Introduction to

Binary Logistic Regression

using Python

Contents

1. Binary Logistic Regression in Python
2. Parameter Estimation and Hypothesis Testing
3. Classification table, Sensitivity & Specificity in Python
4. Classification Report : Precision & Recall values

Binary Logistic Regression

DEPENDENT VARIABLE

Categorical (Binary)

0 or 1

Death or Survival

Absence or Presence

⋮

INDEPENDENT VARIABLE

Categorical or Continuous

Gender

Geographical Regions

Age

Income

Debt Ratio

⋮

Binary logistic regression **models the dependent variable as a logit of p, where p is the probability that dependent variable takes the value 1 or 0**

Statistical Model – For k Predictors

$$\log\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + \cdots + b_kX_k$$

where,

p	:	Probability that Y=1 given X
Y	:	Dependent Variable
$X_1, X_2, \dots, X_k$	:	Independent Variables
$b_0, b_1, \dots, b_k$	:	Parameters of Model

Note that **LHS of the model can lie between - ∞ to ∞**

Parameters of the model are estimated by Maximum Likelihood Method

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

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

Binary Logistic Regression in Python

```
# Import data and check data structure before running model
```

```
import pandas as pd
bankloan=pd.read_csv('BANK LOAN.csv')

bankloan.info()
```

```
# Output:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 8 columns):
SN           700 non-null int64
AGE          700 non-null int64
EMPLOY       700 non-null int64
ADDRESS      700 non-null int64
DEBTINC      700 non-null float64
CREDDEBT     700 non-null float64
OTHDEBT      700 non-null float64
DEFAULTER    700 non-null int64
dtypes: float64(3), int64(5)
memory usage: 43.8 KB
```

Binary Logistic Regression in Python

```
# Change 'AGE' variable into categorical
bankloan['AGE']=bankloan['AGE'].astype('category')

bankloan.info()
```

Age is an integer and we need to convert it into type “category” for modeling purposes.

```
# Output:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 8 columns):
SN          700 non-null int64
AGE         700 non-null category
EMPLOY      700 non-null int64
ADDRESS     700 non-null int64
DEBTINC     700 non-null float64
CREDDEBT    700 non-null float64
OTHDEBT     700 non-null float64
DEFAULTER   700 non-null int64
dtypes: category(1), float64(3), int64(4)
memory usage: 39.1 KB
```

Binary Logistic Regression in Python

```
# Logistic Regression using logit function
import statsmodels.formula.api as smf

riskmodel = smf.logit(formula = 'DEFAULTER ~ AGE + EMPLOY + ADDRESS + DEBTINC + CREDDEBT + OTHDEBT', data = bankloan).fit()
```

- **logit()** fits a logistic regression model to the data.

```
# Model summary

riskmodel.summary()
```

summary() generates detailed summary of the model.

Logit Regression Results

Dep. Variable:	DEFAULTER	No. Observations:	700
Model:	Logit	Df Residuals:	692
Method:	MLE	Df Model:	7
Date:	Tue, 23 Mar 2021	Pseudo R-squ.:	0.3120
Time:	11:41:05	Log-Likelihood:	-276.70
converged:	True	LL-Null:	-402.18
Covariance Type:	nonrobust	LLR p-value:	1.733e-50

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.7882	0.264	-2.985	0.003	-1.306	-0.271
C(AGE)[T.2]	0.2520	0.267	0.946	0.344	-0.270	0.774
C(AGE)[T.3]	0.6271	0.361	1.739	0.082	-0.080	1.334
EMPLOY	-0.2617	0.032	-8.211	0.000	-0.324	-0.199
ADDRESS	-0.0996	0.022	-4.459	0.000	-0.143	-0.056
DEBTINC	0.0851	0.022	3.845	0.000	0.042	0.128
CREDDEBT	0.5634	0.089	6.347	0.000	0.389	0.737
OTHDEBT	0.0231	0.057	0.405	0.685	-0.089	0.135

Interpretation :

- Since p-value is <0.05 for Employ, Address, Debtinc, Creddebt, these independent variables are significant.

Re-run Model in Python

- Re-run the model with employ, address, debtinc, creddebt.

```
riskmodel = smf.logit(formula = 'DEFAULTER ~  EMPLOY + ADDRESS +
DEBTINC + CREDDEBT', data = bankloan).fit()

riskmodel.summary()
```

Re-run Model in Python

```
# Output:
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:              DEFAULTER   No. Observations:                  700
Model:                          Logit   Df Residuals:                      695
Method:                           MLE   Df Model:                            4
Date:                Tue, 23 Mar 2021   Pseudo R-squ.:                  0.3079
Time:                        11:36:38   Log-Likelihood:                -278.37
converged:                       True   LL-Null:                       -402.18
Covariance Type:            nonrobust   LLR p-value:                 2.114e-52
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.7911      0.252     -3.145      0.002      -1.284      -0.298
EMPLOY        -0.2426      0.028     -8.646      0.000      -0.298      -0.188
ADDRESS       -0.0812      0.020     -4.144      0.000      -0.120      -0.043
DEBTINC        0.0883      0.019      4.760      0.000       0.052       0.125
CREDDEBT       0.5729      0.087      6.566      0.000       0.402       0.744
==============================================================================
```

Interpretation :

- Since p-value is <0.05 for Employ, Address, Debtinc and Creddebt, these independent variables are significant.

Odds Ratios In Python

- Final Model is :

$$\log\left(\frac{p}{1-p}\right) = -0.79107 - 0.24258 * (\text{EMPLOY}) - 0.08122 * (\text{ADDRESS}) + 0.08827 * (\text{DEBTINC}) + 0.57290\ (\text{CREDDEBT})$$

- This model is used for predicting the probabilities.

```
import numpy as np
conf = riskmodel.conf_int()
conf['OR'] = riskmodel.params
conf.columns = ['2.5%', '97.5%', 'OR']
print(np.exp(conf))
```

- **conf_int():** calculates confidence intervals for parameters
- **riskmodel.params:** identify the model parameter estimates

Odds Ratios in Python

```
# Output:
```

	2.5%	97.5%	OR
Intercept	0.276905	0.742255	0.453359
EMPLOY	0.742617	0.828950	0.784597
ADDRESS	0.887246	0.958093	0.921989
DEBTINC	1.053295	1.132703	1.092278
CREDDEBT	1.494635	2.104150	1.773397

Interpretation :

- Note that, confidence interval for odds ratio does not include ‘1’ for all variables retained in the model. Which means that all of these variables are significant.
- The odds ratio for CREDDEBT is approximately 1.77
- For one unit change CREDDEBT, the odds of being a defaulter will change by 1.77 folds.

Predicting Probabilities in Python

```
# Predicting Probabilities
```

```
bankloan = bankloan.assign(pred=pd.Series(riskmodel.predict()))
bankloan.head(10)
```

- **predict()** function calculates predicted probabilities which are saved in the same dataset ‘bankloan’ under new column ‘pred’.

```
# Output:
```

	SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER	pred
0	1	3	17	12	9.3	11.36	5.01	1	0.808346726
1	2	1	10	6	17.3	1.36	4	0	0.198114704
2	3	2	15	14	5.5	0.86	2.17	0	0.010062815
3	4	3	15	14	2.9	2.66	0.82	0	0.022159721
4	5	1	2	0	17.3	1.79	3.06	1	0.781808095
5	6	3	5	5	10.2	0.39	2.16	0	0.21646839
6	7	2	20	9	30.6	3.83	16.67	0	0.185631512
7	8	3	12	11	3.6	0.13	1.24	0	0.014726159
8	9	1	3	4	24.4	1.36	3.28	1	0.748212503
9	10	2	0	13	19.7	2.78	2.15	0	0.815255803

Interpretation :

- Last column ‘pred’ gives predicted probabilities.

Classification Table

- Based on **cut-off value** of p, Y is estimated to be either 1 or 0

  Ex. $p>0.5$; $Y=1$

  $p\leq0.5$; $Y=0$

- **Cross tabulation** of observed values of Y and predicted values of Y is called as **Classification Table.**
- The predictive success of the logistic regression can be assessed by looking at the classification table, but classification table is not a good measure of goodness fit since it **varies with the cut off value set.**

- Accuracy Rate measures **how accurate a model is in predicting outcomes.**
- In the adjoining table, 479 times Y=0 was observed as well as predicted. Similarly, Y=1 was observed and predicted 92 times.

  Accuracy Rate = 478+92/700 = 81.43 %

  Here **misclassification rate is : (39 +91) / 700=18.57%**

		Expected	
		0	1
Observed	0	478	39
	1	91	92

Classification Table Terminology

Sensitivity	% of occurrences correctly predicted P(Ypred=1/Y=1)
Specificity	% of non occurrences correctly predicted P(Ypred=0/Y=0)
False Positive Rate (1 – Specificity)	% of non occurrences which are incorrectly predicted. P(Ypred=1/Y=0)
False Negative Rate (1- Sensitivity)	% of occurrences which are incorrectly predicted. P(Ypred=0/Y=1)

		Predicted	
		0	1
Observed	0	**Specificity**	**False Positive (1-Specificity)**
	1	**False Negative (1-Sensitivity)**	**Sensitivity**

Sensitivity and Specificity calculations

Cut-off Value		Accuracy	Sensitivity	Specificity
0.1	FALSE TRUE 0 245 272 1 12 171	(245+171)/700 = 59.43%	171/183=93.4%	245/517=47.4%
0.2	FALSE TRUE 0 349 168 1 26 157	(349+157)/700 = 72.29%	157/183=85.8%	349/517=67.5%
0.3	FALSE TRUE 0 415 102 1 45 138	(415+138)/700 = 84.71%	138/183=75.4%	415/517=80.3%
0.4	FALSE TRUE 0 447 70 1 69 114	(447+114)/700 = 80.14%	114/183=62.3%	447/517=86.5%
0.5	FALSE TRUE 0 478 39 1 91 92	(478+92)/700 =81. 43%	92/183=50.3%	478/517=92.5%

Classification table in Python

```
# Predicting Probabilities
from sklearn.metrics import confusion_matrix

predicted_values1 = riskmodel.predict()
threshold=0.5
predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1
cm1 = confusion_matrix(bankloan['DEFAULTER'],predict
print('Confusion Matrix : \n', cm1)
```

- **confusion_matrix** function creates a cross table of observed Y (defaulter)vs. predicted Y

```
# Output:
```

```
Confusion Matrix :
 [[478  39]
 [ 91  92]]
```

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

- This is how the python output of the confusion matrix appears .

Interpretation :

- There are 478 correctly predicted non-defaulters and 92 correctly predicted defaulters.
- There are 39 wrongly predicted as defaulters and 91 wrongly predicted as non-defaulters.

Sensitivity and Specificity in Python

```
# Sensitivity and Specificity
```

```
sensitivity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Sensitivity : ', sensitivity)

specificity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Specificity : ', specificity )
```

```
# Output:
```

```
Sensitivity :  0.5027322404371585
Specificity :  0.9245647969052224
```

Interpretation :

- The Sensitivity is at 50.27% and the Specificity is at 92.46%. Note that the threshold is set at 0.5

Precision & Recall

- **Precision :** Precision tells us what percentage of predicted positive cases are correctly predicted.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall or Sensitivity :** Recall tells us what percentage of actual positive cases are correctly predicted.

$$Recall = \frac{TP}{TP + FN}$$

Classification Report

```
#Classification Report
from sklearn.metrics import classification_report
print(classification_report(bankloan['DEFAULTER'],predicted_class1))
```

```
# Output:
```

	precision	recall	f1-score	support
0	0.84	0.92	0.88	517
1	0.70	0.50	0.59	183
accuracy			0.81	700
macro avg	0.77	0.71	0.73	700
weighted avg	0.80	0.81	0.80	700

classification_report() gives recall, precision and accuracy along with other measures.

Interpretation :

- Recall is 50% & Precision is 70%.
- Accuracy is 81%.

Quick Recap

In this session, we learned about **Binary Logistic Regression** :

Binary logistic regression	• Dependent variable is binary and independent variables are categorical or continuous or mix of both. • Regression line is sigmoid curve. • Parameters are estimated using MLE.
Classification table	• percentage of correctly predicted observations = accuracy. • Percentage of wrongly predicted observations = misclassification rate
Sensitivity/True Positive rate	• % of occurrences correctly predicted
Specificity/True Negative rate	• % of non occurrences correctly predicted
False Positive Rate	• % of non occurrences which are incorrectly predicted
False Negative Rate	• % of occurrences which are incorrectly predicted
Precision & Recall	• Precision tells us what percentage of predicted positive cases are correctly predicted. • Recall tells us what percentage of actual positive cases are correctly predicted.

Binary Logistic Regression

Checking Model Performance

Contents

1. Receiver Operating Characteristic (ROC) Curve
2. Lift Curve
3. Kolmogorov Smirnov statistics
4. Pearson residuals
5. Residual plot
6. Multicollinearity

Receiver Operating Characteristic Curve

- The **R**eceiver **O**perating **C**haracteristic (ROC) curve is

> A graphical representation of the trade off between the false positive and true positive rates for various cut off values

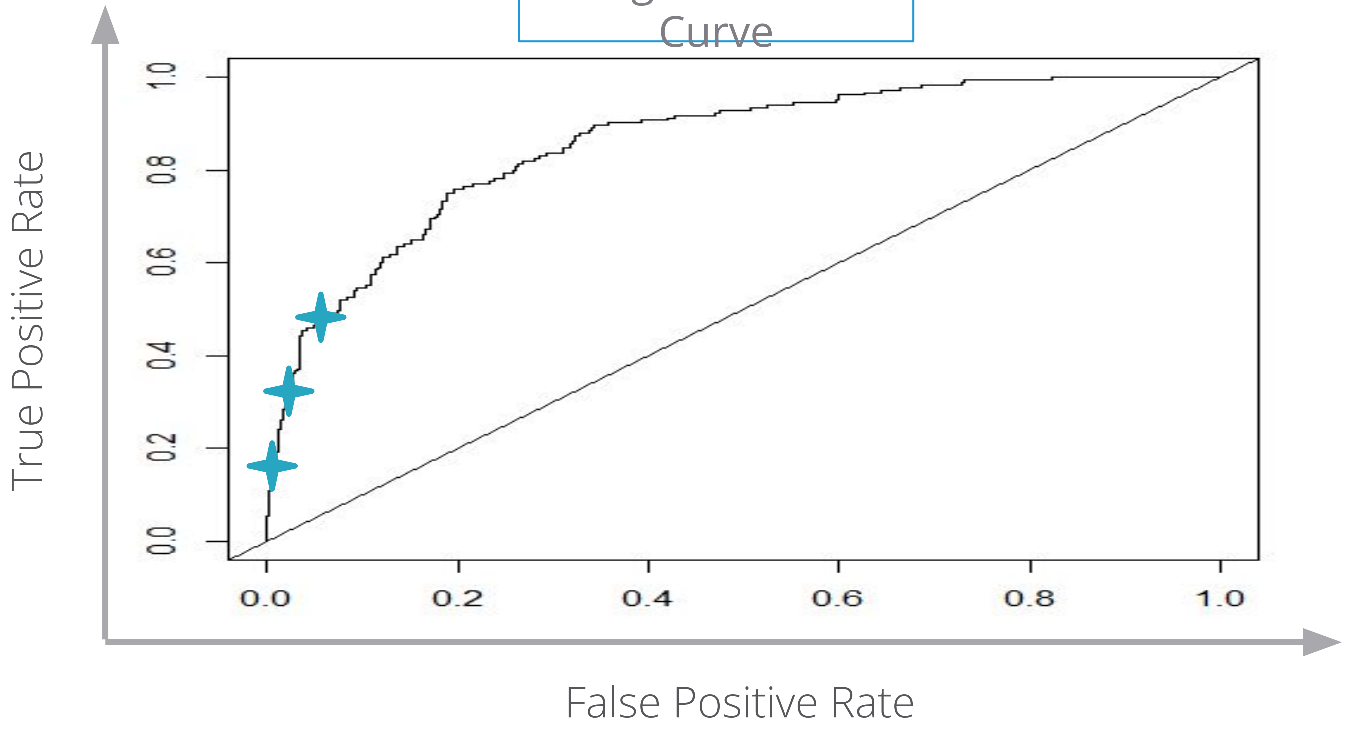
Y- axis: Sensitivity (true positive rate)

X-axis: 1-Specificity (false positive rate)

The performance of the classification model can be assessed by area under the ROC curve (C).

ROC Curve and Area Under ROC Curve

Fig. 01: ROC Curve



High TPR with low FPR is indicative of a good model. This will result in curve that is closer to the Y-axis and top left corner of the plot. It implies higher Area Under the ROC Curve.

ROC Curve and Area Under ROC Curve

Interpreting different versions of an ROC curve

Critical Points	Interpretations
TPR = 0 and FPR = 0	Model predicts every instance to be Non-event
TPR = 1 and FPR = 1	Model predicts every instance to be Event
TPR = 1 and FPR = 0	The Perfect Model

- If the model is perfect, AUC = 1
- If the model is guessing randomly, AUC = 0.5
- Thumb rule: Area Under ROC Curve > 0.65 is considered acceptable

ROC in Python

```
# Importing bank loan data & Fitting Binary Logistic Regression model
```

```
import pandas as pd
bankloan=pd.read_csv('BANK LOAN.csv’)

bankloan['AGE']=bankloan['AGE'].astype('category’)

import statsmodels.formula.api as smf
riskmodel = smf.logit(formula = 'DEFAULTER ~  EMPLOY + ADDRESS +
DEBTINC + CREDDEBT', data = bankloan).fit()

from sklearn.metrics import roc_curve, auc
bankloan=bankloan.assign(pred=riskmodel.predict())
fpr, tpr, thresholds = roc_curve(bankloan['DEFAULTER'],
bankloan['pred'])
```

- Import **roc_curve, auc** from sklearn.metrics
- **predict()** function prepares data required for ROC curve.
- **roc_curve()** computes Receiver operating characteristic (ROC), it returns "tpr“ (True positive rate), "fpr” (False positive rate) and threshold.

ROC in Python

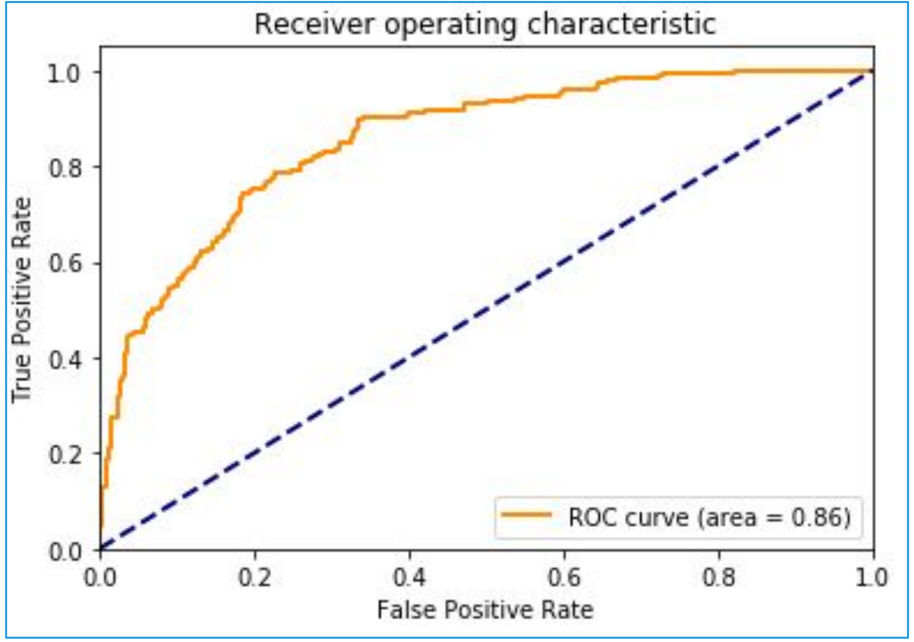
```
# AUC & ROC plot
ruc_auc = auc(fpr,tpr)

import matplotlib.pyplot as plt

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area =
%0.2f)' % ruc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, 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');plt.legend(loc="lower
right")
plt.show()
```

- **auc()** gives area under curve

- **plot()** function plots the objects created using roc_curve. Entire code of plot should be run in single chunk.

ROC in Python

Output:



AUC value

```
print("Area under the ROC curve : %f" % ruc_auc)
```

Output:

```
Area under the ROC curve : 0.855619
```

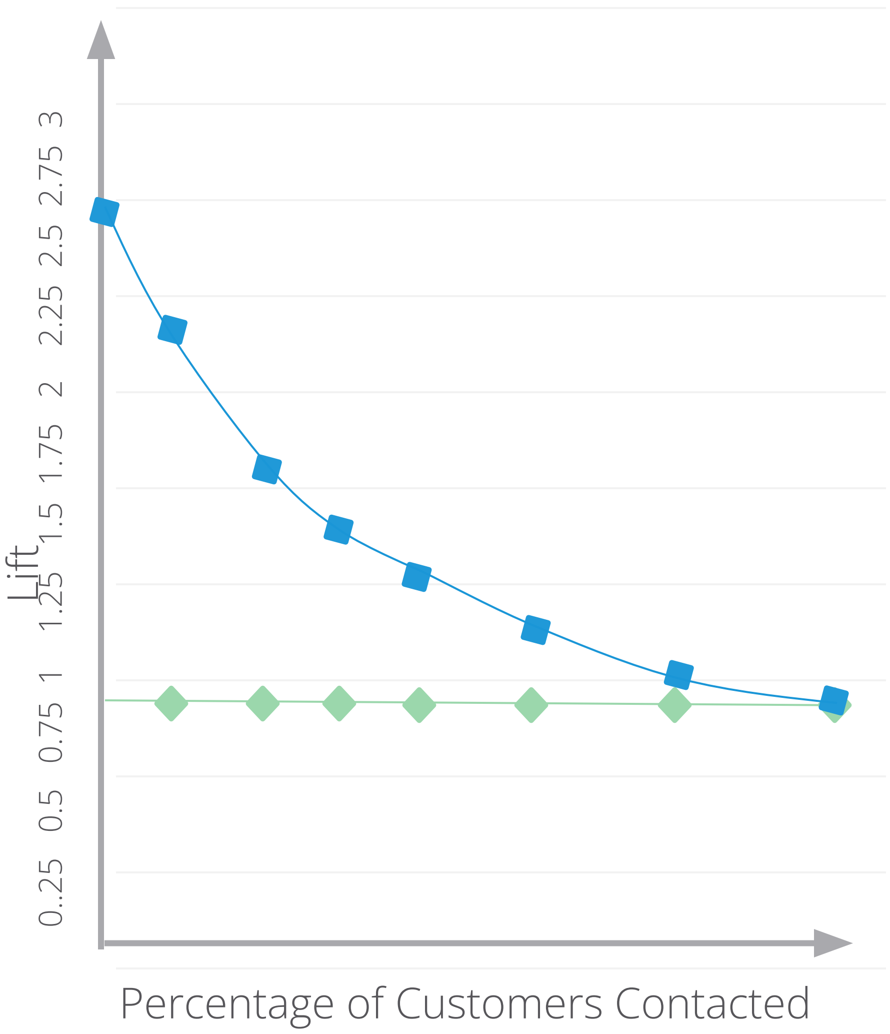
Interpretation :

- Area under the curve is 0.8556 which means model is performing well

Lift Curve

- The idea is to quantify and compare two scenarios-One is using the model to identify certain cases and second using random selection of cases for specific purpose like marketing campaign.
- Lift is the ratio of results obtained **with and without a model**.
- Although primarily used in marketing analytics, the concept finds applicability in other domains as well, such as risk modeling, supply chain analytics, etc.

Lift Curve



Lift Curve: After contacting X% of customers, Y% of respondents will be identified if statistical model is used.

Ratio Y/X is plotted

Baseline: After contacting X% of customers, X% of respondents will be identified if random method is used.

Ratio X/X is plotted

Lift Curve in Python

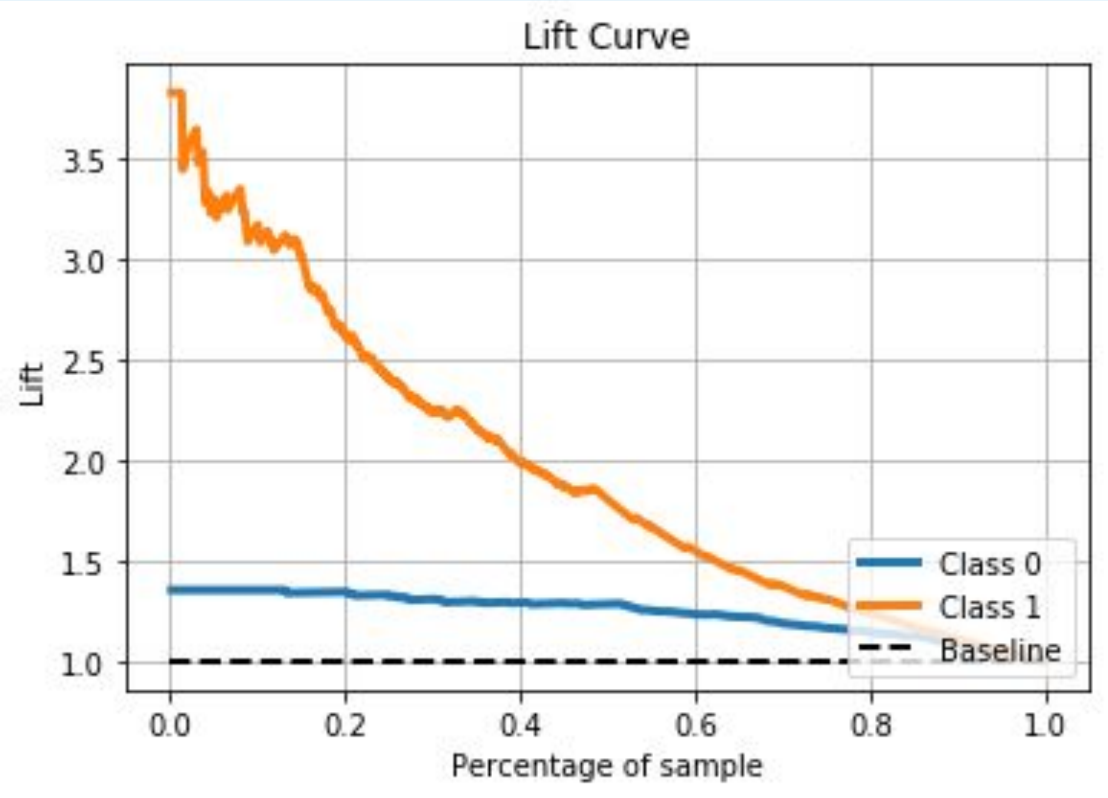
```
# Install "scikit-plot" library in Anaconda Prompt and load in Python
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import scikitplot as skplt

X = bankloan[['EMPLOY', 'ADDRESS', 'DEBTINC', 'CREDDEBT']]
y = bankloan[['DEFAULTER']]
log_model = LogisticRegression()
log_model.fit(X,y)
pred_log = log_model.predict_proba(X)

skplt.metrics.plot_lift_curve(y, pred_log)
plt.show()
```

- **LogisticRegression()** fits a Logistic Regression model
- **predict_proba()** Return probability estimates for the test vector X.
- **scikitplot()** depends on scikit-learn and … s plots for sklearn

* Note : Incase import scikitplot as skplt gives an error, then just run pip install scikit-plot in anaconda promt & then execute import code in python

Lift Curve in Python

```
# Output:
```



Interpretation :

- Model is performing better. As more defaulters identified in earlier buckets.

Kolmogorov-Smirnov Statistic

Kolmogorov-Smirnov (KS) Statistics is one of the most commonly used measures to assess predictive power for marketing or credit risk models

KS is maximum difference between % cumulative Goods and Bads distribution across probability bands

The gains table typically has % cumulative Goods (or Non-Event) and % Cumulative Bads (Or Event) across 10 or 20 probability bands

- **KS is a point estimate**, meaning it is only one value and indicate the probability band where separation between Goods (or Non-Event) and Bads (or Event) is maximum
- Theoretically **K-S can range from 0-100**. **KS less than 25, may not indicate good model**. Too high value should also be evaluated carefully

Kolmogorov-Smirnov Statistic

BAND	Count	Percent	Count(bad)	%(bad)	Count(good)	%(good)	cum% bad	cum% good	KS
0.95-1	10	1.4%	9	4.9%	1	0.2%	4.9%	0.2%	4.7%
0.90-0.95	7	1.0%	7	3.8%	0	0.0%	8.7%	0.2%	8.5%
0.85-0.90	7	1.0%	6	3.3%	1	0.2%	12.0%	0.4%	11.6%
0.80-0.85	7	1.0%	5	2.7%	2	0.4%	14.8%	0.8%	14.0%
0.75-0.80	11	1.6%	9	4.9%	2	0.4%	19.7%	1.2%	18.5%
0.70-0.75	17	2.4%	14	7.7%	3	0.6%	27.3%	1.7%	25.6%
0.65-0.70	17	2.4%	12	6.6%	5	1.0%	33.9%	2.7%	31.2%
0.60-0.65	10	1.4%	7	3.8%	3	0.6%	37.7%	3.3%	34.4%
0.55-0.6	24	3.4%	14	7.7%	10	1.9%	45.4%	5.2%	40.1%
0.5-0.55	21	3.0%	9	4.9%	12	2.3%	50.3%	7.5%	42.7%
0.45-0.5	22	3.1%	9	4.9%	13	2.5%	55.2%	10.1%	45.1%
0.40-0.45	31	4.4%	13	7.1%	18	3.5%	62.3%	13.5%	48.8%
0.35-0.4	29	4.1%	11	6.0%	18	3.5%	68.3%	17.0%	51.3%
0.3-0.35	27	3.9%	13	7.1%	14	2.7%	75.4%	19.7%	55.7%
0.25-0.3	40	5.7%	7	3.8%	33	6.4%	79.2%	26.1%	53.1%
0.2-0.25	45	6.4%	12	6.6%	33	6.4%	85.8%	32.5%	53.3%
0.15-0.2	52	7.4%	10	5.5%	42	8.1%	91.3%	40.6%	50.6%
0.10-0.15	66	9.4%	4	2.2%	62	12.0%	93.4%	52.6%	40.8%
0.05-0.1	80	11.4%	8	4.4%	72	13.9%	97.8%	66.5%	31.3%
0-0.05	177	25.3%	4	2.2%	173	33.5%	100.0%	100.0%	0.0%
Total	**700**	**100%**	183	**100%**	517	**100%**			

Kolmogorov-Smirnov Statistic in Python

```
# Combine observed and expected frequencies
```

```
from scipy.stats import ks_2samp
ks_2samp(bankloan.loc[bankloan.DEFAULTER==0,'pred'],
bankloan.loc[bankloan.DEFAULTER==1,'pred'])
```

- **ks_2samp** computes the kolmogorov-smirnov statistic on 2 samples.
- It returns KS statistic and two-tailed p-value

```
# Output:
```

```
Ks_2sampResult(statistic=0.561552039403452, pvalue=1.909421801103993e-37)
```

Interpretation :

- Maximum difference (K-S statistic) is 0.561552.

Pearson Residuals

- Pearson residual is defined as the standardized difference between the observed and predicted frequency.it measures the relative deviations between the observed and fitted values. :

$$\boxed{r_j = \frac{(Y_j - M_j p_j)}{\sqrt{M_j p_j(1 - p_j)}}}$$

where
Mj : number of observations with jth covariate pattern
Y_j : Observed value (1 or 0) for jth covariate pattern
p_j : Predicted probability for j^{th} covariate pattern

- Binary Logistic Regression **does not require 'Normality' of residuals**

Pearson Residuals in Python

```
# Obtain residuals
bankloan=bankloan.assign(resid=riskmodel.resid_pearson)

bankloan.head()
```

- **resid_pearson()** calculates Pearson residuals.

```
# Output:
```

	SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER	pred	resid
0	1	3	17	12	9.3	11.36	5.01	1	0.808346726	0.486921868
1	2	1	10	6	17.3	1.36	4	0	0.198114704	-0.497052463
2	3	2	15	14	5.5	0.86	2.17	0	0.010062815	-0.100822141
3	4	3	15	14	2.9	2.66	0.82	0	0.022159721	-0.150538706
4	5	1	2	0	17.3	1.79	3.06	1	0.781808095	0.528286162

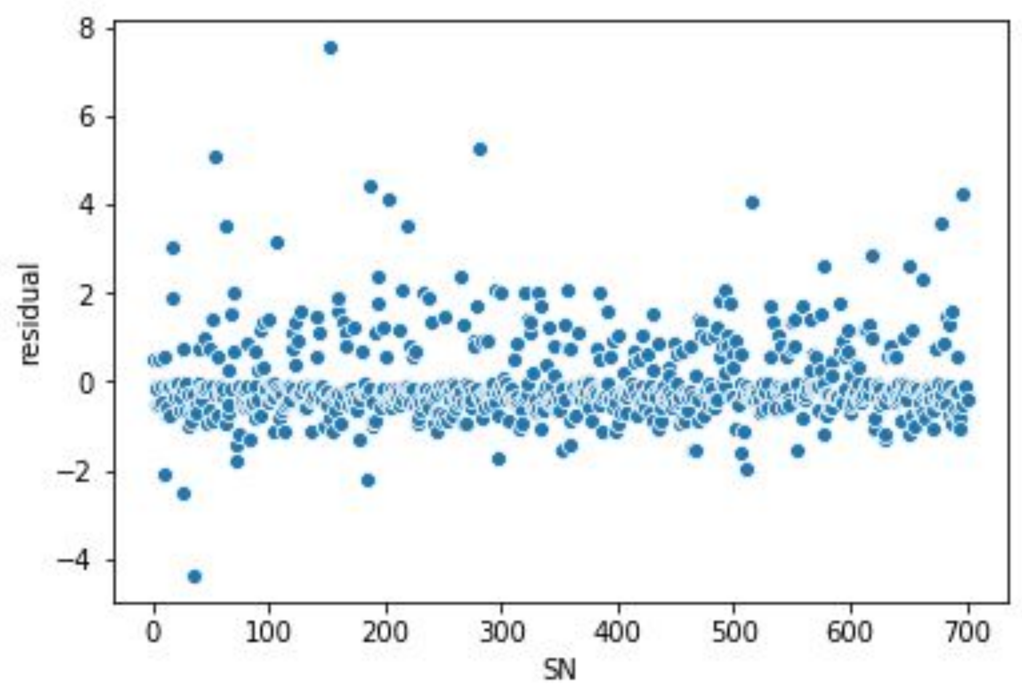
Residuals

Pearson Residuals Plot in Python

```
# Residuals Plot
```

```
import seaborn as sns
sns.scatterplot('SN','resid',data=bankloan); plt.xlabel('SN');
plt.ylabel('residual')
```

```
# Output:
```



Clearly one case has very high residual value.

Multicolinearity

- Multicollinearity exists if there is a strong linear relationship among the continuous independent variables.
- Do not ignore multicollinearity in Binary Logistic Regression .
- Use variance inflation factors to detect multicollieearity.

*** Multicolinearity is explained in MLR module.**

Quick Recap

In this session, we learnt about **checking model performance :**

ROC	• Graphical representation of the trade off between the false positive (FPR) and true positive (TPR) rates for various cut off values.
Lift curve	• Lift Curve Compares model results with baseline without model
K-S statistic	• KS is the maximum difference between % cumulative Goods (event/Y=1) and cumulative Bads (non events/Y=0) distribution across probability groups.
Residual	• Pearson's residual is used for binary logistic regression
Multicollinearity	• Multicollinearity exists if there is a strong linear relationship among the continuous independent variables

Binary Logistic Regression

Model Validation

Contents

1. Cross Validation
2. Hold out validation
3. Performance Measures : Accuracy, Recall, Precision
4. K-fold validation

Cross Validation in Predictive Modeling

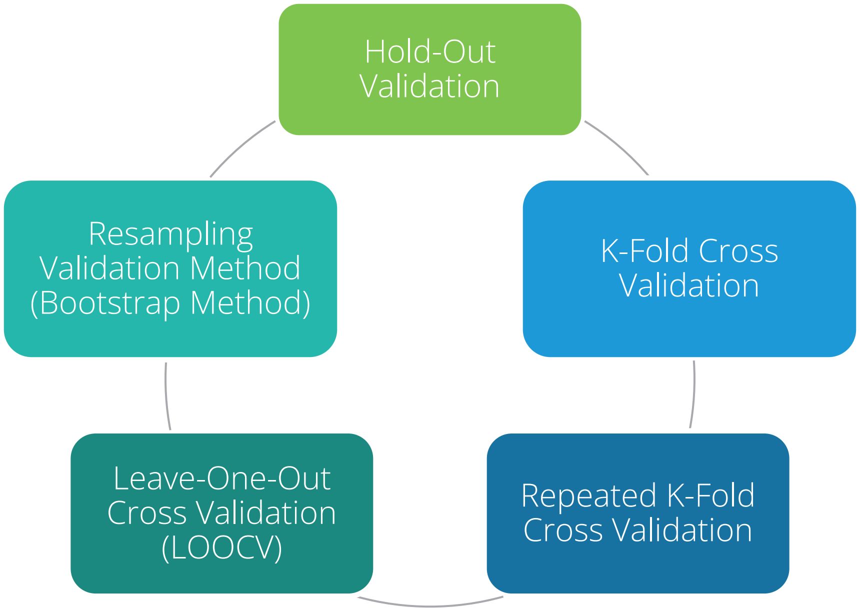
Cross Validation is a
process of evaluating the model on
'Out of Sample' data

- **Model performance measures** for binary logistic regression such as Accuracy rate, Sensitivity, Specificity **tend to be optimistic on 'In Sample Data'**
- More realistic measures of model performance are calculated using "Out of Sample' data
- Cross-validation is a procedure for estimating the generalization performance in this context

Cross validation is important because although a model is built on historical data, ultimately it is to be used on future data. However good the model, if it fails on out of sample data then it defeats the purpose of predictive modeling

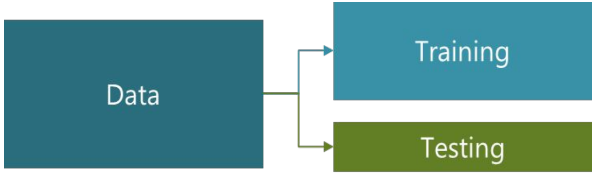
Cross Validation in Predictive Modeling

There are different approaches for cross validation. Five most significant of them are:



We will focus on **Hold Out** and **K-Fold** Cross validation methods.

Hold-Out Validation



In Hold-Out validation method, available data is split into two non-overlapped parts: 'Training Data' and 'Testing Data'

- The model is
  - Developed using training data
  - Evaluated using testing data
- Training data should have more sample size. Typically 70%-80% data is used for model development

* Here we continue to use previous data of bank loan for our further analysis.

Hold Out Validation in Python

```
# Create 2 groups of the data: Training and Testing
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statsmodels.formula.api as smf

bankloan=pd.read_csv('BANK LOAN.csv')

X_train, X_test = train_test_split(bankloan, test_size=0.3)
```

- Import **train_test_split** from sklearn.model_selection
- **train_test_split()** creates Training and Testing data sets
- **test_size=** is the percentage of data to be kept as test data

Hold Out Validation in Python

```
# Check the dimensions training and testing data
X_train.shape
```

```
# Output:
(490, 8)
```

```
X_test.shape
```

```
# Output:
(210, 8)
```

The data of 700 observations are partitioned into 2 parts:
With 490 observations in training (model development) data and remaining 210 observations in testing data (out of sample).

Hold Out Validation

- Model will be run on the training data and predicted probabilities will be generated.
- Same model will be applied to test data to get the predicted probabilities.
- Classification Report will be used to check the performance of the model in training and testing data.

Performance Measures : Accuracy, Precision, Recall

- Accuracy : Accuracy is defined as the ratio of correctly predicted cases by the total cases.

$$\boxed{Accuracy = \frac{TP+TN}{TP+FP+TN+FN}}$$

- Precision : Precision tells us what percentage of predicted positive cases are correctly predicted.

$$\boxed{Precision = \frac{TP}{TP+FP}}$$

- Recall or Sensitivity : Recall tells us what percentage of actual positive cases are correctly predicted.

$$\boxed{Recall = \frac{TP}{TP+FN}}$$

Performance Measures in Python

```
# Generate classification report for training data
```

```
riskmodel=smf.logit(formula = 'DEFAULTER ~  EMPLOY + ADDRESS +
DEBTINC + CREDDEBT', data = X_train).fit()

predicted_values1=riskmodel.predict()
threshold=0.3
predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1

from sklearn.metrics import classification_report
print(classification_report(X_train['DEFAULTER'],predicted_class1)
)
```

```
# Output:
```

```
              precision    recall  f1-score   support

           0       0.89      0.78      0.83       360
           1       0.55      0.75      0.63       130

    accuracy                           0.77       490
   macro avg       0.72      0.76      0.73       490
weighted avg       0.80      0.77      0.78       490
```

Performance Measures in Python

```
# Generate classification report for test data
predicted_values1=riskmodel.predict(X_test)
threshold=0.3
predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1

print(classification_report(X_test['DEFAULTER'],predicted_class1)
)
# Output:
```

	precision	recall	f1-score	support
0	0.84	0.77	0.81	150
1	0.53	0.63	0.58	60
accuracy			0.73	210
macro avg	0.68	0.70	0.69	210
weighted avg	0.75	0.73	0.74	210

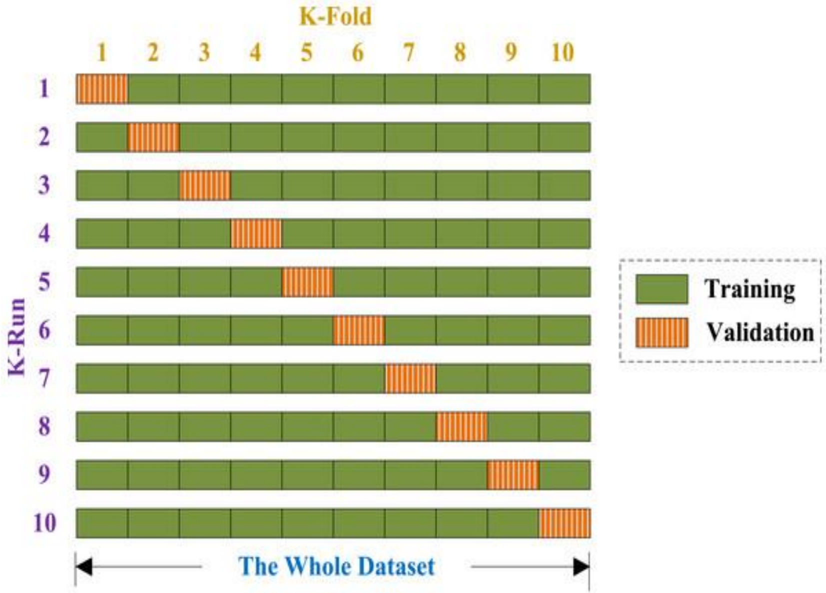
Interpretation :

- Accuracy & Sensitivity of test data is lower than that of train data. However, the values are still acceptable.

K fold Cross Validation

- In k-fold cross-validation the data is first partitioned into k equally (or nearly equally) sized segments or folds.

- Then k iterations of training and testing are performed such that each time one fold is kept aside for testing and model is developed using k-1 folds.



K-fold Validation in Python

```
# Create k-folds
```

```
from sklearn import linear_model
lmreg = linear_model.LogisticRegression()

y=bankloan.DEFAULTER
X=bankloan[['EMPLOY', 'ADDRESS', 'DEBTINC', 'CREDDEBT']]

from sklearn.model_selection import cross_val_predict
from sklearn.metrics.classification import cohen_kappa_score

predicted_prob = cross_val_predict(lmreg, X, y, cv=4,
method='predict_proba')
threshold=0.3
predicted = predicted_prob[:,1]
predicted_class1=np.zeros(predicted.shape)
predicted_class1[predicted>threshold]=1
```

- **cross_val_predict()** generates cross-validated estimates for each input data point.
- **method='predict_proba'** calculates probabilities for both classes.
- cv=4 specifies 4 folds

K-fold Validation in Python

```
# Generate classification report for k-fold validation
print(classification_report(y,predicted_class1))
```

```
# Output:
```

	precision	recall	f1-score	support
0	0.90	0.80	0.85	517
1	0.57	0.75	0.65	183
accuracy			0.79	700
macro avg	0.74	0.77	0.75	700
weighted avg	0.81	0.79	0.80	700

- **classification_report() :** gives accuracy, recall and precision values

Interpretation : accuracy of 0.79 and recall of 0.75 indicate that the model is performing good.

Quick Recap

In this session, we learnt about **Model Validation :**

Cross Validation	• Cross Validation is a process of evaluating the model on 'Out of Sample' data.
Hold out validation	• In Hold-Out validation method, available data is split into two non-overlapped parts: 'Training Data' and 'Testing Data'.
Performance Measures	• Performance measures like Accuracy, recall & precision are calculated to check model performance of train & test data. • `classification_report()` gives all these measures
K-fold validation	• In k-fold cross-validation the data is first partitioned into k equally (or nearly equally) sized segments or folds. • Then k iterations of training and testing are performed such that each time one fold is kept aside for testing and model is developed using k-1 folds.