Table of Contents

[ROC Curves and Precision Recall Curves 1](#_Toc31886698)

[True Positive Rate 1](#_Toc31886699)

[False Positive Rate 1](#_Toc31886700)

[Automated Feature Selection 10](#_Toc31886701)

[Recursive Feature Selection 10](#_Toc31886702)

## ROC Curves and Precision Recall Curves

ROC stands for the Receiver Operating Characteristic curve, or ROC curve. ROC Curves summarize the trade-off between the true positive and false positive rates of a model using different probability thresholds. ROC curves are appropriate when the observations are balanced between each class.

### True Positive Rate

The true positive rate summarizes how well the model identifies all actual positive outcomes.

**True Positive Rate (Recall)** =

### False Positive Rate

The false positive rate, or false alarm rate, summarizes how often positive predictions are made for actual negative outcomes.

**False Positive Rate (*False Alarm Rate*)** =

Exercise (2 marks)

Given the following confusion matrix, what is the true positive rate? What is the false positive rate? Please show your calculations.

|  |  |  |  |
| --- | --- | --- | --- |
| Actual |  | Predicted | |
|  | 0 | 1 |
| 0 | 15 | 5 |
| 1 | 10 | 20 |

20 / 20 + 10 = 20/30

True Positive Rate = 20/30

Exercise (2 marks)

Given the following confusion matrix, what is the true positive rate? What is the false positive rate? Please show your calculations.

|  |  |  |  |
| --- | --- | --- | --- |
| Actual |  | Predicted | |
|  | 0 | 1 |
| 0 | 20 | 15 |
| 1 | 5 | 10 |

False Positive Rate = 5/ 5+20 = 5/25

Exercise (1 mark)

Do the confusion matrices presented in Exercise 1and Exercise 2 rely on probability threshold in any way?

|  |
| --- |
|  |

Figure 1 shows both poor and desirable ROC curves. The area under the line indicates the area of choosing the correct response with a 50/50 guess. When the data and model have been prepared properly, an area under the curve value close to 1 is the best scenario.

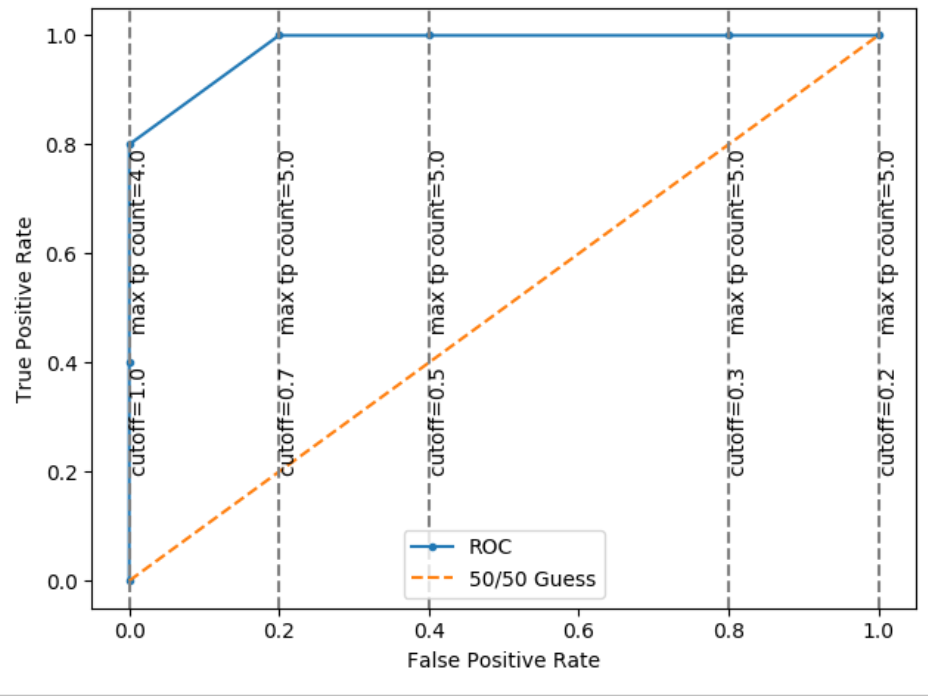
Figure : Poor and Desirable ROC Curves

|  |  |  |  |
| --- | --- | --- | --- |
| Poor AUC=0.482 |  | Good AUC = 0.98 |  |
|  | Predicted  0 2 5  1 1 7 |  | Predicted  0 3 2  1 0 5 |

Example : Simple ROC Curve

This example shows additional content in the ROC curve to show how threshold and true positive counts relate to the ROC curve. The output is displayed in Figure 2.

Figure : ROC Curve with Threshold and True Positive Count Information



Exercise (1 mark)

How does threshold relate to the true positive rate in a ROC curve?

|  |
| --- |
|  |

How is threshold related to the false positive rate in a ROC curve?

|  |
| --- |
|  |

Exercise (1 mark)

As a minimum, assuming you could fine tune results later, given the scenario presented in Figure 2, what cut-off could you use to achieve optimal results?

|  |
| --- |
|  |

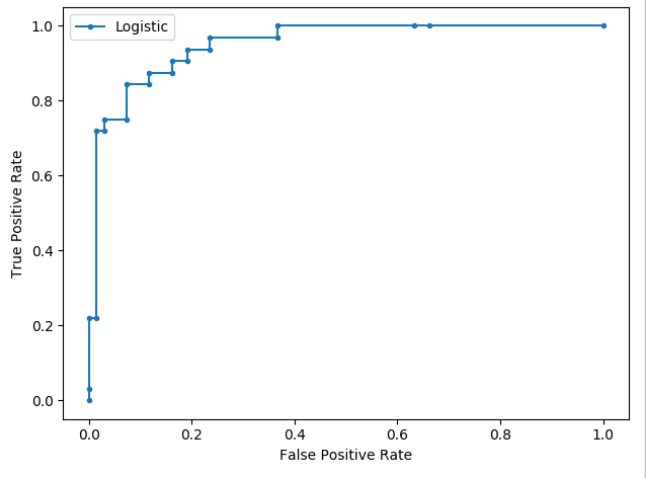
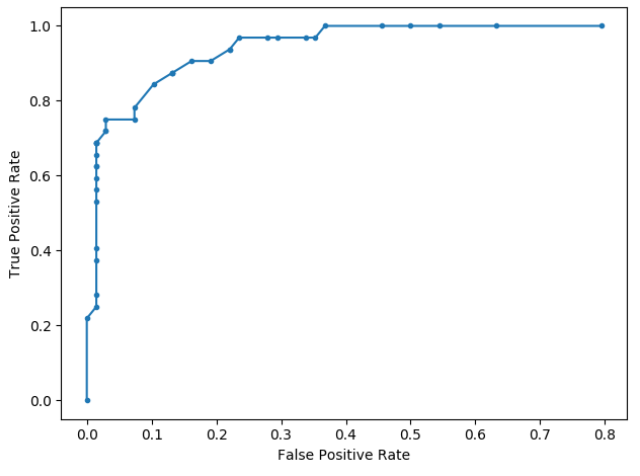
This is the code that is used to draw the output for Example 1.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  import seaborn as sn  from sklearn.metrics import roc\_curve  from sklearn.metrics import roc\_auc\_score  import matplotlib.pyplot as plt  # Setup data.  candidates = {  'work\_experience': [3,4,3,5,4,6,1,4,5,  1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,  5,1,2,1,4,5],  'admitted': [1,1,1,1,1,1,0,1,1,0,0,1,  1,1,1,0,0,1,0,0,0,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,  0,0,1]}  df = pd.DataFrame(candidates,columns= ['work\_experience','admitted'])  print(df)  # Separate into x and y values.  X = df[['work\_experience']]  y = df['admitted']  # Split data.  X\_train,X\_test,y\_train,y\_test = train\_test\_split(  X, y, test\_size=0.25,random\_state=0)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state = 0,  solver='liblinear')  logisticModel.fit(X\_train,y\_train)  y\_pred = logisticModel.predict(X\_test)  y\_prob = logisticModel.predict\_proba(X\_test)  # Show model coefficients and intercept.  print("\nModel Coefficients: ")  print("\nIntercept: ")  print(logisticModel.intercept\_)  print(logisticModel.coef\_)  # Show confusion matrix and accuracy scores.  confusion\_matrix = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  sn.heatmap(confusion\_matrix, annot=True)  plt.show()  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix)  # -------------------------------------------------------------  # Calculates accuracy and shows confusion matrix with custom  # cutoff probability for a response of 1.  # -------------------------------------------------------------  def predictUsingAlternateCutoff(y\_test, probabilities, cutoff, size):  cm = np.zeros((size, size)) # Create empty 5x5 matrix of 0’s  predictions = []  correctCount = 0  for i in range(0, len(probabilities)):  actualRow = y\_test.values[i]  predictCol = 0 # Predicted value is 0 by default.  # Check if probability of zero is high.  if (probabilities[i] >= cutoff):  predictCol = 1  predictions.append(predictCol)  # Increase correct item count when prediction matches actual response.  if (actualRow == predictCol):  correctCount += 1    # Update confusion matrix.  cm[actualRow][predictCol] += 1    accuracy = correctCount / len(probabilities)  tp = cm[1][1]  fn = cm[1][0]  tn = cm[0][0]  fp = cm[0][1]  tpr = tp/(tp + fn)  fpr = fp/(fp + tn)  print("\n\*\*\* CUTOFF: " + str(cutoff))  print("Confusion Matrix: actual (row) vs predicted (col)")  print("TPR: " + str(tpr) + " FPR: " + str(fpr) + " Accuracy: "  + str(accuracy))  print(cm)  return fpr, tp  def showThresholdStats(probabilities):  fprList = []  cutoffList = []  tpList = []  previousFpr = -1  for i in range(10,-1,-1):  SIZE = 2 # for 2x2 matrix  cutoff = i/10  fpr, tp = predictUsingAlternateCutoff(y\_test, probabilities, cutoff, SIZE)  # New fpr rate found.  if(fpr > previousFpr):  tpList.append(tp)  fprList.append(fpr)  cutoffList.append(cutoff)  previousFpr = fpr  # No new fpr but total positives increased.  elif(tp>tpList[len(tpList)-1]):  tpList[len(tpList)-1] = tp  return cutoffList, fprList, tpList  # Extract probabilities.  y\_prob = logisticModel.predict\_proba(X\_test)  cutoffList, fprList, tpList = showThresholdStats(y\_prob[:,1])  # calculate scores  auc = roc\_auc\_score(y\_test, y\_prob[:, 1],)  print('Logistic: ROC AUC=%.3f' % (auc))  # calculate roc curves  lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, y\_prob[:, 1])  plt.plot(lr\_fpr, lr\_tpr, marker='.', label='ROC')  plt.plot([0,1], [0,1], '--', label='50/50 Guess')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  for i in range(0, len(fprList)):  plt.axvline(x=fprList[i],linestyle='--', color='grey')  plt.text(fprList[i], 0.2,  'cutoff=' +str(cutoffList[i])  + ' max tp count='  + str(tpList[i]), rotation=90)  plt.show() |

Example : Drawing the ROC Curve

This example shows how to draw the ROC curve for the computerPurchase.csv dataset. The ROC curve at the left in Figure 3 displays the ROC curve automatically with the roc\_curve() function. The ROC curve on the right is drawn manually to demonstrate more clearly how the numbers are obtained in code. The **high initial true-positive rate** uses a highcut-off for a positive response.By ensuring predictions of 1 with a probability cutoff of say, 90%, we are ensure only the most likely candidates are selected. Also, with a probability cutoff of 90% we are less likely to encounter false positives. However, as we lower our cutoff rate an increasing number of false positive responses could be selected.

Figure : ROC Curves

Exercise (8 marks)

1. Examine the code and output to determine the true positive rate when the cutoff rate is 0.85. List it here:

|  |
| --- |
|  |

1. Examine the code and output to determine the false positive rate when the cutoff rate is 0.85. List it here:

|  |
| --- |
|  |

1. Examine the code and output to determine the true positive rate when the cutoff rate is 0.15. List it here:

|  |
| --- |
|  |

1. Examine the code and output to determine the false positive rate when the cutoff rate is 0.15. List it here:

|  |
| --- |
|  |

1. Summarize your finding for the four results above:

|  |
| --- |
|  |

1. What can we say about the quality of the sample candidates when the false positive rate is low?

|  |
| --- |
|  |

1. What issue could occur if we only choose samples when the false positive rate is low?

|  |
| --- |
|  |

1. Suggest a point on the curve in Figure 3 where the recall score is low.

|  |
| --- |
|  |

This is the code that is used to draw the ROC curve that is shown in Figure 3.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import roc\_curve  from sklearn.metrics import roc\_auc\_score  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "computerPurchase.csv"  df = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding="ISO-8859-1", sep=',',  names=("User ID", "Gender", "Age", "EstimatedSalary",  "Purchased"))  # Create dummy variable for gender.  tempDf = df[['Gender']] # Isolate columns  dummyDf = pd.get\_dummies(tempDf, columns=['Gender']) # Get dummies  df = pd.concat(([df, dummyDf]), axis=1) # Join dummy with original  print(df)  # Separate into x and y values.  X = df[["Gender\_Female", "Gender\_Male", "Age", "EstimatedSalary"]]  y = df['Purchased']  # Show chi-square scores for each feature.  # There is 1 degree freedom since 1 predictor during feature evaluation.  # Generally, >=3.8 is good)  test = SelectKBest(score\_func=chi2, k=3)  chiScores = test.fit(X, y) # Summarize scores  np.set\_printoptions(precision=3)  print("\nPredictor Chi-Square Scores: " + str(chiScores.scores\_))  # Re-assign X with significant columns only after chi-square test.  X = df[["Age", "EstimatedSalary"]]  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.25, random\_state=0)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state=0,  solver='liblinear')  # Scale training set.  sc\_x = StandardScaler()  X\_train = sc\_x.fit\_transform(X\_train)  X\_test = sc\_x.fit\_transform(X\_test)  # Fit the model.  logisticModel.fit(X\_train, y\_train)  # Show model coefficients and intercept.  print("\nModel Coefficients: ")  print("\nIntercept: ")  print(logisticModel.intercept\_)  print(logisticModel.coef\_)  # -------------------------------------------------------------  # Calculates accuracy and shows confusion matrix with custom  # cutoff probability for a response of 1.  # -------------------------------------------------------------  def predictUsingAlternateCutoff(y\_test, probIsOne, cutoff, size):  cm = np.zeros((size, size)) # Create empty 5x5 matrix of 0’s  predictions = []  correctCount = 0  for i in range(0, len(probIsOne)):  actualValue\_Row = y\_test.values[i]  predictValue\_Col = 0 # Predicted value is 0 by default.  prediction = 0  # Check if probability of zero is high.  if (probIsOne[i] >= cutoff):  predictValue\_Col = 1  prediction = 1  cm[actualValue\_Row][predictValue\_Col] += 1  predictions.append(prediction)  # Increase correct item count when prediction matches actual response.  if (actualValue\_Row == predictValue\_Col):  correctCount += 1  accuracy = correctCount / len(probIsOne)  print("\n\*\*\* Accuracy with CUTOFF of " + str(cutoff) + ": " + str(accuracy))  print("\nConfusion Matrix: actual (row) vs predicted (col)")  print(cm)  return predictions, cm  # Extract probabilities.  y\_proba = logisticModel.predict\_proba(X\_test)  SIZE = 2 # for 2x2 matrix  CUT\_OFF = 0.48  updatedPredict = predictUsingAlternateCutoff(y\_test, y\_proba[:, 1], CUT\_OFF, SIZE)  def getPositiveRates(cm):  tn = cm[0][0]  fn = cm[1][0]  tp = cm[1][1]  fp = cm[0][1]  tpr = tp / (tp + fn) # tp over all possible positives  fpr = fp / (fp + tn) # fp over all possible negatives  return fpr, tpr  #----------------------------------------  # Plot the ROC  #----------------------------------------  def graphROC(y\_test, y\_proba, decrement):  tprList = []  fprList = []  SIZE = 2  # Start with cut-off of 1 and decrement down to 0.  # In other words, use from most likely true positive candidates first and then  # to most uncertain candidates.  cutOff = 1  while cutOff >-decrement:  updatedPredictions, cm = predictUsingAlternateCutoff(y\_test,  y\_proba, cutOff, SIZE)  fpr, tpr = getPositiveRates(cm)  tprList.append(tpr)  fprList.append(fpr)  cutOff -= decrement  print("tpr: " + str(tpr))  print("fpr: " + str(fpr))  print("cutoff: " + str(cutOff))  # Plot the list.  plt.plot(fprList, tprList, marker='.', label='Logistic')  plt.plot([0, 1], [0, 1], '--', label='No Skill')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  plt.show()  DECREMENT = 0.025  graphROC(y\_test, y\_proba[:, 1], DECREMENT)  # calculate scores  auc = roc\_auc\_score(y\_test, y\_proba[:, 1],)  print('Logistic: ROC AUC=%.3f' % (auc))  # calculate roc curves  lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, y\_proba[:, 1])  plt.plot(lr\_fpr, lr\_tpr, marker='.', label='Logistic')  plt.plot([0,1], [0,1], '--', label='No Skill')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  plt.show() |

## Automated Feature Selection

We will visit the topic of feature selection several times throughout the year since different algorithms and libraries that enable them offer different feature selection techniques.

Note: Scale your data so it creates a level weighting system

### Recursive Feature Selection

The most popular algorithm for automated feature selection for logistic regression with sklearn is Recursive Feature Elimination (RFE) repeatedly constructs a model by recursively removing the lowest performing features.

Example : Recursive Feature Selection

This example creates dummy variables to generate a very large number of columns. Automated column selection is then used to identify promising columns. After, a chi-square test is performed to identify insignificant columns from the best 20 that are selected. Ideally, before modelling the insignificant columns will be removed. Insignificant candidates are highlighted in yellow. These are the selected columns and corresponding chi-square scores:

\*\*\* Selected names:

duration

campaign

pdays

emp.var.rate

cons.price.idx

cons.conf.idx

euribor3m

job\_blue-collar

default\_no

default\_yes

contact\_cellular

contact\_telephone

month\_aug

month\_jun

month\_mar

month\_may

month\_nov

day\_of\_week\_mon

poutcome\_failure

poutcome\_success

Predictor Chi-Square Scores:

[3.580e+02 1.614e+01 1.580e+02 5.413e+02 7.259e+01 1.080e+01 8.907e+02

1.769e+02 8.488e+01 3.809e-01 3.153e+02 5.480e+02 2.719e+00 3.024e+00

8.429e+02 3.214e+02 5.160e+00 1.478e+01 3.735e+01 3.983e+03]

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, MinMaxScaler  from sklearn.metrics import roc\_auc\_score  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "bank-additional-full.csv"  df = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding="ISO-8859-1", sep=';',  names=(  "age", "job", "marital", "education", "default", "housing", "loan", "contact",  "month", "day\_of\_week", "duration", "campaign", "pdays", "previous", "poutcome",  "emp.var.rate", "cons.price.idx", "cons.conf.idx", "euribor3m", "nr.employed", "y"))  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  print(df.head())  print(df.describe().transpose())  print(df.info())  targetList = []  for i in range(0, len(df)):  if (df.loc[i]['y'] == 'yes'):  targetList.append(1)  else:  targetList.append(0)  df['target'] = targetList  tempDf = df[["job", "marital", "education", "default","housing", "loan", "contact", "month", "day\_of\_week", "poutcome"]] # Isolate columns  dummyDf = pd.get\_dummies(tempDf, columns=["job", "marital", "education", "default",  "housing", "loan", "contact", "month", "day\_of\_week", "poutcome"]) # Get dummies  df = pd.concat(([df, dummyDf]), axis=1) # Join dummy df with original df  X = df[[  "age", "duration", "campaign", "pdays", "previous", "emp.var.rate", "cons.price.idx",  "cons.conf.idx", "euribor3m", "nr.employed", "job\_admin.", "job\_blue-collar",  "job\_entrepreneur", "job\_housemaid", "job\_management", "job\_retired",  "job\_self-employed", "job\_services", "job\_student", "job\_technician", "job\_unemployed",  "job\_unknown", "marital\_divorced", "marital\_married", "marital\_single",  "marital\_unknown", "education\_basic.4y", "education\_basic.6y", "education\_basic.9y",  "education\_high.school", "education\_illiterate", "education\_professional.course",  "education\_university.degree", "education\_unknown", "default\_no",  "default\_unknown", "default\_yes", "housing\_no", "housing\_unknown", "housing\_yes",  "loan\_no", "loan\_unknown", "loan\_yes", "contact\_cellular", "contact\_telephone",  "month\_apr", "month\_aug", "month\_dec", "month\_jul", "month\_jun", "month\_mar",  "month\_may", "month\_nov", "month\_oct", "month\_sep", "day\_of\_week\_fri",  "day\_of\_week\_mon", "day\_of\_week\_thu", "day\_of\_week\_tue", "day\_of\_week\_wed",  "poutcome\_failure", "poutcome\_nonexistent", "poutcome\_success", ]]  y = df[['target']]  from sklearn.feature\_selection import RFE  from sklearn.linear\_model import LogisticRegression  # Scale the data prior to selection. We will discuss scaling  # next week. Basically scaling enables more evenly weighted  # attributes and faster calculations.  print("Please wait for scaling...")  sc\_x = StandardScaler()  X\_scaled = sc\_x.fit\_transform(X)  print("Please wait for automated feature selection...")  logreg = LogisticRegression(max\_iter=200)  rfe = RFE(logreg, 20) # Select top 20 features.  rfe = rfe.fit(X\_scaled, y.values.ravel())  print("Feature selection is complete.")  print(rfe.support\_)  print(rfe.ranking\_)  def getSelectedColumns(ranking):  # Extract selected indices from ranking.  indices = []  for i in range(0, len(ranking)):  if (ranking[i] == 1):  indices.append(i)  # Build list of selected column names.  counter = 0  selectedColumns = []  for col in X:  if (counter in indices):  selectedColumns.append(col)  counter += 1  return selectedColumns  selectedPredictorNames = getSelectedColumns(rfe.ranking\_)  # Show selected names from RFE.  print("\n\*\*\* Selected names: ")  for i in range(0,len(selectedPredictorNames)):  print(selectedPredictorNames[i])  # Separate into x and y values.  X = df[selectedPredictorNames]  X\_scaled = sc\_x.fit\_transform(X)  y = df['target']  # Show chi-square scores for each feature.  # There is 1 degree freedom since 1 predictor during feature evaluation.  # Generally, >=3.8 is good)  from sklearn.feature\_selection import chi2  from sklearn.feature\_selection import SelectKBest  test = SelectKBest(score\_func=chi2, k=20)  XScaled = MinMaxScaler().fit\_transform(X)  chiScores = test.fit(XScaled, y) # Summarize scores  np.set\_printoptions(precision=3)  # Search here for insignificant features.  print("\nPredictor Chi-Square Scores: " + str(chiScores.scores\_))  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.25, random\_state=0)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state=0,  solver='liblinear')  # Fit the model.  logisticModel.fit(X\_train, y\_train)  # Show model coefficients and intercept.  print("\nModel Intercept: ")  print(logisticModel.intercept\_)  print("\nModel Coefficients: ")  print(logisticModel.coef\_) |

Exercise (2 marks)

Add this code to display the confusion matrix and ROC curve.

|  |
| --- |
| y\_pred=logisticModel.predict(X\_test)  y\_prob=logisticModel.predict\_proba(X\_test)  confusion\_matrix = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print(confusion\_matrix)  # calculate scores  from sklearn.metrics import roc\_curve  from sklearn.metrics import roc\_auc\_score  auc = roc\_auc\_score(y\_test, y\_prob[:, 1],)  print('Logistic: ROC AUC=%.3f' % (auc))  # calculate roc curves  lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, y\_prob[:, 1])  plt.plot(lr\_fpr, lr\_tpr, marker='.', label='Logistic')  plt.plot([0,1], [0,1], '--', label='No Skill')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  plt.show() |

Show your confusion matrix and ROC curve here:

|  |
| --- |
| 0.934 |

At what point do significant gains in the true positive rate diminish?

|  |
| --- |
| Where the logistic slope decreases, somewhere around 2.2 to 2.3 |

Exercise (5 marks)

Say you were trying to reach at least 600 customers. What probability cut-off what you need? See Example 1 for a code hint. Show the confusion matrices that is displayed for the threshold that you select.

|  |
| --- |
|  |