X_train, y_train)
```

- **KNeighborsClassifier()** from sklearn.neighbors performs k-nearest neighbour classification
- **n_neighbors=** specifies the value of k.

```
# Output
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=20, p=2,
                     weights='uniform')
```

KNN Classification in Python

```
# Predictions on Test Data
y_pred = KNNclassifier.predict(X_test)
```

predict() predicts the class labels of the provided data. The default threshold is 0.5.

```
# Confusion Matrix
confusion_matrix(y_test, y_pred, labels=[0, 1])
array([[49, 7],
      [24, 37]])

accuracy_score(y_test, y_pred)
0.7350427350427351

precision_score(y_test, y_pred)
0.8409090909090909

recall_score(y_test, y_pred)
0.6065573770491803
```

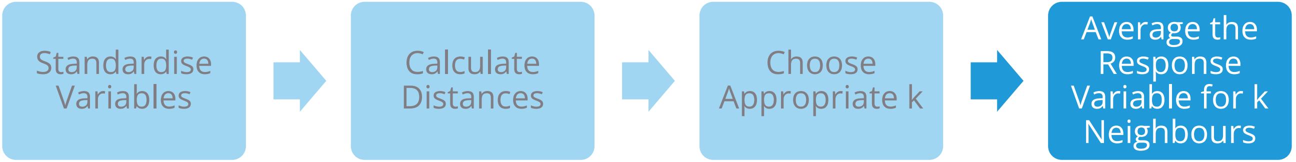
- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as 'Sensitivity' = true positives / (true positives + false negatives)

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

KNN for Regression

KNN algorithm can also be extended to regression problems, i.e. **when the dependent variable is continuous**

Process flow for classification and regression is the same, except for the last step



Average value of the response variable for k neighbours is calculated and assigned to the new case.

`KNeighborsRegressor()` from sklearn.neighbors can be used to run k-nearest neighbour regression in Python.

Get an Edge!

- KNN can be used for categorical variables as well.
- Before executing knn on train-test data, categorical variables have to be converted to numeric variables by creating dummy variables.

Quick Recap

In this session, we learnt about **KNN Classifier** :L

KNN for Classification	• Three most important components of this method are **Distance** between cases, **Value of K** and **Voting** criteria.
KNN for Classification in Python	• `KNeighborsClassifier()` from `sklearn.neighbors`.
KNN for Regression	• KNN algorithm can also be extended to regression problems **when the dependent variable is continuous.**
KNN for Regression in Python	• `KNeighborsRegressor()` from `sklearn.neighbors`.

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,

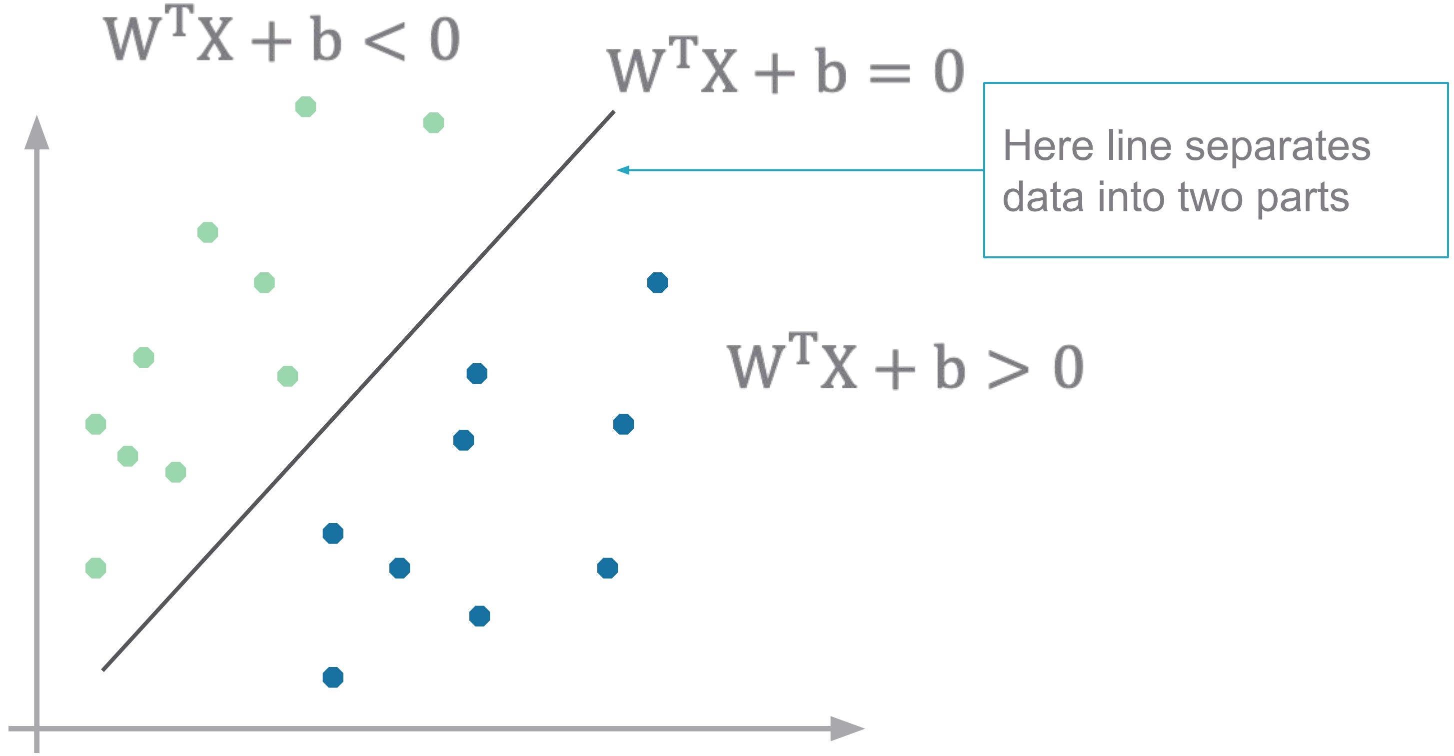
$$\boxed{\mathbf{W^TX + b = 0}}$$

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:

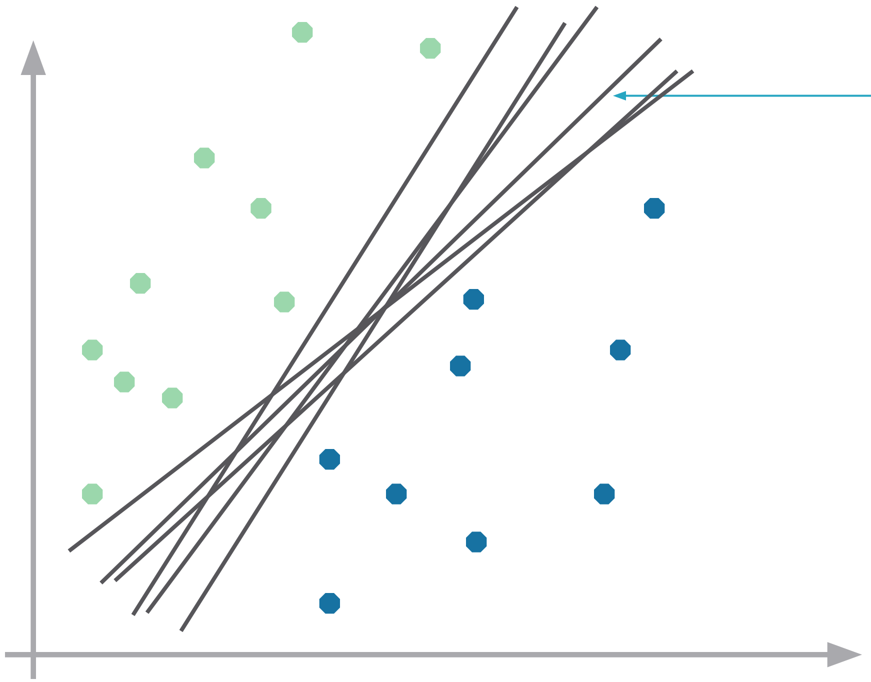
Fig. 01: Binary Classification



Linear Separators

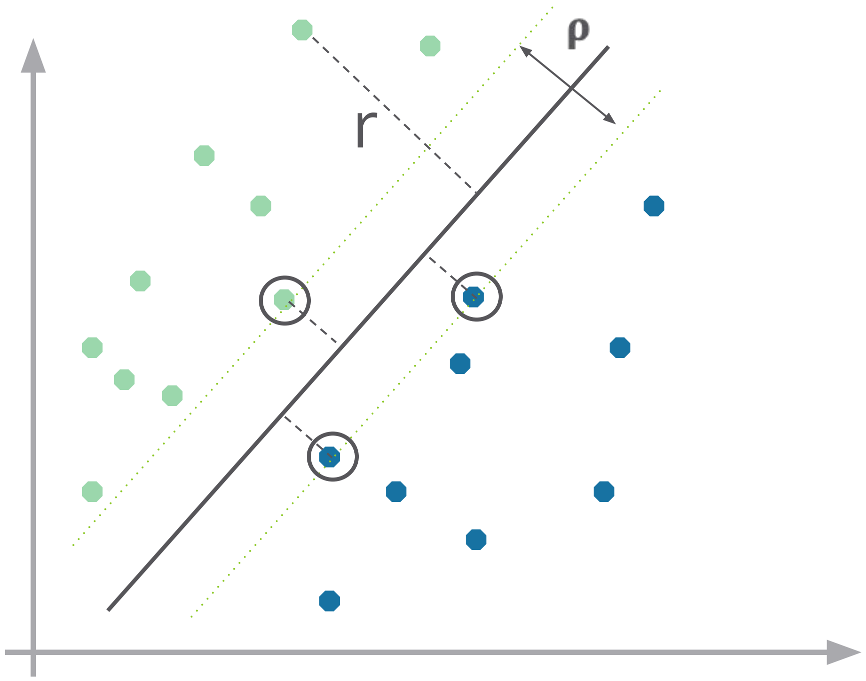
The objective in SVM is to find optimum separator

Fig. 02: Linear Separators



Which of the linear separators is optimal?

Classification Margin



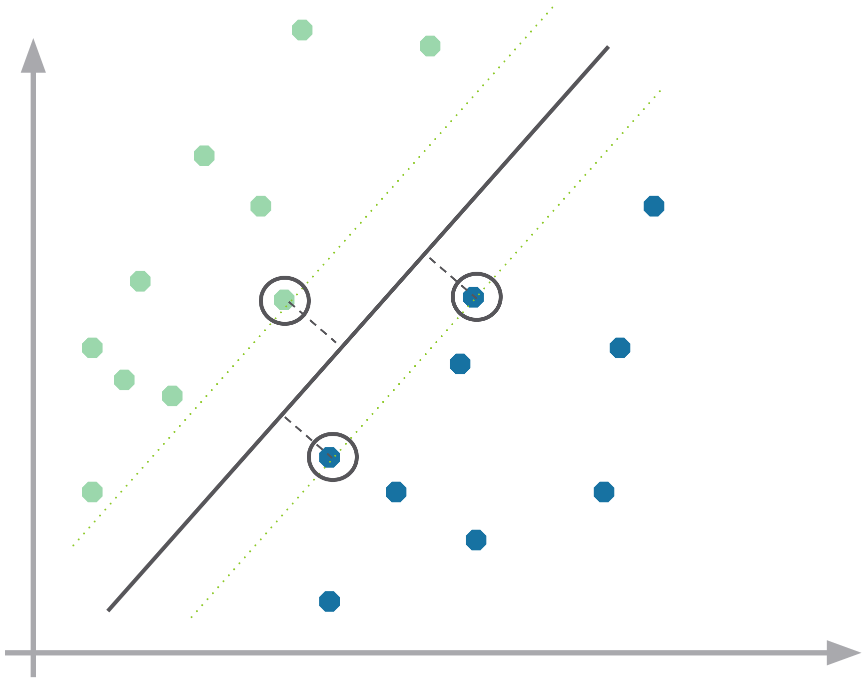
- Distance from case x_i to the separator is

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

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

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

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

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

and

$y_i(w^T x_i + b) \geq 1$

which can be reformulated as:

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

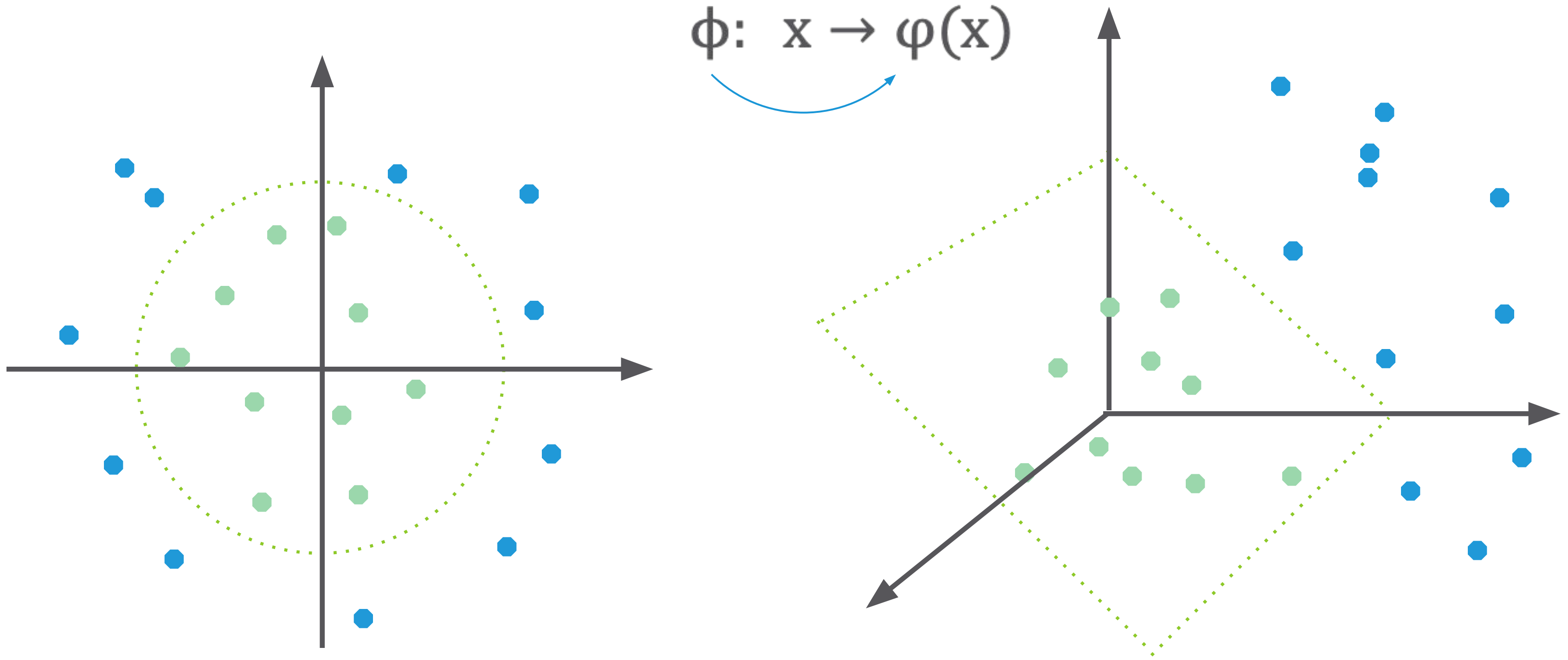
$\phi(w) = w^T w$ is minimised

and

$y_i(w^T x_i + b) \geq 1$

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 $\phi\text{:}\;\; x \rightarrow \varphi(x)$

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

BANK LOAN

Independent Variables

Dependent Variable

SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER

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
```

```
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.

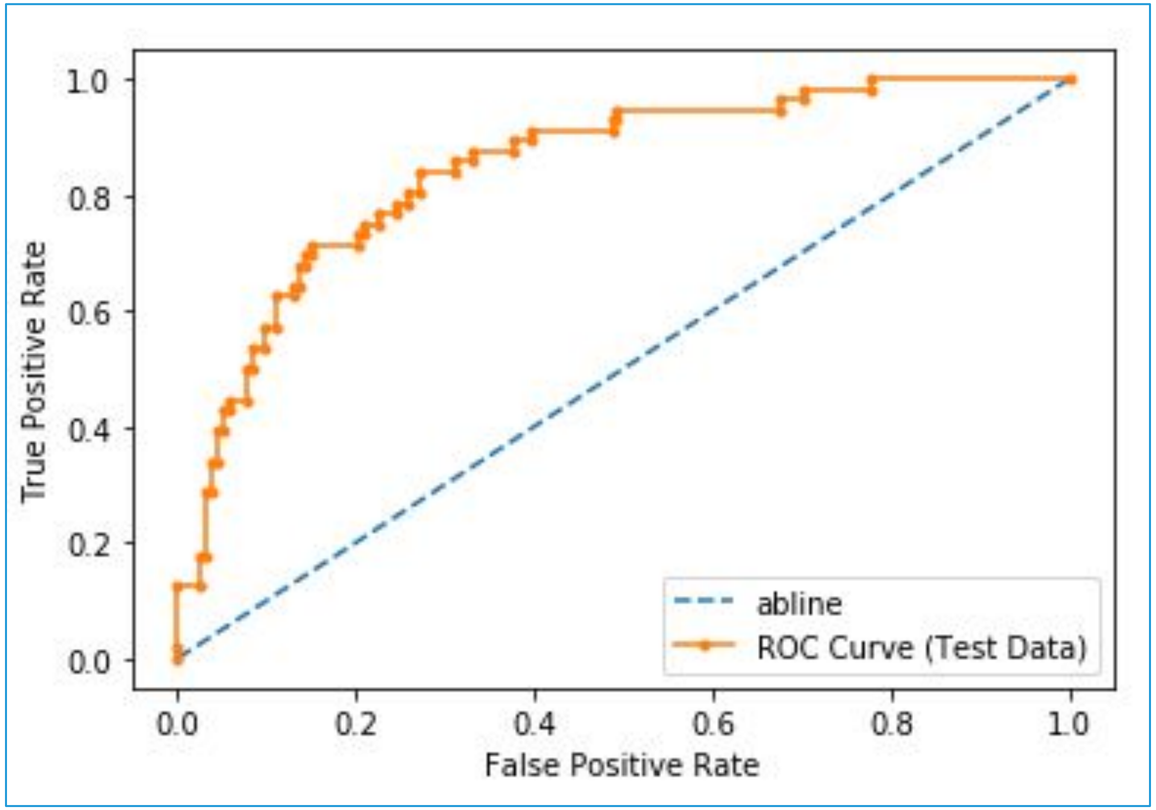
ROC Curve and Area Under ROC Curve

```
# 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