Classification of Hoops' Longwing Populations

Zane Billings

4/1/2020

Introduction

As part of the analysis of the Hoops' Longwing (*Heliconius hoopsus*) study data from my research project in Indonesia, one of the goals was to build a model to predict which subpopulation an individual butterfly is from.

Here, we will use the `caret` package in order to build a *k*-Nearest Neighbors (or kNN for short) model to predict where the butterflies are from.

Building the Model

First things first: let's load in the data and get the `caret` package loaded. Note that loading `caret` automatically loads some useful packages: notably, `data.table`, `ggplot2`, and `dplyr`.

Remember, we want to remove the first column of this particular csv file because it is not useful for us, and we also want to set the last column to be a `character` type vector, because we know it is not a true factor, but a string instead.

```
library(caret)

hoops <- read.csv("hoops_longwing_study.csv")
hoops <- hoops[ ,-1]
hoops$sample_id <- as.character(hoops$sample_id)
```

We might already have an idea of what the Hoops' longwing data looks like from our previous exploration, but let's look at the structure just to make sure everything has imported correctly.

```
str(hoops)
```

```
## 'data.frame':    100 obs. of  16 variables:
##  $ wing_length       : num  28.2 12.2 12.8 16.2 31 ...
##  $ wing_width        : num  12.04 6.23 6.69 7.49 12.31 ...
##  $ age               : int  16 11 47 16 13 26 27 12 10 13 ...
##  $ num_offspring     : int  28 23 37 26 25 31 28 23 24 23 ...
##  $ feeding_range     : num  3.95 3.62 20.23 2.32 3.79 ...
##  $ color_peak        : num  402 393 393 400 390 ...
##  $ num_mates         : int  11 3 2 5 12 3 3 2 9 4 ...
##  $ avg_scale_size    : num  52 27.2 24.5 30.8 56.1 ...
##  $ antenna_length    : num  6.08 2.44 2.68 3.72 6.39 3.69 3.42 2 5.27 3.37 ...
##  $ num_spots         : int  4 8 7 6 4 8 7 11 4 5 ...
##  $ population        : Factor w/ 3 levels "Kayoa","Ternate",..: 3 2 2 2 3 2 2 2 3 2 ...
##  $ dispersal_distance: num  25.4 24.7 24.8 25.1 25.1 ...
##  $ body_length       : num  9.29 4.89 4.7 6.49 8.78 7.17 7.5 3.77 7.95 6.16 ...
##  $ latitude          : num  0.716 0.818 0.818 0.818 0.716 ...
```

```
##  $ longitude         : num  127 127 127 127 127 ...
##  $ sample_id         : chr  "Tid_001_EM" "Ter_002_EM" "Ter_003_ZB" "Ter_004_EM" ...
```

With the changes we've made, everything looks fine. Now we need to get our data ready for the model.

Basically, the way kNN works (I bet you have been eagerly waiting for me to actually explain this) is that we, the experimenters, choose a value, k, and a data point is assigned to a category by looking at the k number of neighboring points which have a Euclidean distance closest to it. (Remember the Euclidean distance is just the regular distance "as the crow flies" in real life, but it can be generalized to n-dimensions.)

So, we consider the set of k points with the lowest Euclidean distances to the point we want to assign to a category, and we ask all of those points "which category are you in", and we assign our new point to the plurality answer.

However, `caret` is so sleek and high-tech that it will even handle choosing the right value of k for us. All we have to do is specify some parameters.

Now, let's make some changes to the dataset that will help us during the modeling process.

So, we only want numeric data, and we'll exclude the latitude and longitude, because we don't want to use the geographic location where the butterfly was collected as our only way to predict what population the butterfly is from.

Then, we'll *normalize* the data, which can help to improve kNN accuracy (see Lantz, Machine Learning With R). Let's look at a summary of our data.

```r
hoops_mod <- subset(hoops, select = -c(latitude, longitude, sample_id))
summary(hoops_mod)
```

```
##   wing_length      wing_width          age       num_offspring
##  Min.   :10.85   Min.   : 4.380   Min.   : 7.0   Min.   :18.00
##  1st Qu.:14.25   1st Qu.: 6.915   1st Qu.:13.0   1st Qu.:24.00
##  Median :16.21   Median : 7.900   Median :16.5   Median :27.00
##  Mean   :19.83   Mean   : 8.692   Mean   :20.8   Mean   :27.31
##  3rd Qu.:25.96   3rd Qu.:10.430   3rd Qu.:26.0   3rd Qu.:30.00
##  Max.   :36.65   Max.   :14.710   Max.   :52.0   Max.   :37.00
##  feeding_range     color_peak      num_mates     avg_scale_size
##  Min.   : 0.610   Min.   :369.1   Min.   : 1.00   Min.   :22.12
##  1st Qu.: 2.638   1st Qu.:383.8   1st Qu.: 3.00   1st Qu.:27.81
##  Median : 3.380   Median :391.5   Median : 4.00   Median :30.82
##  Mean   : 5.102   Mean   :391.4   Mean   : 5.95   Mean   :37.66
##  3rd Qu.: 4.518   3rd Qu.:398.9   3rd Qu.: 9.00   3rd Qu.:47.71
##  Max.   :30.310   Max.   :411.3   Max.   :16.00   Max.   :75.03
##  antenna_length    num_spots       population dispersal_distance
##  Min.   :2.000   Min.   : 3.00   Kayoa  :11   Min.   :22.58
##  1st Qu.:3.188   1st Qu.: 4.00   Ternate:57   1st Qu.:23.99
##  Median :3.700   Median : 6.00   Tidore :32   Median :24.70
##  Mean   :4.286   Mean   : 5.71                Mean   :24.62
##  3rd Qu.:5.670   3rd Qu.: 7.00                3rd Qu.:25.18
##  Max.   :7.140   Max.   :11.00                Max.   :26.48
##   body_length
##  Min.   : 2.750
##  1st Qu.: 4.995
##  Median : 6.430
##  Mean   : 6.764
##  3rd Qu.: 8.330
##  Max.   :11.520
```

We can see that all of our numeric variables are measured on different scales, which can lead to weird notions of distance between points. So, we'll set all of our data to a common scale of measure (this is call normalization). There are two major normalization methods to think about.

We could use *min-max* normalization:

$$X* = \frac{X - \min(X)}{\max(X) - \min(X)},$$

or we could use *Z-score* normalization:

$$X* = \frac{X - \bar{X}}{s_X}.$$

Min-max is more traditional for kNN, and works well if we are not concerned about cpredicting values outside the range of the values we currently have. On the other hand, z-score normalization works well if we think our sample is representative of the population. For now, we'll go with the min-max method, but feel free to see if using the z-score method gives you a different model.

```
normalize <- function(x) {
  xstar <- (x - min(x)) / (max(x) - min(x))
  return(xstar)
}

normalize_df <- function(cols) {
  nlist <- lapply(cols, normalize)
  return(as.data.frame(nlist))
}
```

We'll use these functions to normalize the data. However, note that we do NOT want to normalize our categorical response variable, `population`. So, this will require a little bit of finagling, but it isn't too bad.

```
hoops_temp <- normalize_df(subset(hoops_mod, select = -population))
hoops_norm <- cbind("population" = hoops_mod$population,
                    hoops_temp)
summary(hoops_norm)
```

```
##    population  wing_length       wing_width          age
##  Kayoa   :11   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  Ternate:57    1st Qu.:0.1318   1st Qu.:0.2454   1st Qu.:0.1333
##  Tidore :32    Median :0.2079   Median :0.3408   Median :0.2111
##                Mean   :0.3479   Mean   :0.4174   Mean   :0.3067
##                3rd Qu.:0.5858   3rd Qu.:0.5857   3rd Qu.:0.4222
##                Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##  num_offspring    feeding_range      color_peak       num_mates
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3158   1st Qu.:0.06827   1st Qu.:0.3475   1st Qu.:0.1333
##  Median :0.4737   Median :0.09327   Median :0.5308   Median :0.2000
##  Mean   :0.4900   Mean   :0.15124   Mean   :0.5280   Mean   :0.3300
##  3rd Qu.:0.6316   3rd Qu.:0.13157   3rd Qu.:0.7058   3rd Qu.:0.5333
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##  avg_scale_size   antenna_length     num_spots       dispersal_distance
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1075   1st Qu.:0.2310   1st Qu.:0.1250   1st Qu.:0.3615
##  Median :0.1645   Median :0.3307   Median :0.3750   Median :0.5423
##  Mean   :0.2938   Mean   :0.4447   Mean   :0.3387   Mean   :0.5242
##  3rd Qu.:0.4836   3rd Qu.:0.7140   3rd Qu.:0.5000   3rd Qu.:0.6660
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

```
##   body_length
##  Min.   :0.0000
##  1st Qu.:0.2560
##  Median :0.4196
##  Mean   :0.4577
##  3rd Qu.:0.6363
##  Max.   :1.0000
```

Now, we'll split our data into training and testing datasets. We'll also set a random seed to make our experiment reproducible. The function `caret::createDataPartition()` is an easy way to split up the data by random sampling.

```r
set.seed(340)

training_indices <- createDataPartition(y = hoops_mod$population,
                                        p = 0.7,
                                        list = FALSE)
hoops_train <- hoops_mod[training_indices, ]
hoops_test <- hoops_mod[-training_indices, ]
```

The interested reader will note that if we examine `table(hoops$population)`, we have a class imbalance issue. Good job on spotting that! But I am going to ignore it for now. Something to think about in your spare time would be ways to remedy this issue.

Now we'll train the kNN model. There are some specifics for `caret` here that you can read about if you'd like, but basically you just need to know that `trainControl()` provides some important parameters for how the modeling process works in `caret`, and the function `train(method = "knn")` is what's actually fitting the model for us.

```r
set.seed(340)
parms <- trainControl(method = "repeatedcv",
                      number = 10,
                      repeats = 3)
knn_fit <- train(population ~.,
                 data = hoops_train,
                 method = "knn",
                 trControl = parms,
                 preProcess = c("center", "scale"),
                 tuneLength = 10)
knn_fit
```

```
## k-Nearest Neighbors
##
## 71 samples
## 12 predictors
##  3 classes: 'Kayoa', 'Ternate', 'Tidore'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 64, 64, 63, 64, 64, 64, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8577381  0.7370761
##    7  0.8994048  0.8129549
##    9  0.8625000  0.7460353
```

```
##   11  0.8625000  0.7453439
##   13  0.8535714  0.7271456
##   15  0.8630952  0.7420610
##   17  0.8714286  0.7551841
##   19  0.8563492  0.7199171
##   21  0.8561508  0.7187819
##   23  0.8561508  0.7189022
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

Well, we see that our optimal kNN model is $k = 7$ which gives us an accuray of about 90%. That isn't terrible, but it could probably be better if we messed around a little more. Now let's look at our predictions with the testing data.

```
test <- predict(knn_fit, newdata = hoops_test)
confusionMatrix(test, hoops_test$population)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Kayoa Ternate Tidore
##    Kayoa       0       0      0
##    Ternate     1      17      1
##    Tidore      2       0      8
##
## Overall Statistics
##
##                Accuracy : 0.8621
##                  95% CI : (0.6834, 0.9611)
##     No Information Rate : 0.5862
##     P-Value [Acc > NIR] : 0.00139
##
##                   Kappa : 0.729
##
##  Mcnemar's Test P-Value : 0.26146
##
## Statistics by Class:
##
##                      Class: Kayoa Class: Ternate Class: Tidore
## Sensitivity                0.0000         1.0000        0.8889
## Specificity                1.0000         0.8333        0.9000
## Pos Pred Value                NaN         0.8947        0.8000
## Neg Pred Value             0.8966         1.0000        0.9474
## Prevalence                 0.1034         0.5862        0.3103
## Detection Rate             0.0000         0.5862        0.2759
## Detection Prevalence       0.0000         0.6552        0.3448
## Balanced Accuracy          0.5000         0.9167        0.8944
```

Looking at this data, we can see wthat we are misclassifying almost all of the Kayoa butterflies. This is likely due to the issue with class imbalance where the number of Kayoa butterflies in the testing data is much lower than the number of Tidore and Ternate butterflies.

Conclusions

My guess is that we either need to deal with the class imbalance problem. Maybe z-score normalization would change the results also, but I am skeptical about that. Perhaps another algorithm would be a better classifier as well.

References

- *Machine Learning with R*, 2nd edition, by Brett Lantz. (This book describes how to implement kNN with the `class` package.)
- This website.