Table of Contents

[Cumulative Gains and Lift Charts 1](#_Toc32387059)

[Multinomial Logistic Regression 6](#_Toc32387060)

[Equation Generation 8](#_Toc32387061)

[Scaling 10](#_Toc32387062)

[The Problem with Unscaled Data During Logistic or OLS Modelling 10](#_Toc32387063)

[Comparing MinMax, Standard and Robust Scalers 16](#_Toc32387064)

[Box-Cox Transformation 16](#_Toc32387065)

[Cross Fold Validation 17](#_Toc32387066)

## Cumulative Gains and Lift Charts

Cumulative gains and lift charts are visual aids to help monitor model performance. These charts target the most likely candidates first while showing the diminishing returns of the model. This section is based on the example that is posted at <http://www2.cs.uregina.ca/~dbd/cs831/notes/lift_chart/lift_chart.html>

The example presents a company mail-out campaign. Each mail-out costs $1. You want to contact 100,000 people. You predict 20,000 will respond with your model. Without a model you predict that you will only reach 10,000 out of 100,000 people. You rank the customers into 10 groups in order of their likelihood to respond. The most likely candidates are listed first. In the first bucket the campaign acquires 6,000 out of 20,000 potential responses which is 30% of the total responses. The baseline random guess without a model suggests that you will only reach 10% of the potential customers each bucket. The expected lift is 30/10 = 3. Table 1 shows the % cumulative gain and lift calculations.

Table 1: Cumulative Gains and Lift Calculations

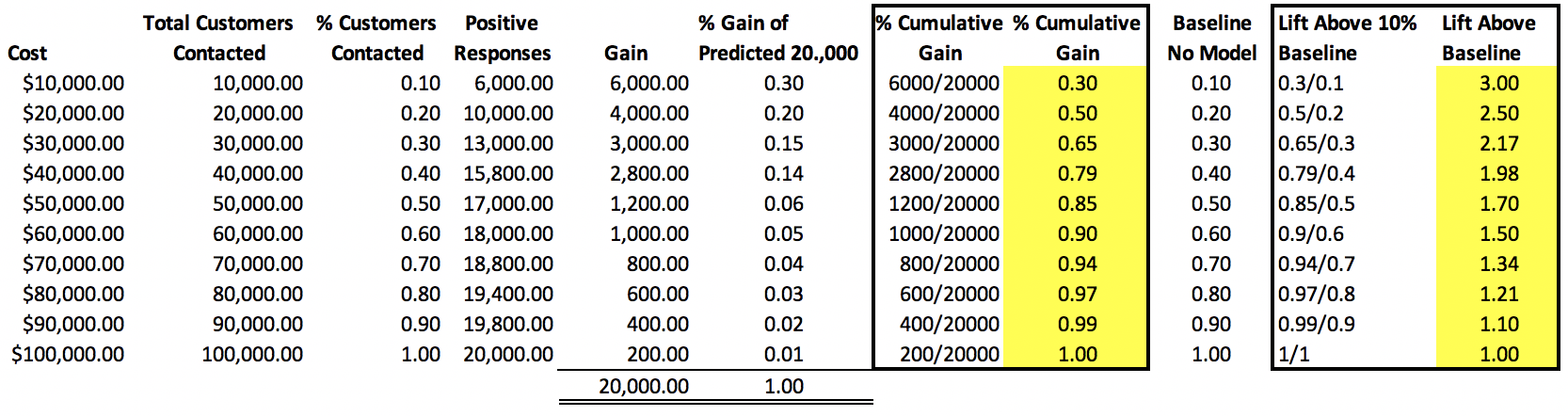


Figure 1: Anticipated Cumulative Gains and Lift for Each Bucket of Target Respondents

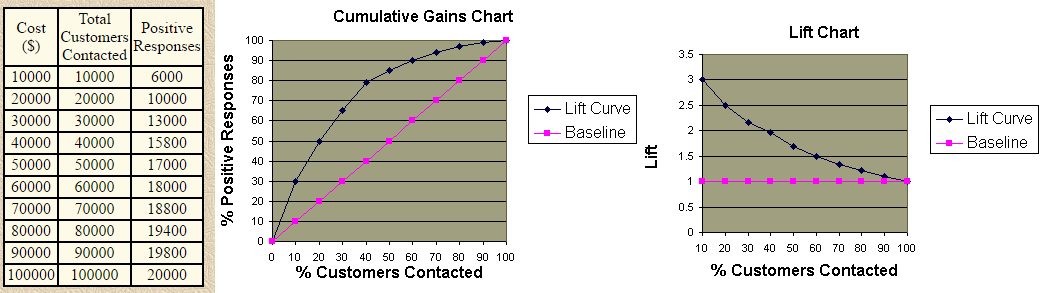


Figure 2 shows a summary of desirable and undesirable cumulative gains curve shapes.

Figure 2: Cumulative Gains Shapes

|  |  |  |  |
| --- | --- | --- | --- |
| Good. | Too perfect. | Overfit. | Worthless. |
|  |  |  |  |

Example 1: Cumulative Gains Chart

This example shows how to automate the process of drawing a cumulative gains chart for a model which predicts whether a breast cancer growth is benign (harmless) or not. The cumulative gains chart in Figure 3 indicates that about most of the potential malignant (harmful) cases can be identified within the first 40% of all samples. Almost all benign samples can be identified within the first 65% of all samples.

Figure 3: Cumulative Gains and Lift Charts Compared with the Confusion Matrix

|  |  |  |
| --- | --- | --- |
|  |  | Predicted 0 1  Actual  0 74 5  1 3 106 |

Here is the code that is used to draw the cumulative gains chart:

|  |
| --- |
| # The usual train-test split mumbo-jumbo  from sklearn.datasets import load\_breast\_cancer  from sklearn.linear\_model import LogisticRegression  from sklearn.model\_selection import train\_test\_split  import pandas as pd  X, y = load\_breast\_cancer(return\_X\_y=True)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.33)  clf = LogisticRegression(  random\_state=0, multi\_class='multinomial', solver='newton-cg')  clf.fit(X\_train, y\_train)  predicted\_probas = clf.predict\_proba(X\_test)  y\_pred = clf.predict(X\_test);  # The magic happens here  import matplotlib.pyplot as plt  import scikitplot as skplt  skplt.metrics.plot\_cumulative\_gain(y\_test, predicted\_probas)  skplt.metrics.plot\_lift\_curve(y\_test, predicted\_probas)  plt.show()  # Show confusion matrix and accuracy scores.  cm = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print(cm) |

Exercise 1 (8 marks)

Run this code for predicting a response to a bank marketing campaign:

|  |
| --- |
| from sklearn.linear\_model import LogisticRegression  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, MinMaxScaler  from sklearn.metrics import roc\_auc\_score, roc\_curve  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  from sklearn import metrics  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[["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\_mar",  "month\_may",  "month\_nov",  "day\_of\_week\_mon",  "poutcome\_failure",  "poutcome\_success"]]  y = df[['target']]  XScaled = MinMaxScaler().fit\_transform(X)  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  XScaled, 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.values.ravel())  # Show model coefficients and intercept.  print("\nModel Intercept: ")  print(logisticModel.intercept\_)  print("\nModel Coefficients: ")  print(logisticModel.coef\_)  y\_pred = logisticModel.predict(X\_test)  y\_prob = logisticModel.predict\_proba(X\_test)  # Show confusion matrix and accuracy scores.  cm = pd.crosstab(np.ravel(y\_test), y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ', metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(cm)  from sklearn.metrics import classification\_report  print(classification\_report(y\_test, y\_pred))  from sklearn.metrics import average\_precision\_score  average\_precision = average\_precision\_score(y\_test, y\_pred)  print('Average precision-recall score: {0:0.2f}'.format(  average\_precision))  # 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()  plt.show()  # The magic happens here  import matplotlib.pyplot as plt  import scikitplot as skplt  skplt.metrics.plot\_cumulative\_gain(y\_test, y\_prob)  #skplt.metrics.plot\_lift\_curve(y\_test, y\_prob)  plt.show() |

Show the confusion matrix and accuracy score here:

|  |
| --- |
|  |

Show the precision and recall scores for a positive response here:

|  |
| --- |
|  |

How do the precision and recall scores offer a clearer view of the model’s effectiveness when compared with the accuracy score?

|  |
| --- |
|  |

Show your ROC curve here:

|  |
| --- |
|  |

Show the cumulative gains curve here:

|  |
| --- |
|  |

What is the baseline for a positive response for the marketing campaign and how is it calculated?

|  |
| --- |
|  |

How does the cumulative gains curve present a clearer view of model effectiveness (if at all) when compared to the ROC curve? Also, how do the negative and positive response curves in the cumulative gains chart differ and why is each curve shaped very differently when compared to the baseline?

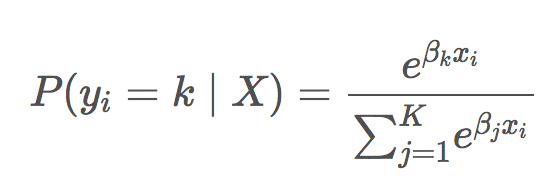
|  |
| --- |
|  |

How do the ROC curve and cumulative gains curve describe a similar view of the model effectiveness?

|  |
| --- |
|  |

## Multinomial Logistic Regression

Multinomial regression allows you to make predictions about more than two classes. In multinomial logistic regression (MLR) the logistic function from last day is replaced with a softmax function:



Where P(=k|X) is the probability that ith observation’s target value, is class k and K is the total number of classes.

Example 2: Multinomial Regression

This example is based on multinomial (multi-class) logistic regression. The tutorial is one of the shortest but most effective tutorials on multinomial regression I have seen. It is a 30-second read.

<https://chrisalbon.com/machine_learning/naive_bayes/multinomial_logistic_regression/>

The goal of this regression is to predict three types of iris flowers: 'setosa', 'versicolor', 'virginica'.



The features are:

* sepal length (cm)
* sepal width (cm)
* petal length (cm)
* petal width (cm)

Here is the code – the explanation for how it works is in the comments:

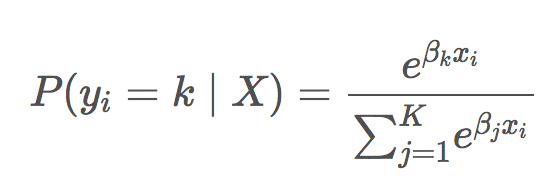
|  |
| --- |
| # Load libraries  from sklearn.linear\_model import LogisticRegression  from sklearn import datasets  from sklearn.preprocessing import StandardScaler  from sklearn.model\_selection import train\_test\_split  from sklearn import metrics  import pandas as pd  # Load data  iris = datasets.load\_iris()  X = iris.data  y = iris.target  # Split data.  X\_train,X\_test,y\_train,y\_test = train\_test\_split(  X, y, test\_size=0.25,random\_state=0)  # Standarize features  scaler = StandardScaler()  Xtrain\_scaled = scaler.fit\_transform(X\_train)  Xtest\_scaled = scaler.fit\_transform(X\_test)  # Create one-vs-rest logistic regression object  clf = LogisticRegression(  random\_state=0, multi\_class='multinomial', solver='newton-cg')  # Train model  model = clf.fit(Xtrain\_scaled, y\_train)  # Predict class  y\_pred = model.predict(Xtest\_scaled)  print(y\_pred)  # View predicted probabilities  y\_prob = model.predict\_proba(Xtest\_scaled)  print(y\_prob)  # Show confusion matrix and accuracy scores.  cm = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print(cm) |

The confusion matrix shows a highly accurate result.

|  |
| --- |
| Predicted 0 1 2  Actual  0 13 0 0  1 0 15 1  2 0 0 9  Accuracy: 0.9736842105263158 |

### Equation Generation

As mentioned, the equation for multinomial logistic regression is:



For our case, k is the class 'setosa', 'versicolor' or 'virginica'. This next example explains how to generate and interpret the model equation.

Example 3: Multinomial Logistic Regression Equation Generation

As explained previously, the logit function is the log of the odds ratio which is: .

Scaled Sepal Length Scaled Sepal Width

Scaled Petal Length Scaled Petal Height

The probability equations are:

P(=setosa|) =

P(=versicolor|) =

P(=virginica|) =

To display the intercepts and model coefficients for the model, add this code to the end of Example 2:

|  |
| --- |
| print("\nIntercept")  print(model.intercept\_)  print("\nModel coefficients")  print(model.coef\_) |

Here are three separate intercepts and three sets of coefficients. One set of coefficients exists for each of the three iris classes:

|  |
| --- |
| Intercept  [-0.42172234 1.76983851 -1.34811617]  Model coefficients  [[-1.02163024 1.0430488 -1.77994737 -1.65927236]  [ 0.5420308 -0.3599358 -0.26681264 -0.71830735]  [ 0.47959944 -0.683113 2.04676001 2.37757971]] |

These intercepts can coefficients can then be used to create the logit (log odds) functions for each class:

Logit(setosa) = 0.42172234 + 1.0430488

Logit(versicolor) = +1.76983851 + 0.5420308

Logit(virginica) = 1.34811617 + 0.47959944

Our probability equation for each class becomes:

P(=k|) =

Example 4: Proofing Out the Equations

To help proof out the probability equations for each Iris class, this code manually calculates probability values for each class with the Xtest\_scaled data as input. The output from these calculations can be compared with the y\_prob data from Example 2. To implement the manual calculations using the model coefficients, append this code to the code from Example 2.

|  |
| --- |
| import numpy as np  def calculateProbabilities(X\_test):  X1 = X\_test[0]  X2 = X\_test[1]  X3 = X\_test[2]  X4 = X\_test[3]    logit1 = -0.42172234 + -1.02163024\*X1 + 1.0430488\*X2 \  -1.77994737\*X3 -1.65927236\*X4  logit2 = 1.76983851 + 0.5420308\*X1 -0.3599358\*X2 -0.26681264\*X3 \  -0.71830735\*X4  logit3 = -1.34811617 + 0.47959944\*X1 -0.683113\*X2 +2.04676001\*X3 \  + 2.37757971\*X4    denominator = 1 + np.exp(logit1) + np.exp(logit2) + np.exp(logit3)  prediction1 = np.exp(logit1)/denominator  prediction2 = np.exp(logit2)/denominator  prediction3 = np.exp(logit3)/denominator    print(str(prediction1) + " " + str(prediction2) + " " + str(prediction3))  calculateProbabilities(Xtest\_scaled[0,:])  calculateProbabilities(Xtest\_scaled[1,:])  calculateProbabilities(Xtest\_scaled[2,:])  calculateProbabilities(Xtest\_scaled[3,:]) |

Table 2 compares the manual calculation output with the auto-generated probabilities. I notice a slight rounding error which I have been unable to eliminate so if you find a better more accurate way please let me know.

Table 2: Manually Calculated Probability Scores (Left) Versus Auto-Generated Probability Scores (Right)

|  |  |
| --- | --- |
| 1.5422e-05 0.0085 0.9864  0.00258 0.8768 0.067195  0.994098 0.00518 7.1195e-08  1.5592e-06 0.02767 0.96886 | [[1.5501e-05 8.584e-03 9.914e-01]  [2.7286e-03 9.2629e-01 7.0986e-02]  [9.9481e-01 5.1899e-03 7.1246e-08]  [1.5646e-06 2.7765e-02 9.7223e-01] |

## Scaling

Since switching to logistic regression, the benefit from scaling and the need for it has been increasingly apparent. Many machine learning algorithms perform better or converge faster when features are on a relatively similar scale and/or close to normally distributed. Last week, to enable recursive feature elimination the scaling was actually required.

### The Problem with Unscaled Data During Logistic or OLS Modelling

Without scaling the weighting of independent features may be may be disproportionate. Linear and logistic regression can often benefit from scaling. A drawback from some forms of scaling though is a loss of interpretability.

Example 5: Unscaled Data

In this example which uses the computer purchase data set, a problem with unscaled data is evident. We previously determined that the significant predictors for a computer purchase are *age* and estimated *salary*. The range of numbers for estimated salary is much larger than age. When creating a logistic regression without scaling the model fails to predict any computer purchases:

Accuracy: 0.68

|  |  |  |
| --- | --- | --- |
| Actual | Predicted 0  0 68  1 32 |  |

Even though salary and age distributions are somewhat shaped the same, the right image in Figure 4 highlights the difference in magnitude which likely is causing unbalanced weights in the model.

Figure 4: Salary and Age Distributions Without Scaling

|  |  |  |
| --- | --- | --- |
| **Salary No Scaling** | **Age No Scaling** | **Together Before Scale** |
|  |  |  |

Here is the code which makes the prediction using age and salary without scaling:

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  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"))  # Separate into x and y values.  X = df[["Age", "EstimatedSalary"]]  y = df['Purchased']  # 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 confusion matrix and accuracy scores.  cm = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(cm) |

To correct the problem described above we will look at three scaling algorithms:

1. MinMaxScaler
2. StandardScaler
3. RobustScaler

#### MinMax Scaler

MinMaxScaler preserves the shape of the original distribution. It does not meaningfully change the information embedded in the original data. MinMaxScaler essentially restricts the range between 0 and 1 (or -1 to 1 if there are negative values). If the distribution is not Gaussian (normal) or the standard deviation is very small, the min-max scaler is usually recommended.

The equation for MinMaxScaler is:

Table 3 highlights predictor values and accuracy results without and with MinMaxScaler.

Table 3: Computer Purchase Prediction Accuracy Without and With MinMaxScaler

|  |  |
| --- | --- |
| **No Scaling** | **MinMaxScaler** |
| Accuracy: 0.68  Predicted 0  Actual  0 68  1 32 | Accuracy: 0.89  Predicted 0 1  Actual  0 67 1  1 10 22 |
|  |  |
| Age EstimatedSalary  mean 37.66 69742.50  std 10.48 34096.96  min 18.00 15000.00  25% 29.75 43000.00  50% 37.00 70000.00  75% 46.00 88000.00  max 60.00 150000.00 | Age Salary  mean 0.47 0.41  std 0.25 0.25  min 0.00 0.00  25% 0.28 0.21  50% 0.45 0.41  75% 0.67 0.54  max 1.00 1.00 |

The second diagram from the right in Figure 5 highlights an enormous imbalance between magnitude for age and salary. The diagram at the very right though shows how MinMaxScaler produces distributions that maintain the original shape while sizing each predictor on a similar scale.

Figure 5: Before and After MinMaxScaler

|  |  |  |  |
| --- | --- | --- | --- |
| **Salary No Scaling** | **Age No Scaling** | **Together Before Scale** | **MinMaxScaler** |
|  |  |  |  |

Example 6: Manually Applying the MinMaxScaler

Here is a sample calculation which applies MinMaxScaler to an estimated salary value of 57,000.

Salary = 57,000

Min. = 15,000

Max. = 60,000

MinMaxScaler = = = = 0.3111

To enable MinMaxScaler, replace the code which splits the data into test and training sets in Example 5 with this version:

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  sc\_x = MinMaxScaler()  X\_Scale = sc\_x.fit\_transform(X)  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X\_Scale, y, test\_size=0.25, random\_state=0) |

Exercise 2 (1 mark)

Show the MinMaxScaler calculation for age of 19. Please show your calculations. Examine the unscaled and scaled data set from Example 5 and Example 6 to check your work.

|  |
| --- |
|  |

Exercise 3 (1 mark)

Show the MinMaxScaler calculation for a salary of 15,000. Examine the unscaled and scaled data set from Example 5 and Example 6 to check your work.

|  |
| --- |
|  |

#### StandardScaler

Deep learning algorithms often call for zero mean and unit variance. Regression-type algorithms also benefit from normally distributed data with small sample sizes. The StandardScaler algorithm outputs something very close to a normal distribution by changing the values so the distribution standard deviation from the mean equals one. StandardScaler does distort the relative distances between the feature values which may not be desirable in many cases.

Example 7: StandardScaler

This equation applies standard scaling:

z =

Table 4 shows results with no scaling and with StandardScalar. Notice how the mean is 0 and standard deviation is 1 for StandardScaler output.

Table 4: Comparing Accuracy and Predictor Values with No Scaling with StandardScalar

|  |  |
| --- | --- |
| **No Scaling** | **StandardScaler** |
| Accuracy: 0.68  Predicted 0  Actual  0 68  1 32 | Accuracy: 0.89  Predicted 0 1  Actual  0 65 3  1 8 24 |
| Age EstimatedSalary  mean 37.66 69742.50  std 10.48 34096.96  min 18.00 15000.00  25% 29.75 43000.00  50% 37.00 70000.00  75% 46.00 88000.00  max 60.00 150000.00 | Age EstimatedSalary  mean -0.00 -0.00  std 1.00 1.00  min -1.88 -1.61  25% -0.76 -0.79  50% -0.06 0.01  75% 0.80 0.54  max 2.13 2.36 |

As you can see in Figure 6, the output from scaling is a transformation which creates similarly sized and shaped distributions for both predictor variables.

Figure 6: Comparing Distributions Before and After StandardScaler

|  |  |  |  |
| --- | --- | --- | --- |
| **Salary No Scaling** | **Age No Scaling** | **Together Before Scale** | **StandardScaled** |
|  |  |  |  |

Here is a sample StandardScaler calculation for a salary of $50,000.

Salary = 50,000

Mean = 69,742.50

StandardScaled Salary = = = -0.571091968316237

To enable StandardScaler, replace the two lines that split the data in Example 5: Unscaled Data with this version:

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  sc\_x = StandardScaler()  X\_Scale = sc\_x.fit\_transform(X)  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X\_Scale, y, test\_size=0.25, random\_state=0) |

Exercise 4 (1 mark)

Show the StandardScaler calculation to scale the age of 19. Please show your calculations. Examine the original and scaled data sets from Example 5 and Example 7 to check your work.

|  |
| --- |
|  |

Show the StandardScaler calculation for a salary of 15,000.

|  |
| --- |
|  |

#### RobustScaler

RobustScaler transforms the feature vector by subtracting the median and then dividing by the interquartile range (75% value — 25% value). RobustScaler users less data so it reduces the influence of outliers.

RobustScaler = =

Here are the results without and with scaling:

|  |  |
| --- | --- |
| **No Scaling** | **RobustScaler** |
| Accuracy: 0.68  Predicted 0  Actual  0 68  1 32 | Accuracy: 0.9  Predicted 0 1  Actual  0 66 2  1 8 24 |
|  |  |
| Age EstimatedSalary  mean 37.66 69742.50  std 10.48 34096.96  min 18.00 15000.00  25% 29.75 43000.00  50% 37.00 70000.00  75% 46.00 88000.00  max 60.00 150000.00 | Age EsitmatedSalary  mean 0.04 -0.01  std 0.65 0.76  min -1.17 -1.22  25% -0.45 -0.60  50% 0.00 0.00  75% 0.55 0.40  max 1.42 1.78 |

For this case, the distributions do not change much since there are no significant outliers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Salary No Scaling** | **Age No Scaling** | **Together Before Scale** | **RobustScaler** |
|  |  |  |  |

Here is a sample RobustScaler calculation for salary:

Salary = 57,000

(x) = 43,000 (x) = 88,000

= =

RobustScaler = = = = -0.29

Example 8: RobustScaler

To enable the RobustScaler replace the test/train split code in Example 5 with this version.

|  |
| --- |
| from sklearn.preprocessing import RobustScaler  sc\_x = RobustScaler()  X\_Scale = sc\_x.fit\_transform(X)  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X\_Scale, y, test\_size=0.25, random\_state=0) |

Exercise 5 (1 mark)

Show the RobustScaler calculation for age of 19. Please show your calculations. Examine the unscaled data in Example 5 and the scaled data set in Example 8 to check your work.

|  |
| --- |
|  |

Exercise 6 (1 mark)

Show the RobustScaler calculation for a salary of 15,000. Examine the unscaled data in Example 5 and the scaled data set in Example 8 to check your work.

|  |
| --- |
|  |

### Comparing MinMax, Standard and Robust Scalers

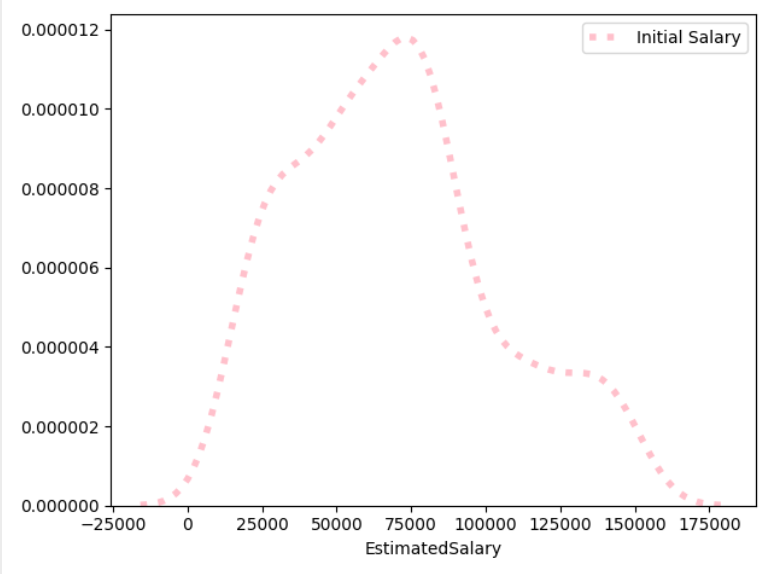
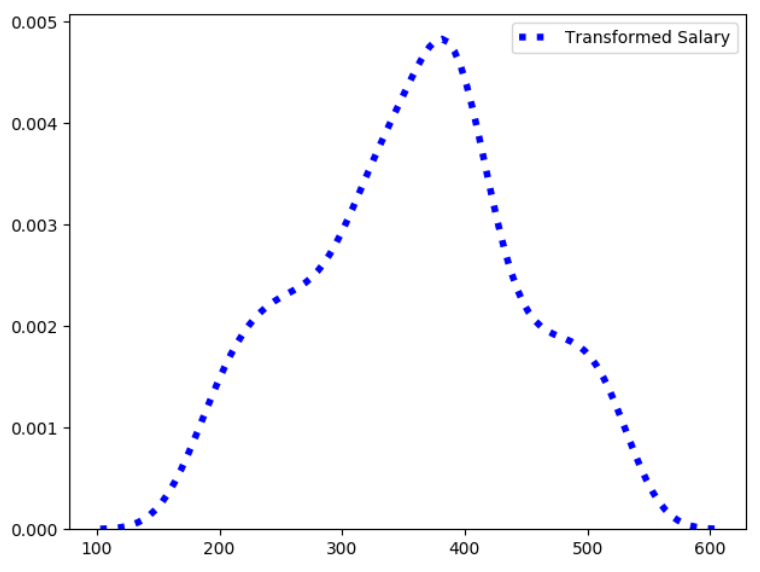
* Use MinMaxScaler as the default if you are transforming a feature. It is non-distorting.
* Use RobustScaler if you have outliers and want to reduce their influence. However, you might be better off removing the outliers, instead.
* Use StandardScaler if you need a relatively normal distribution.

## Box-Cox Transformation

A Box-Cox transformation is a way to transform non-normal dependent variables into a more normal shape. Many of the machine learning algorithms perform better with normalized data. However, the distortion of an original dataset can lead to a loss of interpretability.

Example 9: Implementing the Box-Cox Algorithm

This example demonstrates how to implement a Box-Cox algorithm to normalize the salary data from Example 5.

Replace the test/train split code in Example 5 with this version to enable the transformation:

|  |
| --- |
| from scipy import stats  X\_TransformedSalary = stats.boxcox(X['EstimatedSalary'])  X['TransformedSalary'] = X\_TransformedSalary[0]  # Split data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.25, random\_state=0) |

After the change, the accuracy rises to 92% from 68%. It almost seems like the Box-Cox transformation improved accuracy above the scaling algorithms …or was it luck? Let’s examine cross fold validation.

|  |
| --- |
| Accuracy: 0.92  Predicted 0 1  Actual  0 66 2  1 6 26 |

## Cross Fold Validation

Cross fold validation is a common procedure for splitting data. There are several variations of cross fold validation. Two notable algorithms include:

**Stratified**: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value.

**LOOCV**: Taken to an extreme, k may be set to the total number of observations in the dataset so each observation is given a chance to be the held out of the dataset. This is called **leave-one-out cross-validation**, or LOOCV for short.

Today we will look at a simple implementation which splits the dataset into k groups while holding back data from each group for testing.

Example 10: Cross Fold Validation Introduction

This example demonstrates how cross fold validation splits the data into different data folds.

|  |
| --- |
| # scikit-learn k-fold cross-validation  from numpy import array  from sklearn.model\_selection import KFold  # data sample  data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])  # splits data into 3 randomized folds  kfold = KFold(3, True)  # enumerate splits  for train, test in kfold.split(data):  print('train: %s, test: %s' % (data[train], data[test])) |

The output shows how three separate random folds are created:

|  |
| --- |
| train: [0.1 0.3 0.5 0.6], test: [0.2 0.4]  train: [0.1 0.2 0.4 0.5], test: [0.3 0.6]  train: [0.2 0.3 0.4 0.6], test: [0.1 0.5] |

Exercise 7 (2 marks)

What happens to the test sets if you change the number of folds to 4? Based on this observation what can you say about the importance of choosing a suitable number of splits?

|  |
| --- |
|  |

Example 11: Cross Fold Validation for Logistic Regression

This example shows how to perform iterative cross fold validation for a logistic regression model for predicting purchase responses to a marketing campaign for computers:

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  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"))  count = 0  # Scaled Data  df2 = df.copy() # Preserve original dataframe.  # Normailze estimated salary  from scipy import stats  df2['EstimatedSalary'] = stats.boxcox(df2['EstimatedSalary'])[0]  # Retain predictors and target column only.  df2 = df2[['Age', 'EstimatedSalary', 'Purchased']]  import numpy as np  # enumerate splits - returns train and test arrays of indexes.  # scikit-learn k-fold cross-validation  from sklearn.model\_selection import KFold  # data sample  # prepare cross validation with three folds and 1 as a random seed.  kfold = KFold(3, True, 1)  for train, test in kfold.split(df2):  X = df2[["Age", "EstimatedSalary"]]  y = df2[['Purchased']]  X\_train = X.iloc[train,:] # Gets all rows with train indexes.  y\_train = y.iloc[train,:]  X\_test = X.iloc[test,:]  y\_test = y.iloc[test,:]  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state=0,  solver='liblinear')  # Fit the model.  logisticModel.fit(X\_train, np.ravel(y\_train))  y\_pred = logisticModel.predict(X\_test)  y\_prob = logisticModel.predict\_proba(X\_test)  # Show confusion matrix and accuracy scores.  cm = pd.crosstab(np.ravel(y\_test), y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  count += 1  print("\n\*\*\*K-fold: " + str(count))  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(cm)  from sklearn.metrics import classification\_report, roc\_auc\_score  print(classification\_report(y\_test, y\_pred))  from sklearn.metrics import average\_precision\_score  average\_precision = average\_precision\_score(y\_test, y\_pred)  print('Average precision-recall score: {0:0.2f}'.format(  average\_precision))  # calculate scores  auc = roc\_auc\_score(y\_test, y\_prob[:, 1],)  print('Logistic: ROC AUC=%.3f' % (auc)) |

The output shows modest but arguably stable fluctuations in accuracy and effectiveness.

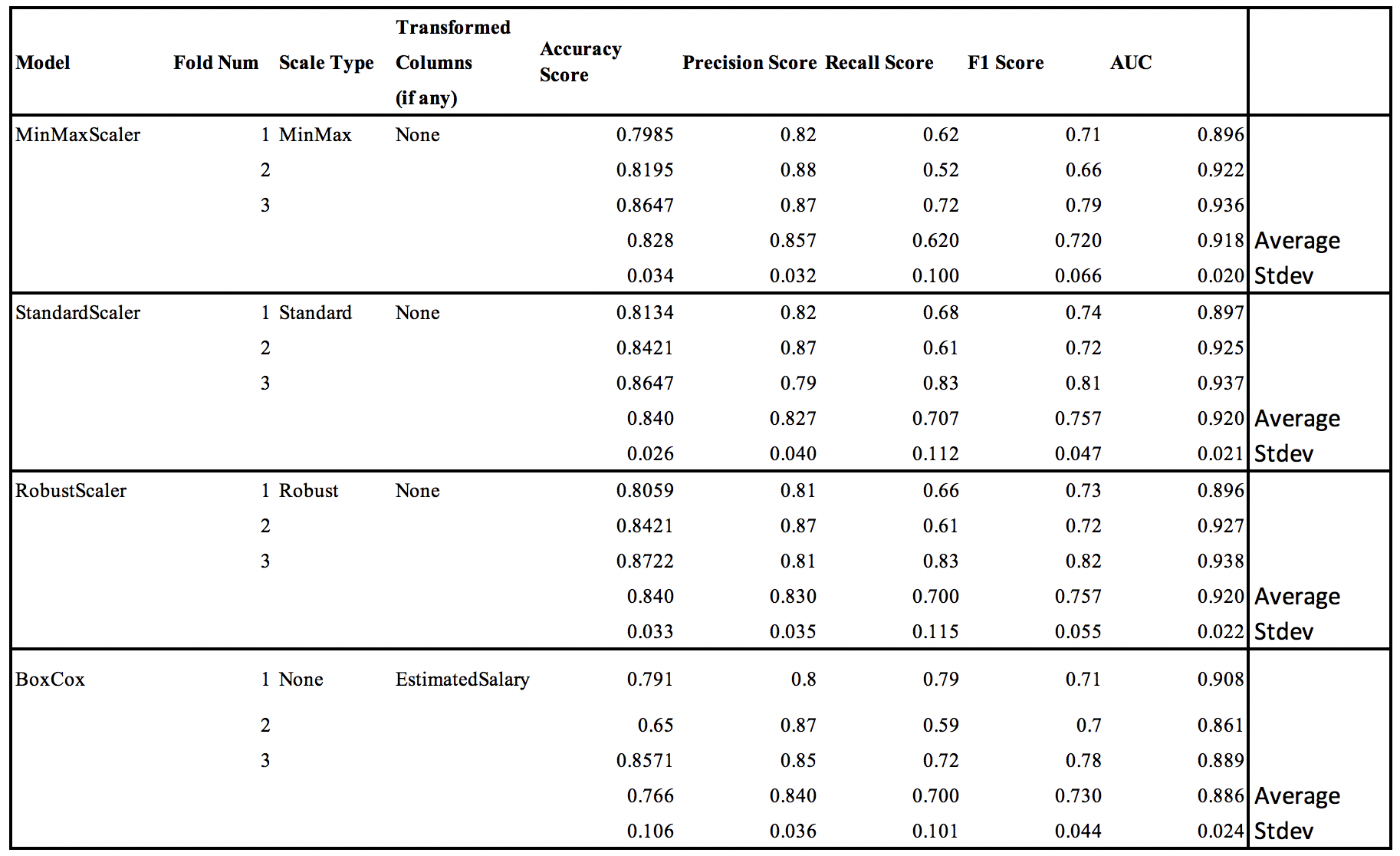
|  |  |  |
| --- | --- | --- |
| K-fold: 1  Accuracy: 0.7910447761194029  Confusion Matrix  Predicted 0 1  Actual  0 74 7  1 21 32 | K-fold: 2  Accuracy: 0.8345864661654135  Confusion Matrix  Predicted 0 1  Actual  0 85 4  1 18 26 | K-fold: 3  Accuracy: 0.8571428571428571  Confusion Matrix  Predicted 0 1  Actual  0 81 6  1 13 33 |

Exercise 8 (8 marks)

Starting with the code from Example 11, use cross fold validation (3 folds each) to test LinearRegression models which use MinMaxScaler, StandardScaler and RobustScaler. Test a fourth model which uses a Box-Cox transformation for estimated salary. (You may apply scaling to the test data of the fourth model as well if you wish). In the table, show results for each fold for all models:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Fold Num** | **Scale Type** | **Transformed**  **Columns**  **(if any)** | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1 Score** | **AUC** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Based on the corrected k-fold code these are the new statistics. Please see the **exercise8\_day11.zip** archive for scaler and transformation code.



Which model performs better if any? You may want to examine a cumulative gains chart or ROC curve to before making a decision.

|  |
| --- |
|  |

Higher variance in scores may indicate unstable models. Do any models exhibit higher variance?

|  |
| --- |
|  |