Homework 5: Classification and its Multiclass Extension

There are six questions (30 total points) in this assignment. The minimum increment is 1 point. Please type in your answers directly in the R Markdown file. After completion, **successfully** knitr it as an html file. Submit **both** the html file and the R Markdown file via Canvas. Please name the R Markdown file in the following format: LastName_FirstName_HW5.Rmd, e.g. Zhao_Zifeng_HW5.Rmd.

Credit Default Dataset [18 points]

The credit default dataset contains information on ten thousand customers. The aim here is to predict which customers will default on their credit card debt. The data is stored in `Default.csv`. It contains 4 variables, `default`, `student`, `balance` and `income`. We would like to build several statistical models to predict the probability of `default` of a person with given personal information. The data description is as follows.

- `default`: A factor with levels No and Yes indicating whether the customer defaulted on their debt
- `student`: A factor with levels No and Yes indicating whether the customer is a student
- `balance`: The average balance that the customer has remaining on their credit card after making their monthly payment
- `income`: Income of customer

Q1 [6 points] Data Partition and Exploration

Q1(a) [2 points] Let's correctly read in the data in `Default.csv` and name it as `total_data`.

```r
#reading in the total data from the csv file
total_data <- read.csv("/Users/TomTheIntern/Desktop/Mendoza/Mod 2/Advanced Stats/Homework 5/Default.csv
                        #setting header as true because the csv has headers
                        header = TRUE,
                        #setting an strings to factors
                        stringsAsFactors = TRUE)
```

Q1(b) [2 points] Among the 10000 customers in the dataset, how many of them default?

```r
#getting the summary of the data to get the number of defaults
default_count <- summary(total_data$default)

#outputting the results of the data
default_count
```

```
##   No  Yes
## 9667  333
```

Of the 10,000 customers, 333 of them are defaults, or roughly 0.033% of the data.

Q1(c) [2 points] Let's partition the data in `total_data` into training **(60%)** and test data **(40%)** and store them as `R` objects `train_data` and `test_data` respectively. Use random seed `set.seed(7)`!

```r
#setting the seed to 7 so the process can be repeated
set.seed(7)

#getting the number of rows of the total data
num_rows <- nrow(total_data)

#determining which rows are training rows
training_rows <- sample(1:num_rows, num_rows * 0.60)

#making the training set
train_data <- total_data[training_rows , ]

#making the testing set
test_data <- total_data[-training_rows, ]
```

Q2 [6 points] Logistic Regression and GAM

Q2(a) [**2 points**] Fit a logistic regression model of `default` w.r.t. all 3 predictors using the **training data**, name it `lm_full`.

```r
#making a logistic regression model
lm_full <- glm(default ~ student + balance + income,
               #setting the data to train data
               data = train_data,
      #setting the family to binomial so that the predictions are scaled correctly
               family = "binomial")
```

Q2(b) [**2 points**] Perform backward selection of `lm_full` via BIC and name the new model `lm_bwd`. Is any variable removed?

```r
#making a backward selection model based on the log regression model
lm_bwd <- step(lm_full, direction = "backward")
```

```
## Start:  AIC=1004.23
## default ~ student + balance + income
##
##             Df Deviance    AIC
## - income     1   996.59 1002.6
## <none>           996.23 1004.2
## - student    1   998.60 1004.6
## - balance    1  1810.88 1816.9
##
## Step:  AIC=1002.59
## default ~ student + balance
##
##             Df Deviance    AIC
## <none>           996.59 1002.6
## - student    1  1007.98 1012.0
## - balance    1  1812.12 1816.1
```

The backward selection process removed the income variable, but kept studentYes and balance.

Q2(c) [**2 points**] Fit a GAM of `default` w.r.t. all 3 predictors using the **training data**, name it `gam1`. Let's use splines with **df=4** for the numerical predictors, which include `balance` and `income`.

```r
#reading in the gam library
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.4.1
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.22-5
```

```r
#making a gam model with splines set to balance and income
gam1 <- gam(default ~ student + s(balance) + s(income),
            #using the train data
            data = train_data,
            #and setting the predictions to binomial
            family = "binomial")
```

Q3 [6 points] Model Evaluation (Prediction)

Q3(a) [**2 points**] Use `lm_full` and `gam1` to generate probability predictions for `default` on the test data and store the predicted probability in `lm_full_pred` and `gam1_pred` respectively.

```r
#making a series of predictions for the log model using the predict function and the test data
lm_full_pred <- predict(lm_full, test_data)

#making a series of predictions for the gam model using the predict function and the test data
gam1_pred <- predict(gam1, test_data)
```

Q3(b) [**2 points**] Use the `confusionMatrix()` function in the `R` package `caret` to evaluate the prediction performance of `lm_full` and `gam1`. What are the sensitivity of `lm_full` and `gam1`?

```r
#writing in the caret library
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```r
#making a confusion matrix based off of the lm_full predictions
#I need to set the variables to as factor so it avoids being set to a character
#and then if else the values above 50 to yes and anything lower to no
confusionMatrix(as.factor(ifelse(lm_full_pred > .50, "Yes", "No")), test_data$default, positive = "Yes"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  3871   87
##        Yes    6   36
##
##                Accuracy : 0.9768
##                  95% CI : (0.9716, 0.9812)
##     No Information Rate : 0.9692
##     P-Value [Acc > NIR] : 0.002527
##
##                   Kappa : 0.4274
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.29268
##             Specificity : 0.99845
##          Pos Pred Value : 0.85714
##          Neg Pred Value : 0.97802
##              Prevalence : 0.03075
##          Detection Rate : 0.00900
##    Detection Prevalence : 0.01050
##       Balanced Accuracy : 0.64557
##
##        'Positive' Class : Yes
##
```

```r
#and then doing the same thing with the gam predictions
confusionMatrix(as.factor(ifelse(gam1_pred> .50, "Yes", "No")), test_data$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  3870   87
##        Yes    7   36
##
##                Accuracy : 0.9765
##                  95% CI : (0.9713, 0.981)
##     No Information Rate : 0.9692
##     P-Value [Acc > NIR] : 0.003428
##
##                   Kappa : 0.4246
##
##  Mcnemar's Test P-Value : 3.693e-16
##
##             Sensitivity : 0.29268
##             Specificity : 0.99819
##          Pos Pred Value : 0.83721
##          Neg Pred Value : 0.97801
##              Prevalence : 0.03075
##          Detection Rate : 0.00900
##    Detection Prevalence : 0.01075
##       Balanced Accuracy : 0.64544
##
##        'Positive' Class : Yes
##
```

lm_full has a sensitivity of 0.29268 and an accuracy of 0.9768

gam1 has a sensitivity of 0.29268 and an accuracy of 0.9765

Q3(c) [**2 points**] Note that the sensitivity of `lm_full` and `gam1` are in the range of 30-40%, which means that the models are having a hard time finding the customers that default. This is not surprising considering that most customers do NOT default. One way to improve sensitivity is to use a threshold **lower** than **0.5** to classify the customer. Let's use a new threshold **0.1**, i.e. we think a customer will default if the predicted probability of default > 0.1.

Use **0.1** as the new classification threshold and calculate the sensitivity for `lm_full` and `gam1`. Did the sensitivity go up? What is the price we need to pay for increasing sensitivity?

```r
#altering the confusion matrix to set the standard for a yes prediction to 10%
confusionMatrix(as.factor(ifelse(lm_full_pred > .1, "Yes", "No")), test_data$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  3865   78
##        Yes   12   45
##
##                Accuracy : 0.9775
##                  95% CI : (0.9724, 0.9819)
##     No Information Rate : 0.9692
##     P-Value [Acc > NIR] : 0.0009516
##
##                   Kappa : 0.4901
##
##  Mcnemar's Test P-Value : 7.303e-12
##
##             Sensitivity : 0.36585
##             Specificity : 0.99690
##          Pos Pred Value : 0.78947
##          Neg Pred Value : 0.98022
##              Prevalence : 0.03075
##          Detection Rate : 0.01125
##    Detection Prevalence : 0.01425
##       Balanced Accuracy : 0.68138
##
##        'Positive' Class : Yes
##
```

```r
#and then doing the same thing with the gam predictions
confusionMatrix(as.factor(ifelse(gam1_pred> .1, "Yes", "No")), test_data$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   No  Yes
##        No  3864   79
##        Yes   13   44
##
##                Accuracy : 0.977
##                  95% CI : (0.9719, 0.9814)
##     No Information Rate : 0.9692
##     P-Value [Acc > NIR] : 0.001844
##
##                   Kappa : 0.4787
##
##  Mcnemar's Test P-Value : 1.229e-11
##
##             Sensitivity : 0.35772
##             Specificity : 0.99665
##          Pos Pred Value : 0.77193
##          Neg Pred Value : 0.97996
##              Prevalence : 0.03075
##          Detection Rate : 0.01100
##    Detection Prevalence : 0.01425
##       Balanced Accuracy : 0.67719
##
##        'Positive' Class : Yes
##
```

The sensitivity shot up for lm_full to 0.36585, but the specificity dropped from 0.99845 to 0.99690.

The sensitivity also improved for the gam model to 0.35772, but the specificity dropped from 0.99819 to 0.99665.

Iris Dataset [12 points]

The famous R.A. Fisher's iris dataset contains the measurements (in centimeters) of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris `setosa`, `versicolor`, and `virginica`.

The data is distributed with `R`. It contains 5 variables, `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width` and `Species`. We would like to build several statistical models to classify the species of a given iris based on its sepal length and width and petal length and width. The data description is as follows.

- `Sepal.Length`: sepal length
- `Sepal.Width`: sepal width
- `Petal.Length`: petal length
- `Petal.Width`: petal width
- `Species`: species

We first read in the data and partition the data in `total_data` into training **(50%)** and test data **(50%)** and store them as `R` objects `train_data` and `test_data` respectively.

```r
library(caret)
total_data <- iris
set.seed(7)
```

```r
total_obs <- nrow(total_data)
train_index <- sample(1:total_obs, 0.5*total_obs)
train_data <- total_data[train_index,]
test_data <- total_data[-train_index,]
```

Q4 [2 points] Multinomial Logistic Regression

Fit a multinomial logistic regression model of `Species` w.r.t. all 4 predictors using the **training data**, name it `lm1`.

```r
#reading in the neural net library
library(nnet)
#making the log regression model
lm1 <- multinom(Species ~ ., data = train_data)
```

```
## # weights:  18 (10 variable)
## initial  value 82.395922
## iter  10 value 0.346201
## iter  20 value 0.256072
## iter  30 value 0.132709
## iter  40 value 0.052666
## iter  50 value 0.033212
## iter  60 value 0.021372
## iter  70 value 0.011800
## iter  80 value 0.009370
## iter  90 value 0.009242
## iter 100 value 0.007993
## final  value 0.007993
## stopped after 100 iterations
```

Q5 [4 points] Multiclass Neural Networks

Fit an NN model of `Species` w.r.t. all 4 predictors using the **training data**, name it `nn1`. For the architecture of NN, let's use one hidden layer with 4 hidden units.

Q5(a) [**2 points**] Let's generate the **training dataset** that are needed for the estimation of NN using the function `model.matrix()` and store it in `x_train_nn`. In addition, use the `scale()` function to standardize the predictors by centering with mean and scaling with sd. Let's further combine the dependent variable `Species` with the standardized predictors `x_train_nn` generated. Let's further generate the **test dataset** that are needed for the out-of-sample prediction evaluation of NN using the function `model.matrix` and store it in `x_test_nn`. Use the `scale()` function to standardize the predictors by centering with mean and scaling with sd.

```r
#beginning to make the data into a matrix form
x_train_nn <- model.matrix( ~. , data = train_data)[ , -7]

#pulling the mean of the data from each of the columns
mean <- apply(x_train_nn, 2, FUN = mean)

#pulling the sd from each of the columns
sd <- apply(x_train_nn, 2, FUN = sd)
```

```r
#scaling the data on the mean and the sd of the data
x_train_nn <- scale(x_train_nn, center = mean, scale = sd)[ ,-1]

#binding label to the data
x_train_nn <- data.frame(Species = as.factor(train_data$Species), x_train_nn)[ , -6]

#renaming the label
colnames(x_train_nn)[1] <- "Species"

#making the test data frame
x_test_nn <- model.matrix(~., data = test_data)[ , -7]

#scaling using the old predictors
x_test_nn <- scale(x_test_nn, center = mean, scale = sd)[ , -1]

#bidning label to the data
x_test_nn <- data.frame(Species = as.factor(test_data$Species), x_test_nn)[ , -6]

#renaming the data
colnames(x_test_nn)[1] <- "Species"
```

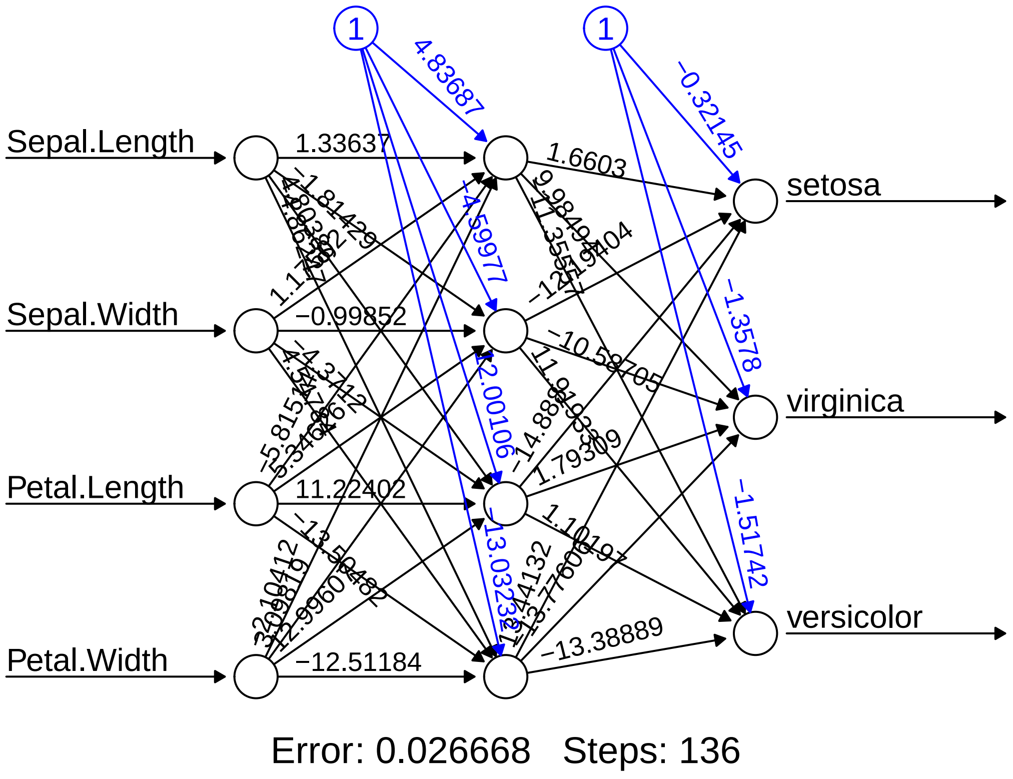
Q5(b) [**2 points**] Let's fit an NN that has one hidden layer with 4 hidden units and name it `nn1`. Make sure to use random seed **`set.seed(7)`**!

```r
#loading the neural net library
library(neuralnet)

#setting the seed to 7 for repeatability
set.seed(7)

#making the neural net model based on the train data, setting the output to non-linear so it returns mu
nn1 <- neuralnet(Species ~ ., hidden = 4, data = x_train_nn, linear.output = FALSE)

#ploting the neural network model
plot(nn1, rep = "best")
```



Q6 [6 points] Model Evaluation (Prediction)

Q6(a) [2 points] Use `lm1` and `nn1` to generate probability predictions for `Species` on the test data and store the predicted probability in `lm1_pred` and `nn1_pred` respectively.

```r
#making predictions on the log model
lm1_pred <- predict(lm1, newdata = test_data, type = "probs")

#making predictions on the neural network model
nn1_pred <- predict(nn1, newdata = x_test_nn, type = "probs")
```

Q6(b) [2 points] Use the `confusionMatrix()` function in the `R` package `caret` to evaluate the prediction performance of `lm1` and `nn1`.

```r
#reading in caret so I can make a confusion matrix
library(caret)

#pulling in the class labels
class_label <- levels(x_train_nn$Species)

#taking the most likely prediction and making it the label
nn1_pred <- factor(class_label[apply(nn1_pred, 1, which.max)])

#making a confusion matrix
confusionMatrix(nn1_pred, x_test_nn$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         27         0
##   virginica       0          4        22
##
## Overall Statistics
##
##                Accuracy : 0.9467
##                  95% CI : (0.869, 0.9853)
##     No Information Rate : 0.4133
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9196
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.8710           1.0000
## Specificity                 1.0000            1.0000           0.9245
## Pos Pred Value              1.0000            1.0000           0.8462
## Neg Pred Value              1.0000            0.9167           1.0000
## Prevalence                  0.2933            0.4133           0.2933
## Detection Rate              0.2933            0.3600           0.2933
## Detection Prevalence        0.2933            0.3600           0.3467
## Balanced Accuracy           1.0000            0.9355           0.9623
```

```r
#taking the most likely preidction and making it the label
lm1_pred <- factor(class_label[apply(lm1_pred, 1, which.max)])

#making a confusion matrix from the data
confusionMatrix(lm1_pred, test_data$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         27         0
##   virginica       0          4        22
##
## Overall Statistics
##
##                Accuracy : 0.9467
##                  95% CI : (0.869, 0.9853)
##     No Information Rate : 0.4133
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9196
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.8710           1.0000
## Specificity                 1.0000            1.0000           0.9245
## Pos Pred Value              1.0000            1.0000           0.8462
## Neg Pred Value              1.0000            0.9167           1.0000
## Prevalence                  0.2933            0.4133           0.2933
## Detection Rate              0.2933            0.3600           0.2933
## Detection Prevalence        0.2933            0.3600           0.3467
## Balanced Accuracy           1.0000            0.9355           0.9623
```

Q6(c) [2 points] Which statistical model do you prefer, `lm1` or `nn1`? Give reasons.

Answer:

Both models have an accuracy of 0.9467 and perform equally well, so the tiebreaker is picking the model with more of a parsominious structure, which would be the log model.