R - An Approach to Multi-class Classification Problem using XGboost

Arnab Panja

2021-04-15

1. Introduction

Multi-class non-linear tree based classification problem is quite common in the real business world of machine learning and statistical modeling. Its just a few days back I thought upon a requirement to classify a set of data into one among a set of classications. This notebook attempts to devise an approach to solve this problem.

We take the palmerpenguins data set from Allison Horst to identify a multi-level classification problem and then attempt to devise a statistical model for predicting the classes for a test data set.

This paper is inspired by the above problem definition and the Rpubs Link **https://rpubs.com/dalekube/XGBoost-Iris-Classification-Example-in-R**. This link has applied the same principles on the IRIS data set. The below approach has applied the same principles on the Palmer Penguins data set and in the process has touched upon the uses of the below concepts of R

1. Handling missing data
2. apply() function
3. vector look ups
4. Xgboost's multi-class classification
5. data visualization with ggplot

So we first load the palmer penguins data.

```r
# Load the data into a data frame
penguins_data_full <- palmerpenguins::penguins


penguins_data_full <- as.data.frame(penguins_data_full,
                                    stringsAsFactors = FALSE)


# describe the data
dim(penguins_data_full)
```

```
## [1] 344   8
```

```r
names(penguins_data_full)
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

As we can observe the palmer penguins data set contains 344 observations and 8 variables. The names of the variables are also as printed by the dim function above.

The summary of the data can also be checked using the summary function as below

```
# Summary
summary(penguins_data_full)
```

```
##       species          island    bill_length_mm  bill_depth_mm  
##  Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10  
##  Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60  
##  Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30  
##                                  Mean   :43.92   Mean   :17.15  
##                                  3rd Qu.:48.50   3rd Qu.:18.70  
##                                  Max.   :59.60   Max.   :21.50  
##                                  NA's   :2       NA's   :2      
##  flipper_length_mm  body_mass_g       sex           year     
##  Min.   :172.0     Min.   :2700   female:165   Min.   :2007  
##  1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007  
##  Median :197.0     Median :4050   NA's  : 11   Median :2008  
##  Mean   :200.9     Mean   :4202                Mean   :2008  
##  3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009  
##  Max.   :231.0     Max.   :6300                Max.   :2009  
##  NA's   :2         NA's   :2
```

2. Handling Missing Data

The summary function quite clearly indicates the presence of NA values (missing values) in the data set. The missing values can also be found by using the apply function on all the variables of the data set. The **MAGIC !!!** of the apply() function is put to good use here.

```
v_na_values <- apply(X = penguins_data_full,
                     MARGIN = 2,
                     FUN = function(x) sum(is.na(x)))


v_na_values[v_na_values > 0]
```

```
##    bill_length_mm     bill_depth_mm flipper_length_mm       body_mass_g 
##                 2                 2                 2                 2 
##               sex 
##                11
```

The first statement stores the count of NA values for all the variables using the apply() function (is this not a wonderful way !!!) and then the second statement outputs just the variables which have NA values in them.

We now attempt to handle the missing values in these variables.

1. For numeric variables we take the mean value and replace the missing values with this mean value.
2. For character variable sex we replace the missing observation with the sex value that has occured the most in the data set.

This is done in just a few steps as below

```
# populate missing values with mean of each numeric variables

penguins_data_full[, c(3, 4, 5, 6)] <-
  apply(X = penguins_data_full[, c(3, 4, 5, 6)], MARGIN = 2,
  FUN = function(x) ifelse(is.na(x),
                           round(mean(x, na.rm = TRUE),digits = 1),x))
```

```r
# populate missing sex values with maximum occurrences of sex type
penguins_data_full$sex <- ifelse(
  is.na(penguins_data_full$sex),
  names(table(penguins$sex)[table(penguins$sex) == max(table(penguins$sex))]),
                                as.character(penguins_data_full$sex))
```

Once again we see the **MAGIC !!!** of the apply() functions to replace the missing values in the data set with the mean value. The vectorized functions in R are its most impressive aspects though it can be daunting to understand it first and then apply them to suitable situations. The above is one such situation where in other programming languages this would have been done by using loops but in R the vectorized apply() function is quite fast and does the work.

3. Statistical Modeling

We would now like to build a tree-based non-linear model that would predict the species of a penguin given the 5 predictors as below

1. bill depth
2. bill length
3. flipper length
4. body mass
5. sex

So with this objective we set out to prepare the predictor variables and the response variables so that it can fed into the xgboost algortihm.

```r
# removing unwanted variables island and year
# from modeling - island and year
penguins_data <- penguins_data_full[, -c(2, 8)]

# converting the factor variables into a sparse integer matrix
# for xgboost inputs

penguins_data$sex <- as.integer(as.factor(penguins_data$sex))

# IMPORTANT TO NOTE :-
# the response classes needs to be converted to
# integers STARTING from 0. Hence a -1 is done

penguins_data$species <- as.integer(as.factor(penguins_data$species)) - 1
```

3a. Split Train and Test Data

We now start the modeling process by performing the below 3 steps

1. Split the data set into a 80:20 ratio, where the 80% of the observations will form the training set and the remaining 20% the test set
2. We define the paramerts of the xgboost algorithm
3. We then train the xgboost model with the selected parameters and the training data set

```r
set.seed(1234)


# splitting the data set into train and test
train_index <- sample(1:nrow(penguins_data), round(0.8*nrow(penguins_data),
                                                    digits = 0))
test_index <- setdiff(1:nrow(penguins_data),
                      train_index)


# create the training variables and label variables for training
penguins_train <- as.matrix(penguins_data[train_index, -1])
label_train <- as.matrix(penguins_data[train_index, 1])

# create the test variables and label variables for testing
penguins_test <- as.matrix(penguins_data[test_index, -1])
label_test <- as.matrix(penguins_data[test_index, 1])


# create the xgb.Dmatrix objects for train and test
xgb_train <- xgb.DMatrix(data = penguins_train,
                         label = label_train)
xgb_test <- xgb.DMatrix(data = penguins_test,
                        label = label_test)
```

Please note that for xgboost model we need the input dataset to be transformed into a sparse matrix (xgb.Dmatrix object). This is accomplished as above. The response variable is also converted into an integer matrix starting with 0. These are all pre-requisites for training the xgboost model.

3b. Parameters of the Model

We now define the parameters of the model and train the model using the below set of scripts

```
# Find the number of classification outcomes
no_classes <- length(unique(penguins_data$species))

# Create the parameter list for xgboost
param_list <- list(booster = "gbtree",
                   eta = 0.001,
                   max_depth = 5,
                   gamma = 3,
                   subsample = 0.75,
                   colsample_bytree = 1,
                   objective = "multi:softprob",
                   eval_metric = "mlogloss",
                   num_class = no_classes)
```

As can be seen the objective is set as multi:softprob. This will help determine the probabilities for an observation to be in the different species.

3c. Model Formation

We now would fit an xgboost model with the above parameters.

```
xgb_fit <- xgb.train(params = param_list,
                     data = xgb_train,
                     nrounds = 10000,
                     early_stopping_rounds = 10,
                     watchlist = list(val1 = xgb_train,
                                      val2 = xgb_test),
                     verbose =  0)

# Results converging after 4035 iterations
xgb_fit
```

```
## ##### xgb.Booster
## raw: 13.6 Mb
## call:
##   xgb.train(params = param_list, data = xgb_train, nrounds = 10000,
##     watchlist = list(val1 = xgb_train, val2 = xgb_test), verbose = 0,
##     early_stopping_rounds = 10)
## params (as set within xgb.train):
##   booster = "gbtree", eta = "0.001", max_depth = "5", gamma = "3", subsample = "0.75", colsample_byt
## xgb.attributes:
##   best_iteration, best_msg, best_ntreelimit, best_score, niter
## callbacks:
##   cb.evaluation.log()
##   cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
##     verbose = verbose)
## # of features: 5
## niter: 4340
## best_iteration : 4330
## best_ntreelimit : 4330
## best_score : 0.097098
## best_msg : [4330]    val1-mlogloss:0.108302  val2-mlogloss:0.097098
## nfeatures : 5
## evaluation_log:
##     iter val1_mlogloss val2_mlogloss
##        1      1.097282      1.097277
##        2      1.095973      1.095935
## ---
##     4339      0.108300      0.097106
##     4340      0.108300      0.097107
```

We see that the model converges after about 4000 iterations which is an indicator that the predictions of the model have shown a very low error consistently.

3c. Model Predictions

We now use the trained model to predict the outputs on the test observations.

```
# predicting new responses
xgb_pred <- predict(object = xgb_fit,
                    newdata = penguins_test,
                    reshape = TRUE)

xgb_pred <- as.data.frame(xgb_pred, stringsAsFactors = FALSE)

xgb_pred$max_pred <- apply(X = xgb_pred,
                           MARGIN = 1,
                           FUN = function(x)
                           colnames(xgb_pred)[which.max(x)])
```

As we had mentioned the predictions below in columns V1, V2 and V3 are the probability of a test observation to belong to these classes. V1, V2 and V3 would correspond to the levels of the factor variable species. The output predicted data with probailities is as below.

```
##          V1         V2         V3 max_pred
## 1 0.9570312 0.02801801 0.01495086       V1
## 2 0.9563202 0.02870170 0.01497813       V1
## 3 0.9564791 0.02783346 0.01568738       V1
## 4 0.9471916 0.02817016 0.02463823       V1
## 5 0.9567070 0.02832416 0.01496883       V1
## 6 0.8851076 0.07749156 0.03740082       V1
```

We observe that each of the records have a probaility and the apply() function above has selected the class corresponding to the maximum probability. We definitely need to map the class variables V1, V2, V3 to their actual species which we will do below using another of R's **MAGIC !!!** of vector lookups.
We also now calculate the prediction accuracy of the model. All of these steps are accomplished by the below scripts.

3d. Prediction Accuracy

The accuracy of the predictions from the model can be determined with the following steps and visualizations.

```r
# Calculating prediction accuracy

v_species_lkp <- levels(penguins_data_full$species)
names(v_species_lkp) <- c("V1", "V2", "V3")

xgb_pred$species <- unname(v_species_lkp[xgb_pred$max_pred])

final_penguins_data <- cbind(penguins_data_full[test_index, ],
                             "pred_species" = xgb_pred$species)

# first 6 rows of the predicted data set
head(final_penguins_data[, c(1, 3, 4, 5, 6, 7, 9)])
```

```
##    species bill_length_mm bill_depth_mm flipper_length_mm body_mass_g    sex
## 1   Adelie           39.1          18.7               181        3750   male
## 3   Adelie           40.3          18.0               195        3250 female
## 8   Adelie           39.2          19.6               195        4675   male
## 11  Adelie           37.8          17.1               186        3300   male
## 16  Adelie           36.6          17.8               185        3700 female
## 18  Adelie           42.5          20.7               197        4500   male
##    pred_species
## 1        Adelie
## 3        Adelie
## 8        Adelie
## 11       Adelie
## 16       Adelie
## 18       Adelie
```

The test observations with the predicted species alongside in the last variable shows a glimpse of the output above. As we can see from the scripts the levels of the species have defined a look up vector and this has been used to look up and determine the predicted species.

The confusion matrix of the model is as determined below

```r
# distribution & reconciliation counts
# per group
```

```r
df_conf_matrix <- as_tibble(final_penguins_data) %>%
  count(species,
        pred_species,
        sort = TRUE) %>%
  mutate(recon = ifelse(species == pred_species,
                        "matched",
                        "not matched")) %>%
  arrange(recon, species, pred_species)

df_conf_matrix
```

```
## # A tibble: 4 x 4
##   species   pred_species     n recon
##   <fct>     <chr>        <int> <chr>
## 1 Adelie    Adelie          24 matched
## 2 Chinstrap Chinstrap       12 matched
## 3 Gentoo    Gentoo          32 matched
## 4 Chinstrap Adelie           1 not matched
```

```r
v_matched <- as_vector(df_conf_matrix %>%
                         filter(recon == "matched") %>%
  group_by() %>% summarise(matched = sum(n)))

v_unmatched <- as_vector(df_conf_matrix %>%
                           filter(recon == "not matched") %>%
  group_by() %>% summarise(matched = sum(n)))
```

And the prediction accuracy as below

```r
# prediction accuracy figure
pred_accuracy <-
  round(sum(ifelse(final_penguins_data$species ==
                     final_penguins_data$pred_species,
                     1,
                     0))/nrow(final_penguins_data) * 100, digits = 2)

# Print - Prediction Accuracy value of xgboost prediction
paste0("Prediction Accuracy = ", pred_accuracy, "%")
```

```
## [1] "Prediction Accuracy = 98.55%"
```

We observe that the prediction accuracy of the model is significantly high.

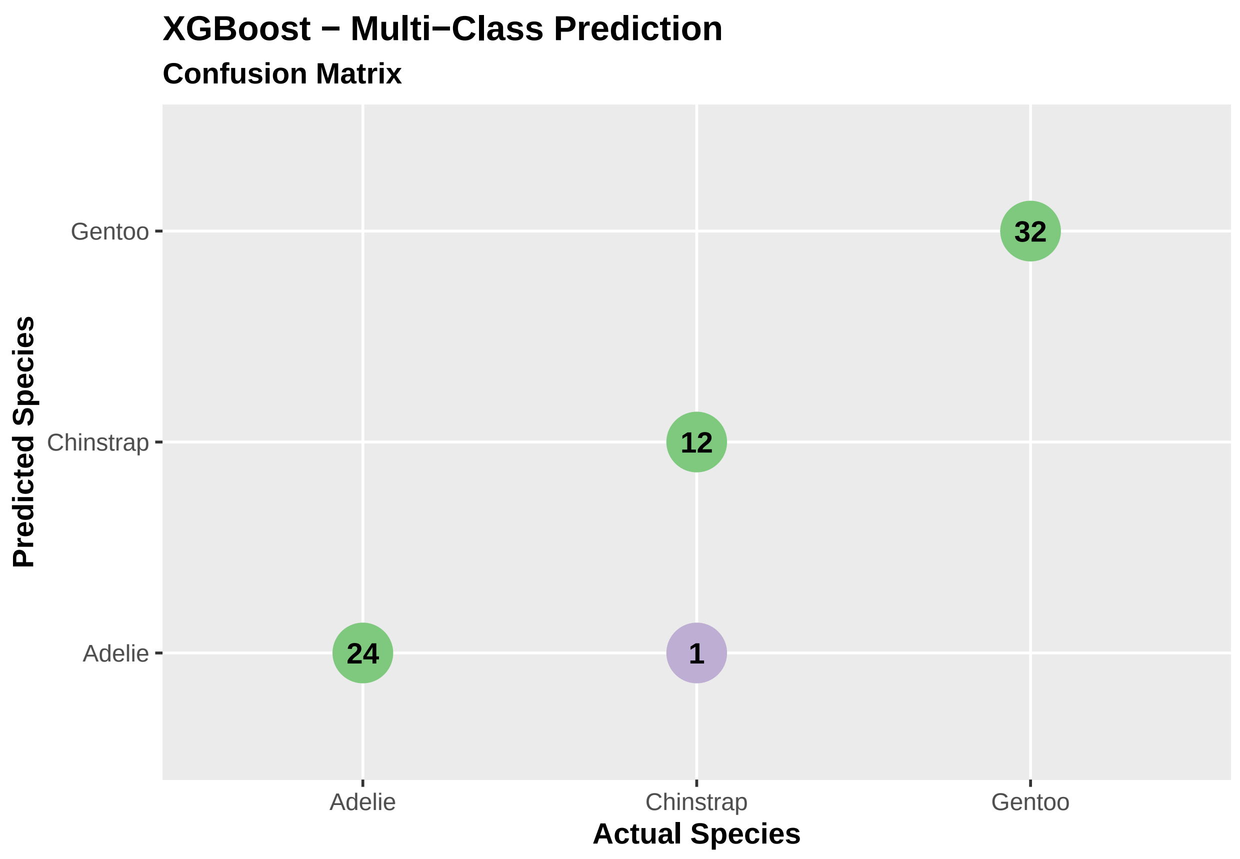
The confusion matrix is a good visualization to understand the number of observations where the predictions have come good and where they have failed. Let us plot this data using the lovel ggplot package and the two geoms - geom_point and geom_text.

No work in R can be complete without the awesome **ggplot() package** plot.

```r
# Visualization of Confusion Matrix

p_confusion_matrix <- as_tibble(final_penguins_data) %>%
  count(species,
        pred_species,
        sort = TRUE) %>%
  mutate(recon = ifelse(species == pred_species,
                        "matched",
                         "not matched")) %>%
  arrange(recon, species, pred_species) %>%
  ggplot() +
  geom_point(mapping = aes(x = species,
                           y = pred_species,
                           color = recon),
             show.legend = FALSE,
             size = 10) +
  geom_text(mapping = aes(x = species,
                          y = pred_species,
                          label = n),
            show.legend = FALSE,
            size = 4,
            color = "black",
            fontface = "bold") +
  scale_color_brewer(palette = "Accent") +
  theme(axis.title = element_text(face = "bold"),
        plot.subtitle = element_text(face = "bold"),
        plot.title = element_text(face = "bold")) +
  labs(x = "Actual Species",
       y = "Predicted Species",
       title = "XGBoost - Multi-Class Prediction",
       subtitle = "Confusion Matrix")

p_confusion_matrix
```



As the above visualizations show, the number of observations where the predictions have matched are 68 and where they have not matched are 1. This essentially means that given the predictors as above the xgboost model is able to predict the species of the penguins with a very high degree of accuracy.

4. Conclusions & References

The analysis here has helped us understand the below concepts and its applications using the R Programing Language

1. Handling missing data
2. Using R's apply() function and vector look ups
3. Applying xgboost algorithm to approach a mulit-class classification problem

The details of the data set and the reference link which inspired me to write this and apply them on the penguins data set is mentioned as below.

Data Set :- https://allisonhorst.github.io/palmerpenguins/

Reference Link :- https://rpubs.com/dalekube/XGBoost-Iris-Classification-Example-in-R

Tidy Tuesday :- https://github.com/rfordatascience/tidytuesday