Assignment 3.1: Module 3 Exercise Questions | R

Ryan S Dunn

11/13/2021

Data Science Using Python and R: Chapter 5 - Page 78: Questions #28, 29, 30, 31, 32, 33, & 34

28. Partition the data set, so that 67% of the records are included in the training data set and 33% are included in the test data set. Use a bar graph to confirm your proportions.

```
#load the libraries
library(ggplot2)

#load the data set
churn = read.csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data Mining 502/Module 3/Datasets/Chu
head(churn)
```

```
##   State Account.Length Area.Code    Phone Intl.Plan VMail.Plan VMail.Message
## 1    KS            128       415 382-4657        no        yes            25
## 2    OH            107       415 371-7191        no        yes            26
## 3    NJ            137       415 358-1921        no         no             0
## 4    OH             84       408 375-9999       yes         no             0
## 5    OK             75       415 330-6626       yes         no             0
## 6    AL            118       510 391-8027       yes         no             0
##   Day.Mins Day.Calls Day.Charge Eve.Mins Eve.Calls Eve.Charge Night.Mins
## 1    265.1       110      45.07    197.4        99      16.78      244.7
## 2    161.6       123      27.47    195.5       103      16.62      254.4
## 3    243.4       114      41.38    121.2       110      10.30      162.6
## 4    299.4        71      50.90     61.9        88       5.26      196.9
## 5    166.7       113      28.34    148.3       122      12.61      186.9
## 6    223.4        98      37.98    220.6       101      18.75      203.9
##   Night.Calls Night.Charge Intl.Mins Intl.Calls Intl.Charge CustServ.Calls
## 1          91        11.01      10.0          3        2.70              1
## 2         103        11.45      13.7          3        3.70              1
## 3         104         7.32      12.2          5        3.29              0
## 4          89         8.86       6.6          7        1.78              2
## 5         121         8.41      10.1          3        2.73              3
## 6         118         9.18       6.3          6        1.70              0
##   Old.Churn Churn
## 1    False. False
## 2    False. False
## 3    False. False
## 4    False. False
## 5    False. False
## 6    False. False
```

```r
#set the random seed and identify the number of records in the data
set.seed(7)
n <- dim(churn)[1]
n
```

```
## [1] 3333
```

```r
#develop the partition of 67% of the data
percent_index <- runif(n) < 0.67

#create training and test sets with the .67 and .37 records
churn_train <- churn[ percent_index,]
churn_test <- churn[ !percent_index,]

#print the number of records in the training and testing sets
print(dim(churn_train)[1])
```

```
## [1] 2227
```

```r
print(dim(churn_test)[1])
```

```
## [1] 1106
```

```r
print("Training set percentage is:")
```

```
## [1] "Training set percentage is:"
```

```r
round((dim(churn_train)[1] / dim(churn)[1]) * 100,2)
```

```
## [1] 66.82
```

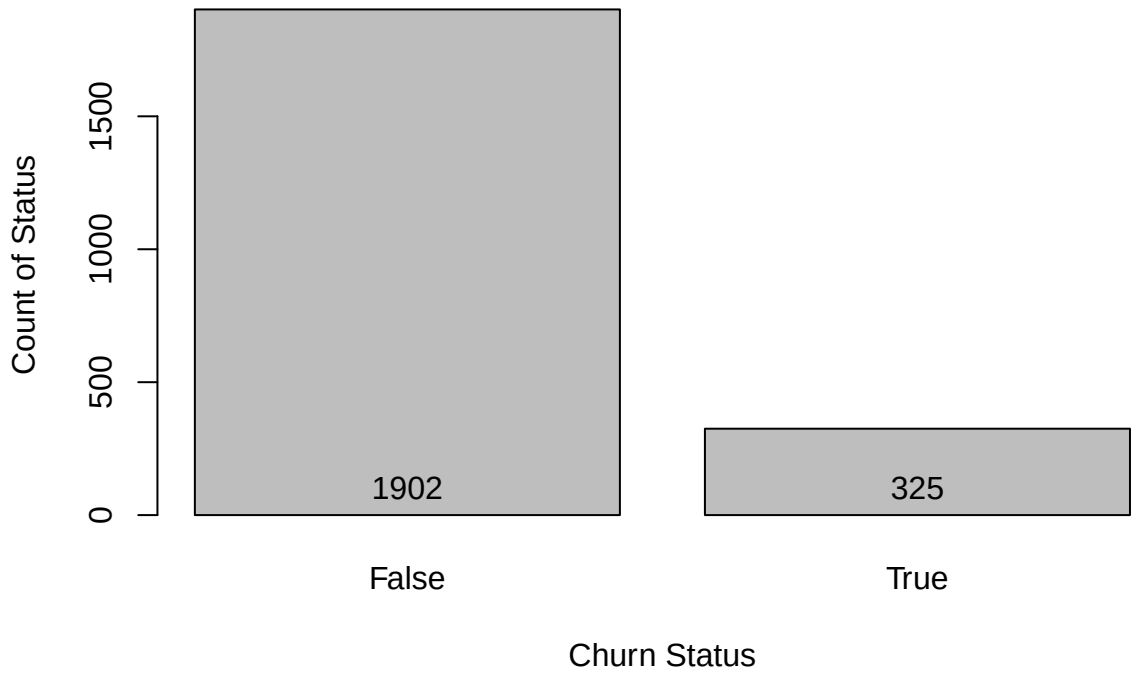
```r
print("Test set percentage is:")
```

```
## [1] "Test set percentage is:"
```

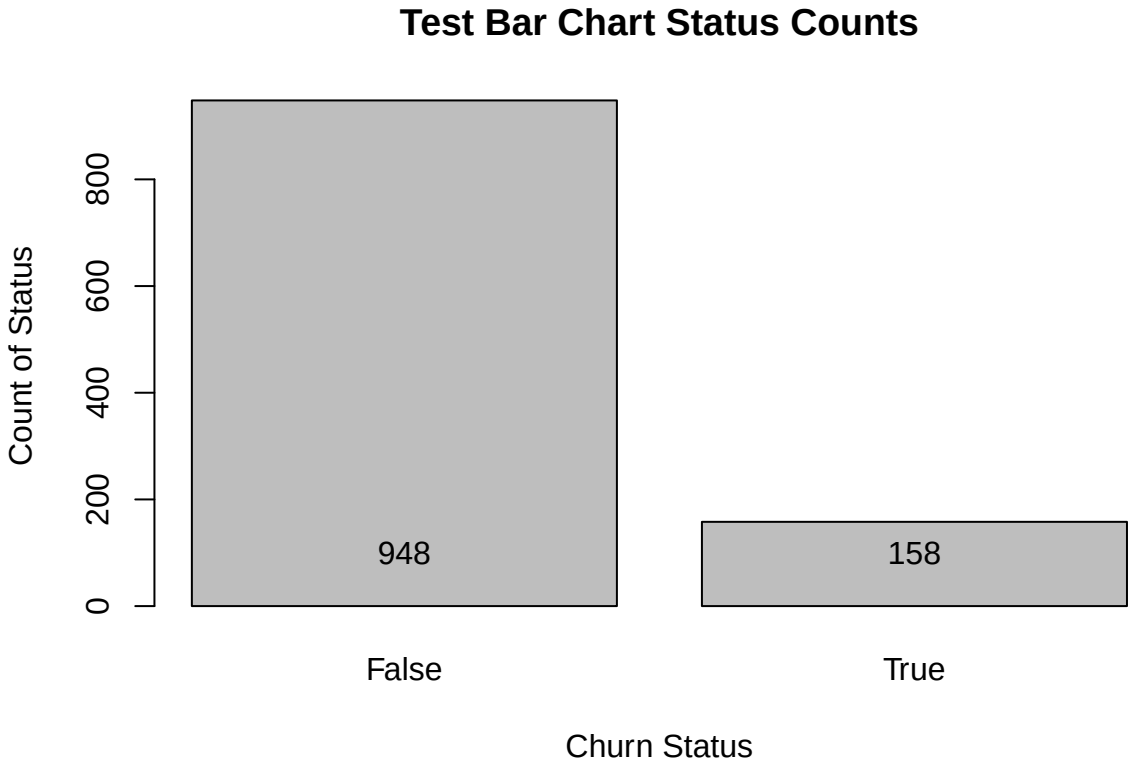
```r
round((dim(churn_test)[1] / dim(churn)[1]) * 100,2)
```

```
## [1] 33.18
```

```r
#confirm the partitions with bar graph
status_training <- table(churn_train$Churn)
b <- barplot(status_training, main = "Training Bar Chart Value Counts",
        xlab = "Churn Status",
        ylab = "Count of Status")

#create the numbers inside of barchart
text(b,  + 100, status_training, font=1)
```




```
#confirm the partitions with bar graph
status_test <- table(churn_test$Churn)
c <- barplot(status_test, main = "Test Bar Chart Status Counts",
        xlab = "Churn Status",
        ylab = "Count of Status")

#create the numbers inside of barchart
text(c,  + 100, status_test, font=1)
```



29. Identify the total number of records in the training data set and how many records in the training data set have a churn value of true.

```
#print the number of records in the training set
dim(churn_train)[1]
```

```
## [1] 2227
```

```
#make an index of only True churn values in the training set
true_index <- which(churn_train$Churn == 'True')
length(true_index)
```

```
## [1] 325
```

30. Use your answers from the previous exercise to calculate how many true churn records you need to resample in order to have 20% of the rebalanced data set have true churn values.

```
#view the False and True records
table(churn_train$Churn)
```

```
##
## False  True
##  1902   325
```

```
#calculate the amount of records needed to add
needed_number <- ((0.2)*(2227)-325)/(0.8)
print("The number of records we need to add are:")
```

```
## [1] "The number of records we need to add are:"
```

```
needed_number
```

```
## [1] 150.5
```

31. Perform the rebalancing described in the previous exercies and confirm that 20% of the records in the rebalanced data set have true churn values.

```
to.resample <- which(churn_train$Churn == 'True')
our.resample <- sample( x = to.resample, size = 150, replace = TRUE)
```

```
our.resample <- churn_train[our.resample,]

train_churn_rebalance <- rbind(churn_train, our.resample)
```

```
t.v1 <- table(train_churn_rebalance$Churn)
t.v2 <- rbind(t.v1, round(prop.table(t.v1),4) * 100)
colnames(t.v2) <- c("Churn = False", "Churn = True");
rownames(t.v2) <- c("Count", "proportion")
t.v2
```

```
##            Churn = False Churn = True
## Count             1902.00       475.00
## proportion          80.02        19.98
```

32. Which baseline model do we use to compare our classification model performance against? To whihc value does this baseline modelassign all predictions? What is the accuracy of this baseline model?

The base line model that we would use to compare our classification model against is where the model is an even 50/50 split. With a classification model - (is this true?), or would it be the all positive/all negative model?

33. Validate your partition by testing for the difference in mean "day minutes for the training set versus the test set.

```
#mean of balanced training set
mean(train_churn_rebalance$Day.Mins)
```

```
## [1] 182.4444
```

```
#mean of test data set
mean(churn_test$Day.Mins)
```

```
## [1] 177.4879
```

34. Validate your partition by testing for the difference in proportion of true churn records for the training set versus the test set.

```
#validate the partitions by testing difference in proportion
```

Data Science Using Python and R: Chapter 7 - Page 109: Questions 23, 24, 25, 26, 27, 28, 29, 30

23. Using the training data set, create a C5.0 model (Model 1) to predict a customer's Income using Marital Status and Capital Gains Loses. Obtain the predicted responses.

```
library(C50)

#import the training and test data
training_data <- read.csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data Mining 502/Module 3/Data
test_data <- read.csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data Mining 502/Module 3/Datasets
head(training_data)
```

```
##   Marital.status Income Cap_Gains_Losses
## 1  Never-married  <=50K          0.02174
## 2       Divorced  <=50K          0.00000
## 3        Married  <=50K          0.00000
## 4        Married  <=50K          0.00000
## 5        Married  <=50K          0.00000
## 6        Married   >50K          0.00000
```

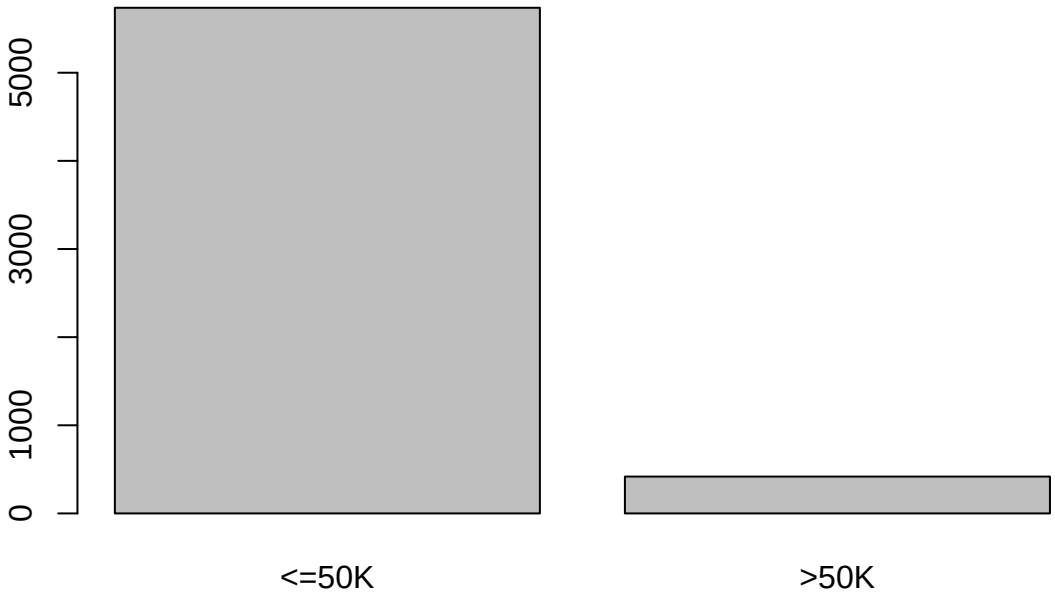
```
#change the types to factor for the C5.0 algorithm for training and test data
training_data$Income <- factor(training_data$Income)
training_data$Marital.status <- factor(training_data$Marital.status)

test_data$Income <- factor(test_data$Income)
test_data$Marital.status <- factor(test_data$Marital.status)
```

```
#run the C5.0 Model
C5 <- C5.0(formula = Income ~ Marital.status + Cap_Gains_Losses, data = training_data,
           control = C5.0Control(minCases = 75))
```

24. Evaluate Model 1 using the test data set. Construct a contingency table to compare the actual and predicted values of Income.

```
#subset the predictor values into their own dataframe
test.X <- subset(x = test_data, select = c("Marital.status","Cap_Gains_Losses"))
```

```
#run the test data through the training data model
y_pred <- predict(object = C5, newdata = test.X)
plot(y_pred)
```




```
#compare predictions to the actual income values in the test data set
t1 <- table(test_data$Income, y_pred)

#assign the names for the rows and columns in contigency table
row.names(t1) <- c("Actual: <=50K", "Actual: >50K")
colnames(t1) <- c("Predicted: <=50K", "Predicted: >50K")
t1 <- addmargins( A = t1, FUN = list(Total = sum), quiet = TRUE) ;
t1
```

```
##                y_pred
##                 Predicted: <=50K Predicted: >50K Total
##   Actual: <=50K             4632              42  4674
##   Actual: >50K              1105             376  1481
##   Total                     5737             418  6155
```

25. For Model 1, recapitulate Table 7.4 from the text, calculating all of the model evaluation measures shown in the table. Call this table the Model Evaluation Table. Leave space for Model 2.

```
#develop the model 1 metrics to input into the confusion matrix
TP_1 <- t1[1,1]
FN_1 <- t1[1,2]
FP_1 <- t1[2,1]
TN_1 <- t1[2,2]

#totals for the corresponding row and column values of confusion matrix
TAN_1 <- FP_1 + TN_1
TAP_1 <- TP_1 + FN_1
TPN_1 <- TN_1 + FN_1
TPP_1 <- FP_1 + TP_1
GT_1<- TP_1 + FN_1 + FP_1 + TN_1

#evaluation methods for model 1
accuracy_1 <- round((TN_1 + TP_1) / (GT_1),4)
error_rate_sensitivity_1 <- (1 - accuracy_1)
```

```r
specificity_1 <- round((TN_1/TAN_1),4)
precision_1 <- round((TP_1/TPP_1),4)
f1_1 <- round(2* ((precision_1 * specificity_1) / (precision_1 + specificity_1)),4)
f2_1 <- round(5 * ((precision_1 * specificity_1) / (4 * precision_1 + specificity_1)),4)
f0.5_1 <- round(1.25 * ((precision_1 * specificity_1) / (0.25 * precision_1 + specificity_1)),4)
```

```r
#put the calculation of the metrics in the below array
eval_metrics_1 <- c(" ","(TN + TP)/ GT", accuracy_1, NA,
                    " ","1 - accuracy", error_rate_sensitivity_1, NA,
                    " ","(TN / TAN", specificity_1, NA,
                    " ","(TP / TPP)", precision_1, NA,
                    " ","2 * (precision * specificity) / (precision + specificity)",f1_1, NA,
                    " ", "5 * (precision * specificity) / ( 4 *precision + specificity)",f2_1, NA,
                    " ","1.25 * (precision * specificity) / (0.25 *precision + specificity)", f0.5_1, NA)

#create a matrix to input the eval metrics into
model_eval_table_1 <- matrix(eval_metrics_1, ncol=4 ,nrow = 7, byrow=TRUE)

#make the column names
colnames(model_eval_table_1) <- c(" ", "Equation", "Model_1", "Module_2")
row.names(model_eval_table_1) <- c("Accuracy", "Error rate Sensitivity", "Specificity", "Precision",
                                   "F1", "F2", "F0.5")

#view the table
met_1 <- as.table(model_eval_table_1)
met_1
```

```
##
## Accuracy
## Error rate Sensitivity
## Specificity
## Precision
## F1
## F2
## F0.5
##                          Equation
## Accuracy                 (TN + TP)/ GT
## Error rate Sensitivity 1 - accuracy
## Specificity              (TN / TAN
## Precision                (TP / TPP)
## F1                        2 * (precision * specificity) / (precision + specificity)
## F2                       5 * (precision * specificity) / ( 4 *precision + specificity)
## F0.5                     1.25 * (precision * specificity) / (0.25 *precision + specificity)
##                          Model_1 Module_2
## Accuracy                 0.8136
## Error rate Sensitivity 0.1864
## Specificity              0.2539
## Precision                0.8074
## F1                       0.3863
## F2                       0.2942
## F0.5                     0.5623
```

26. Clearly and completely interpret each of the Model 1 evaluation measures from the Model Evaluation Method.

Accuracy: The accuracy of model 1 is 0.8136. Accuracy represents the proportion of correct classifications being made. Ultimately, this translates to the true positive + true negative amounts divided by the grand total.
Error rate Sensitivity: The sensitivity of model 1 is 0.1864. The error rate, or sensitivity, is simply 1 minus the accuracy, which translates to the the proportion of incorrect classifications being made by the model. Specificity (Recall): The specificity, or Recall of model 1 is 0.2539. Specificity measures what proportion of all negative records are captured by the model. This translates to the true negative / total actually negative. Why specificity matters, is that when a classification model is specific, it identifies a large proportion of attributes in the model that are negative. Precision: The precision of our model is 0.8074. Precision measures the proportion of records that are classified as positive as actually being a true positive. This translates to the true positive / true predicted positive values. F1: F1 for the model is 0.3863. F scores combine precision and recall into a single measure. For the F1 value, this is the harmonic mean of the precision and recall values. F2: The F2 for the model is 0.2942. When the value is more than 1 (as in this case), recall is weighted higher than precision. F0.5 The F0.5 for the model is 0.5623. When the value is less than 1 (as in this case), recall is weighted lower than precision.

27. Create a cost matrix, called the 3x cost matrix, that specifies a false positive is four times as bad as a false negative.

```
cost.C5 <- matrix(c(0,1,4,0), byrow = TRUE, ncol=2)
dimnames(cost.C5) <- list(c('<=50K', '>50K'),c('<=50K', '>50K'))

cost.C5
```

```
##       <=50K >50K
## <=50K     0    1
## >50K      4    0
```

28. Using the training data set, build a C5.0 model (Model 2), to repredict a customers' Income using Marital Status and Capital Gains and Losses, using the 3x cost matrix.

```
#create the cost matrix to account for unequal error costs
C5.costs <- C5.0(formula = Income ~ Marital.status + Cap_Gains_Losses, data = training_data, control = C
           costs = cost.C5)

test.X <- subset(x = test_data, select = c("Marital.status","Cap_Gains_Losses"))

y_pred <- predict(object = C5.costs, newdata = test.X)

t2 <- table(test_data$Income, y_pred)

#assign the names for the rows and columns in contingency table
row.names(t2) <- c("Actual: <=50K", "Actual: >50K")
colnames(t2) <- c("Predicted: <=50K", "Predicted: >50K")
t2 <- addmargins( A = t2, FUN = list(Total = sum), quiet = TRUE) ;
t2
```

```
##                y_pred
##                 Predicted: <=50K Predicted: >50K Total
##   Actual: <=50K             4663              11  4674
##   Actual: >50K              1117             364  1481
##   Total                     5780             375  6155
```

29. Evaluate your predictions from Model 2 using the actual response values from the test data set. Add Overall Model Cost and Profit per Customer to Model Evaluation Table. Calculate all the measures from the Model Evaluation Table.

```
#develop the model 1 metrics to input into the matrix
TP_2 <- t2[1,1]
FN_2 <- t2[1,2]
FP_2 <- t2[2,1]
TN_2 <- t2[2,2]

#totals for the corresponding row and column values of confusion matrix
TAN_2 <- FP_2 + TN_2
TAP_2 <- TP_2 + FN_2
TPN_2 <- TN_2 + FN_2
TPP_2 <- FP_2 + TP_2
GT_2<- TP_2 + FN_2 + FP_2 + TN_2

#evaluation methods for model 1
accuracy_2 <- round((TN_2 + TP_2) / (GT_2),4)
error_rate_sensitivity_2 <- (1 - accuracy_2)
specificity_2 <- round((TN_2/TAN_2),4)
precision_2 <- round((TP_2/TPP_2),4)
f1_2 <- round(2* ((precision_2 * specificity_2) / (precision_2 + specificity_2)),4)
f2_2 <- round(5 * ((precision_2 * specificity_2) / (4 * precision_2 + specificity_2)),4)
f0.5_2 <- round(1.25 * ((precision_2 * specificity_2) / (0.25 * precision_2 + specificity_2)),4)
```

```
#put the calculation of the metrics in the below array
eval_metrics_2 <- c(" ", "(TN + TP)/ GT", accuracy_1, accuracy_2,
                    " ","1 - accuracy", error_rate_sensitivity_1, error_rate_sensitivity_2,
                    " ", "(TN / TAN", specificity_1, specificity_2,
                    " ", "(TP / TPP)", precision_1, precision_2,
                    " ", " 2 * (precision * specificity) / (precision + specificity)", f1_1, f1_2,
                    " ", "5 * (precision * specificity) / ( 4 *precision + specificity)", f2_1, f2_2,
                    " ", "1.25 * (precision * specificity) / (0.25 *precision + specificity)", f0.5_1, f0

#create a matrix to input the eval metrics into
model_eval_table_2 <- matrix(eval_metrics_2, ncol=4 ,nrow = 7, byrow=TRUE)

#make the column names
colnames(model_eval_table_2) <- c(" ", "Equation","Model_1", "Module_2")
row.names(model_eval_table_2) <- c("Accuracy", "Error rate Sensitivity", "Specificity", "Precision",
                                   "F1", "F2", "F0.5")

#view the table
met_2 <- as.table(model_eval_table_2)
met_2
```

```
##
## Accuracy
## Error rate Sensitivity
## Specificity
## Precision
## F1
## F2
## F0.5
```

```
##                          Equation
## Accuracy                 (TN + TP)/ GT
## Error rate Sensitivity 1 - accuracy
## Specificity              (TN / TAN
## Precision                (TP / TPP)
## F1                        2 * (precision * specificity) / (precision + specificity)
## F2                       5 * (precision * specificity) / ( 4 *precision + specificity)
## F0.5                     1.25 * (precision * specificity) / (0.25 *precision + specificity)
##                          Model_1 Module_2
## Accuracy                 0.8136  0.8167
## Error rate Sensitivity 0.1864  0.1833
## Specificity              0.2539  0.2458
## Precision                0.8074  0.8067
## F1                       0.3863  0.3768
## F2                       0.2942  0.2855
## F0.5                     0.5623  0.5539
```

30. Compare the evaluation measures from Model 1 and Model 2 using the 3x cost matrix. Discuss the strengths and weaknesses of each model.

When compared to model 1, model 2 only provides a marginal benefit in Accuracy and Sensitivity. Essentially, with the cost matrix included into the development of the model, model two yields a 81.67% accuracy rate, versus an 81.36% accuracy rate of model 1. For both specificity and precision however, model 2 is not as good at determining what proportion of all negative records are captured by the model, nor what proportion of all positive records are captured by the model. In fact, the specificity of model to relative to model 1 decreases by almost 1% (0.81%). Lastly, F1, F2 and F0.5 are all less significant in model 2. In summation, even though model 2 is marginally more accurate than model 1, it loses the ability to accurately assing the proper positive and negative classes to the attribute being predicted.

```
a  <- 0.2458 - 0.2539
a
```

```
## [1] -0.0081
```

Data Science Using Python and R: Chapter 8 - Page 126: Questions #31, 32, 33, & 34

31. Run the Naive Bayes classifier to classify persons as living or dead based on sex and education.

```
#install.packages("gridExtra") #run this package first to install the library
library(gridExtra)
```

```
#import death data sets
death_tr <- read.csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data Mining 502/Module 3/Datasets/
death_test <- read.csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data Mining 502/Module 3/Dataset

#head(death_training)

#death_tr$Age <- as.factor(death_tr$Age)
#death_tr$Death   <- as.factor((death_tr$Death))
```

```
#to validate table names below
a <- table(death_tr$Death)
#plot(a)
```

```r
#create a contingency table for Sex
ts<- table(death_tr$Death, death_tr$Sex)
colnames(ts) <- c("Sex = 1", "Sex = 2")
rownames(ts) <- c("Death = 0", "Death = 1")
addmargins(A = ts, FUN = list(Total = sum), quiet = TRUE)
```

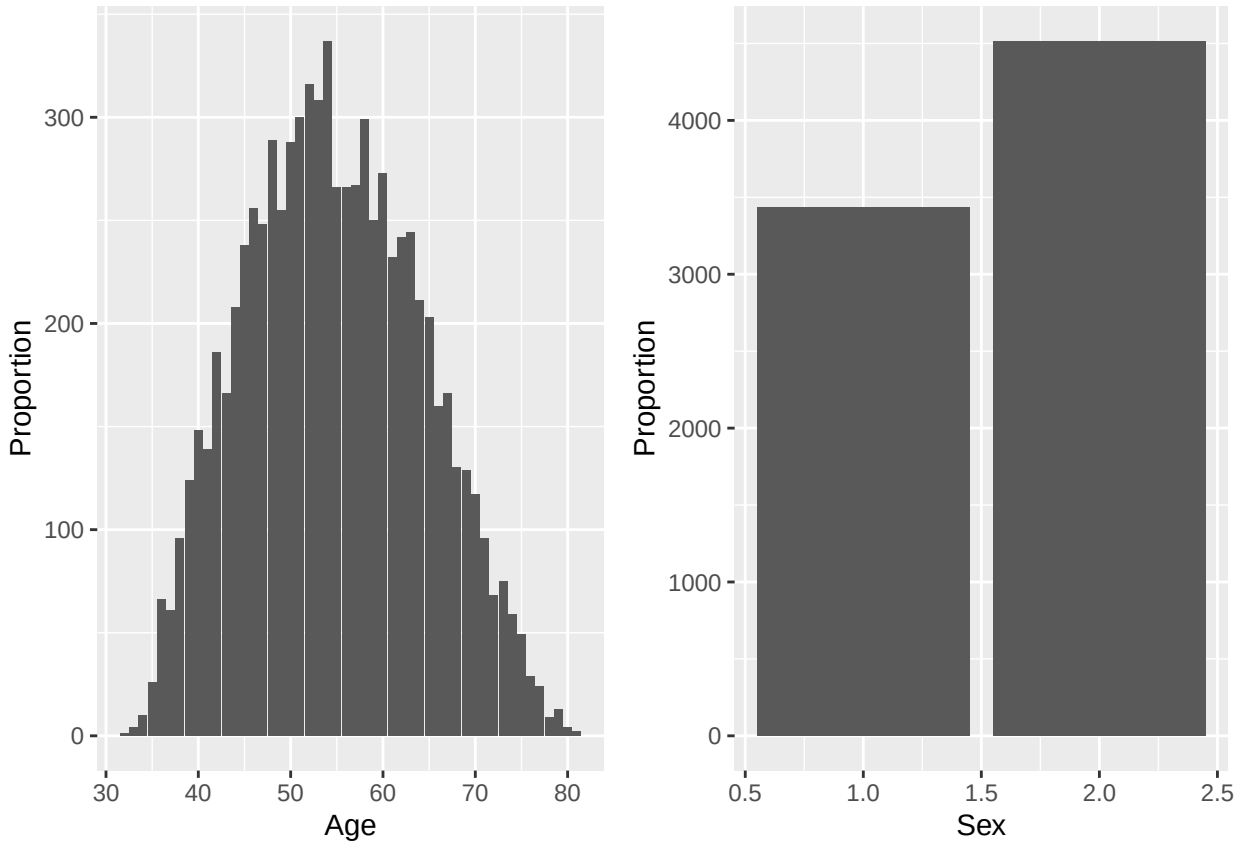
```
##
##             Sex = 1 Sex = 2 Total
##   Death = 0    2113    3422  5535
##   Death = 1    1324    1094  2418
##   Total        3437    4516  7953
```

```r
#ta
```

```r
#create a contingency table for Age
ta <- table(death_tr$Death, death_tr$Age)
#colnames(ta) <- c("Ages")
rownames(ta) <- c("Death =0", "Death =1")
#addmargins(A = ta, FUN = list(Total = sum_, quiet = TRUE)
ta
```

```
##
##             32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
##   Death =0   1   4   9  24  60  55  82 109 122 120 158 138 168 197 209 204 239
##   Death =1   0   0   1   2   6   6  14  15  26  19  28  28  40  41  47  44  50
##
##             49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
##   Death =0 202 232 234 247 237 255 191 180 186 197 173 187 138 127 139 106 106
##   Death =1  53  56  66  69  71  82  75  86  81 102  77  86  94 115 105 105  97
##
##             66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
##   Death =0  76  92  53  54  48  45  31  34  19  24   9  10   2   1   0   1
##   Death =1  84  74  77  75  69  51  37  41  40  25  20  14   7  12   4   1
```

```r
age_plot <- ggplot(death_tr, aes(Age)) + geom_bar( aes(fill = Age)) +
  ylab("Proportion")

sex_plot <- ggplot(death_tr, aes(Sex)) + geom_bar( aes(fill = Sex)) +
  ylab("Proportion")
grid.arrange(age_plot, sex_plot, nrow = 1)
```




```
#install.packages("e1071") #install this package to run the Naive Bayes algorithm
library(e1071)
```

```
#run the Naive Bayes estimator
nb_1 <- naiveBayes(formula = Death ~ Age + Sex, data = death_tr)
nb_1
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.6959638 0.3040362
##
## Conditional probabilities:
##    Age
## Y       [,1]     [,2]
##   0 52.80903 8.930191
##   1 59.00124 9.316727
##
##    Sex
## Y       [,1]      [,2]
##   0 1.618248 0.4858602
##   1 1.452440 0.4978359
```

32. Evaluate the Naive Bayes model on the framingham_nb_test data set. Display the results in a contingency table. Edit the row and column names of the table to make the table more readable. Include a total row and column.

```
#run the Naive Bayes estimator on the test data
ypred <- predict(object = nb_1, newdata = death_test)
```

```
#create the contingency table
preds <- table(death_test$Death, ypred)
rownames(preds) <- c("Actual: 1", "Actual: 2")
colnames(preds) <- c("Predicted: 1", "Predicted: 2")
addmargins( A = preds, FUN = list(Total =sum), quiet = TRUE )
```

```
##            ypred
##             Predicted: 1 Predicted: 2 Total
##   Actual: 1         1475          122  1597
##   Actual: 2          493          167   660
##   Total             1968          289  2257
```

```
#barplot(preds)
```

33. According to your table in the previous exercise, find the following values for the Naive Bayes model:

```
#develop the TP, TN, FP, FN values from confusion matrix
TP_nb <- preds[1,1]
TN_nb <- preds[2,2]
FP_nb <- preds[2,1]
FN_nb <- preds[1,2]

#totals for the corresponding row and column values of confusion matrix
TAN_nb <- FP_nb + TN_nb
TAP_nb <- TP_nb + FN_nb
TPN_nb <- TN_nb + FN_nb
TPP_nb <- FP_nb + TP_nb
GT_nb<- TP_nb + FN_nb + FP_nb + TN_nb
```

a. Accuracy

```
#evaluation methods for Naive Bayes model - accuracy
print(accuracy_nb <- round((TN_nb + TP_nb) / (GT_nb),4) )
```

```
## [1] 0.7275
```

b. Error Rate

```
#evaluation methods for Naive Bayes model - error rate
print(error_rate_sensitivity_nb <- (1 - accuracy_nb))
```

```
## [1] 0.2725
```

34. According to your contingency table, find the following values for the Naive Bayes model:

a. How often it correctly classifies a dead persons.

```
correct_dead <- TN_nb / TPN_nb
correct_dead
```

```
## [1] 0.5778547
```

b. How often it correctly classifies living persons

```
correct_living <- TP_nb / TPP_nb
correct_living
```

```
## [1] 0.7494919
```

ADS502_Module_3_Homework

November 15, 2021

1 Assignment 3.1: Module 3 Exercise Questions | Python

1.0.1 Ryan S. Dunn | University of San Diego | M.S. Applied Data Science

for detailed explainations on each question, please see the R output above

[96]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import random
```

1.1 Data Science Using Python and R: Chapter 5 - Page 78: Questions 28, 29, 30 , 31, 32, 33, 34

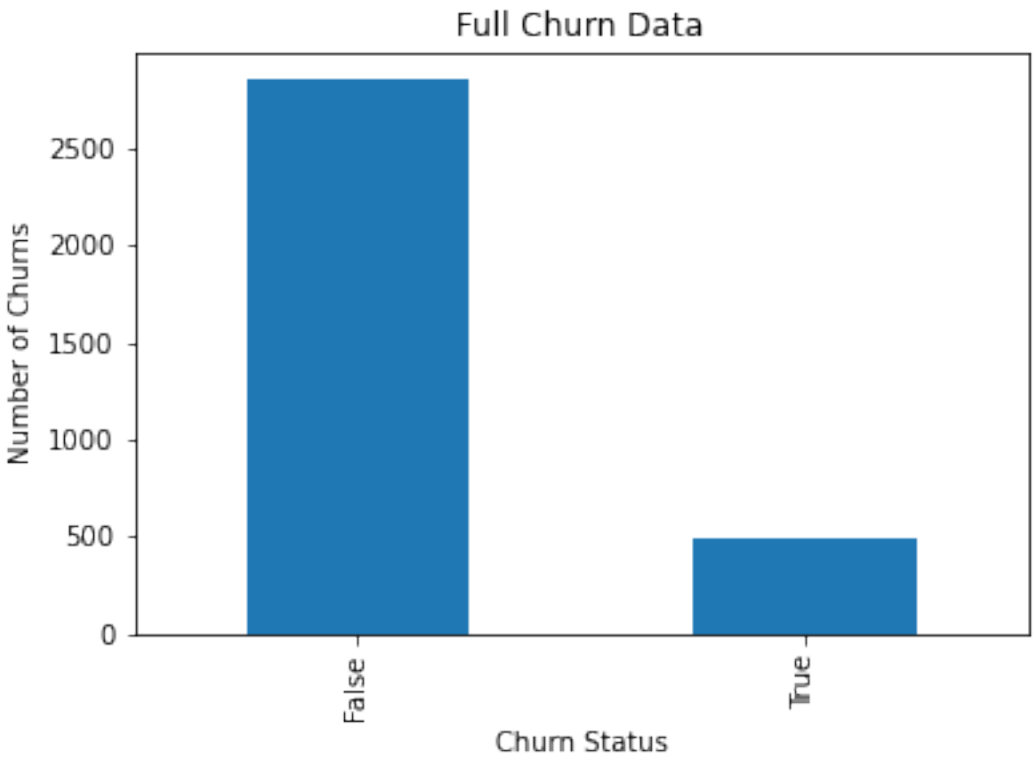
[41]:
```
#import the data
churn = pd.read_csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data␣
↪Mining 502/Module 3/Datasets/Churn", header = 0)
#churn.head()
```

1.1.1 28. Partition the data set, so that 67% of the records are included in the training data set and 33% are included in the test data set. Use a bar graph to confirm your proportions.

[42]:
```
#partition the data set into train and test
churn_train, churn_test = train_test_split(churn, test_size = 0.33,␣
↪random_state = 7)
```

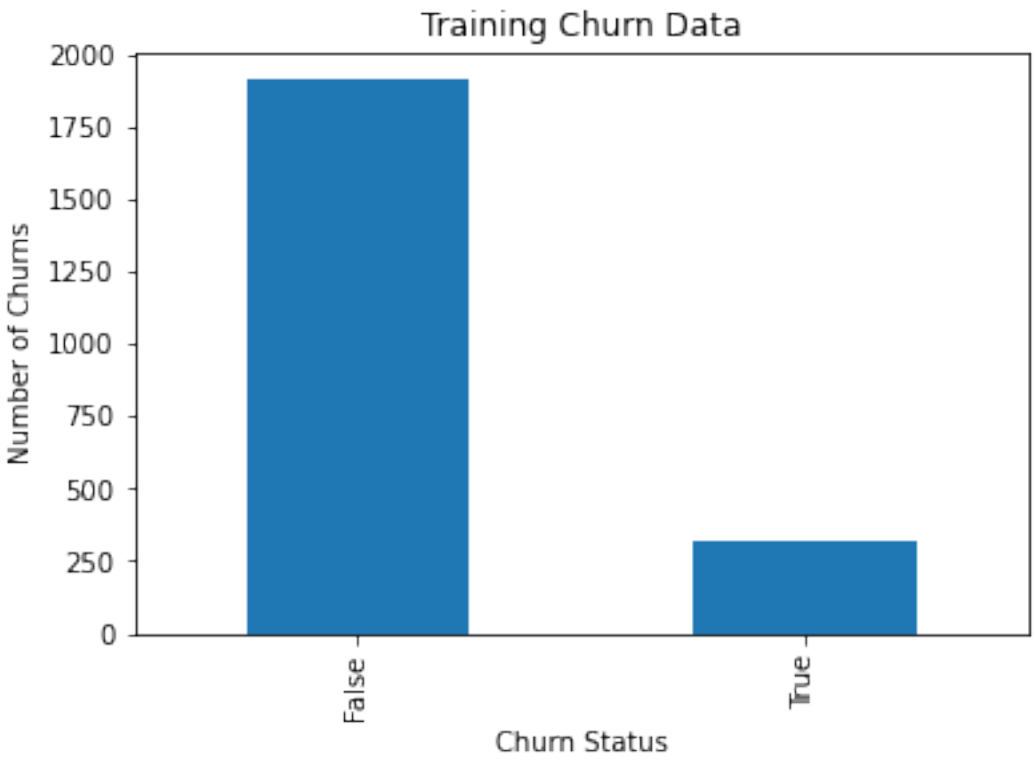
[43]:
```
#total data set
churn.Churn.value_counts().plot(kind='bar', title = 'Full Churn Data')
plt.ylabel('Number of Churns')
plt.xlabel('Churn Status')
print('The total count of rows are:' ,  churn.shape[0])
```

```
The total count of rows are: 3333
```



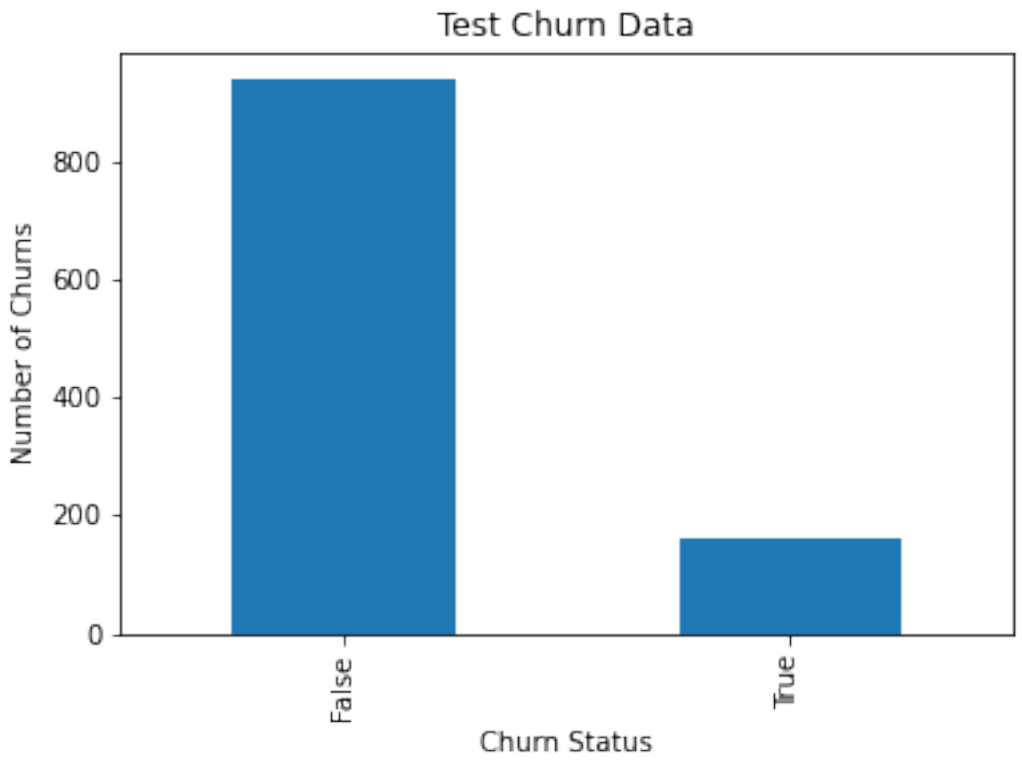
[44]:
```
#training barcharts
churn_train.Churn.value_counts().plot(kind='bar', title = 'Training Churn Data')
plt.ylabel('Number of Churns')
plt.xlabel('Churn Status')
print('The total count of rows are:' ,  churn_train.shape[0])
```

```
The total count of rows are: 2233
```



[45]:
```
#test barcharts
churn_test.Churn.value_counts().plot(kind='bar', title = 'Test Churn Data')
plt.ylabel('Number of Churns')
plt.xlabel('Churn Status')
print('The total count of rows are:' ,  churn_test.shape[0])
```

```
The total count of rows are: 1100
```



[46]:
```
#validated percentages of total data:
print("The training data set contains", round(churn_train.shape[0]/churn.
 ↪shape[0] * 100 ,2),"% of the data.")
print("The test data set contains" , round(churn_test.shape[0]/churn.shape[0] *␣
 ↪100 ,2),"% of the data.")
```

```
The training data set contains 67.0 % of the data.
The test data set contains 33.0 % of the data.
```

1.1.2 29. Identify the total number of records in the training data set and how many records in the training data set have a churn value of true.

[47]:
```
#total number of records in training data set
print("The training data set contains", round(churn_train.shape[0]/churn.
 ↪shape[0] * 100 ,2),"% of the data.")
```

```
The training data set contains 67.0 % of the data.
```

[48]:
```
#how many records have a churn value of true
training_churn_true = churn_train.loc[churn_train['Churn'] == True]
print("The training data set contains", round(training_churn_true.shape[0]),␣
 ↪"records.")
```

```
The training data set contains 320 records.
```

1.1.3 30. Use your answers from the previous exercise to calculate how many true churn records you need to resample in order to have 20% of the rebalanced data set have true churn values.

[49]:
```
#total records in training data set
churn_train.shape[0]
```

[49]:
```
2233
```

[50]:
```
#values of True and False within training set
churn_train['Churn'].value_counts()
```

[50]:
```
False    1913
True      320
Name: Churn, dtype: int64
```

[51]:
```
#find the value of how many we need with 20 %
n_new = round(((0.2 * churn_train.shape[0]) - training_churn_true.shape[0])/0.
 ↪8,0)
print("We need to resample", n_new, "records whose response is 'True' and add␣
 ↪them to our training set")
```

```
We need to resample 158.0 records whose response is 'True' and add them to our
training set
```

1.1.4 31. Perform the rebalancing described in the previous exercies and confirm that 20% of the records in the rebalanced data set have true churn values.

[52]:
```
#isolate records we want to resample (only True Churn's)
to_resample = churn_train.loc[churn_train['Churn'] == True]
```

[53]:
```
#Sample from our records of interest
our_resample = to_resample.sample(n = 158 , replace = True)
```

[15]:
```
#concat two data sets by putting rows on top of eachother (union)
rebalanced_train = pd.concat([churn_train, our_resample])
```

[54]:
```
#view the total count of records in the new data set
rebalanced_train.shape[0]
```

[54]:
```
2391
```

[55]:
```
#check new True and False values in the new rebalanced dataframe
rebalanced_train['Churn'].value_counts()
```

[55]:
```
False    1913
True      478
Name: Churn, dtype: int64
```

[56]:
```
#create a dataframe of just the rebalanced True Churn values and obtain count
total_true = rebalanced_train.loc[rebalanced_train['Churn'] == True]
total_true.shape[0]
```

[56]: 478

[57]:
```
#validate that new percentage of data is 20%
new_test_percentage =  total_true.shape[0]/ rebalanced_train.shape[0]
print(round(new_test_percentage,2)*100,"% of the data is now 'True'.")
```

```
20.0 % of the data is now 'True'.
```

1.1.5 32. Which baseline model do we use to compare our classification model performance against? To which value does this baseline model assign all predictions? What is the accuracy of this model?

The answer is:

1.1.6 33. Validate your partition by testing for the difference in mean "day minutes for the training set versus the test set.

[58]:
```
print("Rebalanced training data 'Day Mins' mean is:",␣
 ↪round(rebalanced_train['Day Mins'].mean(axis=0),2))
print("Churn test data 'Day Mins' mean is:", round(churn_test['Day Mins'].
 ↪mean(axis=0),2))
```

```
Rebalanced training data 'Day Mins' mean is: 181.38
Churn test data 'Day Mins' mean is: 179.82
```

1.1.7 34. Validate your partition by testing for the difference in proportion of true churn records for the training set versus the test set.

[95]:
```
# how do you do this?
```

1.2 Data Science Using Python and R: Chapter 7 - Page 109: Questions 23, 24, 25, 26, 27, 28, 29, 30

1.2.1 23. Using the training data set, create a C5.0 model (Model 1) to predict a customer's Income using Marital Status and Capital Gains Loses. Obtain the predicted responses.

[69]:
```
#import libraries for analysis
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

[86]:
```
#import the training data set
training_data = pd.read_csv(trainin, header = 0)
```

```
#training_data.head()
```

[61]:
```
#set outcome variable as income
y = training_data[['Income']]
```

[62]:
```
#create a numpy array of the different values within the Marital status␣
↪attribute
mar_np = np.array(training_data['Marital status'])
(mar_cat, mar_cat_dict) = stattools.categorical(mar_np, drop=True, dictnames␣
↪=True)
```

```
/Users/ryan_s_dunn/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tools/tools.py:158: FutureWarning: categorical is
deprecated. Use pandas Categorical to represent categorical data and can
get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
```

[88]:
```
mar_cat_pd = pd.DataFrame(mar_cat)

#new dataframe with marital status categorical variables as numbers and capital␣
↪gains/lossees
X = pd.concat((training_data[['Cap_Gains_Losses']],mar_cat_pd), axis = 1)
```

[89]:
```
#assign the names back to the variables
X_names = ['Cap_Gains_Losses', "Divorced", "Married", "Never-married",␣
↪"Separated", "Widowed"]
y_names = ["<=50K",">50K"]
```

[65]:
```
#develop the C5.0 model
model_1 = DecisionTreeClassifier(criterion = "entropy", max_leaf_nodes=5).
↪fit(X,y)
```

[92]:
```
#predict the income variable from the mode.
k = model_1.predict(X)
```

[67]:
```
#develop a C5.0 model to predict a customers Income
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=.30,␣
↪random_state=1)
```

1.2.2 24. Evaluate Model 1 using the test data set. Construct a contingency table to compare the actual and predicted values of Income.

[129]:
```
#evaluate the model
clf = DecisionTreeClassifier()

clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.92      0.83      1597
           1       0.57      0.26      0.35       660

    accuracy                           0.73      2257
   macro avg       0.66      0.59      0.59      2257
weighted avg       0.70      0.73      0.69      2257
```

[124]:
```
#display the confusion matrix
cm = pd.DataFrame(confusion_matrix(y_test, y_pred))
cm
```

[124]:

	0	1
0	1597	0
1	660	0

[147]:
```
#totals for the corresponding metrics
TN_1 = 1597
TP_1 = 0
FN_1 = 660
FP_1 = 0
GT_1 = TN_1 + TP_1 + FN_1 + TP_1
#GT_1
```

1.2.3 25. For Model 1, recapitulate Table 7.4 from the text, calculating all of the model evaluation measures shown in the table. Call this table the Model Evaluation Table. Leave space for Model 2.

[155]:
```
metrics = ["Accuracy", "Error Rate", "Specificity", "Precision","F1"]
m1_eval = [(TN_1 + TP_1)/GT_1, (1 - ((TN_1 + TP_1)/GT_1)), TN_1 / (TN_1 +␣
↪FN_1), 0.00, 2*(0.00 * TN_1 / (TN_1 + FN_1))]
m2_eval = ['NA','NA','NA','NA']

df1 = pd.DataFrame(list(zip(metrics, m1_eval,m2_eval)),columns = ['Metrics',␣
↪'Module 1 Results', 'Module 2 Results'])
df1
```

[155]:

	Metrics	Module 1 Results	Module 2 Results
0	Accuracy	0.707576	NA
1	Error Rate	0.292424	NA
2	Specificity	0.707576	NA
3	Precision	0.000000	NA

1.2.4 26. Clearly and completely interpret each of the Model 1 evaluation measures from the Model Evaluation Method.

For detialed explainations, see the R output.

1.2.5 27. Create a cost matrix, called the 3x cost matrix, that specifies a false positive is four times as bad as a false negative.

[161]:
```
list_1 = [0,1,4,0]
cost_matrix = pd.DataFrame(list_1, rows = 2)
cost_matrix
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-161-fcf8bf71eb30> in <module>
      1 list_1 = [0,1,4,0]
----> 2 cost_matrix = pd.DataFrame(list_1, rows = 2)
      3 cost_matrix

TypeError: __init__() got an unexpected keyword argument 'rows'
```

1.3 Data Science using Python and R: Chapter 8 - Page 126: Questions 31, 32, 33, 34

1.3.1 31. Run the Naive Bayes classifier to classify persons as living or dead based on sex and education.

[71]:
```
from sklearn.naive_bayes import MultinomialNB
```

[72]:
```
#training data
death_training = pd.read_csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied␣
↪Data Mining 502/Module 3/Datasets/Framingham_Training", header = 0)

#test data
death_test = pd.read_csv("/Users/ryan_s_dunn/Documents/USD MS-ADS/Applied Data␣
↪Mining 502/Module 3/Datasets/Framingham_Test", header = 0)
```

[93]:
```
print(death_training.shape)
print(death_test.shape)
```

```
(7953, 5)
(2257, 5)
```

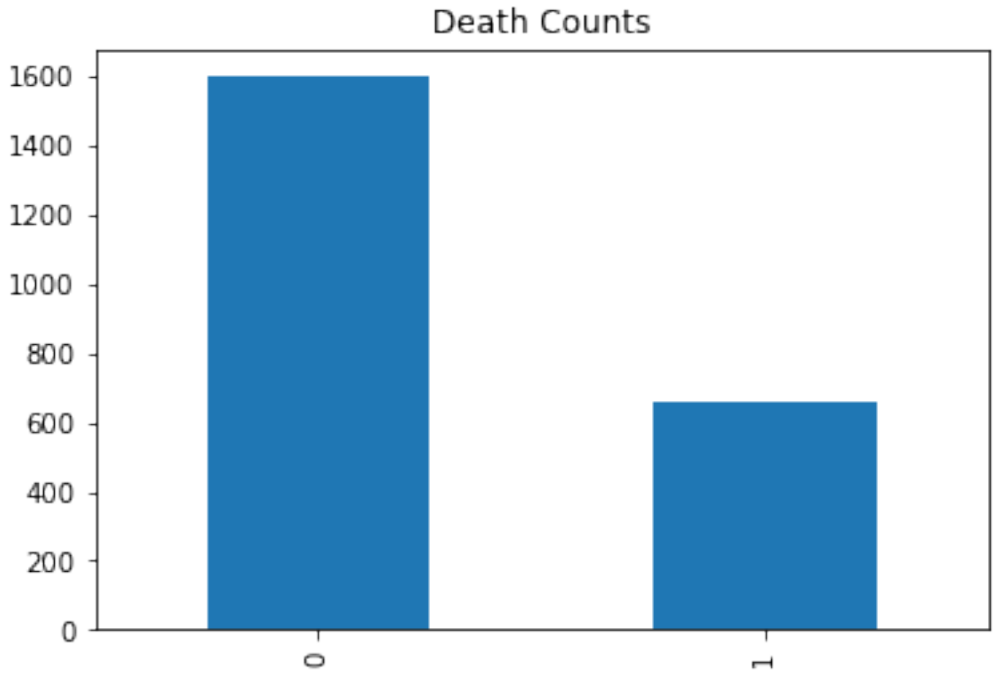
[97]:
```
death_training.head(1)
```

[97]:

	Index	Sex	Age	Educ	Death
0	1	1	39	4	0

[75]:
```
#review the actual count of deaths by 0 & 1 for validation
death_test.Death.value_counts().plot(kind='bar', title = 'Death Counts')
```

[75]:
```
<AxesSubplot:title={'center':'Death Counts'}>
```



[76]:
```
#develop the training Xn and Y variables
X_train = death_training[['Sex','Age']]
y_train = death_training['Death']
```

[77]:
```
#run the Navie Bayes algorithm
nb = MultinomialNB(alpha = 0.5).fit(X_train, y_train)
```

1.3.2 32. Evaluate the Naive Bayes model on the framingham_nb_test data set. Display the results in a contingency table. Edit the row and column names of the table to make the table more readable. Include a total row and column.

[78]:
```
#set up the x variables within the test set
X_test = death_test[['Sex','Age']]
y_test = death_test['Death']
```

[100]:
```
#predict the test variables with the Naive Bayes Model
y_pred = nb.predict(X_test)
```

[101]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      1.00      0.83      1597
           1       0.00      0.00      0.00       660

    accuracy                           0.71      2257
   macro avg       0.35      0.50      0.41      2257
weighted avg       0.50      0.71      0.59      2257
```

```
/Users/ryan_s_dunn/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/ryan_s_dunn/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/ryan_s_dunn/opt/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[106]:
```
cf_dm = pd.DataFrame(confusion_matrix(y_test, y_pred))
#cf_dm = pd.crosstab("True", "False")
cf_dm
```

[106]:

	0	1
0	1597	0
1	660	0

1.3.3 33. According to your table in the previous exercise, find the following values for the Naive Bayes model:

(a) Accuracy:

[81]:
```
from sklearn.metrics import accuracy_score
print('The accuracy is',accuracy_score(y_pred,y_test))
```

```
accuracy is 0.7075764288879043
```

(b) Error Rate:

[120]:
```
print('The error rate is', (1 - accuracy_score(y_pred,y_test)))
```

```
The error rate is 0.2924235711120957
```

1.3.4 34. According to your contingency table, find the following values for the Naive Bayes model:

[162]:
```
#create the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[162]:
```
array([[1471,  126],
       [ 491,  169]])
```

[112]:
```
#select the individual evaluation metrics from the confusion matrix
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]

#review each evaluation method for accuracy
print(TN)
print(FP)
print(FN)
print(TP)
```

```
1597
0
660
0
```

[84]:
```
t1 = pd.crosstab(death_training['Sex'], death_training['Age'])
t1['Total'] = t1.sum(axis=1)
t1.loc['Total'] = t1.sum()
t1
```

[84]:

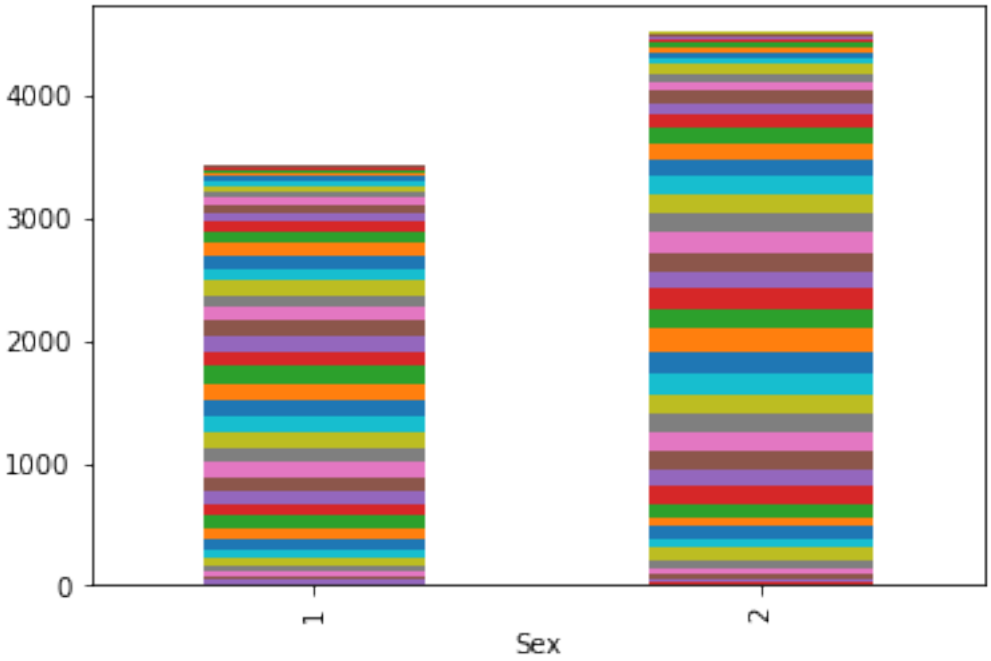
Age	32	33	34	35	36	37	38	39	40	41	…	73	74	75	76	77	78	\
Sex											…							
1	0	1	2	9	33	27	45	55	58	60	…	29	28	17	12	9	0	
2	1	3	8	17	33	34	51	69	90	79	…	46	31	32	17	15	9	
Total	1	4	10	26	66	61	96	124	148	139	…	75	59	49	29	24	9	

Age	79	80	81	Total
Sex				
1	5	2	0	3437
2	8	2	2	4516
Total	13	4	2	7953

[3 rows x 51 columns]

[85]:
```
t1_plot = pd.crosstab(death_training['Sex'], death_training['Age'])
t1_plot.plot(kind='bar', legend = None, stacked = True)
```

[85]: <AxesSubplot:xlabel='Sex'>



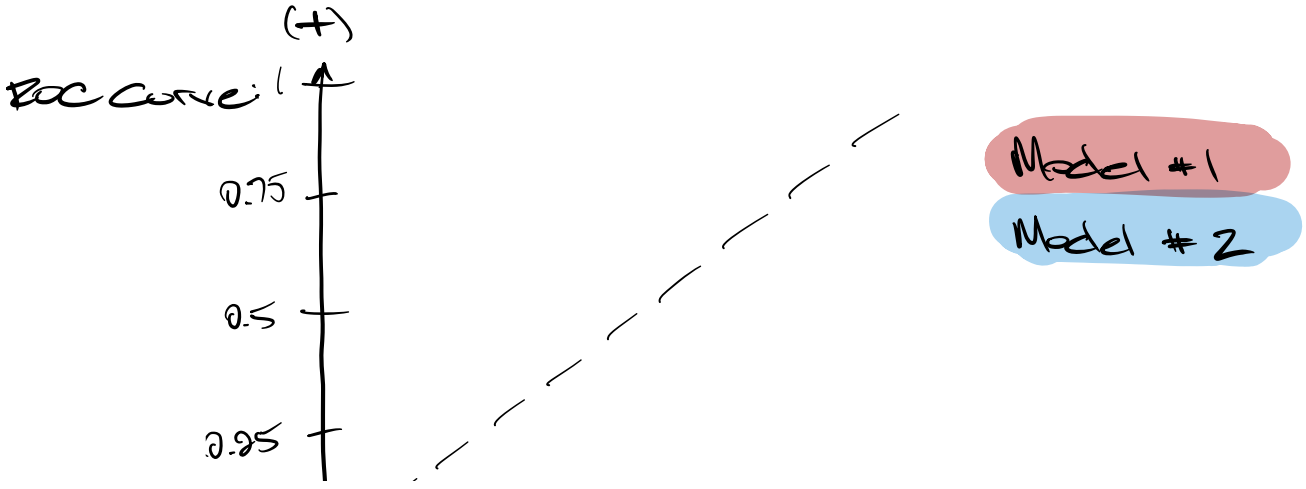
(a) How often it correctly classifies dead persons.

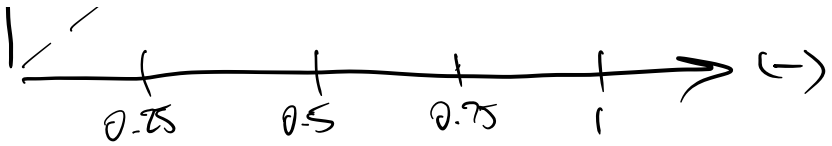
(b) How often it correctly classifies living persons.

Ryan S. Dunn
Module 3 | MIS 502

16. (a) Plot the ROC curve for both M_1 & M_2. You should plot them on the same graph. Which model do you think is better?

Instance	True Class	$P(+\mid A_1 \ldots A_n, M_1)$		$P(-\mid A_1 \ldots A_n, M_2)$	
1	+	0.73	+	0.61	+
2	+	0.69	+	0.03	-
3	-	0.44	-	0.68	+
4	-	0.55	+	0.31	-
5	+	0.67	+	0.45	-
6	+	0.47	-	0.09	-
7	-	0.08	-	0.38	-
8	-	0.15	-	0.05	-
9	+	0.45	-	0.01	-
10	-	0.35	-	0.04	-





(b) For model M_1, suppose you choose the cutoff threshold to be $t = 0.5$. In other words, any test instances whose posterior probability is greater than t, will be classified as a positive example. Compute precision, recall, & F-measure for the model at this threshold value.

Actual \ Predicted	+	−
+	3	2
−	1	4

Precision $= 3/4 = 0.75$

Recall $= 3/5 = 0.60$

$F_1 = 2 \cdot \left(\frac{(.60) \cdot (.75)}{.60 + .75} \right) = 0.666$

(c) Repeat the analysis for part (b) using the same cutoff threshold on model M_2. Compare the F-measure results for both models. Which model is better? Are the results consistent with what you expect from the ROC curve?

Actual \ Predicted	+	−
+	1	4

−	1	4

Precision = 1/2 = 0.50

Recall = 1/5 = 0.20

$F_1 = 2 \cdot \left(\frac{(0.50) \cdot (0.20)}{0.50 + 0.20} \right) = 0.29$

The F_1 score of 0.66 is preferable to the F_1 score of 0.29.

(d) Repeat part (b) for model M_1 using the threshold $t = 0.1$. Which threshold do you prefer, $t = 0.5$ or $t = 0.1$? Are the results consistent with what you expect from the ROC curve?

		Predicted +	Predicted −
Actual	+	2	3
	−	3	2

Precision = 2/5 = 0.4

Recall $= \frac{7}{5} = 0.4$

$F_1 = 2 \cdot \left(\frac{(0.4) \cdot (0.4)}{0.4 + 0.4} \right) = 0.4$

I would prefer the $t = 0.1$ value, as the Recall & F_1 value are both larger.

21. Given the datasets shown in Figures 4.59, explain how the decision tree, naive bayes, and k-nearest neighbor classifiers would perform with these data sets.

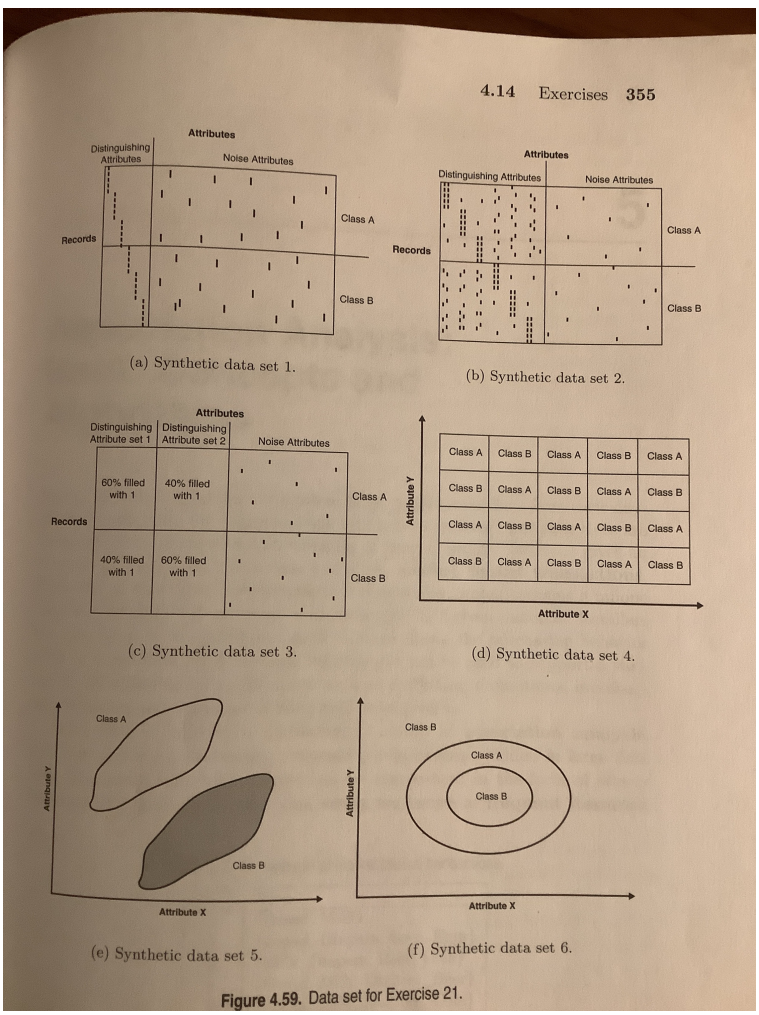


Figure 4.59. Data set for Exercise 21.

(a) Decision Tree - with scattered data, the decision tree algorithm would struggle here.

Naïve Bayes - because we are looking at probabilities, given a class attribute, this algorithm would work well.

KNN - the lines are all in the same general space, so this would work well.

(b) Decision Tree - similar to (a) decision tree's would not work well.

Naïve Bayes - With the attributes too clumped together this algorithm would struggle

KNN - because the data is very scattered, but scattered in the same locations, this algorithm would not work well.

(c) Decision Tree - Decision tree would work well, because the data is already grouped in a decision like box that can be assigned values.

Naïve Bayes - Naïve Bayes would also work well, as we have probabilities assigned to classes already.

KNN - KNN would not work with this data, as it is not plotted, or KNN generally would need to be applied.

(d) Decision Tree - This could possibly work with this data set, since we have some general paths the data appears to follow.

but it would still struggle

Naive Bayes: Similar to D.T.'s, NB may possibly work for this data set.

KNN - KNN would not work here, as the algorithm may contradict the observable pattern

(e) DecisionTree - A D.T. may work here, as the data just needs to know which area it needs to land.

Naive Bayes - works well, since we know the areas that the $P(\text{Class A} \mid X)$ & $P(\text{Class B} \mid X)$ are

KNN - works best here, because the data is in regions where all points are the same.

(f) Decision Tree may struggle, since the data follows a ring path, however because it is categorical & within a defined area, it may work

Naive Bayes - would work here, since we would just need to understand both probabilities of the classes given their dots

KNN - works best here, because the areas are clearly defined.