Random forest

Zexuan Zhao

2022-12-05

Replicate original random forest analysis in the paper

The followings are codes modified from `scripts_from_original_paper/7_Random_forest/random_forest_NEW_3.r`.

Load CSV data and exclude redundant variables

```
data <- read_csv("AppendixS1.csv") %>%
# Exclude low-level taxonomic variables
  select(-Species, -Genus, -Family, -Order) %>%
# Exclude redundant statistical variables
  select(-pmtr, -dgr, -dgp, -der, -dep, -ger, -gep) %>%
# Exclude variables of genetic sequence
  select(-bp, -pi) %>%
# Exclude variables generated by downstream analysis
  select(-r, -c, -t, -p,    -f, -fp,    -cgeo,  -tgeo,  -pgeo,  -cenv,  -tenv,  -penv) %>%
# Exclude rows with NAs
  na.omit()
```

```
## Rows: 19197 Columns: 67
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (10): Species, Gene, Genus, Family, Order, Class, Phylum, Kingdom, Metab...
## dbl (57): n, pmtr, pmtp, dgr, dgp, der, dep, ger, gep, bp, pi, Postfloodingo...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(data)
```

```
## [1] 12252    42
```

In total there are 41 predictors, 1 response variable and 12252 data points.

Convert p-value to labels

pmtp is generated by `mantel.partial(mydata.dnadist, mydata.geodist, mydata.envdist, method = "pearson", permutations = 999)` from `scripts_from_original_paper/8_IBE/IBE.r`.

```r
data_labeled <- data %>%
# Convert p-value to reject/accpet null hypothesis by a threshold of 0.05
  mutate(pmtp = ifelse(pmtp <= 0.05, "yes", "no") %>% as.factor())
```

The data set is highly unbalanced because the positive cases are only 14.9118511%.

```r
# Calculate percentage of yes/no
data_labeled %>%
  group_by(pmtp) %>%
  summarize(n = n(), p = n/12252)
```

```
## # A tibble: 2 x 3
##   pmtp      n     p
##   <fct> <int> <dbl>
## 1 no    10425 0.851
## 2 yes    1827 0.149
```

Run random forest and imporance analysis

```r
rf_unfiltered_unbalanced <- randomForest(pmtp ~ .,
                   data=data_labeled,
                   ntree=1000, importance=TRUE, nPerm=100)
rf_unfiltered_unbalanced
#saveRDS(rf_unfiltered_unbalanced, "rf_unfiltered_unbalanced.RDS")
```

```
##
## Call:
##  randomForest(formula = pmtp ~ ., data = data_labeled, ntree = 1000,      importance = TRUE, nPerm =
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 14.5%
## Confusion matrix:
##        no yes class.error
## no  10177 248  0.02378897
## yes  1528 299  0.83634373
```

If no filtering and balancing is applied, the overall accuracy is relatively high (0.1467067) but highly biased (false negative rate = 0.8363437). See important variables:

```r
importance(rf_unfiltered_unbalanced) %>%
  as_tibble(rownames = "var") %>%
# Sort by MDA descendantly
  arrange(desc(MeanDecreaseAccuracy)) %>%
  select(var, MeanDecreaseAccuracy)
```

```
## # A tibble: 41 x 2
##    var         MeanDecreaseAccuracy
##    <chr>                      <dbl>
##  1 n                          100.
##  2 abs_mid_lat                 51.9
##  3 area                        50.2
##  4 abs_min_lat                 48.3
##  5 max_lon                     47.0
##  6 length                      46.7
##  7 min_lat                     45.9
##  8 min_lon                     45.8
##  9 abs_max_lat                 45.2
## 10 max_lat                     43.3
## # ... with 31 more rows
```

Re-run random forest using balanced data and set `n >= 20`

Apparently `n` is an important predictor, but it is an artifect that has nothing to do with the scientific question I'm addressing. `n` is the number of individuals whose genetic sequences were used to infer genetic structure. `n` too low means the inference of genetic structure is not reliable. In the paper, they claimed that `n>=20` is a good threshold.

```
data_labeled_20 <- data_labeled %>%
  filter(n >= 20)
dim(data_labeled_20)
```

```
## [1] 2363   42
```

After applying `n >= 20`, there are only 2363 (0.1928665%) data points left. See how filtering by `n >=20` changes accuracy:

```
rf_filtered_unbalanced <- randomForest(pmtp ~ .,
                                       data=data_labeled_20,
                                       ntree=1000, importance=TRUE, nPerm=100)
rf_filtered_unbalanced
saveRDS(rf_filtered_unbalanced, "rf_filtered_unbalanced.RDS")
```

```
##
## Call:
##  randomForest(formula = pmtp ~ ., data = data_labeled_20, ntree = 1000,      importance = TRUE, nPerr
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 28.78%
## Confusion matrix:
##       no yes class.error
## no  1463 173   0.1057457
## yes  507 220   0.6973865
```

As shown above, the model is still predicting badly on positive cases. What's more, it's predicting worse on true negative cases and the overall error rate is even higher. It could be because of the low number of positive cases in the training data set and applying filter makes both positive cases and negative cases rarer, so the model does not have much data to train. We can (partially) solve this problem by resampling from the original data set and sample equal numbers of positive and negative cases.

```r
# Sample from original data set to get balanced data set.
# By default the resampled data set will be 2 times the size of the original one.
# Return the resampled dataframe
generate_balanced_data <- function(data, N = 2*nrow(data)){
  data_pos <- data %>%
    filter(pmtp == "yes")
  data_neg <- data %>%
    filter(pmtp == "no")
  data_pos_sample <- data_pos %>%
    slice_sample(n = round(N/2), replace = TRUE)
  data_neg_sample <- data_neg %>%
    slice_sample(n = N - round(N/2), replace = TRUE)
  data_pos_sample %>%
    bind_rows(data_neg_sample)
}
```

See how balanced training data set can affect accuracy:

```r
rf_unfiltered_balanced <- randomForest(pmtp ~ .,
                                       data=generate_balanced_data(data_labeled),
                                       ntree=1000, importance=TRUE, nPerm=100)
saveRDS(rf_unfiltered_balanced, "rf_unfiltered_balanced.RDS")
```

```
##
## Call:
##  randomForest(formula = pmtp ~ ., data = generate_balanced_data(data_labeled),      ntree = 1000, im
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 1.46%
## Confusion matrix:
##        no   yes class.error
## no  11918   334 0.027260855
## yes    24 12228 0.001958864
```

As shown above, the accuracy is elevated so much and the bias is also eliminated!

Putting it together, I made a pipeline function that does balancing sampling, random forest building and importance analysis:

```r
# Run random forest and importance analysis once on resampled balanced data.
# Return a dataframe merging the error rate from rf and MeanDecreaseAccuracy from imp
random_forest_importance_once <- function(i, data){
  rf <- randomForest(pmtp ~ .,
                     data = generate_balanced_data(data),
                     ntree=1000, importance=TRUE, nPerm=100)
  error_rate <- rf$confusion[,3] %>% as_tibble_row() %>%
    mutate(index = i) %>%
    gather(key = "type", value = "value", -index)
  imp <- importance(rf2) %>%
    as_tibble(rownames = "var") %>%
    arrange(desc(MeanDecreaseAccuracy)) %>%
    select(var, MeanDecreaseAccuracy) %>%
    mutate(index = i) %>%
    select(index, type = var, value = MeanDecreaseAccuracy)
  error_rate %>%
    bind_rows(imp)
}
```

Here I did resampling 100 times, pooled the results and only showed the mean of all metrics.

```
# Use parallel computing in R
random_forests <- mclapply(1:100,
                           random_forest_importance_once,
                           data = data_labeled_20,
                           mc.cores = 6)
random_forests_tbl <- do.call(bind_rows, random_forests)
#saveRDS(random_forests_tbl, "random_forests_tbl.RDS")
```

Summarizing the error rates from 100 resamplings

```
imp_filtered_balanced <- random_forests_tbl %>%
  filter(type == "yes" | type == "no") %>%
  group_by(type) %>%
  summarize(mean_error = mean(value))
imp_filtered_balanced
```

```
## # A tibble: 2 x 2
##   type  mean_error
##   <chr>      <dbl>
## 1 no        0.0563
## 2 yes       0.0261
```

Summarizing the mean decrease accuracy of predictors from 100 resamplings

```
MDA <- random_forests_tbl %>%
  filter(type != "yes" & type != "no") %>%
  group_by(type) %>%
  summarize(meanDecreaseAccuracy = mean(value)) %>%
  arrange(desc(meanDecreaseAccuracy))
saveRDS(MDA, "MDA.RDS")
```

Comparing accuracies based on four data set:

```
summary_accuracy <-
rf_unfiltered_unbalanced$confusion[,3] %>%
  as_tibble_row() %>%
  gather(key = true, value = unfiltered_unbalanced) %>%
  left_join(rf_unfiltered_balanced$confusion[,3] %>%
              as_tibble_row() %>%
              gather(key = true, value = unfiltered_balanced)
            ) %>%
  left_join(rf_filtered_unbalanced$confusion[,3] %>%
              as_tibble_row() %>%
              gather(key = true, value = filtered_unbalanced)
            ) %>%
  left_join(imp_filtered_balanced %>%
              select(true = type, filtered_balanced = mean_error)
            )
```

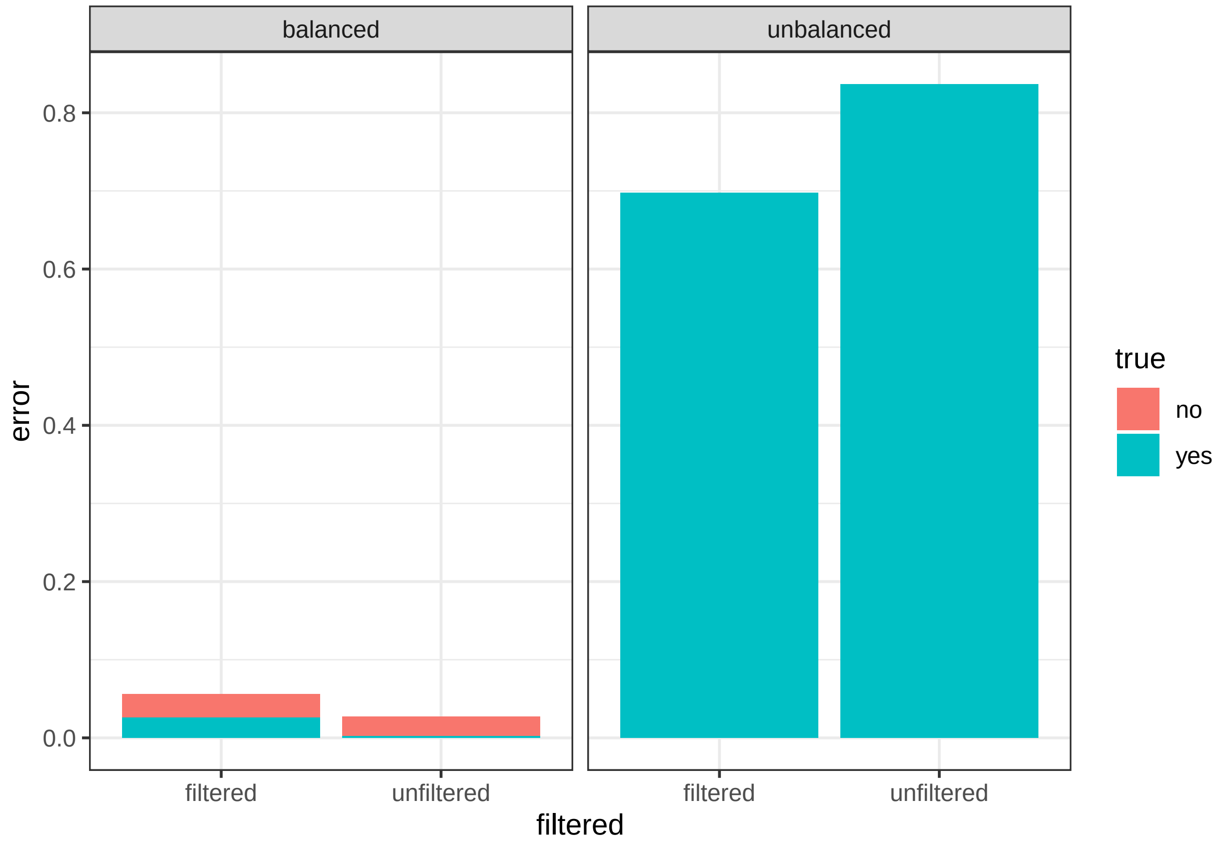
```
## Joining, by = "true"
## Joining, by = "true"
## Joining, by = "true"
```

```
saveRDS(summary_accuracy, "summary_accuracy_randomForest.RDS")
summary_accuracy
```

```
## # A tibble: 2 x 5
##   true  unfiltered_unbalanced unfiltered_balanced filtered_unbalanced filtered~1
##   <chr>                 <dbl>               <dbl>               <dbl>      <dbl>
## 1 no                   0.0238             0.0273                0.106     0.0563
## 2 yes                  0.836              0.00196               0.697     0.0261
## # ... with abbreviated variable name 1: filtered_balanced
```

Plotting the error rate with combination of filtered and balanced

```
summary_accuracy_plot <- summary_accuracy %>%
  gather(key = type, value = error, -true) %>%
  separate(type, into = c("filtered", "balanced"), sep = "_") %>%
  mutate(grouper = interaction(filtered, balanced)) %>%
  group_by(grouper) %>%
  ggplot(aes(x = filtered, y = error, group = grouper, fill = true)) +
    geom_bar(stat = "identity",
             position = "dodge") +
  facet_grid(. ~ balanced) +
  theme_bw()
summary_accuracy_plot
```



```
ggsave("../report/img/summary_accuracy_plot.png",
       plot = summary_accuracy_plot,
       width = 4, height = 3, units = "in")
```