Class 9 Activity Sheet V1 – Random Forest - Software Tools – U Guelph

Activity adapted from Sally Adamowicz; last updated September 29, 2024

Learning Outcomes

By the end of completing this activity, you should be able to:

- Explain the difference between the OR "|" and AND "&" operators and apply these correctly when performing data filtering.
- Explain code for DNA sequence filtering and trimming line by line and character by character, including several commonly used regular expression characters
- Describe the value of using named variables and performing calculations from data (as contrasted with hard coding values into our scripts)
- Build a random forest classifier in R using the randomForest package and explain what are the key inputs and outputs
- Adapt the provided example code to fit a new classifier

1. Line 106. Here, we are seeing the logical operator OR "|". Explain what that is doing on line 106. Why can't we use AND "&" there instead? Try it. What happens if we use "&" instead? What do lines 109 and 110 yield and why?

Line 106 looks for records where both the COI-5P or the 28S markers are included. If you use the & instead, lines 109 and 110 yield nothing, because there are no instances where both are recorded.

2. Walk through lines 116-120 carefully. By the end, you should be able to explain what every line and actually every single character does in that block of code.

dfCOI28S <- dfBOLD %>% - create a new data frame called dfSeq where the data originates from the dfCOI28S data frame including only sequences with the COI-5P or 28S markers

mutate(nucleotides2 = str_remove(nucleotides, "^[-N]+")) %>% - create a new column by taking the existing nucleotides column and remove any instances with any number of "N" at the beginning of the sequence, which are ambiguous nucleotides.

mutate(nucleotides2 = str_remove(nucleotides2, "[-N]+$")) %>% - remove any instances with any number of "N" at the end of the sequence in the nucleotides2 column created in the previous step

mutate(nucleotides2 = str_remove_all(nucleotides2, "-+")) %>% - remove any instances of "-" from the sequence

*filter(str_count(nucleotides2, "N") <= (0.05 * str_count(nucleotides)))* - only include the sequences in the new column where the ambiguous nucleotides "N" are less than or equal to 5% of the original sequence length

3. Read through and run lines 126-130. Explain WHY we are using q1 and q3 as new variables here. What is the benefit of this approach?

We are using q1 and q3 as new variables because they are easier to read and interpret. Additionally, by including them as their own variables, you can always reference the length of sequences at the first and third quartiles. This becomes advantageous when you want to do calculations with these numbers; you do not need to re-write the calculation for the quartiles, instead we just reference the variables themselves.

4. Lines 139-144. Explain briefly the output from each line. WHY was each output correct? (I strongly encourage everyone to build in such "checks" into your code regularly to check that previous steps worked as intended, prior to proceeding with your main plotting, statistical analysis, etc.)

dim(dfSeq) - look at the data frame's dimensions (i.e., number of rows and columns). This is correct because the number of variables is the same (81) and since we are using a constrained sequence length, we are filtering rows where sequences are less than the first quartile's length

unique(dfSeq$markercode) - checking to see the unique marker codes. This is correct because we did not change anything about the markers being used (COI-5P and 28S)

table(dfSeq$markercode) - showing the number of records for each marker code. This is correct because the 28S has more constrained sequence length, and the COI-5P had more values recorded from the original data.

sum(is.na(dfSeq$nucleotides2)) - counting the number of n.a values in the nucleotides2 column. This is correct because there are none, since we filtered these out in a previous step.

summary(str_count(dfSeq$nucleotides2[dfSeq$markercode == "COI-5P"])) - looking at the summary statistics for the COI-5P marker. This is correct because the minimum is the q1 value we calculated before and the maximum is the q3 value we calculated.

summary(str_count(dfSeq$nucleotides2[dfSeq$markercode == "28S"])) - looking at the summary statistics for the 28S marker. This is correct because we know the 28S marker was the constraint, meaning that their lengths will be smaller as shown by the statistics.

5. Up to line 178. Many of the middle sections of this script are similar to our previous example script. So, run through the middle, but please ASK if you have any remaining questions about the code.

NOTE: There is no answer provided in the V2 file. Please ASK if you have any questions about this section or about any other code lines not explicitly covered in the activity sheet questions.

6. Line 178. After running this line, have a look at dfSeq in the viewer. Note that you may need to use the arrows in RStudio to scroll through different sets of variables (columns). Scroll through to the end of the columns. Have a look at what you notice over the last few columns. Also, write code to calculate the mean frequency of "TTA" and "TTG". What do you notice about the frequencies of 3-mer "TTA" vs. "TTG"? How do these values compare to the monomer (nucleotide) frequencies?

avg_TTA <- mean(dfSeq$TTA) - 0.0778

avg_TTG <- mean(dfSeq$TTG) - 0.0208

avg_T <- mean(dfSeq$T) - 282.5801

avg_A <- mean(dfSeq$A) - 195.5541

avg_G <- mean(dfSeq$G) - 85.1475

The monomer frequencies are much larger than the 3-mer sequences, as they are less likely to occur together than individually. There are many combinations of sequences as the k-mer length gets longer

7. Line 206-208. Explain WHY we are specifying a separate validation set. What is the point of this?

We are specifying a separate validation set because we want to ensure our model is working properly, using data we have already classified. We know there should be 20 of each marker, so if our model works correctly, then we can validate that the model classifies them as expected. This validation set is removed from model training, so that the model has never seen it before and serves as an appropriate test of the model's accuracy.

8. Line 208. We could have written this line to be more generalizable, to a data set of any size. Re-write this line of code so that we would sample 0.25 (25%) of the data, for a data set of any size. (If you get stuck, then still make notes about how you might go about this at a high level, and then have a look at the example answer.)

You could use the sample_frac() function in dplyr to sample a percentage of the data set rather than hard coding the number of samples to take.

```
dfValidation <- dfCotesia %>%
  group_by(markercode) %>%
  sample_frac(0.25)
```

9. Line 216. We are seeing another new piece of syntax. What is %in% doing? What is it doing from an R syntax point of view? And, what are we doing here from the point of view of our machine learning activity?

The %in% is an operator that checks whether one set of values in a vector is in another set of values in a vector. In this scenario, we are checking to make sure that for our training data set, we do not include any rows/records that already exist in the testing data we created previously.

10. Line 218. In this example, we are sampling 62 rows per marker. Why are we doing this? Why are we not keeping all remaining COI data not in the validation set? What could go wrong if we have severe class imbalance? (Tip: Imagine training a classifier if we keep one thousand times more data of one category than the other category… what would happen?).

We are sampling 62 rows per marker because there are only 82 records total that have the 28S marker. Since we set aside 20 28S and COI-5P sequences, that leaves only 62 28S records. Although there are more than 82 records with the COI-5P marker, we don't want to over-represent this class, because the model will be trained on an unbalanced data set and learn to classify most things as COI-5P to achieve good performance.

11. Interpret line 227. What sequence features are we using as our candidate predictor variables? What are we trying to predict (i.e. what is our response variable)? How many decision trees are we generating? How many features (predictor variables) are sampled for creating each split?

We are using the proportion of A, T and G in each sequence as candidate predictor variables and the response variable is the marker code. We are creating 50 decision trees, and 1 feature (predictor variable) is sampled for each split

12. Next, adapt the code provided in the Script 9 V1 to build a classifier to distinguish the two genera in the wasp data set. Or, come up with your own classification problem to solve! See V2 of the script for example code for part 6.

```
#_Question 12 - Build own classifier to distinguish genera ----

###_ Load data and investigate ----

df_wasps<- read_tsv("../data/wasps.tsv")

names(df_wasps)

summary(df_wasps)

table(df_wasps$markercode)


df_wasps_cleaned <- df_wasps %>%
  filter(!is.na(nucleotides)) %>%
  filter(!is.na(genus_name))


unique(df_wasps_cleaned$genus_name)

sum(is.na(df_wasps_cleaned$nucleotides))


###_ Create a new data frame ----

df_sequences <- df_wasps_cleaned %>%
  mutate(nucleotides2 = str_remove(nucleotides, "^[-N]+")) %>%
  mutate(nucleotides2 = str_remove(nucleotides2, "[-N]+$")) %>%
  mutate(nucleotides2 = str_remove_all(nucleotides2, "-+")) %>%
  filter(str_count(nucleotides2, "N") <= (0.05 * str_count(nucleotides)))


df_compare_seqs <- cbind(df_sequences$nucleotides, df_sequences$nucleotides2)
```

```
q1 <- quantile(nchar(df_sequences$nucleotides2[df_sequences$genus_name == "Cotesia"]), probs = 0.25, na.rm = TRUE)

q1

q3 <- quantile(nchar(df_sequences$nucleotides2[df_sequences$genus_name == "Cotesia"]), probs = 0.75, na.rm = TRUE)

q3


table(df_sequences$genus_name)


df_sequences <- df_sequences %>%

  filter(((str_count(nucleotides2) >= q1 & str_count(nucleotides2) <= q3 & genus_name == "Cotesia") | genus_name == "Campoletis"))

dim(df_sequences)

unique(df_sequences$genus_name)

table(df_sequences$genus_name)

sum(is.na(df_sequences$nucleotides2))

summary(str_count(df_sequences$nucleotides2[df_sequences$genus_name == "Cotesia"]))

summary(str_count(df_sequences$nucleotides2[df_sequences$genus_name == "Campoletis"]))



###_ Create model ----

## Converting tibble to data frame for biostrings to use

df_sequences <- as.data.frame(df_sequences)

df_sequences$nucleotides2 <- DNAStringSet(df_sequences$nucleotides2)

class(df_sequences$nucleotides2)


df_sequences <- cbind(df_sequences, as.data.frame(letterFrequency(df_sequences$nucleotides2, letters = c("A", "C","G", "T"))))


## Calculating proportions of each nucleotide

df_sequences$Aprop <- (df_sequences$A) / (df_sequences$A + df_sequences$T + df_sequences$C + df_sequences$G)
```

```
df_sequences$Tprop <- (df_sequences$T) / (df_sequences$A + df_sequences$T + df_sequences$C +
df_sequences$G)

df_sequences$Gprop <- (df_sequences$G) / (df_sequences$A + df_sequences$T + df_sequences$C +
df_sequences$G)

df_sequences$nucleotides2 <- as.character(df_sequences$nucleotides2)

table(df_sequences$genus_name)

## Create validation data set
set.seed(217)

df_validation <- df_sequences %>%
 group_by(genus_name) %>%
 sample_frac(0.25)

table(df_validation$genus_name)

## Create training data set
set.seed(13)
df_training <- df_sequences %>%
 filter(!processid %in% df_validation$processid) %>%
 group_by(genus_name) %>%
 sample_n(1024)

table(df_training$genus_name)

names(df_training)
```

```
## Create final model

genus_classifier <- randomForest::randomForest(x = df_training[, 86:88], y =
base::as.factor(df_training$genus_name), ntree = 50, importance = TRUE)


genus_classifier


genus_classifier$importance


## Validate

predict_validation <- predict(genus_classifier, df_validation[, c(70, 86:88)])

predict_validation


## Confusion matrix

table(observed = df_validation$genus_name, predicted = predict_validation)
```

We hope that you enjoyed this activity. You should now understand that k-mer profiles can be helpful features that can be calculated quickly, and we can use them as candidates for sequence classification. We encourage you to go deeper to explore the random forest settings and also try out other machine learning approaches, such as available through the R package caret.