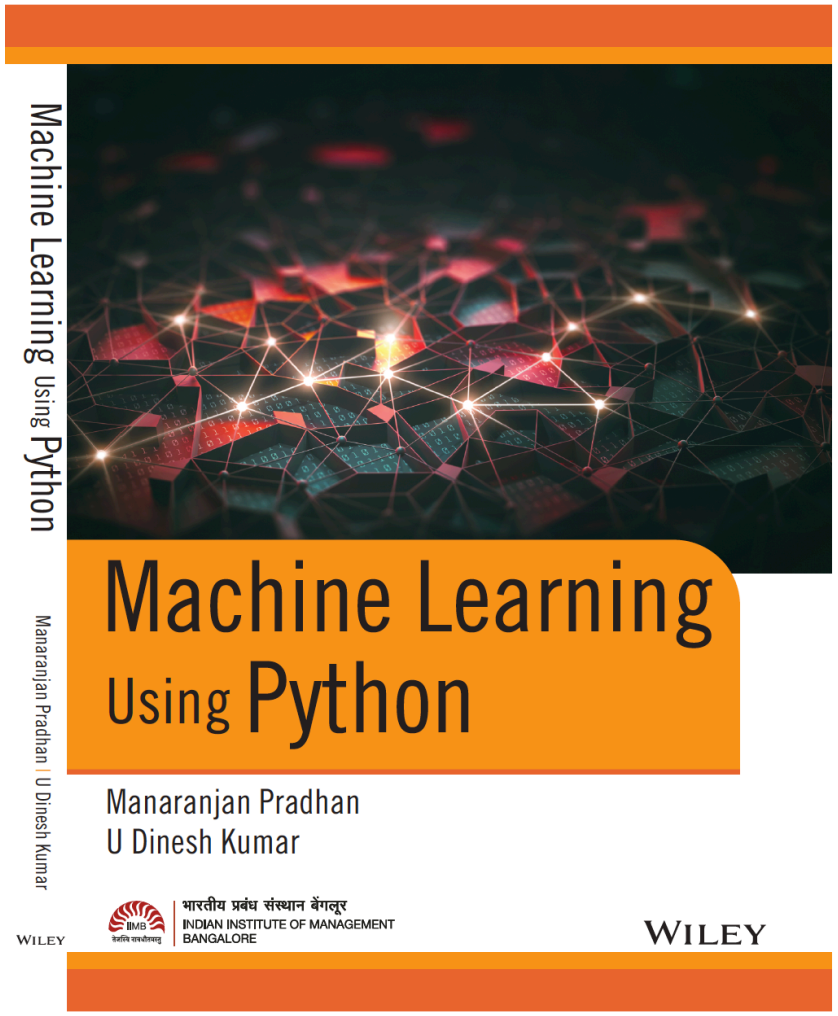

Chapter 05: Classification Problems

Prepared By: Purvi Tiwari

Learning Objectives

- Learning the concept of classification problems and their applications across different sectors.
- Learn to build logistic regression and decision tree models using the Python package *statsmodel* and *sklearn* APIs.
- Learn to measure model accuracies using confusion matrix, sensitivity, specificity, precision, receiver operating characteristic (ROC) curve, and area under ROC curve.
- Learn about statistical significance, model diagnostics and finding the optimal cut-off probabilities for classification.

Classification Overview

- Classification problems are an important category of problems where outcome variable takes discrete values.
- Primary objective is to predict the probability of an observation belonging to a class, known as class probability.
- Few examples of classification problems are:
    1. A bank would to classify the low-risk and high-risk customers.
    2. E-commerce providers would predict whether a customer is likely to churn or not.
    3. The HR department of a firm may want to predict if an applicant would accept an offer or not.

Classification Overview (Cntd.)

- Classification problems with binary outcomes are called binary classification.
- Classification problems with multiple outcomes are called multinomial classification.
- Techniques used for solving classification problems
    1. Logistic Regression
    2. Classification Trees
    3. Discriminant Analysis
    4. Neural Networks
    5. Support Vector Machines (SVM)

Binary Logistic Regression

- A statistical model in which the response variable takes only two values, 0 or 1.
- The explanatory variable can either be continuous or discrete.
- Assume that the outcomes are called positive (Y=1) and negative (Y=0). The probability that a record belongs to a positive class, using the binary logistic regression model is given by

$$P(Y=1)=\frac{e^{Z}}{1+e^{Z}}$$

$$Z=\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_3+\cdots+\beta_mX_m$$

Binary Logistic Regression (Cntd.)

- The logistic regression has an S-shaped curve.
- The class probability of an observation belonging to class labelled as 1, that is P(Y=1) can be written as

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = Z = \beta_0 + \beta_1 X_1 + \cdots + \beta_m X_m$$

- The left hand side of the above equation is known as logit function; the right-hand side is a linear function.
- Such models are called **generalized linear models** (GLM).
- The errors may not follow normal distribution.

Credit Classification

- Dataset: German credit rating dataset
- Available at University of California Irvin machine learning laboratory to predict whether a credit is a good or bad credit.
- Predicting the probability of default, when a customer applies for a loan in a bank.
- The data contains several attributes of the persons who availed the credit.

Credit Classification (Cntd.)

TABLE 5.1 Data description for German credit rating dataset

Variable	Variable Type	Description	Categories
checkin_acc	categorical	Status of existing checking account	• A11 : … < 0 DM • A12 : 0 <= … < 200 DM • A13 : … >= 200 DM / salary assignments for at least 1 year • A14 : no checking account
duration	numerical	Duration of the credit given in months	
credit_history	categorical	Credit History	• A30 : no credits taken/all credits paid back duly • A31 : all credits at this bank paid back duly • A32 : existing credits paid back duly till now • A33 : delay in paying off in the past • A34 : critical account/other credits existing (not at this bank)
amount	numerical	Amount of credit/loan	
savings_acc	categorical	Balance in savings account	• A61 : … <100 DM • A62 : 100 <= … <500 DM • A63 : 500 <= … < 1000 DM • A64 : .. >=1000 DM • A65 : unknown/no savings account
present_emp_since	categorical	Employment in years	• A71 : unemployed • A72 : … < 1 year • A73 : 1 <= … < 4 years • A74 : 4 <= … < 7 years • A75 : .. >= 7 years
inst_rate	numerical	Installment rate	
personal_status	categorical	Marital status	• A91 : male : divorced/separated • A92 : female : divorced/separated/married • A93: male : single • A94 : male : married/widowed • A95 : female : single
residing_since	numerical	Residing since in years	
age	numerical	Age in years	
inst_plans	categorical	Other installment plans of the applicant	• A141 : bank • A142 : stores • A143 : none
checkin_acc	categorical	Balance in checking account	• A11 : … < 0 DM • A12 : 0 <= … < 200 DM • A13 : … >= 200 DM/salary assignments for at least 1 year • A14 : no checking account
job	categorical	Job	• A171 : unemployed/unskilled - non-resident • A172 : unskilled - resident • A173 : skilled employee/official • A174 : management/self-employed/highly qualified employee/officer
status	categorical	Credit status	• 0: Good Credit • 1: Bad Credit

Credit Classification (Cntd.)

- Reading and displaying few records from the dataset

```
import pandas as pd
import numpy as np

credit_df = pd.read_csv( "German Credit Data.csv" )
```

```
credit_df.iloc[0:5,1:7]
```

	duration	credit_history	Amount	savings_acc	present_emp_since	inst_rate
0	6	A34	1169	A65	A75	4
1	48	A32	5951	A61	A73	2
2	12	A34	2096	A61	A74	2
3	42	A32	7882	A61	A74	2
4	24	A33	4870	A61	A73	3

```
credit_df.iloc[0:5,7:]
```

	personal_status	residing_since	age	inst_plans	num_credits	job	status
0	A93	4	67	A143	2	A173	0
1	A92	2	22	A143	1	A173	1
2	A93	3	49	A143	1	A172	0
3	A93	4	45	A143	1	A173	0
4	A93	4	53	A143	2	A173	1

Credit Classification (Cntd.)

- There are few columns which are categorical and have been inferred as objects

1. A11 : … < 0 DM
2. A12 : 0 <= … < 200 DM
3. A13 : … >= 200 DM/salary assignments for at least 1 year
4. A14 : no checking account

- Finding the number of observations in the dataset for good/ bad credit

```
credit_df.status.value_counts()
```

```
0  700
1  300
Name: status, dtype: int64
```

Credit Classification (Cntd.)

- Creating features and storing all independent variables

```
X_features = list( credit_df.columns )
X_features.remove( 'status' )
X_features
```

```
['checkin_acc',
 'duration',
 'credit_history',
 'amount',
 'savings_acc',
 'present_emp_since',
 'inst_rate',
 'personal_status',
 'residing_since',
 'age',
 'inst_plans',
 'num_credits',
 'job']
```

Credit Classification (Cntd.)

- Encoding Categorical Features

```
encoded_credit_df = pd.get_dummies(credit_df[X_features],
                                   drop_first = True)
```

```
['duration',
 'amount',
 'inst_rate',
 'residing_since',
 'age',
 'num_credits',
 'checkin_acc_A12',
 'checkin_acc_A13',
 'checkin_acc_A14',
 'credit_history_A31',
 'credit_history_A32',
 'credit_history_A33',
 'credit_history_A34',
 'savings_acc_A62',
 'savings_acc_A63',
 'savings_acc_A64',
 'savings_acc_A65',
 'present_emp_since_A72',
 'present_emp_since_A73',
 'present_emp_since_A74',
 'present_emp_since_A75',
 'personal_status_A92',
 'personal_status_A93',
 'personal_status_A94',
 'inst_plans_A142',
 'inst_plans_A143',
 'job_A172',
 'job_A173',
 'job_A174']
```

Credit Classification (Cntd.)

- For example, checkin_acc variable is encoded into three dummy variables and the base category.
  1. checkin_acc_A12
  2. checkin_acc_A13
  3. checkin_acc_A14

```
encoded_credit_df[['checkin_acc_A12',
                   'checkin_acc_A13',
                   'checkin_acc_A14']].head(5)
```

	checkin_acc_A12	checkin_acc_A13	checkin_acc_A14
0	0	0	0
1	1	0	0
2	0	0	1
3	0	0	0
4	0	0	0

Credit Classification (Cntd.)

- Building logistic regression model
  1. Set X (features) and Y (outcome) variables.
  2. Add a new column and set its value to 1, to estimate the intercept.
  3. Regression parameters are estimated using maximum likelihood estimation.

```
import statsmodels.api as sm

Y = credit_df.status
X = sm.add_constant(encoded_credit_df)
```

Credit Classification (Cntd.)

- Splitting Dataset into Training and Test Sets

```
from sklearn.model_selection import
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 Y,
                                                 test_size = 0.3,
                                                 random_state = 42)
```

- Building Logistic Regression Model

```
import statsmodels.api as sm

logit = sm.Logit(y_train, X_train)
logit_model = logit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.488938
         Iterations 6
```

Credit Classification (Cntd.)

- Printing Model Summary

```
logit_model.summary2()
```

Model:	Logit	Pseudo R-squared:	0.198
Dependent Variable:	status	AIC:	744.5132
Date:	2018-05-06 18:46	BIC:	881.0456
No. Observations:	700	Log-Likelihood:	–342.26
Df Model:	29	LL-Null:	–426.75
Df Residuals:	670	LLR p-value:	1.0630e-21
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

Credit Classification (Cntd.)

	Coef.	Std.Err.	z	P > |z|	[0.025	0.975]
const	−0.1511	1.1349	−0.1331	0.8941	−2.3754	2.0733
duration	0.0206	0.0104	1.9927	0.0463	0.0003	0.0409
amount	0.0001	0.0000	2.3765	0.0175	0.0000	0.0002
inst_rate	0.3064	0.0986	3.1083	0.0019	0.1132	0.4996
residing_since	0.0967	0.0920	1.0511	0.2932	−0.0836	0.2771
age	−0.0227	0.0103	−2.2131	0.0269	−0.0428	−0.0026
num_credits	0.2854	0.2139	1.3342	0.1821	−0.1338	0.7045
checkin_acc_A12	−0.4126	0.2391	−1.7260	0.0843	−0.8812	0.0559
checkin_acc_A13	−0.9053	0.4338	−2.0868	0.0369	−1.7556	−0.0550
checkin_acc_A14	−1.6052	0.2586	−6.2073	0.0000	−2.1120	−1.0983
credit_history_A31	0.1532	0.5795	0.2643	0.7916	−0.9827	1.2890
credit_history_A32	−0.4960	0.4411	−1.1245	0.2608	−1.3604	0.3685
credit_history_A33	−0.8881	0.5022	−1.7683	0.0770	−1.8724	0.0962
credit_history_A34	−1.4124	0.4528	−3.1190	0.0018	−2.2999	−0.5249
savings_acc_A62	−0.0496	0.3208	−0.1545	0.8772	−0.6782	0.5791
savings_acc_A63	−0.6640	0.4818	−1.3779	0.1682	−1.6084	0.2804
savings_acc_A64	−1.1099	0.6019	−1.8439	0.0652	−2.2896	0.0699
savings_acc_A65	−0.6061	0.2745	−2.2080	0.0272	−1.1441	−0.0681
present_emp_since_A72	0.0855	0.4722	0.1810	0.8564	−0.8401	1.0110
present_emp_since_A73	−0.0339	0.4492	−0.0754	0.9399	−0.9142	0.8465
present_emp_since_A74	−0.3789	0.4790	−0.7910	0.4289	−1.3178	0.5600
present_emp_since_A75	−0.2605	0.4554	−0.5721	0.5673	−1.1532	0.6321
personal_status_A92	−0.0069	0.4841	−0.0142	0.9887	−0.9557	0.9419
personal_status_A93	−0.4426	0.4764	−0.9291	0.3528	−1.3762	0.4911
personal_status_A94	−0.3080	0.5554	−0.5546	0.5792	−1.3967	0.7806
inst_plans_A142	−0.2976	0.5157	−0.5772	0.5638	−1.3084	0.7131
inst_plans_A143	−0.4458	0.2771	−1.6086	0.1077	−0.9889	0.0974
job_A172	−0.0955	0.7681	−0.1243	0.9011	−1.6009	1.4100
job_A173	−0.0198	0.7378	−0.0269	0.9786	−1.4658	1.4262
job_A174	−0.0428	0.7371	−0.0581	0.9537	−1.4876	1.4019

Credit Classification (Cntd.)

- Model Diagnostics

1. Wald's test (Chi-square test) for checking the statistical significance of individual predictor variables.
2. Likelihood ration test for checking the statistical significance of the overall model. It is used for variable (feature) selection.
3. Pseudo R^2 : It is a measure of goodness of the model. It does not have the same interpretation of R^2 as in the MLR model.

Credit Classification (Cntd.)

- Defining the method to get significant variables

```
def get_significant_vars( lm ):
    #Store the p-values and corresponding column names in a dataframe
    var_p_vals_df = pd.DataFrame( lm.pvalues )
    var_p_vals_df['vars'] = var_p_vals_df.index
    var_p_vals_df.columns = ['pvals', 'vars']
    # Filter the column names where p-value is less than 0.05
    return list( var_p_vals_df[var_p_vals_df.pvals <= 0.05]['vars'] )
```

```
significant_vars = get_significant_vars( logit_model )

significant_vars
```

```
['duration',
 'amount',
 'inst_rate',
 'age',
 'checkin_acc_A13',
 'checkin_acc_A14',
 'credit_history_A34',
 'savings_acc_A65']
```

Credit Classification (Cntd.)

- Building Logistic regression model using only significant variables

```
final_logit = sm.Logit( y_train,
              sm.add_constant( X_train [significant_vars] ) ).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.511350
         Iterations 6
```

```
final_logit.summary2()
```

Model:	Logit	Pseudo R-squared:	0.161
Dependent Variable:	status	AIC:	733.8898
Date:	2018-05-06 18:50	BIC:	774.8495
No. Observations:	700	Log-Likelihood:	–357.94
Df Model:	8	LL-Null:	–426.75
Df Residuals:	691	LLR p-value:	7.4185e-26
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

Credit Classification (Cntd.)

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	−0.8969	0.4364	−2.0551	0.0399	−1.7523	−0.0415
duration	0.0197	0.0098	2.0033	0.0451	0.0004	0.0390
amount	0.0001	0.0000	2.3205	0.0203	0.0000	0.0002
inst_rate	0.2811	0.0929	3.0264	0.0025	0.0991	0.4632
age	−0.0216	0.0089	−2.4207	0.0155	−0.0392	−0.0041
checkin_acc_A13	−0.8038	0.4081	−1.9697	0.0489	−1.6037	−0.0040
checkin_acc_A14	−1.5452	0.2187	−7.0649	0.0000	−1.9738	−1.1165
credit_history_A34	−0.8781	0.2319	−3.7858	0.0002	−1.3327	−0.4235
savings_acc_A65	−0.5448	0.2581	−2.1108	0.0348	−1.0507	−0.0389

Credit Classification (Cntd.)

- Observations from the model output

1. The negative sign in coefficient value indicates that as the value this variable increases, the probability of being a bad credit decreases.
2. The log of odds ratio or probability of being a bad credit increases as duration, amount, inst_rate increases.
3. The probability of being a bad credit decreases as age increases. This means older people tend to pay back their credits ontime compared to younger people.

Credit Classification (Cntd.)

- Predicting on Test Data

```
y_pred_df=pd.DataFrame({"actual": y_test,
                        "predicted_prob": final_logit.predict(
                        sm.add_constant( X_test[significant_vars]))})
```

```
y_pred_df.sample(10, random_state = 42)
```

	actual	predicted_prob
557	1	0.080493
798	0	0.076653
977	0	0.345979
136	0	0.249919
575	0	0.062264
544	0	0.040768
332	1	0.833093
917	1	0.370667
678	0	0.388392
363	0	0.088952

Credit Classification (Cntd.)

- Iterating through predicted probability of each observation and tagging as bad credit (1) if probability value is more than 0.5 or as good credit (0) otherwise.

```
y_pred_df['predicted'] = y_pred_df.predicted_prob.map(
                         lambda x: 1 if x > 0.5 else 0)

y_pred_df.sample(10, random_state = 42)
```

	actual	predicted_prob	predicted
557	1	0.080493	0
798	0	0.076653	0
977	0	0.345979	0
136	0	0.249919	0
575	0	0.062264	0
544	0	0.040768	0
332	1	0.833093	1
917	1	0.370667	0
678	0	0.388392	0
363	0	0.088952	0

Credit Classification (Cntd.)

- Creating a Confusion Matrix

A matrix formed by checking the actual values and predicted values of observations in the dataset.

```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
```

```
from sklearn import metrics
def draw_cm( actual, predicted ):
    ## Cret
    cm = metrics.confusion_matrix( actual, predicted, [1,0] )
    sn.heatmap(cm, annot=True,    fmt='.2f',
               xticklabels = ["Bad credit", "Good Credit"],
               yticklabels = ["Bad credit", "Good Credit"] )
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

Credit Classification (Cntd.)

```
draw_cm( y_pred_df.actual,
         y_pred_df.predicted )
```

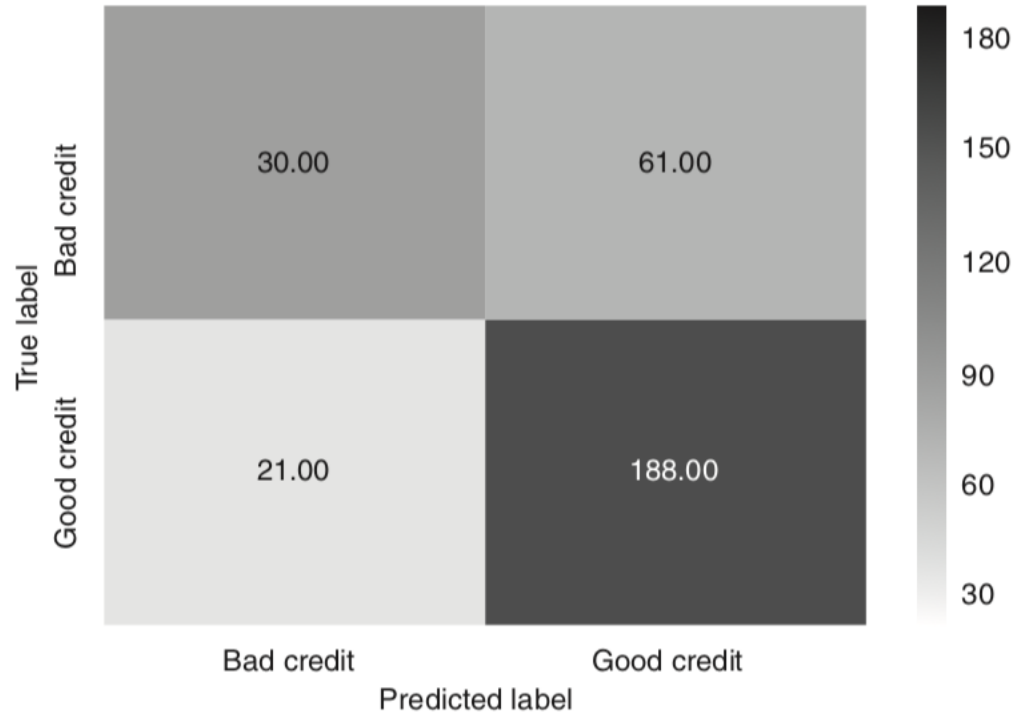

FIGURE 5.2 Confusion matrix with cut-off probability of 0.5.

Credit Classification (Cntd.)

- Observations from confusion matrix

1. Left-top quadrant represents actual bad credit and is correctly classified as bad credit. This is called True Positive (TP).
2. Left-down quadrant represents actual good credit and is incorrectly classified as bad credit. This called False Positive (FP).
3. Right-top quadrant represents actual bad credit and is incorrectly classified as good credit. This is called False Negative (FN).
4. Right-down quadrant represents actual good credit and is correctly classified as good credit. This is called True Negative (TN).

Credit Classification (Cntd.)

- Measuring Accuracies

1. Sensitivity or Recall – the conditional probability that the predicted class is positive given that the actual class is positive.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

2. Specificity – the conditional probability that the predicted class is negative given that the actual class is negative

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Credit Classification (Cntd.)

3. Precision – the conditional probability that the actual value is positive given that the prediction by the model is positive.

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

4. F-Score – a measure that combines precision and recall.

$$\text{F-Score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

Credit Classification (Cntd.)

- Calculating measures in Python

```
print( metrics.classification_report( y_pred_df.actual,
                                      y_pred_df.predicted ) )
```

	precision	recall	f1-score	support
0	0.76	0.90	0.82	209
1	0.59	0.33	0.42	91
avg/total	0.70	0.73	0.70	300

- The model is very good at identifying the good credits, but not very good at identifying bad credits.
- This the result for cut-off probability of 0.5%. This can be improved by choosing the right cut-off probability.

Credit Classification (Cntd.)

- Distributions of predicted probabilities for bad/ good credit

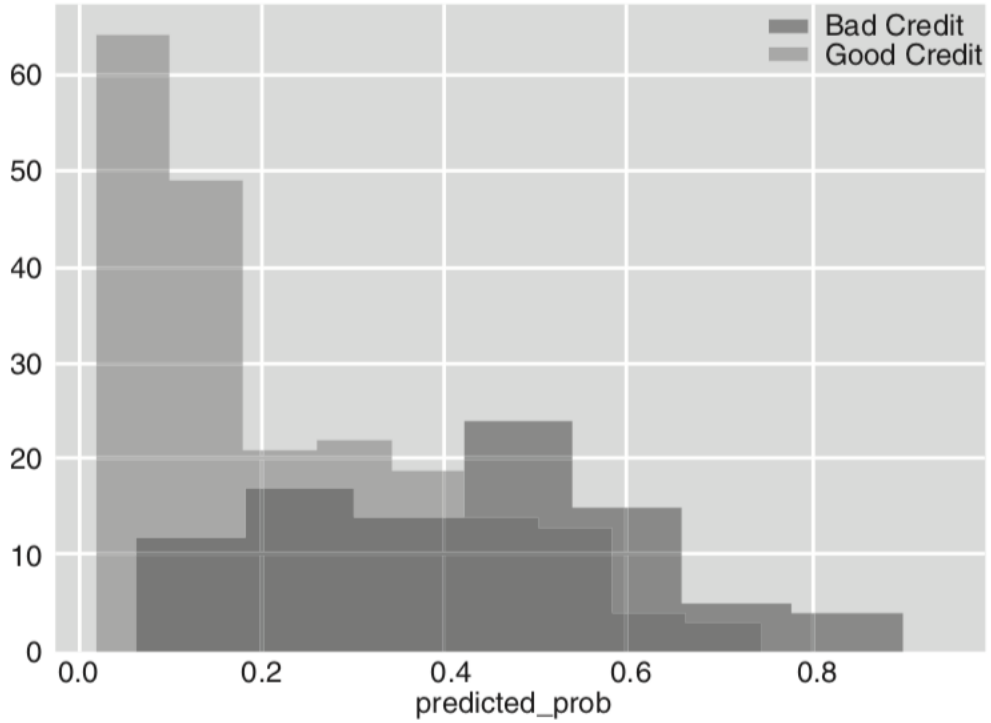

FIGURE 5.3 Distribution of predicted probability values by the model for both good and bad credits.

Credit Classification (Cntd.)

- Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC)

1. ROC curve can be used to understand the overall performance of a logistic regression model and used for model selection.
2. ROC curve is a plot between sensitivity on the vertical axis and 1-specificity on the horizontal axis.

Credit Classification (Cntd.)

- Creating the ROC curve

```
def draw_roc( actual, probs ):
    # Obtain fpr, tpr, thresholds
    fpr,
    tpr,
    thresholds = metrics.roc_curve( actual,
                                    probs,
                                    drop_intermediate = False )
auc_score = metrics.roc_auc_score( actual, probs )
plt.figure(figsize=(8, 6))
# Plot the fpr and tpr values for different threshold values
plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
# draw a diagonal line connecting the origin and top right most point
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
# Setting x and y labels
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()

return fpr, tpr, thresholds
```

```
fpr, tpr, thresholds = draw_roc( y_pred_df.actual,
                                 y_pred_df.predicted_prob)
```

Credit Classification (Cntd.)

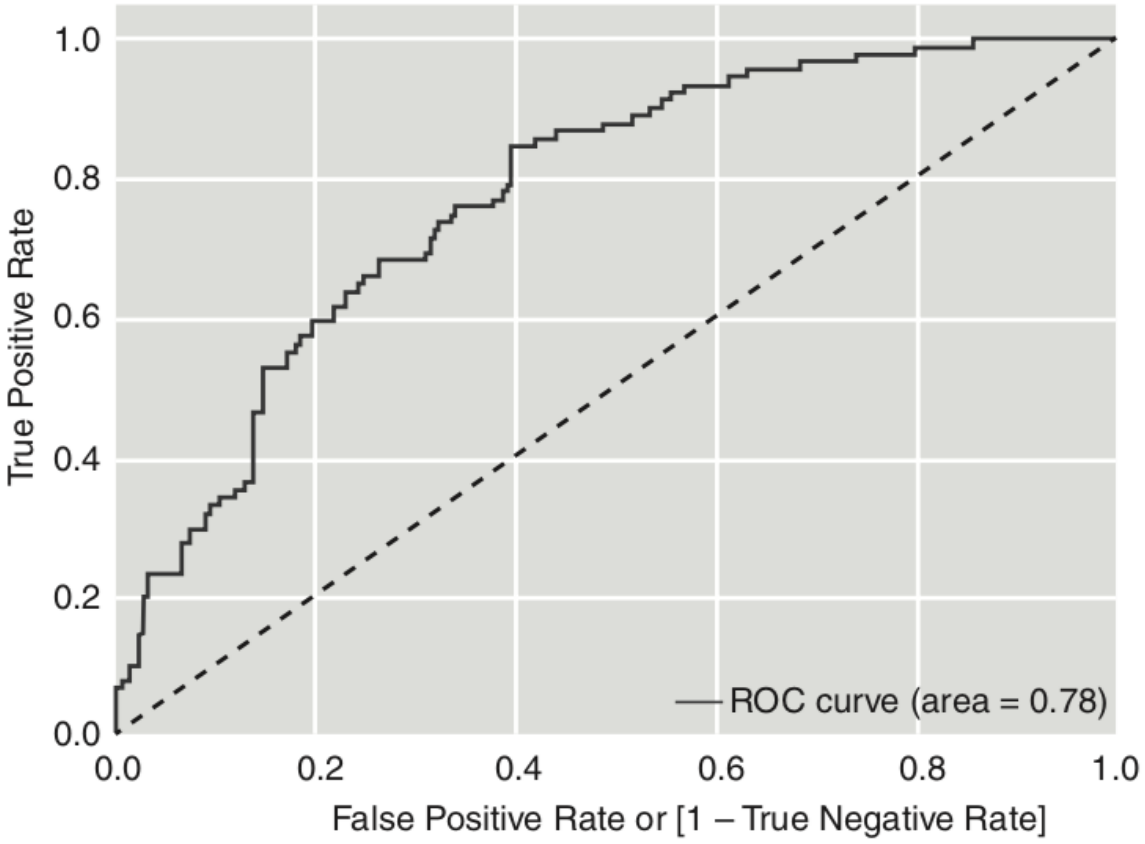

FIGURE 5.4 ROC curve.

Credit Classification (Cntd.)

```
auc_score = metrics.roc_auc_score( y_pred_df.actual,
                                   y_pred_df.predicted_prob )
round( float( auc_score ), 2 )
```

```
0.78
```

- Model with higher AUC is preferred and AUC is frequently used for model selection.
- As a thumb rule. AUC of at least 0.7 is required for practical application of the model.

Credit Classification (Cntd.)

- Finding Optimal Classification Cut-off

1. Youden's Index
   - A classification cut-off probability for which the following function is maximized (aka J-statistic)
   - Select the cut-off probability for which (sensitivity + specificity – 1) is maximum.

$$\text{Youden's Index} = \text{J-Statistic} = \max_{p} \left[\text{Sensitivity}\,(p) + \text{Specificity}\,(p) - 1\right]$$

Credit Classification (Cntd.)

```
tpr_fpr = pd.DataFrame( { 'tpr': tpr,
                          'fpr': fpr,
                          'thresholds': thresholds } )

tpr_fpr['diff'] = tpr_fpr.tpr - tpr_fpr.fpr
tpr_fpr.sort_values( 'diff', ascending = False )[0:5]
```

	fpr	thresholds	tpr	diff
159	0.397129	0.221534	0.846154	0.449025
160	0.401914	0.216531	0.846154	0.444240
161	0.406699	0.215591	0.846154	0.439455
158	0.397129	0.223980	0.835165	0.438036
165	0.421053	0.207107	0.857143	0.436090

Credit Classification (Cntd.)

```
y_pred_df['predicted_new'] = y_pred_df.predicted_prob.map(
                             lambda x: 1 if x > 0.22 else 0)
```

```
draw_cm( y_pred_df.actual,
         y_pred_df.predicted_new)
```

Now print the report with cut-off probability of 0.22.

```
print(metrics.classification_report( y_pred_df.actual,
                                     y_pred_df.predicted_new ))
```

	precision	recall	f1-score	support
0	0.90	0.60	0.72	209
1	0.48	0.85	0.61	91
avg/total	0.77	0.68	0.69	300

Credit Classification (Cntd.)

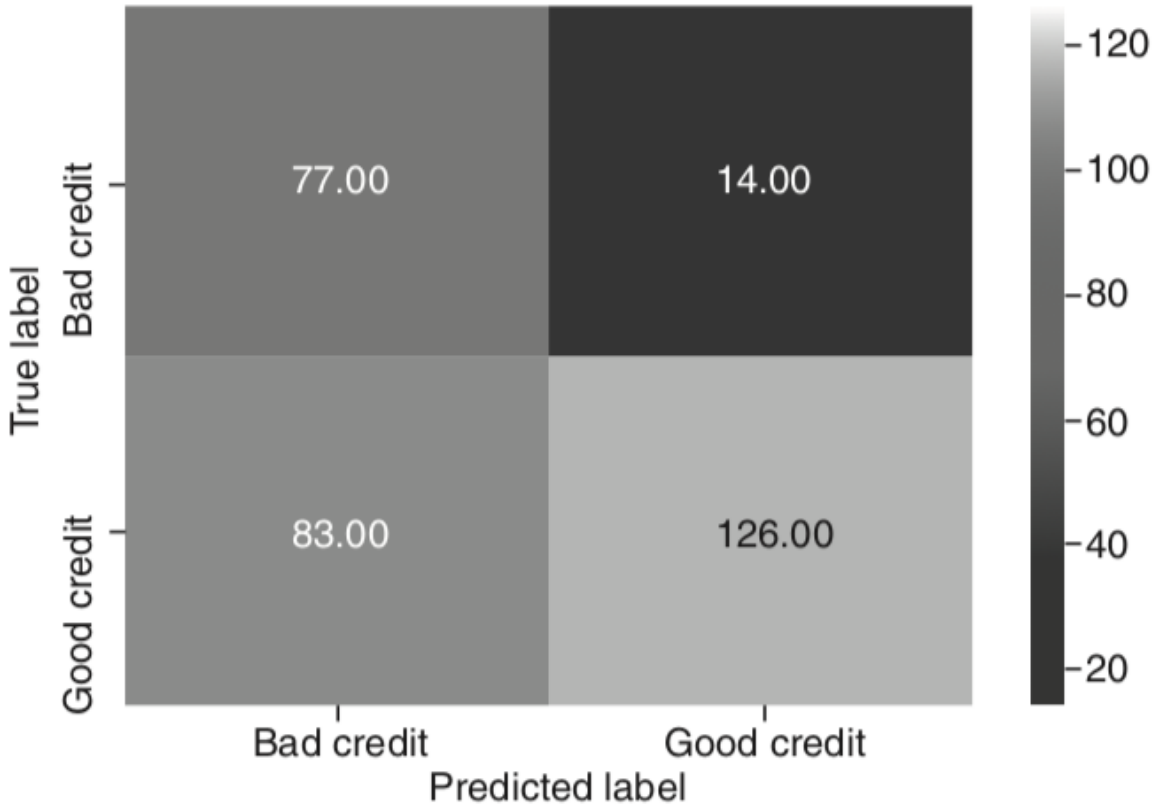

FIGURE 5.5 Confusion matrix with optimal cut-off using Yoden's index.

Credit Classification (Cntd.)

1. Cost-based Approach
   - A measure that finds the cut-off where the total penalty cost is minimum.
   - Assuming cost of a false positive is C1 and that of a false negative is C2
   - Total cost will be

$$\text{Total cost} = FN * C1 + FP * C2$$

```
def get_total_cost( actual, predicted, cost_FPs, cost_FNs ):
    # Get the confusion matrix and calculate cost
    cm = metrics.confusion_matrix( actual, predicted, [1,0] )
    cm_mat = np.array( cm )
    return cm_mat[0,1] * cost_FNs + cm_mat[1,0] * cost_FPs
```

Create a DataFrame which will capture the cost against different cut-off probability values.

```
cost_df = pd.DataFrame( columns = ['prob', 'cost'])
```

Credit Classification (Cntd.)

- Finding the optimal cut-off probability at lowest cost

```
idx = 0

## iterate cut-off probability values between 0.1 and 0.5
for each_prob in range(10, 50):
    cost = get_total_cost( y_pred_df.actual,
                           y_pred_df.predicted_prob.map(
           lambda x: 1 if x > (each_prob/100)      else 0), 1, 5 )
    cost_df.loc[idx] = [(each_prob/100), cost]
    idx += 1
```

```
cost_df.sort_values( 'cost', ascending = True )[0:5]
```

	Prob	Cost
4	0.14	150.0
12	0.22	153.0
2	0.12	154.0
10	0.20	154.0
9	0.19	156.0

Credit Classification (Cntd.)

- Lowest cost is achieved at cut-off probability of 0.14 if false negatives are assumed to be five times costlier than false positives.

```
y_pred_df['predicted_using_cost'] = y_pred_df.predicted_prob.map(
  lambda x: 1 if x > 0.14 else 0)
```

```
draw_cm( y_pred_df.actual,
         y_pred_df.predicted_using_cost )
```

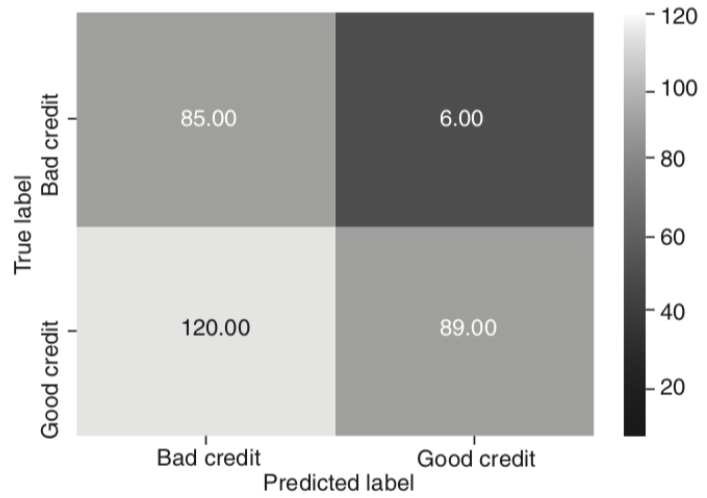

FIGURE 5.6 Confusion matrix with cost-based optimal cut-off.

Gain and Life Chart

- Used for measuring the benefits of using logistic regression model
- Business context such as target marketing
- Steps for using Gain and Lift charts
    1. Predict the probability Y=1 (positive) using the logistic regression model and arrange the observation in the decreasing order of predicted probability.
    2. Divide the datasets into deciles. Calculate the number of positives (Y=1) in each decile and cumulative number of positives up to a decile.
    3. Gain is the ratio between cumulative number of the positive observations up to a decile to total number of positive observations in the data.
    4. Gain chart is a chart drawn between gain on the vertical axis and decile on the horizontal axis

Gain and Lift Chart (Cntd.)

5. Lift is the ratio of the number of positive observations up to a decile I using the LR model to the expected number of positives up to that decile I based on a random model.
6. Lift chart is the chart between lift on the vertical axis and the corresponding decile on the horizontal axis.

$$\text{Gain} = \frac{\text{Cumulative number of positive observations upto decile } i}{\text{Total number of positive observations in the data}}$$

$$\text{Lift} = \frac{\text{Cumulative number of positive observations upto decile } i \text{ using LR model}}{\text{Cumulative number of positive observations upto decile } i \text{ based on random model}}$$

Gain and Lift Chart (Cntd.)

- Bank Marketing dataset

TABLE 5.2 Data description for bank marketing dataset

Variable	Variable Type	Description
age	numeric	Age of the client who is the targe of this marketing exercise
job	categorical	type of job (categories: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self- employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
marital	categorical	marital status (categories: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
education	categorical	education qualification (categories: "unknown", "secondary", "primary", "tertiary")
default	categorical	customer has credit in default? (categories: 'no', 'yes')
balance	numerical	average yearly balance, in euros
housing-loan	categorical	has housing loan? (categories: 'no', 'yes')
personal-loan	categorical	has personal loan? (categories: 'no', 'yes')
previous-campaign	numerical	number of contacts performed before this campaign and for this client
subscribed	categorical	has the client subscribed a term deposit? (binary: "yes", "no")

Gain and Lift Chart (Cntd.)

- Loading and Preparing the dataset

```
import pandas as pd
bank_df = pd.read_csv( 'bank.csv' )
bank_df.head( 5 )
```

	age	job	marital	education	default	balance	housing-loan	personal-loan	current-campaign	previous-campaign	subscribed
0	30	unemployed	married	primary	no	1787	no	no	1	0	no
1	33	services	married	secondary	no	4789	yes	yes	1	4	no
2	35	management	single	tertiary	no	1350	yes	no	1	1	no
3	30	management	married	tertiary	no	1476	yes	yes	4	0	no
4	59	blue-collar	married	secondary	no	0	yes	no	1	0	no

Gain and Lift Chart (Cntd.)

- Capturing the features into a list

```
X_features = list( bank_df.columns )
X_features.remove( 'subscribed' )
X_features
```

```
['age',
 'job',
 'marital',
 'education',
 'default',
 'balance',
 'housing-loan',
 'personal-loan',
 'current-campaign',
 'previous-campaign']
```

Gain and Lift Chart (Cntd.)

- Encoding the categorical features

```
encoded_bank_df = pd.get_dummies( bank_df[X_features],
                                  drop_first = True )
```

- Encoding the outcome variable as 1 (yes) and 0 (no)

```
Y = bank_df.subscribed.map( lambda x: int( x == 'yes') )
X = encoded_bank_df
```

Gain and Lift Chart (Cntd.)

- Building the Logistic Regression (LR) Model

```
logit_model = sm.Logit( Y, sm.add_constant( X ) ).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.335572
         Iterations 7
```

```
logit_model.summary2()
```

Model:	Logit	Pseudo R-squared:	0.061
Dependent Variable:	subscribed	AIC:	3082.2384
Date:	2018-02-26 21:42	BIC:	3236.2341
No. Observations:	4521	Log-Likelihood:	−1517.1
Df Model:	23	LL-Null:	−1615.5
Df Residuals:	4497	LLR p-value:	1.4866e-29
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

Gain and Lift Chart (Cntd.)

	Coef.	Std.Err.	z	P > |z|	[0.025	0.975]
const	−1.7573	0.3799	−4.6251	0.0000	−2.5019	−1.0126
age	0.0078	0.0058	1.3395	0.1804	−0.0036	0.0191
balance	−0.0000	0.0000	−0.2236	0.8231	−0.0000	0.0000
current-campaign	−0.0905	0.0238	−3.8042	0.0001	−0.1371	−0.0439
previous-campaign	0.1414	0.0212	6.6569	0.0000	0.0998	0.1830
job_blue-collar	−0.3412	0.2000	−1.7060	0.0880	−0.7331	0.0508
job_entrepreneur	−0.2900	0.3161	−0.9175	0.3589	−0.9096	0.3295
job_housemaid	−0.0166	0.3339	−0.0497	0.9603	−0.6711	0.6379
job_management	−0.0487	0.1984	−0.2455	0.8061	−0.4375	0.3401
job_retired	0.5454	0.2503	2.1794	0.0293	0.0549	1.0360
job_self-employed	−0.2234	0.2895	−0.7715	0.4404	−0.7909	0.3441
job_services	−0.2248	0.2245	−1.0012	0.3167	−0.6648	0.2152
job_student	0.3888	0.3181	1.2223	0.2216	−0.2346	1.0122

	Coef.	Std.Err.	z	P > |z|	[0.025	0.975]
job_technician	−0.2101	0.1874	−1.1213	0.2622	−0.5773	0.1571
job_unemployed	−0.3723	0.3336	−1.1162	0.2643	−1.0261	0.2815
job_unknown	0.3193	0.4620	0.6913	0.4894	−0.5861	1.2248
marital_married	−0.4012	0.1440	−2.7857	0.0053	−0.6835	−0.1189
marital_single	−0.0463	0.1676	−0.2763	0.7823	−0.3749	0.2822
education_secondary	0.2128	0.1680	1.2670	0.2052	−0.1164	0.5420
education_tertiary	0.3891	0.1935	2.0103	0.0444	0.0098	0.7684
education_unknown	−0.1956	0.2927	−0.6682	0.5040	−0.7693	0.3781
default_yes	0.2286	0.3670	0.6228	0.5334	−0.4908	0.9479
housing-loan_yes	−0.5355	0.1024	−5.2273	0.0000	−0.7362	−0.3347
personal-loan_yes	−0.7139	0.1689	−4.2268	0.0000	−1.0449	−0.3829

Gain and Lift Chart (Cntd.)

- Finding out the significant variables

```
significant_vars = get_significant_vars(logit_model)

significant_vars
```

```
['const',
 'current-campaign',
 'previous-campaign',
 'job_retired',
 'marital_married',
 'education_tertiary',
 'housing-loan_yes',
 'personal-loan_yes']
```

Gain and Lift Chart (Cntd.)

- Building LR model using only significant variables

```
logit_model_2 = sm.Logit( Y, sm.add_constant( X[X_features] ) ).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.337228
         Iterations 7
```

```
logit_model_2.summary2()
```

Model:	Logit	Pseudo R-squared:	0.056
Dependent Variable:	subscribed	AIC:	3065.2182
Date:	2018-02-26 21:42	BIC:	3116.5501
No. Observations:	4521	Log-Likelihood:	−1524.6
Df Model:	7	LL-Null:	−1615.5
Df Residuals:	4513	LLR p-value:	8.1892e-36
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

Gain and Lift Chart (Cntd.)

	Coef.	Std.Err.	z	$P > \|z\|$	[0.025	0.975]
const	−1.4754	0.1133	−13.0260	0.0000	−1.6974	−1.2534
current-campaign	−0.0893	0.0236	−3.7925	0.0001	−0.1355	−0.0432
previous-campaign	0.1419	0.0211	6.7097	0.0000	0.1004	0.1833
job_retired	0.8246	0.1731	4.7628	0.0000	0.4853	1.1639
marital_married	−0.3767	0.0969	−3.8878	0.0001	−0.5667	−0.1868
education_tertiary	0.2991	0.1014	2.9500	0.0032	0.1004	0.4978
housing-loan_yes	−0.5834	0.0986	−5.9179	0.0000	−0.7767	−0.3902
personal-loan_yes	−0.7025	0.1672	−4.2012	0.0000	−1.0302	−0.3748

Gain and Lift Chart (Cntd.)

```
y_pred_df = pd.DataFrame( { 'actual': Y,
                            'predicted_prob': logit_model_2.predict(
                            sm.add_constant( X[X_features] ) ) } )
```

- Sorting the observations by their predicted probabilities in descending order

```
sorted_predict_df = y_pred_df[['predicted_prob',
                               'actual']].sort_values('predicted_prob',
                               ascending = False)
```

- Segmenting the observations into decile

```
num_per_decile = int( len( sorted_predict_df ) / 10 )
print("Number of observations per decile:", num_per_decile)
```

```
Number of observations per decile:      452
```

Gain and Lift Chart (Cntd.)

```
def get_deciles(df):
    # Set first decile
    df['decile'] = 1

    idx = 0
    # Iterate through all 10 deciles
    for each_d in range(0, 10):
        # Setting each 452 observations to one decile in sequence
        df.iloc[idx:idx+num_per_decile, df.columns.get_
            loc('decile')] = each_d
        idx += num_per_decile

    df['decile'] = df['decile'] + 1

    return df
```

```
deciles_predict_df = get_deciles( sorted_predict_df )
```

Gain and Lift Chart (Cntd.)

```
deciles_predict_df[0:10]
```

	predicted_prob	actual	decile
3682	0.864769	0	1
97	0.828031	0	1
3426	0.706809	0	1
1312	0.642337	1	1
3930	0.631032	1	1
4397	0.619146	0	1
2070	0.609129	0	1
3023	0.573199	0	1
4080	0.572364	0	1
804	0.559350	0	1

Gain and Lift Chart (Cntd.)

- Calculating Gain

```
gain_lift_df = pd.DataFrame(
     deciles_predict_df.groupby(
              'decile')['actual'].sum() ).reset_index()
gain_lift_df.columns = ['decile', 'gain']
```

```
gain_lift_df['gain_percentage'] = (100 * gain_lift_df.gain.
                                   cumsum()/gain_lift_df.gain.sum())
```

```
gain_lift_df
```

	decile	gain	gain_percentage
0	1	125	23.992322
1	2	83	39.923225
2	3	73	53.934741
3	4	53	64.107486
4	5	31	70.057582
5	6	46	78.886756
6	7	37	85.988484
7	8	28	91.362764
8	9	25	96.161228
9	10	20	100.000000

Gain and Lift Chart (Cntd.)

- If we target only first 50% of the customers, we have almost 70% subscriptions. Now, plotting the change in gain percentage

```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
```

```
plt.figure( figsize = (8,4))
plt.plot( gain_lift_df['decile'],
          gain_lift_df['gain_percentage'], '-' )

plt.show()
```

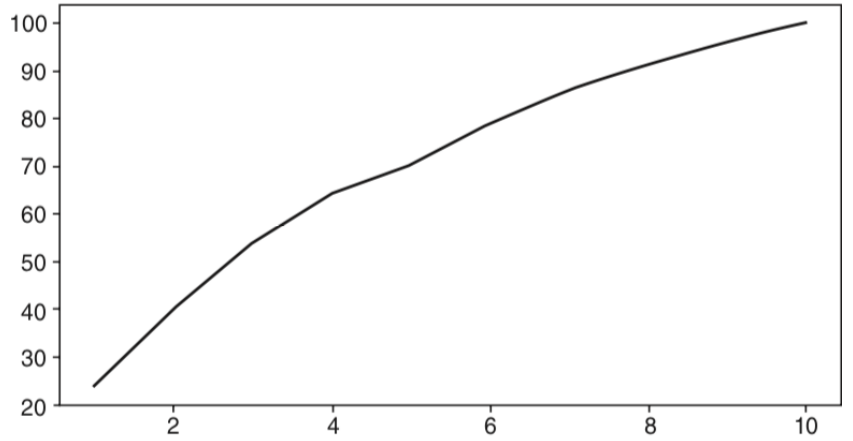

FIGURE 5.7 Gain chart.

Gain and Lift Chart (Cntd.)

- Calculating Lift

```
gain_lift_df['lift'] = ( gain_lift_df.gain_percentage
                       / ( gain_lift_df.decile * 10 ) )
gain_lift_df
```

	decile	gain	gain_percentage	lift
0	1	125	23.992322	2.399232
1	2	83	39.923225	1.996161
2	3	73	53.934741	1.797825
3	4	53	64.107486	1.602687
4	5	31	70.057582	1.401152
5	6	46	78.886756	1.314779
6	7	37	85.988484	1.228407
7	8	28	91.362764	1.142035
8	9	25	96.161228	1.068458
9	10	20	100.000000	1.000000

Gain and Lift Chart (Cntd.)

```
plt.figure( figsize = (8,4))
plt.plot( gain_lift_df['decile'], gain_lift_df['lift'], '-' )
plt.show()
```

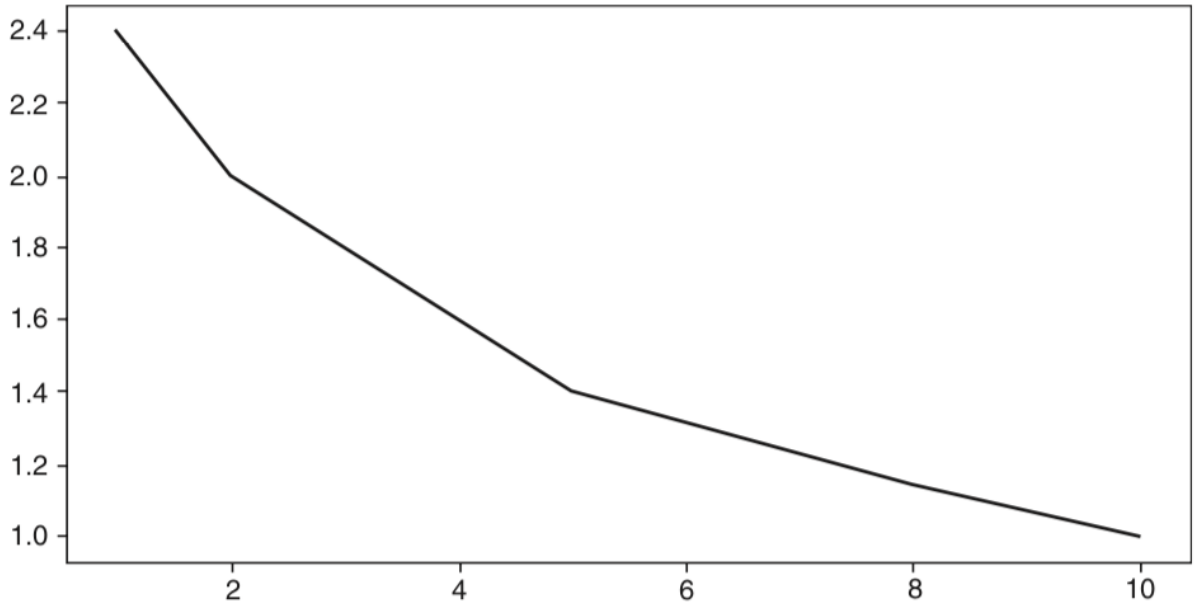

FIGURE 5.8 Lift chart.

- Targeting customers using the model can capture 2.39 times the number of subscribers compared to a random model in decile 1, 1.99 times up to decile 2, and so on and so forth.

Classification Tree (Decision Tree Learning)

- Collection of divide and conquer problem-solving strategies.
- Uses tree-like structure to predict the value of an outcome variable.
- Root node consists of the complete data
- Multiple branches are created using intelligent strategies.
- Classification and Regression Tree (CART) is one of the classification tree techniques.
  - Classification tree
    1. if the outcome variable is discrete.
    2. Impurity measures used such as Gini Impurity Index and Entropy
  - Regression tree
    1. if the outcome variable is continuous.
    2. Minimizes the Sum of Squared Errors (SSE)

Classification Tree (Decision Tree Learning)

- Steps to generate classification and regression trees:
  1. Start with the complete training data in the root node.
  2. Decide on the measure of impurity. Search for a predictor variable that minimizes the impurity when the parent node is split into children nodes.
  3. Repeat step 2 for each subset of the data using the independent variables until
     1. All the dependent variables are exhausted.
     2. The stopping criteria are met. Few stopping criteria used are number of levels of tree from the root node, minimum number of observations in parent/child node, and minimum reduction in impurity index.
  4. Generate business rules for the leaf nodes of the tree.

Classification Tree (Decision Tree Learning)

- *sklearn.tree.DecisionTreeClassifier* provides decision tree algorithm. It takes the following key parameters:
  1. criteion: string – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
  2. max_depth: int – The maximum depth of the tree.
  3. min_samples_split: int or float – The minimum number of samples required to split an internal node. If *int,* then number of samples or if *float,* percentage of total number of samples.
  4. in_samples_leaf: int – The minimum number of samples required to be at a leaf node.

Classification Tree (Decision Tree Learning)

- Building the Model

```
Y = credit_df.status
X = encoded_credit_df
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, Y,
                                                 test_size = 0.3,
                                                 random_state = 42 )
```

Classification Tree (Decision Tree Learning)

- Decision Tree Classifier using Gini Criteria

```
from sklearn.tree import DecisionTreeClassifier
clf_tree = DecisionTreeClassifier(criterion = 'gini',
                                  max_depth = 3 )
```

```
clf_tree.fit( X_train, y_train )
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_
     depth=3,
     max_features=None, max_leaf_nodes=None,
     min_impurity_decrease=0.0, min_impurity_split=None,
     min_samples_leaf=1, min_samples_split=2,
     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
     splitter='best')
```

Classification Tree (Decision Tree Learning)

- Measuring test Accuracy and Displaying the Tree

```
tree_predict = clf_tree.predict( X_test )
metrics.roc_auc_score( y_test, tree_predict )
```

```
0.58
```

- To visualize the tree, use *graphviz* software. Install the following
  - GraphViz
  - pip install pydotplus
- The export the tree model to a file using *export_graphviz()* function

Classification Tree (Decision Tree Learning)

- Measuring test Accuracy and Displaying the Tree

```
from sklearn.tree import export_graphviz
import pydotplus as pdot
from IPython.display import Image

# Export the tree into odt file
export_graphviz( clf_tree,
                 out_file = "chd_tree.odt",
                 feature_names = X_train.columns,
                 filled = True )

# Read the create the image file
chd_tree_graph = pdot.graphviz.graph_from_dot_file( 'chd_tree.odt' )
chd_tree_graph.write_jpg( 'chd_tree.png' )
# Render the png file
Image(filename='chd_tree.png')
```

Classification Tree (Decision Tree Learning)

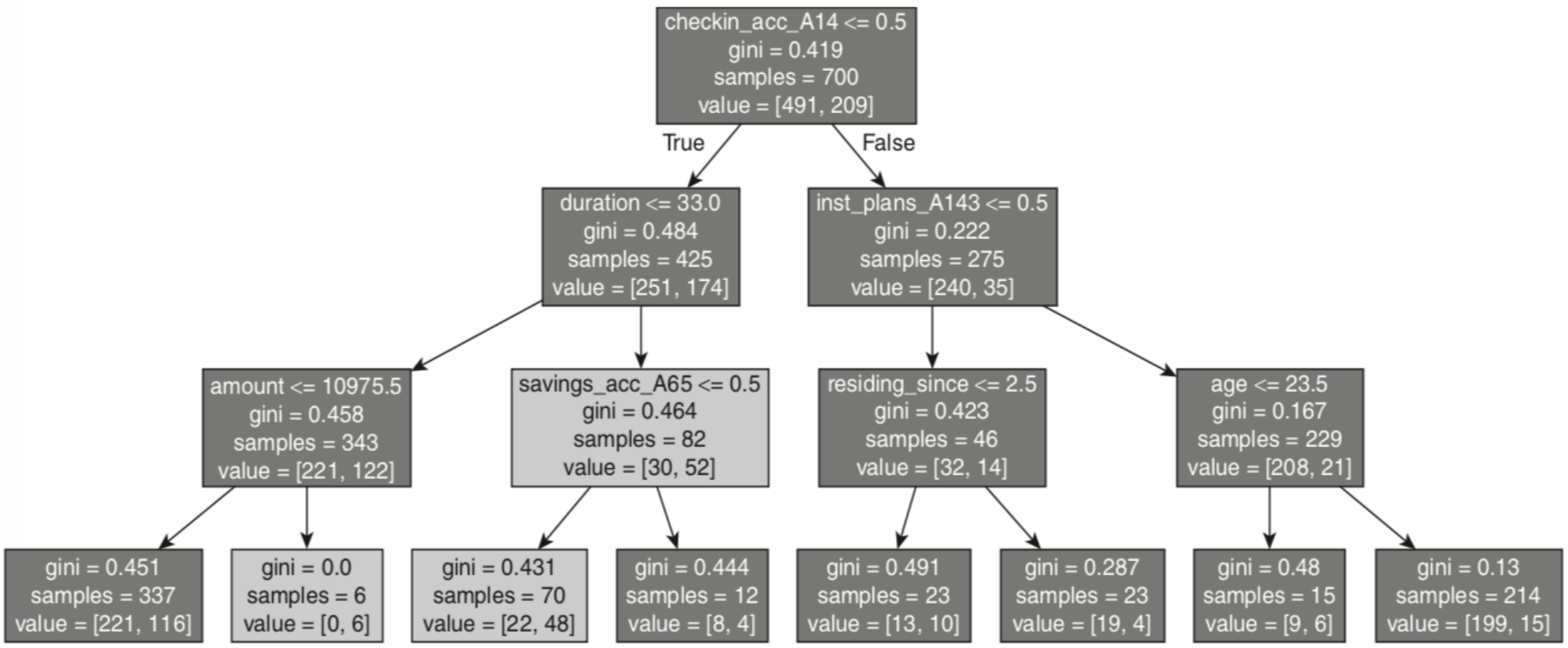

FIGURE 5.9 Decision tree graph created using Gini impurity splitting criteria.

Classification Tree (Decision Tree Learning)

- Interpretation from the decision tree diagram
  1. At the top node, there are 700 observations of which 491 are good credits. The corresponding Gini Index is 0.419.
  2. *checkin_acc_A14* is the most important feature for splitting the good/bad credits and hence, chosen as the top splitting criteria.
  3. The first rule (*checkin_acc_A14 < 0.5)* means if the customer has *checkin_acc_A14* or not.
  4. The nodes represented by dark shades depict good credits while the light shades are bad credit.
  5. Rule 1: if the customer doesn't have *checkin_acc_A14* and credit duration is less than 33 and doesn't have *saving_acc_A65*, then there is high probability of being a bad credit.
  6. Rule 2: If the customer has *checkin_acc_A14* and *Inst_plans_A143* and age is more than 23.5, the thee is high probability of being good credit.

Classification Tree (Decision Tree Learning)

- Understanding Gini Impurity
  - Probability of a ransom sample being classified correctly if we randomly pick a label according to the distribution in a branch.
  - The Gini impurity index for a classification problem with C classes is given by

$$Gini(k) = \sum_{i=1}^{C} p_i(k)(1 - p_i(k))$$

  - It reaches its minimum when all cases in the node belong to a specific category.

```
gini_node_1 = 1 - pow(491/700, 2) - pow (209/700, 2)
print( round( gini_node_1, 4) )
```

```
0.419
```

The Gini index is 0.419 as shown in Figure 5.9.

Classification Tree (Decision Tree Learning)

- Building Decision Tree using Entropy Criteria

$$\text{Entropy}(k) = -\sum_{j=1}^{J} p(j|k)\log_2(j|k)$$

  - The value of entropy lies between 0 and 1
  - Higher entropy indicates a higher impurity at the node.

```
clf_tree_entropy = DecisionTreeClassifier( criterion = 'entropy',
                                           max_depth = 3 )
clf_tree_entropy.fit( X_train, y_train )

# Export the tree into odt file
export_graphviz( clf_tree_entropy,
                 out_file = "chd_tree_entropy.odt",
                 feature_names = X_train.columns )

# Read the create the image file
chd_tree_graph = pdot.graphviz.graph_from_dot_file( 'chd_tree_
entropy.odt' )
chd_tree_graph.write_jpg ( 'chd_tree_entropy.png' )
# Render the png file
Image(filename='chd_tree_entropy.png')
```

Classification Tree (Decision Tree Learning)

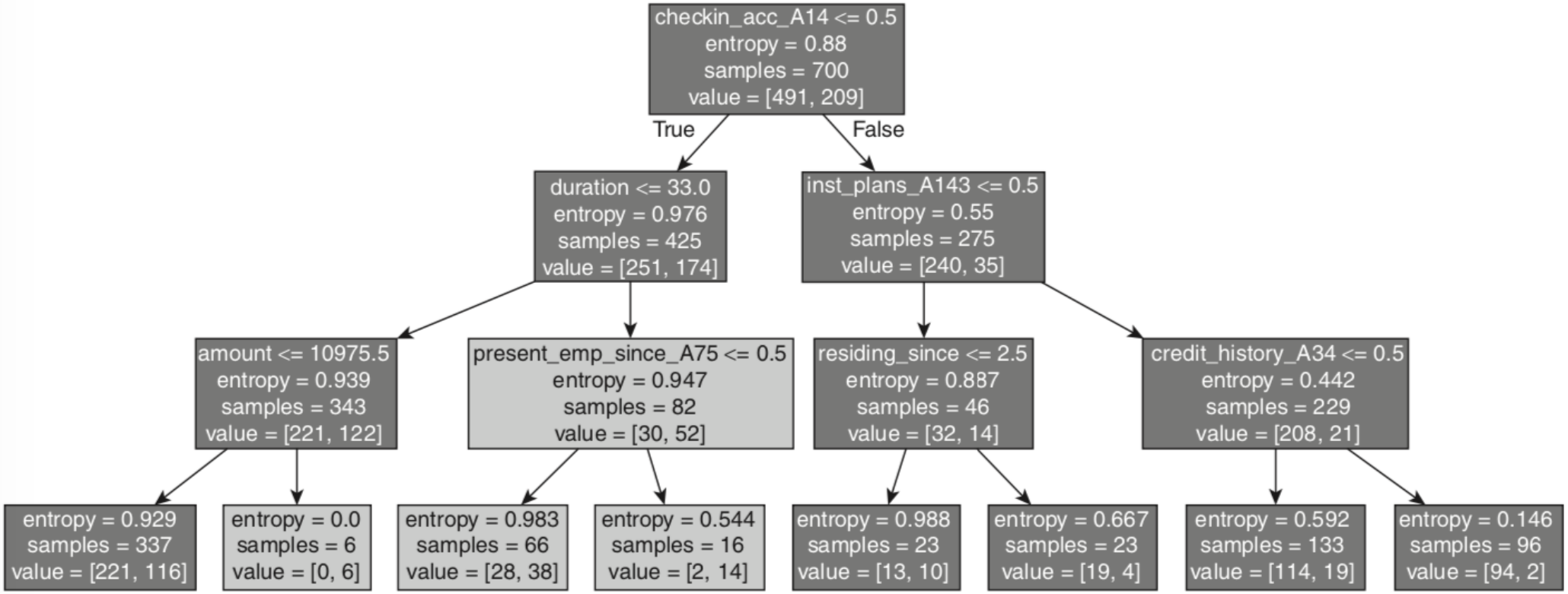

FIGURE 5.10 Decision tree graph created using entropy splitting criteria.

Classification Tree (Decision Tree Learning)

1. The nodes represented by dark shades depict good credits, while the nodes represented by light shades are bad credits.
2. The rules generated at the top two level nodes are same as rules generated by decision tree built using Gini index. The rules at the third level nodes are different.
3. Rule 1: if the customer has *checkin_acc_A14, Inst_plans_code* is A143 and credit history code is A34, then there is high probability of being good credit.

Classification Tree (Decision Tree Learning)

- Calculating Entropy

```
import math

entropy_node_1 = - (491/700)*math.log2(491/700) - (209/700)*math.
                   log2(209/700)
print(round( entropy_node_1, 2))
```

```
0.88
```

The entropy at the top node is 0.88, as shown in the Figure 5.10.

- Measuring Test Accuracy using AUC

```
tree_predict = clf_tree_entropy.predict( X_test )
metrics.roc_auc_score( y_test, tree_predict )
```

```
0.5763972869236027
```

Classification Tree (Decision Tree Learning)

- Finding Optimal Criteria and Max_depth
  1. GridSearchCV is used to find the most optimal hyperpatamers.
  2. For classification models, roc_auc values are used as accuracy measure.
  3. It used k-fold cross validation to measure and validate the accuracy.
  4. It takes the following arguments:
     1. estimator – A scikit-learn model which implements the estimator interface.
     2. param_grid – A dictionary with parameter names (string) as keys and lists of parameter settings to try as values.
     3. scoring – Is a string; is the accuracy measure, i.e. roc_auc.
     4. cv – is an integer; gives the number of folds in k-fold.
  5. Configuring the grid search to search for optimal parameters
     1. Splitting criteria: gini or entropy
     2. Maximum depth of decision tree ranging from 2 to 10

Classification Tree (Decision Tree Learning)

```
from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'criterion': ['gini','entropy'],
                     'max_depth': range(2,10)}]
clf_tree = DecisionTreeClassifier()
clf = GridSearchCV(clf_tree,
      tuned_parameters,
      cv=10,
      scoring='roc_auc')
clf.fit(X_train, y_train)
```

```
GridSearchCV(cv=10, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None,
           criterion='gini', max_depth=None,
           max_features=None, max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0,presort=False,random_state=None,
           splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'max_depth': range(2, 10), 'criterion': ['gini',
       'entropy']}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Classification Tree (Decision Tree Learning)

```
clf.best_score_
```

```
0.68242993197278912
```

```
clf.best_params_
```

```
{'criterion': 'gini', 'max_depth': 2}
```

- The tree with *gini* criteria and *max_depth* = 2 is the best model.

Classification Tree (Decision Tree Learning)

- Benefits of using Decision Tree

1. Rules generated are simple and interpretable.
2. Trees can be visualized.
3. Work well with both numerical and categorical data. Do not require data to be normalized or creation of dummy variables.
4. Rules can help create business strategies.

Thank You !