**Unit 4 Assignment**

Laurence T. Burden

Purdue Global University

IN402: Modeling and Predictive Analysis

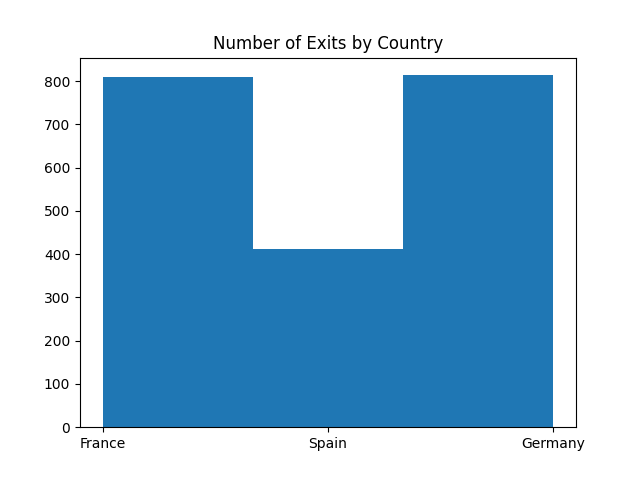
Steward Huang

December 3, 2023

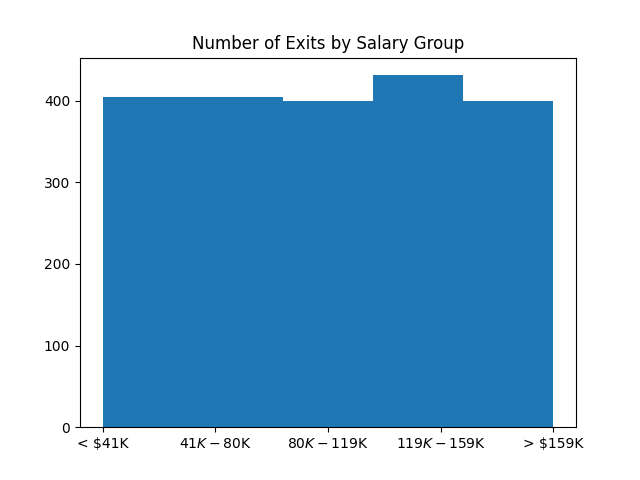
The code creates two classification algorithms to find if a customer will churn or not.  The dataset contains 10,000 observations with 14 features.  The target variable that will be predicted is the “Exited” value.  A one in this column means that a customer left and a zero means they stayed.  The three variables that will be used in the models are “Geography,” “Age,” and “EstimatedSalary.”

The following histograms show the number of exits for each of the three independent variables.  The geography variable shows that France and Germany are roughly the same in their exits, but Spain is much lower than either.  The number of exits remains consistent across salary ranges.  Finally, the number of exits follows a normal distribution for the age variable.  The exits peak between the ages of 40 and 50.

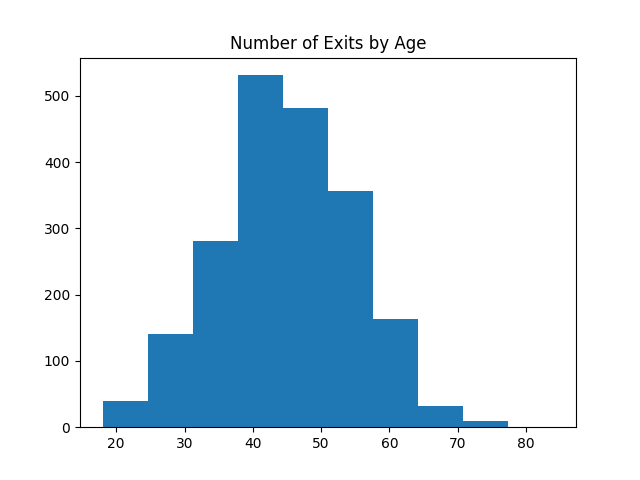
**Figure 1**

****

**Figure 2**

****

**Figure 3**

****

The important category of metrics for this analysis is accurately predicting whether a customer will churn.  If we predict that a customer will churn but they don’t, then the company is only down the small incentive given in an unneeded attempt to keep the customer.  On the other hand, predicting that customers will stay but actually leave will leave the company down a customer.

The logistic regression model shows an accuracy of 79%.  The confusion matrix shows the true positives as 2,311, false positives as 62, the false negatives as 575, and the true negatives as 52.  The recall of this model is 80%, which is a good level for the metric.  The precision is 97%.  The F1 score is .97 for the logistic regression model.  This gives us high confidence in this model.

The SVM model has a precision of 70%.  This means that customers the model predicted would churn, only 70% did.  This is a decent prediction rate.  The recall rate is 16%, meaning that only 16% of the predicted churn customers were correctly predicted.  The overall accuracy rate was 81%.  Finally, the F1 score is only .26 for customers that churn.  This is a poor F1 score since we want to get as close to 1 as possible.  This model is not as good as the logistic regression model.

This exercise created two models to analyze a customer retention data set for a fictitious bank.  One model used logistic regression and the other was built on the SVM algorithm.  The two models were tested using the metrics of recall, precision, accuracy, and F1 score.  The logistic regression model was found to be much better based on these metrics.

**Appendix A**

**Code Output**

Unit 4 Assignment / Module 3 Part 2 Competency Assessment Output

12/03/2023 07:34:28

### Data Head ###

RowNumber CustomerId Surname ... IsActiveMember EstimatedSalary Exited

0 1 15634602 Hargrave ... 1 101348.88 1

1 2 15647311 Hill ... 1 112542.58 0

2 3 15619304 Onio ... 0 113931.57 1

3 4 15701354 Boni ... 0 93826.63 0

4 5 15737888 Mitchell ... 1 79084.10 0

5 6 15574012 Chu ... 0 149756.71 1

6 7 15592531 Bartlett ... 1 10062.80 0

7 8 15656148 Obinna ... 0 119346.88 1

8 9 15792365 He ... 1 74940.50 0

9 10 15592389 H? ... 1 71725.73 0

[10 rows x 14 columns]

### Data Tail ###

RowNumber CustomerId ... EstimatedSalary Exited

9990 9991 15798964 ... 53667.08 0

9991 9992 15769959 ... 69384.71 1

9992 9993 15657105 ... 195192.40 0

9993 9994 15569266 ... 29179.52 0

9994 9995 15719294 ... 167773.55 0

9995 9996 15606229 ... 96270.64 0

9996 9997 15569892 ... 101699.77 0

9997 9998 15584532 ... 42085.58 1

9998 9999 15682355 ... 92888.52 1

9999 10000 15628319 ... 38190.78 0

[10 rows x 14 columns]

### Number of Null Values in Each Column ###

RowNumber 0

CustomerId 0

Surname 0

CreditScore 0

Geography 0

Gender 0

Age 0

Tenure 0

Balance 0

NumOfProducts 0

HasCrCard 0

IsActiveMember 0

EstimatedSalary 0

Exited 0

dtype: int64

### Data Structure ###

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 RowNumber 10000 non-null int64

1 CustomerId 10000 non-null int64

2 Surname 10000 non-null object

3 CreditScore 10000 non-null int64

4 Geography 10000 non-null object

5 Gender 10000 non-null object

6 Age 10000 non-null int64

7 Tenure 10000 non-null int64

8 Balance 10000 non-null float64

9 NumOfProducts 10000 non-null int64

10 HasCrCard 10000 non-null int64

11 IsActiveMember 10000 non-null int64

12 EstimatedSalary 10000 non-null float64

13 Exited 10000 non-null int64

dtypes: float64(2), int64(9), object(3)

memory usage: 1.1+ MB

None

### Data Description ###

RowNumber CustomerId ... EstimatedSalary Exited

count 10000.00000 1.000000e+04 ... 10000.000000 10000.000000

mean 5000.50000 1.569094e+07 ... 100090.239881 0.203700

std 2886.89568 7.193619e+04 ... 57510.492818 0.402769

min 1.00000 1.556570e+07 ... 11.580000 0.000000

25% 2500.75000 1.562853e+07 ... 51002.110000 0.000000

50% 5000.50000 1.569074e+07 ... 100193.915000 0.000000

75% 7500.25000 1.575323e+07 ... 149388.247500 0.000000

max 10000.00000 1.581569e+07 ... 199992.480000 1.000000

[8 rows x 11 columns]

### First 5 Rows of X and Y ###

[['France' 42 101348.88]

['Spain' 41 112542.58]

['France' 42 113931.57]

['France' 39 93826.63]

['Spain' 43 79084.1]]

[1 0 1 0 0]

### Copy of X ###

[['France' 42 101348.88]

['Spain' 41 112542.58]

['France' 42 113931.57]

...

['France' 36 42085.58]

['Germany' 42 92888.52]

['France' 28 38190.78]]

### First 10 Rows of X Copy After Transformation ###

[[1.0 0.0 0.0 42 101348.88]

[0.0 0.0 1.0 41 112542.58]

[1.0 0.0 0.0 42 113931.57]

[1.0 0.0 0.0 39 93826.63]

[0.0 0.0 1.0 43 79084.1]

[0.0 0.0 1.0 44 149756.71]

[1.0 0.0 0.0 50 10062.8]

[0.0 1.0 0.0 29 119346.88]

[1.0 0.0 0.0 44 74940.5]

[1.0 0.0 0.0 27 71725.73]]

### Y after Label Encoding ###

[1 0 1 ... 1 1 0]

### X Train ###

7000

### X Test ###

3000

Variance of CreditScore is 9341.860156575658

Variance of Age is 109.99408416841685

Variance of Tenure is 8.364672627262726

Variance of Balance is 3893436175.990742

Variance of EstimatedSalary is 3307456784.134512

### X Train After Transformation ###

[[1.0 0.0 0.0 -0.18527928339653538 -1.3768671286602732]

[1.0 0.0 0.0 0.005323256654276214 -1.3123385709418818]

[1.0 0.0 0.0 -0.18527928339653538 -0.3188434876153574]

...

[1.0 0.0 0.0 0.577130876806711 -0.1404845747229453]

[1.0 0.0 0.0 0.005323256654276214 0.018567238358042602]

[0.0 1.0 0.0 0.29122706673049364 -1.1594776478647715]]

### Log Reg Prediction Length ###

3000

### Log Reg Prediction ###

Predicted Churn: 114

Predicted Stay: 2886

### Log Reg Confusion Matrix ###

[[2311 62]

[ 575 52]]

### Log Reg Accuracy Score ###

0.7876666666666666

accuracy: 0.81

precision: 0.696551724137931

recall 0.16108452950558214

precision recall f1-score support

0 0.82 0.98 0.89 2373

1 0.70 0.16 0.26 627

accuracy 0.81 3000

macro avg 0.76 0.57 0.58 3000

weighted avg 0.79 0.81 0.76 3000

**Appendix C**

**Python Code**

###############################################################  
# Author: Laurence Burden  
# for Purdue University Global  
#  
# Unit 4 Assignment / Module 3 / Part 2 Competency Assessment  
#  
# Prediction Customer Churn  
###############################################################  
  
# Imports  
import sys  
  
# Ignoring warnings  
if not sys.warnoptions:  
 import warnings  
  
warnings.simplefilter("ignore")  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
# import seaborn as sns  
  
# Output Header  
print('Unit 4 Assignment / Module 3 Part 2 Competency Assessment Output\n')  
  
from datetime import datetime  
print(datetime.now().strftime("%m/%d/%Y %H:%M:%S"), '\n')  
  
# Import the dataset into the development environment.  
df = pd.read\_csv('Churn\_Modelling.csv')  
  
# In the paper, describe the datasource and what you intend to use the libraries for  
  
# Explorative Analysis.  
# Explore the content of the dataset using .head()  
print('### Data Head ###')  
print(df.head(10))  
print()  
  
# Explore the content of the dataset using .tail()  
print('### Data Tail ###')  
print(df.tail(10))  
print()  
  
# Check if there are any missing values using isnull() functions, and remove them  
# using .dropna() function (if any)  
# Check if there are null values anywhere  
print('### Number of Null Values in Each Column ###')  
print(df.isnull().sum())  
print()  
  
# Check the structure and if there are any missing values using .info() function  
print('### Data Structure ###')  
print(df.info())  
print()  
  
# Check the descriptive statistics on numeric variables using .describe() function.  
print('### Data Description ###')  
print(df.describe())  
print()  
  
# For each variable in the dataset that you intend to use in the modeling  
# phase, create an appropriate chart (barchart, histogram, etc.) to explore the  
# relationships between variables  
  
# Create 5 bins for estimated salary  
labels = ['< $41K', '$41K - $80K', '$80K - $119K', '$119K - $159K', '> $159K']  
df['SalaryGroup'] = pd.qcut(df['EstimatedSalary'], q=5, labels=labels)  
  
# Create new dataframes  
# only observations that exited  
df\_exited = df[df['Exited'] == 1]  
  
# Plot histogram of those that exited by salary groups  
plt.hist(df\_exited['SalaryGroup'], bins=5)  
plt.xticks([0, 1, 2, 3, 4], labels=labels)  
plt.title('Number of Exits by Salary Group')  
plt.show()  
  
# Plot exits vs exited by country  
plt.hist(df\_exited['Geography'], bins=3)  
plt.title('Number of Exits by Country')  
plt.show()  
  
# Plot exits vs age  
plt.hist(df\_exited['Age'])  
plt.title('Number of Exits by Age')  
plt.show()  
  
# Wrangle the data to transform the variables.  
# Split the data into testing and training subsets.  
# Assume that "Geography", "Age" and "EstimatedSalary" are the variables you  
# believe are predicting the outcome variable "Exited" the best. Define target  
# and independent variables and assign them into X (independent variables) and  
# Y(target variable).  
# Define dependent and independent variables  
X = df.loc[:, ['Geography', 'Age', 'EstimatedSalary']].values  
y = df.loc[:, 'Exited'].values  
print('### First 5 Rows of X and Y ###')  
print(X[:5])  
print(y[:5])  
  
# Make a copy of X to work on.  
X\_cp = X.copy()  
  
print('### Copy of X ###')  
print(X\_cp)  
print()  
  
# Encode independent variable (categorical data):  
# Transform the encoded column to One hot vectors  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.compose import ColumnTransformer  
  
columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [0])], remainder='passthrough')  
X\_cp = np.array(columnTransformer.fit\_transform(X\_cp))  
  
print('### First 10 Rows of X Copy After Transformation ###')  
print(X\_cp[:10])  
print()  
  
# Encoding the Dependent Variable  
from sklearn.preprocessing import LabelEncoder  
  
labelencoder\_y = LabelEncoder()  
y = labelencoder\_y.fit\_transform(y)  
  
print('### Y after Label Encoding ###')  
print(y)  
print()  
  
# Split the data into the testing and training subsets using train\_test\_split.  
# Use the 30/70 split for training/testing subsets.  
# Split the data into training and testing  
from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_cp, y, test\_size=0.3, random\_state=1)  
  
print('### X Train ###')  
print(len(X\_train))  
print()  
print('### X Test ###')  
print(len(X\_test))  
print()  
  
# Check the variance of variables Age, Geography and EstimatedSalary.  
# Decide whether the scaling is required.  
from statistics import variance  
  
creditScore = df['CreditScore']  
age = df['Age']  
tenure = df['Tenure']  
balance = df['Balance']  
estimatedSalary = df['EstimatedSalary']  
  
# Display variance values  
print("Variance of CreditScore is % s " % (variance(creditScore)))  
print()  
print("Variance of Age is % s " % (variance(age)))  
print()  
print("Variance of Tenure is % s " % (variance(tenure)))  
print()  
print("Variance of Balance is % s " % (variance(balance)))  
print()  
print("Variance of EstimatedSalary is % s " % (variance(estimatedSalary)))  
print()  
  
# Scale the variables to the same scale (feature scaling).  
from sklearn.preprocessing import StandardScaler  
  
sc = StandardScaler()  
X\_train[:,-2: ] = sc.fit\_transform(X\_train[:,-2: ])  
X\_test[:,-2:] = sc.transform(X\_test[:,-2:])  
  
print('### X Train After Transformation ###')  
print(X\_train)  
print()  
  
# Build and evaluate a Logistic Regression Model  
# Import the package  
from sklearn.linear\_model import LogisticRegression  
  
# Create a new model using LogisticRegression() construct.  
logreg\_model = LogisticRegression()  
  
# Fit the model into the training subsets.  
logreg\_model.fit(X\_train, y\_train)  
  
# Make a prediction using .predict() on the test dataset.  
# Make a prediction using Logistic Regression  
logreg\_prediction = logreg\_model.predict(X\_test)  
predicted\_churn = np.count\_nonzero(logreg\_prediction == 1)  
predicted\_stay = len(logreg\_prediction) - predicted\_churn  
  
print('### Log Reg Prediction Length ###')  
print(len(logreg\_prediction))  
print('### Log Reg Prediction ###')  
print('Predicted Churn: ', predicted\_churn)  
print('Predicted Stay: ', predicted\_stay)  
print()  
  
# Evaluate the model using cross validation; build confusion matrix and perform  
# an accuracy score to determine if the accuracy of the result is high enough.  
# Evaluate how the model has been performing  
# Cross validation  
# Display confusion matrix  
from sklearn.metrics import confusion\_matrix  
  
cm = confusion\_matrix(y\_test, logreg\_prediction)  
  
print('### Log Reg Confusion Matrix ###')  
print(cm)  
print()  
  
# Display accuracy score  
from sklearn.metrics import accuracy\_score  
accScore = accuracy\_score(y\_test, logreg\_prediction)  
  
print('### Log Reg Accuracy Score ###')  
print(accScore)  
print()  
  
# Build and evaluate a Support Vector Machine Model  
  
# Import libraries  
from sklearn.svm import SVC  
from sklearn import metrics  
  
# Create a new model using SVC construct (Radial Basis Function kernel as an argument)  
svm\_model = SVC(kernel = "rbf")  
  
# Fit the model into the training subsets  
# Train the model  
svm\_model.fit(X\_train, y\_train)  
  
# Make a prediction using .predict() function on the test dataset.  
# Predict the response  
svm\_prediction = svm\_model.predict(X\_test)  
  
print("accuracy: ", metrics.accuracy\_score(y\_test, y\_pred = svm\_prediction))  
print()  
  
# Evaluate the model using the precision score and recall score and determine  
# if the accuracy of the result is high enough.  
# Display precision score  
print("precision: ", metrics.precision\_score(y\_test, y\_pred = svm\_prediction))  
print()  
  
# Display recall score  
print("recall", metrics.recall\_score(y\_test, y\_pred = svm\_prediction))  
print()  
  
# Display classification report  
print(metrics.classification\_report(y\_test, y\_pred = svm\_prediction))