

MSD 2019 Final Project

An extension (out of sample testing) of Greed and Grievance in Civil War by Paul Collier and Anke Hoeffler, 2000

Kiran Ramesh (kr2789), Sai Srujan Chinta (sc4401), Bhavya Shahi (bs3118)

2019-05-13 05:59:43

Contents

Reading Data	1
Helper Functions	1
k-fold Cross Validation	2
Opportunity Models	2
Grievance Models	5
Combined Model	8
Computing Averages of the k-fold Validation	11

Reading Data

```
setwd(".")
options(scipen = 100, digits = 4)

data <- read.dta("data/G&G.dta")
data <- data[!is.na(data$warsa), ]
```

Helper Functions

```
summarize_into_table <- function(summary_obj) {
  # takes summary object as input, and returns a DF
  # res. to print res, use print(summ(obj), quote =
  # FALSE) is you don't want quotes

  options(digits = 4)
  res <- t(round(summary_obj$coefficients, digits = 4))
  z.values <- res[4, ]
  Signif <- symnum(z.values, corr = FALSE, na = FALSE,
    cutpoints = c(0, 0.01, 0.05, 0.1, 1), symbols = c("***",
      "**", "*", ""))
  res <- rbind(res, Signif)
  res <- t(res)
  res <- res[-1, ]
}
```

```

    res
  }

comma_sep = function(x) {
  x = strsplit(x, "")
}

convert2dArrayToDf = function(all_tests) {
  model_names <- all_tests[, 1]

  invisible(apply(all_tests, 2, as.numeric))
  invisible(sapply(all_tests, as.numeric))
  class(all_tests) <- "numeric"
  storage.mode(all_tests) <- "numeric"

  all_tests <- as.data.frame(all_tests)

  all_tests[, 1] <- model_names

  return(all_tests)
}

```

k-fold Cross Validation

```

k <- 5

all_tests <- matrix(data = 0, nrow = 15 * k, ncol = 6)
colnames(all_tests) <- c("model", "test_index", "sens",
  "spec", "auc", "accuracy")

set.seed(42)
shuffled_data <- data[sample(nrow(data)), ]
folds <- cut(seq(1, nrow(shuffled_data)), breaks = k,
  labels = FALSE)

thresholding_flag = TRUE
threshold_value = as.numeric(params$threshold)

sens_index <- 3
spec_index <- 4
auc_index <- 5
accuracy_index <- 6

```

Opportunity Models

Generating the various opportunity models

```
# Opportunity Models
```

```
filtering_columns_list <- list("warsa,sxp,sxp2,coldwar,secm,gy1,peace,prevwara,mount,geogia,frac,lnpop"
```

```

"warsa,sxp,sxp2,coldwar,secm,gy1,peace,mount,geogia,frac,lnpop",
"warsa,sxp,sxp2,coldwar,lngdp_,gy1,peace,mount,geogia,frac,lnpop",
"warsa,sxp,sxp2,lngdp_,peace,lnpop,diaspeaa", "warsa,sxp,sxp2,lngdp_,peace,lnpop,difdpeaa,diahpeaa".

regression_formula_list <- list("warsa ~  sxp + sxp2 + coldwar + secm + gy1 + peace + prevwara + mount +
  "warsa ~  sxp + sxp2 + coldwar + secm + gy1 + peace + mount + geogia + frac + lnpop",
  "warsa ~  sxp + sxp2 + coldwar + lngdp_ + gy1 + peace + mount + geogia + frac + lnpop",
  "warsa ~  sxp + sxp2 + lngdp_ + peace + lnpop + diaspeaa",
  "warsa ~  sxp + sxp2 + lngdp_ + peace + lnpop + difdpeaa + diahpeaa")

for (i in c(1:5)) {

  for (testIndex in 1:k) {

    filtering_columns <- strsplit(filtering_columns_list[[i]],
      ",")[[1]]

    opportunity.data <- shuffled_data[, filtering_columns]

    # Segement your data by fold using the which()
    # function
    testIndexes <- which(folds == testIndex, arr.ind = TRUE)
    testData <- opportunity.data[testIndexes, ]
    trainData <- opportunity.data[-testIndexes,
      ]

    # trainData <- na.omit(trainData) testData <-
    # na.omit(testData)

    opportunity_fit <- glm(as.formula(regression_formula_list[[i]]),
      family = binomial(link = "logit"), data = trainData)

    opportunity_predict <- predict(opportunity_fit,
      newdata = testData, type = "response")

    opportunity_y.hat <- as.matrix(opportunity_predict)

    y <- as.matrix(testData$warsa)

    all_tests[(i - 1) * k + testIndex, 1] <- paste(c("opportunity",
      i), collapse = ".")
    all_tests[(i - 1) * k + testIndex, 2] <- as.numeric(testIndex)

    if (thresholding_flag == TRUE) {

      opportunity_y.hat_normalized <- opportunity_y.hat

      opportunity_y.hat_normalized[opportunity_y.hat_normalized >=
        threshold_value] <- 1
      opportunity_y.hat_normalized[opportunity_y.hat_normalized <
        threshold_value] <- 0
    }
  }
}

```

```

    opp_predict_normalized <- prediction(opportunity_y.hat_normalized,
                                         y)

    len <- length(opp_predict_normalized@fp[[1]])
    fp <- as.numeric(opp_predict_normalized@fp[[1]][len -
1])
    tp <- as.numeric(opp_predict_normalized@tp[[1]][len -
1])
    fn <- as.numeric(opp_predict_normalized@fn[[1]][len -
1])
    tn <- as.numeric(opp_predict_normalized@tn[[1]][len -
1])

    all_tests[(i - 1) * k + testIndex, sens_index] <- tp/(tp +
fn)
    all_tests[(i - 1) * k + testIndex, spec_index] <- tn/(tn +
fp)

    all_tests[(i - 1) * k + testIndex, accuracy_index] <- (tp +
tn)/(tp + tn + fp + fn)

    opp_predict <- prediction(opportunity_y.hat,
                              y)
    opp_auc <- performance(opp_predict, measure = "auc")
    all_tests[(i - 1) * k + testIndex, auc_index] <- as.numeric(unlist(slot(opp_auc,
"y.values"))))
  } else {

    opp_predict <- prediction(opportunity_y.hat,
                              y)

    opp_f <- performance(opp_predict, measure = "f")
    opp_where.F <- which.max(as.numeric(unlist(slot(opp_f,
"y.values"))))
    opp_what.F <- performance(opp_predict,
                              measure = "sens", x.measure = "spec")

    all_tests[(i - 1) * k + testIndex, sens_index] <- as.numeric(unlist(slot(opp_what.F,
"y.values")))[opp_where.F]
    all_tests[(i - 1) * k + testIndex, spec_index] <- as.numeric(unlist(slot(opp_what.F,
"x.values")))[opp_where.F]

    opp_auc <- performance(opp_predict, measure = "auc")
    all_tests[(i - 1) * k + testIndex, auc_index] <- as.numeric(unlist(slot(opp_auc,
"y.values"))))
  }
}
}

```

```

lower_lim = 1
upper_lim = 5 * k
print(convert2dArrayToDf(all_tests[lower_lim:upper_lim,
1:6]))

```

##	model	test_index	sens	spec	auc	accuracy
## 1	opportunity.1	1	0.00000	0.9922	0.7642	0.9078
## 2	opportunity.1	2	0.00000	1.0000	0.9179	0.9640
## 3	opportunity.1	3	0.00000	0.9758	0.8767	0.9237
## 4	opportunity.1	4	0.00000	0.9924	0.7816	0.9097
## 5	opportunity.1	5	0.20000	1.0000	0.8829	0.9398
## 6	opportunity.2	1	0.00000	0.9922	0.7745	0.9078
## 7	opportunity.2	2	0.00000	1.0000	0.9134	0.9640
## 8	opportunity.2	3	0.00000	0.9758	0.8664	0.9237
## 9	opportunity.2	4	0.00000	0.9924	0.7822	0.9097
## 10	opportunity.2	5	0.20000	1.0000	0.8854	0.9398
## 11	opportunity.3	1	0.07692	0.9926	0.7483	0.9128
## 12	opportunity.3	2	0.00000	1.0000	0.8951	0.9662
## 13	opportunity.3	3	0.00000	0.9860	0.7920	0.9338
## 14	opportunity.3	4	0.00000	0.9790	0.7902	0.8917
## 15	opportunity.3	5	0.16667	1.0000	0.9029	0.9310
## 16	opportunity.4	1	0.40000	1.0000	0.8931	0.9752
## 17	opportunity.4	2	0.00000	1.0000	0.7446	0.9573
## 18	opportunity.4	3	0.00000	0.9725	0.9358	0.9464
## 19	opportunity.4	4	0.00000	0.9911	0.8036	0.9024
## 20	opportunity.4	5	0.00000	1.0000	0.8410	0.9344
## 21	opportunity.5	1	0.40000	1.0000	0.8914	0.9752
## 22	opportunity.5	2	0.00000	1.0000	0.7464	0.9573
## 23	opportunity.5	3	0.00000	0.9725	0.9358	0.9464
## 24	opportunity.5	4	0.09091	0.9821	0.7979	0.9024
## 25	opportunity.5	5	0.00000	1.0000	0.8410	0.9344

Grievance Models

Generating the various grievance models

```

# Grievance Models

filtering_columns_list <- list("warsa,elfo,rf,pol16,etdo4590,dem,peace,mount,geogia,lnpop",
  "warsa,elfo,rf,pol16,etdo4590,dem,peace,mount,geogia,lnpop,ygini",
  "warsa,elfo,rf,pol16,etdo4590,dem,peace,mount,geogia,lnpop,lgini")

regression_formula_list <- list("warsa ~ elfo + rf + pol16 + etdo4590 + dem + peace + mount + geogia +
  "warsa ~ elfo + rf + pol16 + etdo4590 + dem + peace + mount + geogia + lnpop + ygini",
  "warsa ~ elfo + rf + pol16 + etdo4590 + dem + peace + mount + geogia + lnpop + lgini")

for (i in c(1:3)) {

  for (testIndex in 1:k) {

    filtering_columns <- strsplit(filtering_columns_list[[i]],
      ",")[[1]]

```

```

grievance.data <- shuffled_data[, filtering_columns]

# Segment your data by fold using the which()
# function
testIndexes <- which(folds == testIndex, arr.ind = TRUE)
testData <- grievance.data[testIndexes, ]
trainData <- grievance.data[-testIndexes, ]

# trainData <- na.omit(trainData) testData <-
# na.omit(testData)

grievance_fit <- glm(as.formula(regression_formula_list[[i]]),
  family = binomial(link = "logit"), data = trainData)

grievance_predict <- predict(grievance_fit,
  newdata = testData, type = "response")

grievance_y.hat <- as.matrix(grievance_predict)

y <- as.matrix(testData$warsa)

all_tests[5 * k + (i - 1) * k + testIndex,
  1] <- paste(c("grievance", i), collapse = ".")
all_tests[5 * k + (i - 1) * k + testIndex,
  2] <- as.numeric(testIndex)

if (thresholding_flag == TRUE) {
  grievance_y.hat_normalized <- grievance_y.hat

  grievance_y.hat_normalized[grievance_y.hat_normalized >=
    threshold_value] <- 1
  grievance_y.hat_normalized[grievance_y.hat_normalized <
    threshold_value] <- 0

  griev_predict_normalized <- prediction(grievance_y.hat_normalized,
    y)

  len <- length(griev_predict_normalized@fp[[1]])
  fp <- as.numeric(griev_predict_normalized@fp[[1]][[len -
    1]])
  tp <- as.numeric(griev_predict_normalized@tp[[1]][[len -
    1]])
  fn <- as.numeric(griev_predict_normalized@fn[[1]][[len -
    1]])
  tn <- as.numeric(griev_predict_normalized@tn[[1]][[len -
    1]])

  all_tests[5 * k + (i - 1) * k + testIndex,
    sens_index] <- tp/(tp + fn)
  all_tests[5 * k + (i - 1) * k + testIndex,
    spec_index] <- tn/(tn + fp)

  all_tests[5 * k + (i - 1) * k + testIndex,

```

```

        accuracy_index] <- (tp + tn)/(tp +
        tn + fp + fn)

    griev_predict <- prediction(grievance_y.hat,
    y)
    griev_auc <- performance(griev_predict,
    measure = "auc")
    all_tests[5 * k + (i - 1) * k + testIndex,
    auc_index] <- as.numeric(unlist(slot(griev_auc,
    "y.values"))))

  } else {

    griev_predict <- prediction(grievance_y.hat,
    y)

    griev_f <- performance(griev_predict, measure = "f")
    griev_where.F <- which.max(as.numeric(unlist(slot(griev_f,
    "y.values"))))
    griev_what.F <- performance(griev_predict,
    measure = "sens", x.measure = "spec")

    all_tests[5 * k + (i - 1) * k + testIndex,
    sens_index] <- as.numeric(unlist(slot(griev_what.F,
    "y.values")))[griev_where.F]
    all_tests[5 * k + (i - 1) * k + testIndex,
    spec_index] <- as.numeric(unlist(slot(griev_what.F,
    "x.values")))[griev_where.F]

    griev_auc <- performance(griev_predict,
    measure = "auc")
    all_tests[5 * k + (i - 1) * k + testIndex,
    auc_index] <- as.numeric(unlist(slot(griev_auc,
    "y.values"))))

  }

}

lower_lim = 5 * k + 1
upper_lim = 5 * k + 3 * k
print(convert2dArrayToDf(all_tests[lower_lim:upper_lim,
1:6]))

```

```

##      model test_index sens   spec   auc accuracy
## 1 grievance.1         1    0 0.9935 0.6625   0.9102
## 2 grievance.1         2    0 1.0000 0.8019   0.9636
## 3 grievance.1         3    0 1.0000 0.7737   0.9474
## 4 grievance.1         4    0 1.0000 0.7286   0.9064
## 5 grievance.1         5    0 1.0000 0.7712   0.9205
## 6 grievance.2         1    0 1.0000 0.5584   0.9402
## 7 grievance.2         2    0 1.0000 0.7723   0.9655

```

## 8	grievance.2	3	0	1.0000	0.6473	0.9333
## 9	grievance.2	4	0	1.0000	0.6810	0.9062
## 10	grievance.2	5	0	1.0000	0.7159	0.9187
## 11	grievance.3	1	0	0.9910	0.6381	0.9402
## 12	grievance.3	2	0	1.0000	0.8103	0.9508
## 13	grievance.3	3	0	1.0000	0.8609	0.9421
## 14	grievance.3	4	0	1.0000	0.7468	0.9174
## 15	grievance.3	5	0	1.0000	0.7237	0.9262

Combined Model

Generating the combined opportunity and grievance models

```
# Combined Models
```

```
filtering_columns_list <- list("warsa,sxp,sxp2,coldwar,secm,gy1,peace,mount,georgia,lnpop,frac,grievxb",
  "warsa,peace,mount,georgia,lnpop,elfo,rf,pol16,etdo4590,dem,greedxb",
  "warsa,sxp,sxp2,coldwar,secm,gy1,peace,mount,georgia,lnpop,frac,elfo,rf,pol16,etdo4590,dem,ygini",
  "warsa,sxp,sxp2,coldwar,secm,gy1,peace,mount,georgia,lnpop,frac,elfo,rf,pol16,etdo4590,dem",
  "warsa,sxp,sxp2,secm,gy1,peace,georgia,lnpop,frac,etdo4590",
  "warsa,sxp,sxp2,lngdp_,gy1,peace,georgia,lnpop,frac,etdo4590",
  "warsa,sxp,sxp2,secm,gy1,peace,georgia,lnpop,frac,etdo4590,oilsxp,oilsxp2")
```

```
regression_formula_list <- list("warsa ~ sxp + sxp2 + coldwar + secm + gy1 + peace + mount + georgia + lnpop + frac + elfo + rf + pol16 + etdo4590 + dem + greedxb",
  "warsa ~ peace + mount + georgia + lnpop + elfo + rf + pol16 + etdo4590 + dem + greedxb",
  "warsa ~ sxp + sxp2 + coldwar + secm + gy1 + peace + mount + georgia + lnpop + frac + elfo + rf + pol16 + etdo4590 + dem + greedxb",
  "warsa ~ sxp + sxp2 + coldwar + secm + gy1 + peace + mount + georgia + lnpop + frac + elfo + rf + pol16 + etdo4590 + dem + greedxb",
  "warsa ~ sxp + sxp2 + secm + gy1 + peace + georgia + lnpop + frac + etdo4590",
  "warsa ~ sxp + sxp2 + lngdp_ + gy1 + peace + georgia + lnpop + frac + etdo4590",
  "warsa ~ sxp + sxp2 + secm + gy1 + peace + georgia + lnpop + frac + etdo4590 + oilsxp + oilsxp2")
```

```
for (i in c(1:7)) {
```

```
  for (testIndex in 1:k) {
```

```
    filtering_columns <- strsplit(filtering_columns_list[[i]],
      ",")[[1]]
```

```
    combined.data <- shuffled_data[, filtering_columns]
```

```
    # Segment your data by fold using the which()
```

```
    # function
```

```
    testIndexes <- which(folds == testIndex, arr.ind = TRUE)
```

```
    testData <- combined.data[testIndexes, ]
```

```
    trainData <- combined.data[-testIndexes, ]
```

```
    # trainData <- na.omit(trainData) testData <-
```

```
    # na.omit(testData)
```

```
    combined_fit <- glm(as.formula(regression_formula_list[[i]]),
      family = binomial(link = "logit"), data = trainData)
```



```

combined_predict <- predict(combined_fit, newdata = testData,
                             type = "response")

combined_y.hat <- as.matrix(combined_predict)

y <- as.matrix(testData$warsa)

all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
           1] <- paste(c("combined", i), collapse = ".")
all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
           2] <- as.numeric(testIndex)

if (thresholding_flag == TRUE) {
  combined_y.hat_normalized <- combined_y.hat

  combined_y.hat_normalized[combined_y.hat_normalized >=
                             threshold_value] <- 1
  combined_y.hat_normalized[combined_y.hat_normalized <
                             threshold_value] <- 0

  comb_predict_normalized <- prediction(combined_y.hat_normalized,
                                         y)

  len <- length(comb_predict_normalized@fp[[1]])
  fp <- as.numeric(comb_predict_normalized@fp[[1]][[len -
    1]])
  tp <- as.numeric(comb_predict_normalized@tp[[1]][[len -
    1]])
  fn <- as.numeric(comb_predict_normalized@fn[[1]][[len -
    1]])
  tn <- as.numeric(comb_predict_normalized@tn[[1]][[len -
    1]])

  all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
             sens_index] <- tp/(tp + fn)
  all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
             spec_index] <- tn/(tn + fp)

  all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
             accuracy_index] <- (tp + tn)/(tp +
    tn + fp + fn)

  comb_predict <- prediction(combined_y.hat,
                              y)
  comb_auc <- performance(comb_predict, measure = "auc")
  all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
             auc_index] <- as.numeric(unlist(slot(comb_auc,
    "y.values"))))
} else {

  comb_predict <- prediction(combined_y.hat,
                              y)

```

```

comb_f <- performance(comb_predict, measure = "f")
comb_where.F <- which.max(as.numeric(unlist(slot(comb_f,
  "y.values"))))
comb_what.F <- performance(comb_predict,
  measure = "sens", x.measure = "spec")

all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
  sens_index] <- as.numeric(unlist(slot(comb_what.F,
  "y.values")))[comb_where.F]
all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
  spec_index] <- as.numeric(unlist(slot(comb_what.F,
  "x.values")))[comb_where.F]

comb_auc <- performance(comb_predict, measure = "auc")
all_tests[(5 + 3) * k + (i - 1) * k + testIndex,
  auc_index] <- as.numeric(unlist(slot(comb_auc,
  "y.values"))))

}

}

}

lower_lim = 5 * k + 3 * k + 1
upper_lim = 5 * k + 3 * k + 7 * k
print(convert2dArrayToDf(all_tests[lower_lim:upper_lim,
  1:6]))

```

##	model	test_index	sens	spec	auc	accuracy
## 1	combined.1	1	0.00000	0.9758	0.7628	0.8897
## 2	combined.1	2	0.00000	1.0000	0.9190	0.9618
## 3	combined.1	3	0.00000	0.9756	0.8897	0.9231
## 4	combined.1	4	0.00000	0.9840	0.7813	0.8978
## 5	combined.1	5	0.10000	1.0000	0.8769	0.9313
## 6	combined.2	1	0.00000	0.9839	0.7601	0.8971
## 7	combined.2	2	0.00000	1.0000	0.9492	0.9618
## 8	combined.2	3	0.00000	0.9756	0.8455	0.9231
## 9	combined.2	4	0.00000	0.9920	0.7927	0.9051
## 10	combined.2	5	0.10000	1.0000	0.8835	0.9313
## 11	combined.3	1	0.00000	0.9886	0.5966	0.9255
## 12	combined.3	2	0.00000	1.0000	0.9396	0.9579
## 13	combined.3	3	0.00000	0.9765	0.4765	0.9121
## 14	combined.3	4	0.11111	1.0000	0.7589	0.9223
## 15	combined.3	5	0.14286	1.0000	0.8283	0.9375
## 16	combined.4	1	0.00000	0.9758	0.7124	0.8897
## 17	combined.4	2	0.00000	1.0000	0.9333	0.9618
## 18	combined.4	3	0.00000	0.9756	0.8304	0.9231
## 19	combined.4	4	0.00000	0.9760	0.7833	0.8905
## 20	combined.4	5	0.10000	1.0000	0.8810	0.9313
## 21	combined.5	1	0.00000	0.9845	0.7455	0.9007
## 22	combined.5	2	0.00000	1.0000	0.9463	0.9640
## 23	combined.5	3	0.00000	0.9758	0.8744	0.9237
## 24	combined.5	4	0.00000	1.0000	0.7967	0.9167

```
## 25 combined.5      5 0.10000 1.0000 0.8951 0.9323
## 26 combined.6      1 0.07692 0.9926 0.7477 0.9128
## 27 combined.6      2 0.00000 1.0000 0.8951 0.9662
## 28 combined.6      3 0.00000 0.9790 0.7727 0.9272
## 29 combined.6      4 0.00000 0.9930 0.7947 0.9045
## 30 combined.6      5 0.00000 1.0000 0.9079 0.9172
## 31 combined.7      1 0.08333 0.9758 0.7796 0.8971
## 32 combined.7      2 0.00000 1.0000 0.9297 0.9624
## 33 combined.7      3 0.00000 0.9748 0.8860 0.9206
## 34 combined.7      4 0.00000 0.9919 0.8574 0.9104
## 35 combined.7      5 0.10000 0.9913 0.9435 0.9200
```

Computing Averages of the k-fold Validation

```
all_tests <- convert2dArrayToDf(all_tests)

result <- aggregate(all_tests[, 3:6], list(all_tests$model),
  mean)

names(result)[1] <- "model"

print(result)
```

```
##      model      sens      spec      auc accuracy
## 1 combined.1 0.02000 0.9871 0.8459 0.9207
## 2 combined.2 0.02000 0.9903 0.8462 0.9237
## 3 combined.3 0.05079 0.9930 0.7199 0.9311
## 4 combined.4 0.02000 0.9855 0.8281 0.9193
## 5 combined.5 0.02000 0.9921 0.8516 0.9275
## 6 combined.6 0.01538 0.9929 0.8236 0.9256
## 7 combined.7 0.03667 0.9868 0.8792 0.9221
## 8 grievance.1 0.00000 0.9987 0.7476 0.9296
## 9 grievance.2 0.00000 1.0000 0.6750 0.9328
## 10 grievance.3 0.00000 0.9982 0.7560 0.9353
## 11 opportunity.1 0.04000 0.9921 0.8447 0.9290
## 12 opportunity.2 0.04000 0.9921 0.8444 0.9290
## 13 opportunity.3 0.04872 0.9915 0.8257 0.9271
## 14 opportunity.4 0.08000 0.9927 0.8436 0.9432
## 15 opportunity.5 0.09818 0.9909 0.8425 0.9432
```

```
write.csv(result, file = paste0("Project_Extension_2_Threshold_",
  params$threshold, ".csv"))
```

The following is a list of all packages used to generate these results. (Leave at very end of file.)

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-apple-darwin17.7.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS/LAPACK: /usr/local/Cellar/openblas/0.3.5/lib/libopenblas-r0.3.5.dylib
##
```

```

## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] ROCR_1.0-7      gplots_3.0.1.1  lme4_1.1-21
## [4] Matrix_1.2-15  DescTools_0.99.28 foreign_0.8-71
## [7] forcats_0.3.0  stringr_1.4.0   dplyr_0.8.0
## [10] purrr_0.3.0    readr_1.3.1     tidyr_0.8.2
## [13] tibble_2.0.1   ggplot2_3.1.0   tidyverse_1.2.1
## [16] scales_1.0.0   here_0.1
##
## loaded via a namespace (and not attached):
## [1] Rcpp_1.0.0      lubridate_1.7.4  mvtnorm_1.0-10
## [4] lattice_0.20-38 gtools_3.8.1     assertthat_0.2.0
## [7] rprojroot_1.3-2 digest_0.6.18    R6_2.4.0
## [10] cellranger_1.1.0 plyr_1.8.4       backports_1.1.3
## [13] evaluate_0.13   httr_1.4.0       pillar_1.3.1
## [16] rlang_0.3.1     lazyeval_0.2.1   readxl_1.3.0
## [19] rstudioapi_0.9.0 minqa_1.2.4      gdata_2.18.0
## [22] nloptr_1.2.1    rmarkdown_1.11   splines_3.5.2
## [25] munsell_0.5.0   broom_0.5.1      compiler_3.5.2
## [28] modelr_0.1.3    xfun_0.4         pkgconfig_2.0.2
## [31] manipulate_1.0.1 htmltools_0.3.6  tidyselect_0.2.5
## [34] expm_0.999-4    crayon_1.3.4     withr_2.1.2
## [37] MASS_7.3-51.1   bitops_1.0-6     grid_3.5.2
## [40] nlme_3.1-137    jsonlite_1.6     gtable_0.2.0
## [43] formatR_1.6     magrittr_1.5     KernSmooth_2.23-15
## [46] cli_1.0.1       stringi_1.3.1    xml2_1.2.0
## [49] generics_0.0.2  boot_1.3-20      tools_3.5.2
## [52] glue_1.3.0      hms_0.4.2        yaml_2.2.0
## [55] colorspace_1.4-0 caTools_1.17.1.2 rvest_0.3.2
## [58] knitr_1.21      haven_2.0.0

```