Table of Contents

[Hypothesis Testing for Coefficients - OLS 1](#_Toc31781664)

[Logistic Regression 3](#_Toc31781665)

[Logistic Regression Assumptions 3](#_Toc31781666)

[Odds Ratio 3](#_Toc31781667)

[Logistic Model Introduction 4](#_Toc31781668)

[Selecting Predictors with Chi-Square 4](#_Toc31781669)

[Displaying Model Coefficients and Intercept 5](#_Toc31781670)

[Model Validation 5](#_Toc31781671)

[Adjusting the Cut-Off 8](#_Toc31781672)

[Interpreting the Result 9](#_Toc31781673)

[Precision, Recall, F-Score 11](#_Toc31781674)

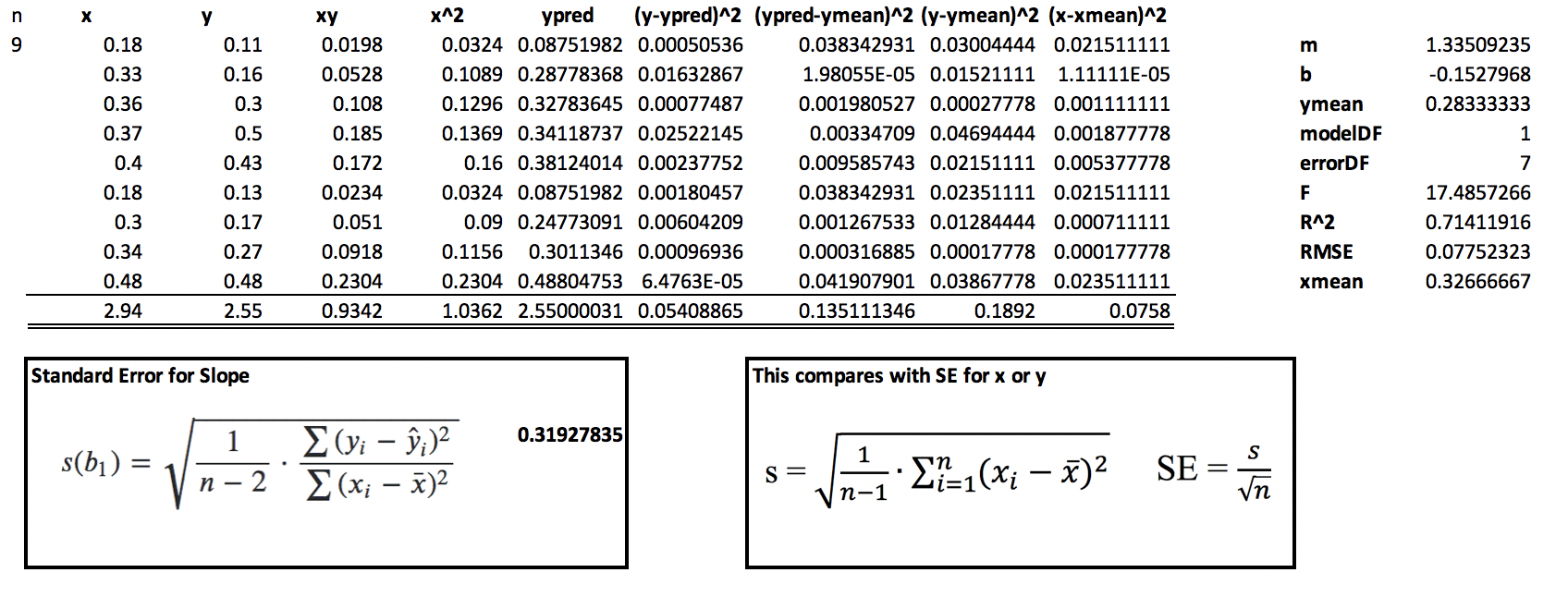
[Identifying Highly Likely Candidates 11](#_Toc31781675)

[F1-Score 12](#_Toc31781676)

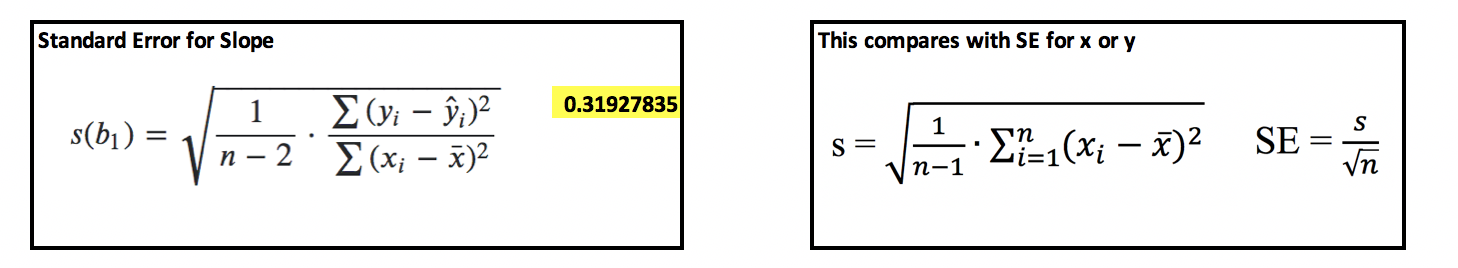
[Standardization 15](#_Toc31781677)

## Hypothesis Testing for Coefficients - OLS

This section explains the hypothesis test for the slope coefficients with a little more detail in case you are interested. First, here are the basic calculations which are required for determining the slope and intercept coefficients.

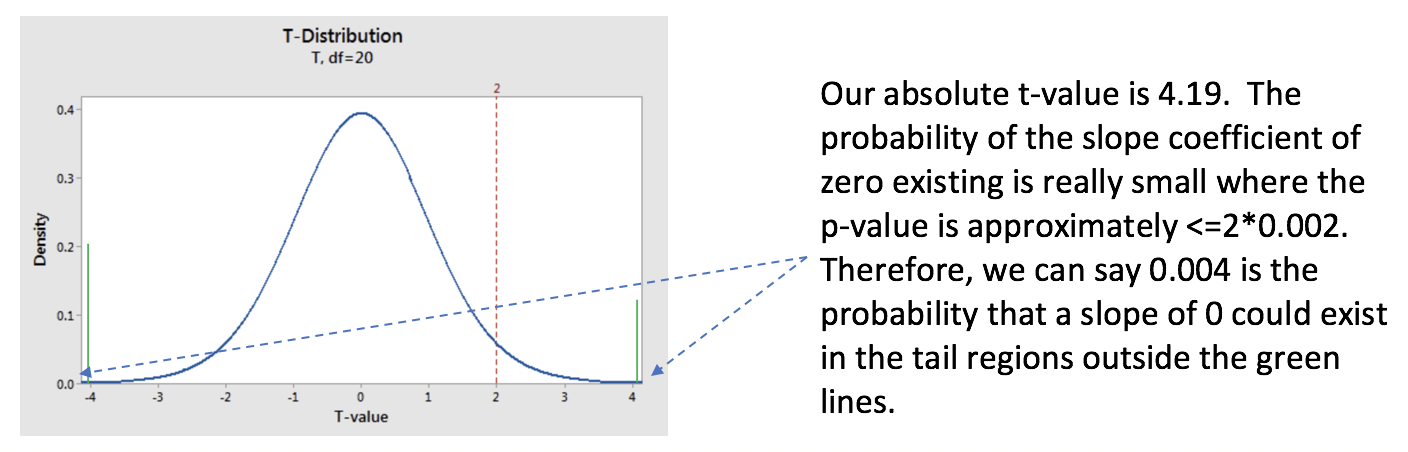


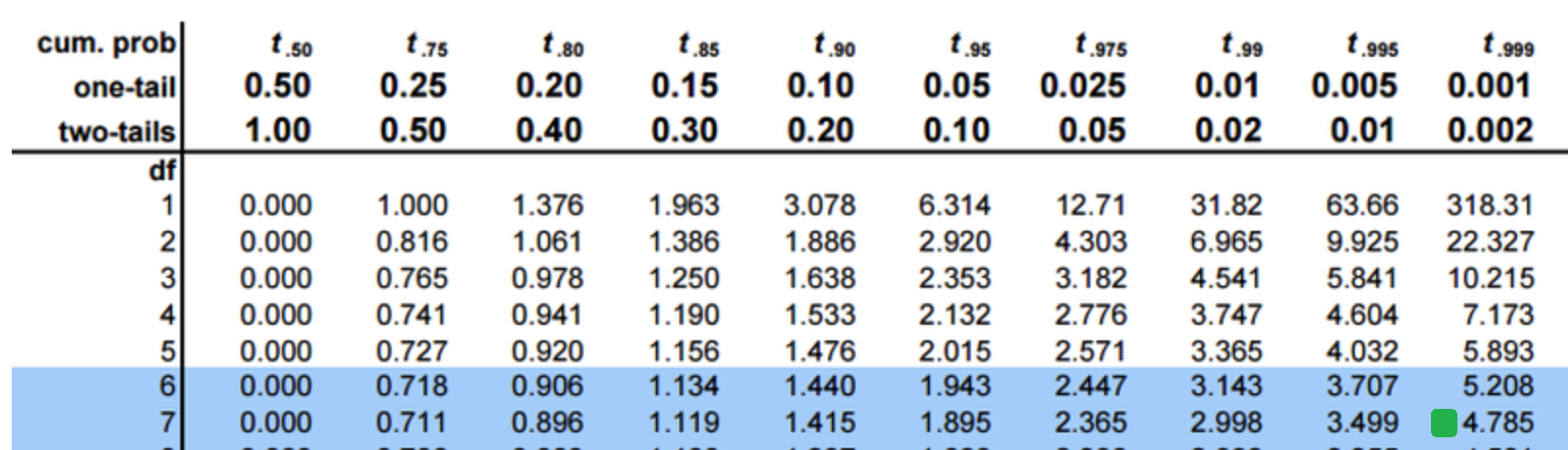
This is the calculation for the standard error of the slope on the left. We can plug in values from our spreadsheet above. We are looking at the average error for X and Y and there is a penalty of 2 degrees of freedom. The standard error equation is a little more complicated for the slope but it is comparable to the standard error equations for X and Y.



|  |  |
| --- | --- |
| These are our statistics for our model if we were to use all of the data for training. We can use this output to check our work. The SE value in our manual calculation is correct according to this output. | There are many possible slopes for our scatter of data but we want to know if our slope is 0 for the null hypothesis |
|  |  |

|  |  |
| --- | --- |
| We can test our slope by calculating the z score for it where our null hypothesis says = 0. is the standard deviation. | We don’t know the population standard deviation so we have to use the standard error and because of that we must use the t distribution. |
|  | t = = = 4.185266457680251 |





Since the probability the slope is zero is very small we can reject the null hypothesis and conclude that the slope of 1.335 is significant.

See: <https://www.khanacademy.org/math/ap-statistics/inference-slope-linear-regression/inference-slope/v/t-statistic-slope>

Explains the p-value:

<https://www.khanacademy.org/math/ap-statistics/inference-slope-linear-regression/inference-slope/v/making-conclusions-for-regression-slope-hypothesis-test>

## Logistic Regression

Logistic regression is a fundamental algorithm for making classifications. Unlike least squares regression, logistic regression **does not** predict continuous numeric variables such as the sale price of a home. Instead, logistic regression predicts a probability that ranges between 0 and 1. For example, predictions might determine:

* If an email is spam.
* If fraud has occurred.
* If a person has contracted a specific illness.
* Whether a user will respond to a marketing campaign.v

Slightly more advanced Logistic regression, which we won’t cover today, can select one of multiple categories such as:

* Whether a wine is 1, 2, 3, 4, 5, 6, 7 or 8 stars.
* Whether a spruce tree is a white, red or black spruce tree.

There are many algorithms that can perform classification and sometimes with better results. However, logistic regression uses a linear model, which has decent explanatory power.

## Logistic Regression Assumptions

* Binary logistic regression requires the dependent variable to be binary.
* For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
* Only the meaningful variables should be included.
* The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
* Logistic regression requires quite large sample sizes.

## Odds Ratio

At a very simple level, we could consider using an odds ratio to measure probability that an event occurred:

**Odds Ratio** = =

So, if a team has a 0.7 probability of winning the odds ratio is = 2.3333.

The problem with odds ratios is, as Probability(Occurred) increases, the odds ratio approaches . As Probability(!Occurred) increases, the odds ratio approaches . It is difficult to build a predictive model for such a wide range.

To avoid the issue of an infinite range, logistic regression uses a sigmoid function to restrict all predictions between 0 and 1. Because of the range, the sigmoid function is ideal for a probabilistic prediction (see Equation 1).

Equation : Sigmoid Function

p =

Figure 2 shows the classic S-shape for a sigmoid function where 0 In other words, for all possible values of x, y is always greater than 0 and less than 1.

Figure : Classic Sigmoid S-Shape With 0 1

|  |  |
| --- | --- |
|  | import matplotlib.pyplot as plt  import numpy as np  x = np.array([-20,-10,-6,-4,  -1,0,2,4,6,10,20])  y = 1/(1+np.exp(-x))  plt.plot(x, y)  plt.title('Sigmoid Function')  plt.xlabel("Y")  plt.ylabel("X")  plt.show() |

You may recognize **x in** Equation 1. This is a linear function called a logit. The coefficients of the logit function are weights. The sigmoid function basically takes our logit output and scales it to a range that is always between 0 and 1.

## Logistic Model Introduction

This section introduces basic logistic model regression and validation techniques.

Example : Logistic Regression Introduction

Here is a simple logistic regression example which uses GMAT, GPA and work experience to predict if the application will be admitted to an academic program.

### Selecting Predictors with Chi-Square

Before modeling, a quick blunt chi-square test runs for each predictor variable checks if each is significant in the model. For the current case, our chi-square scores appear as:

# GMT. # GPA #Work

﻿ Predictor Chi-Square Scores: [123.062 3.307 21.457]

For a p-value of 5% or less we would want the chi-square value to be 3.841 or higher. Based on this statistic, GMT and Work variables are selected for the model. GPA may be useful but we will leave it out for now.

### Displaying Model Coefficients and Intercept

The coefficients of the logit function include -0.004 for GMT and 1.177 for work experience. The intercept for the logit function is -0.727.

Model Coefficients:

[[-0.004 1.177]]

Intercept:

[-0.727]

### Model Validation

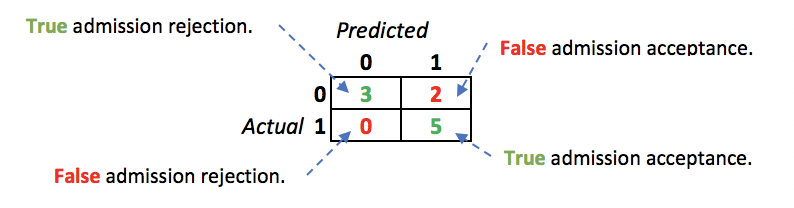
After the model is created, some basic accuracy statistics are displayed. For this example, 8 out of 10 predictions are correct.

Accuracy: 0.8

#### Confusion Matrices

With many classification algorithms, a confusion matrix compares actual versus predicted values. For this example, our target prediction is admission acceptance. The actual response row values are mapped with predicted response column values. The best possible outcome is to have all values in the diagonal from the top left to the bottom right where admissions rejections and acceptances are accurately predicted. For our example, the 8 correct predictions exist in the diagonal but two incorrect predictions exist at the top right of Figure 3.

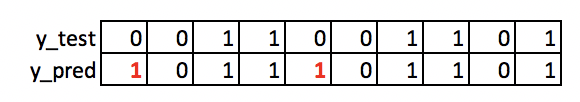
Figure : Confusion Matrix



The matrix accounts for all predictions in our test set.

Printing out our actual test and predicted values shows a clear breakdown of where differences between actual and predicted values occur:

Figure : Comparing Actual and Predicted Values

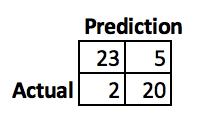


Here is the code for the example:

|  |
| --- |
| 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  # Setup data.  candidates = {'gmat': [780,750,690,710,680,730,690,720,  740,690,610,690,710,680,770,610,580,650,540,590,620,  600,550,550,570,670,660,580,650,660,640,620,660,660,  680,650,670,580,590,690],  'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,  3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,  3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,  3.3,3.3,2.3,2.7,3.3,1.7,3.7],  '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= ['gmat', 'gpa',  'work\_experience','admitted'])  print(df)  # Separate into x and y values.  X = df[['gmat', 'gpa','work\_experience']]  y = df['admitted']  # Import the necessary libraries first  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  #You imported the libraries to run the experiments. Now, let's see it in action.  # 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[['gmat', 'work\_experience']]  # 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)  # 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)  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix) |

Exercise (1 mark)

What is the accuracy of logistic regression with the following confusion matrix?



|  |
| --- |
| 23 + 20 are accurate  7 are false.  Observed rows = 43+ 7 = 50  43/50 = 86/100 = 0.86  Accuracy is 86% |

Exercise (2 marks)

Manually create a confusion matrix for the following set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Actual** | 1 | 0 | 1 | 1 | 0 |
| **Predicted** | 1 | 1 | 0 | 1 | 1 |

Show your confusion matrix output here:

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual** |  | **Predicted** | |
| **0** | 0 | 1 |
| **1** | 2 | 2 |
|  | **0** | **1** |

Exercise (1 mark)

What is the accuracy rate for the confusion matrix that is drawn in Exercise 2?

|  |
| --- |
| 2/5 = |

### Adjusting the Cut-Off

The prediction set in Example 1 is actually based on probabilities where the cut-off between a 0 and 1 response is a default of 0.5. We can, however, actually manually compute the predictions and a confusion matrix with a different cut-off. Using a different cut-off can sometimes improve accuracy.

Example : Adjusting the Cut-Off

This example shows that we can slightly boost accuracy with a custom probability cut-off. For this case, if we adjust the required probability for a response of ‘1’ to 60% the accuracy improves.

The output first displays the prediction probabilities of 0 and the probabilities of 1. These values are generated with the *logisticModel.predict\_proba(X\_test)* instruction.

﻿Prediction probability set:

[[0.153 0.847] # Actual value is 0 <- We still fail to predict this value correctly.

[0.717 0.283] # Actual value is 0

[0.027 0.973] # Actual value is 1

[0.235 0.765] # Actual value is 1

[0.428 0.572] # Actual value is 0 <- This prediction now accurate with a 60% cutoff.

[0.887 0.113] # Actual value is 0

[0.025 0.975] # Actual value is 1

[0.09 0.91 ] # Actual value is 1

[0.645 0.355] # Actual value is 0

[0.08 0.92 ]] # Actual value is 1

\*\*\* Accuracy with CUTOFF of 0.6: 0.9

Confusion Matrix: actual (row) vs predicted (col)

[[4. 1.]

[0. 5.]]

Here is the code which generates the output. To implement this code, place it after the code in Example 1.

|  |
| --- |
| #-------------------------------------------------------------  # 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    # Extract probabilities.  y\_proba = logisticModel.predict\_proba(X\_test)  print("\nPrediction probability set:")  print(y\_proba)  RESPONSE\_IS\_ONE = 1  probaOne = y\_proba[:, RESPONSE\_IS\_ONE]  SIZE = 2 # for 2x2 matrix  CUT\_OFF = 0.6  updatedPredict = predictUsingAlternateCutoff(y\_test, probaOne, CUT\_OFF, SIZE) |

## Interpreting the Result

We have generated coefficients for a logit function that plugs into a sigmoid function. The logit function is actually the log of the odds radio.

Logit Function = ln( +

Log(odds ratio) = -0.727 - 0.004

At a glance the result may seem difficult to interpret but actually the logistic model can be explained nicely. When we interpret the coefficients of our model from Example 1 we can say the odds of an academic acceptance actually decrease with the GMAT score but acceptance probability increases with work experience. Perhaps a higher GMAT score indicates that the applicant is overqualified for the academic program. Ideally this finding would be confirmed with a subject matter expert.

#### Proofing Out the Logic

There is no need to commit this proof to memory but I will explain it at a high level so you can trust the logic. The logit function is actually the log of the odds ratio:

Logit Function = ln( +

Since we can say;

Odds Ratio = = =

By re-arranging the equations for probability and the odds-ratio we can also say:

(1-p) (odds ratio) = p

(odds ratio) – p (odds ratio) = p

odds ratio = p + p (odds ratio)

odds ratio = p (1 + odds ratio)

p =

By subbing in the logit function inside the odds-ratio we get the sigmoid function which restricts the result to a range between 0 and 1.

p = =

That’s pretty tough to follow at a first glance but there is no need to memorize the logic. What really matters is that you have confidence in one of the most basic classification algorithms.

Example : Confirming Our Equation Results

To help verify the equations are set properly for Example 1, this example plugs in the **last** *gmat* and *work\_experience* pair from *X\_test*. The listing below shows the last two *gmat* and *work\_experience* pairs of *X\_test*.

﻿ **gmat work\_experience**

…

﻿540 2

**660 5**

The output observed confirms that the probability of a response of acceptance (1) can be calculated three different ways.

﻿Probability calculated with sigmoid: 0.9253940925245717

Probability calculated with odds ratio: 0.9253940925245717

Probability calculated by model: [0.08 0.92]

To implement this solution, add this code to the end of the code from Example 1.

|  |
| --- |
| ﻿﻿# Calculate probability of a 1 response with sigmoid function.  probSigmoid = 1/(1+np.exp( -1\*(INTERCEPT-0.004\*GMAT+1.177\*WORK)))  print("\nProbability calculated with sigmoid: " + str(probSigmoid))  # Calculate probability of a 1 response with odds ratio.  odds = np.exp(INTERCEPT-0.004\*GMAT+1.177\*WORK)  probOdds = odds/(1+odds)  print("Probability calculated with odds ratio: " + str(probOdds))  # Calculate probability of a 1 response with model.  y\_proba = logisticModel.predict\_proba(X\_test)  print("Probability calculated by model: " + str(y\_proba[TEST\_INDEX])) |

Exercise (2 marks)

Run the code for Example 3 again but this time use the 9th pair of GMAT and work\_experience values from the X\_test set. Show your output here:

|  |
| --- |
|  |

Print out the values for y\_test and show it here. The 9th value should correspond nicely with the prediction output displayed above.

|  |
| --- |
|  |

## Precision, Recall, F-Score

In addition to validating a model with an accuracy score and confusion matrix, we have other measures to obtain a clearer understanding of model reliability. When making predictions for large data sets, the accuracy score of a classifier model may not be high. However, the model may still be able to identify sample candidates that have a high likelihood of yielding a positive response of 1.

### Identifying Highly Likely Candidates

For a case like a marketing campaign, the estimated response for many candidates may be quite neutral but a segment of the market could be highly likely to respond. When a marketing budget is limited it may still be very profitable to target a small number of candidates who are most likely to respond. On the other hand, you may also determine that there is a desirable cost-benefit for contacting all candidates in the market.

#### Precision

The precision is the ratio of true positive results over the sum of true and false positive results:

The precision score rates classifier’s ability to not label a sample as positive if it is negative.

#### Recall

The recall ratio is:

The recall score rates the ability of the classifier to find all the positive samples. A low recall rate indicates that a significant number of positive responses where missed. For a marketing campaign, a low recall rate means that a large amount of potential revenue has been overlooked.

### F1-Score

A high F-score is as a weighted mean of the precision and recall scores. The F-score is at its best value at 1 and worst at 0.

*F1 Score =*

Example : Measuring Precision, Recall and F Score

This example builds a model to predict whether consumers will respond to a computer marketing campaign to make a purchase. Accuracy, precision and F1 scores are used to evaluate the model.

The output shows an accuracy rating of 61% which seems decent.However, the confusion matrix shows only 3 positive responses were predicted.

\*\*\* Accuracy with CUTOFF of 0.48: 0.61

Confusion Matrix: actual (row) vs predicted (col)

[[58. 10.]

[29. 3.]]

The score output from the program displays other numbers but the ones we normally care about during a binary classification are for a prediction of 1.

precision recall f1-score support

0 0.67 0.85 0.75 68

1 **0.23** **0.09** **0.13**32

Precision Rate

The low precision rate shows the model is actually very poor at predicting true positive responses of 1.

= = = **0.23**

Recall Rate

The low recall rate indicates that a very large proportion of potential consumers were missed.

= = = **0.09**

F1-Score

The low F1 score indicates a poor performance:

*F1 Score = = =* **0.13**

Here is the code that generates the model and displays the model score results:

|  |
| --- |
| 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  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  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')  # Fit the model.  logisticModel.fit(X\_train,y\_train)  # y\_pred=logisticModel.predict(X\_test)  # 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    # Extract probabilities.  y\_proba = logisticModel.predict\_proba(X\_test)  print("\nPrediction probability set:")  RESPONSE\_IS\_ONE = 1  probaOne = y\_proba[:, RESPONSE\_IS\_ONE]  SIZE = 2 # for 2x2 matrix  CUT\_OFF = 0.48  updatedPredict = predictUsingAlternateCutoff(y\_test, probaOne, CUT\_OFF, SIZE)  from sklearn.metrics import classification\_report  print(classification\_report(y\_test, updatedPredict))  from sklearn.metrics import average\_precision\_score  average\_precision = average\_precision\_score(y\_test, updatedPredict)  print('Average precision-recall score: {0:0.2f}'.format(  average\_precision)) |

Show output with precision etc.

|  |
| --- |
|  |

Exercise (8 marks)

Given the following confusion matrix, manually calculate the accuracy score, precision score, recall score and f1-score:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Predicted | |
| Actual |  | 0 | 1 |
| 0 | 32 | 10 |
| 1 | 3 | 15 |

Show your accuracy score here (please include the calculations):

|  |
| --- |
| 32+15 = 47  10+3+47 = 60  Accuracy Score = 47/60 |

Show your precision score here (please include the calculations):

|  |
| --- |
|  |

Show your recall score here(please include the calculations):

|  |
| --- |
|  |

Show your f1-score here (please include the calculations):

|  |
| --- |
|  |

## Standardization

Standardization is a useful technique to transform attributes with normal-like distributions and differing means and standard deviations to a standard normal-like distributions with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis.

Example : Standardizing the Computer Marketing Dataset

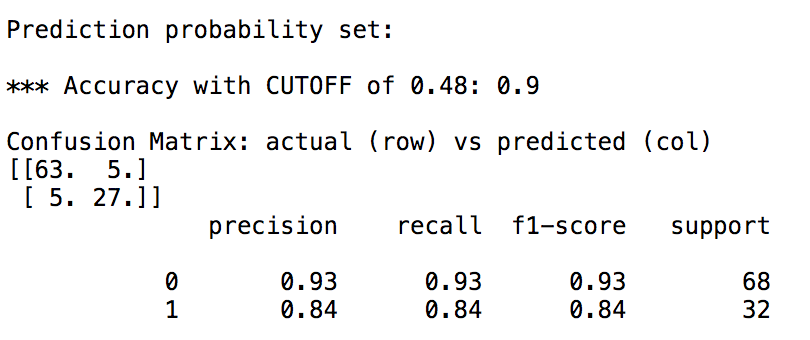
The Age and EstimatedSalary columns of the computerPurchase.csv data set are somewhat normal like but the scales are very different. These predictors may be good candidates for scaling.



Later add this highlighted code to Example 4 just above the highlighted instruction logisticModel.fit()

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  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) |

The output shows noticeably better results:



Exercise (8 marks)

Perform a quick logistic regression analysis using the fluDiagnosis.csv dataset. You may use 0.5 if you want. You can also apply scaling if you want or not. Show your confusion matrix and state your cut-off value here:

|  |
| --- |
|  |

State the accuracy rating and manually calculate it here:

|  |
| --- |
|  |

Show your manual calculations for precision here. Explain why the precision is good or bad.

|  |
| --- |
|  |

Show your manual calculations for recall here. Explain why the recall rate is good or bad.

|  |
| --- |
|  |