Final Assignment

Philipp Habicht

2022-11-11

```r
# Read in data from excel
data = read_xlsx("/Users/hawk/Documents/Unimelb/Practice of Statistics and Data Science/assignment5/art

# Cleaning and summarizing
# Grouping x and y variables
x = data.frame(data$Age, data$Gender, data$Diabetes, data$`Ever smoked`,
               data$PVD, data$CVD)
colnames(x) = c("Age", "Gender", "Diabetes", "Ever_smoked", "PVD", "CVD")
y = data.frame(data$`RA medial calcification`, data$`ITA intimal abnormality`)
colnames(y) = c("RA", "ITA")

# Summarize data
summary(x)
```

```
##       Age            Gender          Diabetes       Ever_smoked
##  Min.   :42.00   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:60.00   1st Qu.:1.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :68.50   Median :1.0000   Median :0.0000   Median :1.0000
##  Mean   :65.77   Mean   :0.8909   Mean   :0.2455   Mean   :0.6636
##  3rd Qu.:72.75   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :81.00   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       PVD              CVD
##  Min.   :0.0000   Min.   :0.0
##  1st Qu.:0.0000   1st Qu.:0.0
##  Median :0.0000   Median :0.0
##  Mean   :0.1727   Mean   :0.1
##  3rd Qu.:0.0000   3rd Qu.:0.0
##  Max.   :1.0000   Max.   :1.0
```

```r
summary(y)
```

```
##        RA              ITA
##  Min.   :0.0000   Min.   :0.0
##  1st Qu.:0.0000   1st Qu.:0.0
##  Median :0.0000   Median :1.0
##  Mean   :0.1273   Mean   :0.7
##  3rd Qu.:0.0000   3rd Qu.:1.0
##  Max.   :1.0000   Max.   :1.0
```

```r
# Classification problem: Building 4 groups from y variable
y %>%
  mutate(ycat = ifelse(RA==0 & ITA ==0, 1, 0), #no, no
         ycat = ifelse(RA==0 & ITA ==1, 2, ycat), #no, yes
         ycat = ifelse(RA==1 & ITA ==0, 3, ycat), #yes, no
```

```r
         ycat = ifelse(RA==1 & ITA ==1, 4, ycat)) -> y #yes, yes

y_categories = y$ycat

# Explanatory data analysis (EDA)

# Combining x and y to data frame
all_data = data.frame(x,y_categories)

# Frequencies RA
table(y$RA)
```

```
##
##  0  1
## 96 14
```

```r
round(table(y$RA)/length(y$RA),3)
```

```
##
##     0     1
## 0.873 0.127
```

```r
# Frequencies ITA
table(y$ITA)
```

```
##
##  0  1
## 33 77
```

```r
round(table(y$ITA)/length(y$ITA),3)
```

```
##
##   0   1
## 0.3 0.7
```

```r
# 4 groups
abs_freq = data.frame(table(as.factor(y_categories)))
colnames(abs_freq) = c("Groups", "Freq")
abs_freq
```

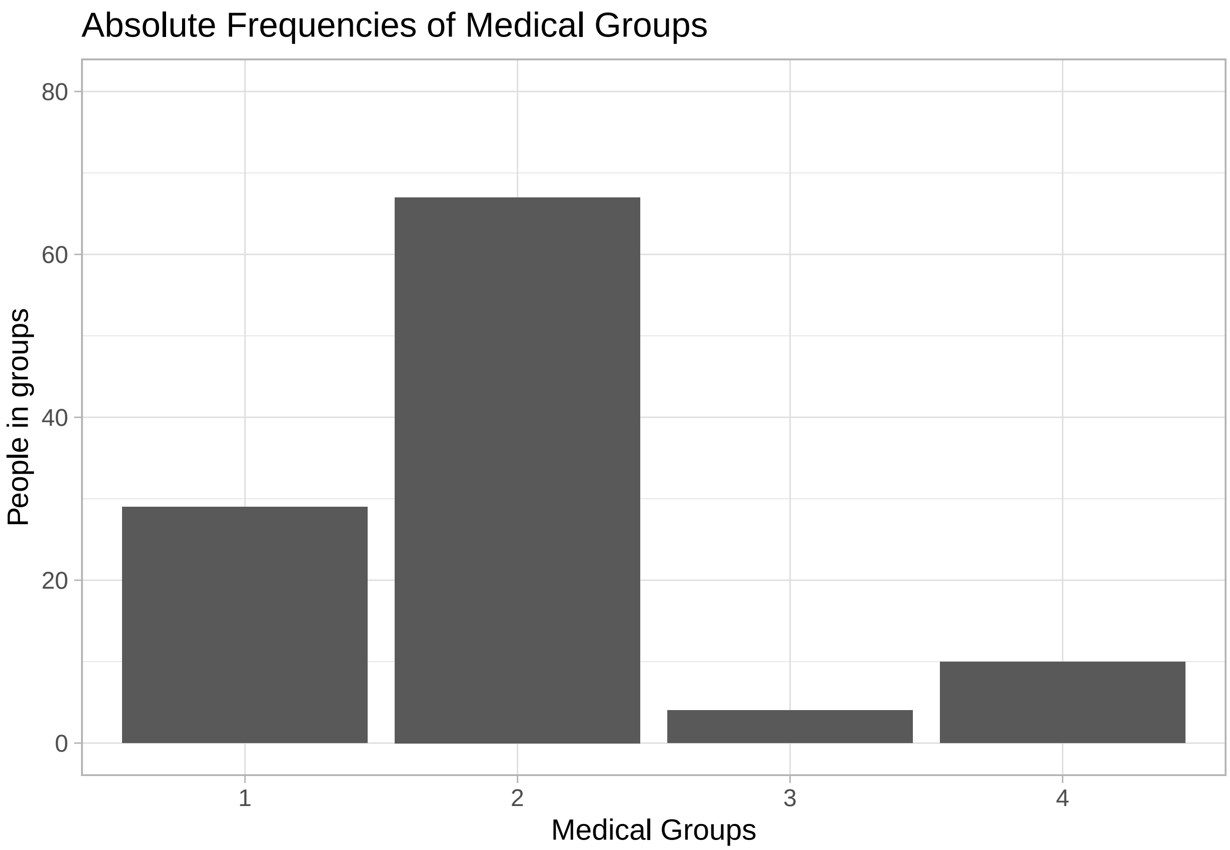
```
##   Groups Freq
## 1      1   29
## 2      2   67
## 3      3    4
## 4      4   10
```

```r
# Relative frequencies of classes in %
round((table(all_data$y_categories)/sum(table(all_data$y_categories)))*100,2)
```

```
##
##     1     2     3     4
## 26.36 60.91  3.64  9.09
```

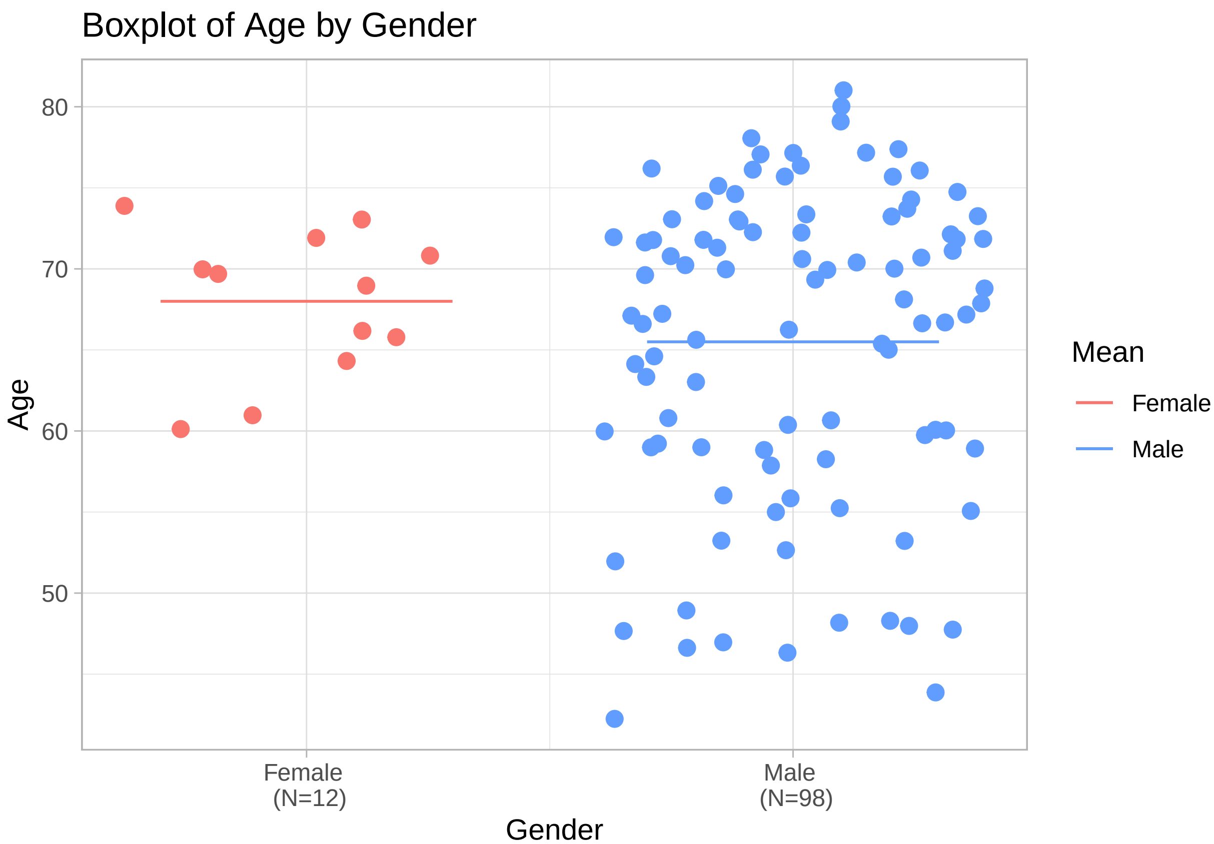
```r
# Plot
abs_freq %>%
  ggplot(aes(x = Groups, y = Freq, label = Freq)) +
  geom_bar(stat="identity") +
  ggtitle("Absolute Frequencies of Medical Groups") +
```

```
  theme(plot.title = element_text(hjust = 0.7)) +
  theme_light() +
  xlab("Medical Groups") +
  ylab("People in groups") +
  ylim(0,80)
```




```
# Groups are highly unbalanced with less data

# Age and gender information
xlab_fem = paste("Female \n (N=", nrow(x[x$Gender==0,]), ")", sep="")
xlab_male = paste("Male \n (N=", nrow(x[x$Gender==1,]), ")", sep="")
aver_age_fem = mean(x$Age[x$Gender==0])
aver_age_mal = mean(x$Age[x$Gender==1])

ggplot() +
  geom_jitter(data=x[x$Gender ==0,], aes(x = Gender, y=Age),
              color="#F8766D", size =2.5) +
  geom_jitter(data=x[x$Gender ==1,], aes(x = Gender, y=Age),
              color="#619CFF", size=2.5) +
  geom_segment(aes(x = -0.3, xend = 0.3, y = mean(aver_age_fem),
                   yend = mean(aver_age_fem), colour = "Female")) +
  geom_segment(aes(x = 0.7, xend = 1.3, y = mean(aver_age_mal),
                   yend = mean(aver_age_mal), colour = "Male")) +
  scale_color_manual(values = c("Female"= "#F8766D", "Male" = "#619CFF"),
                     name ="Mean") +
  scale_x_continuous(breaks = c(0,1), label=c(xlab_fem, xlab_male)) +
  labs(title = "Boxplot of Age by Gender", y = "Age") +
```

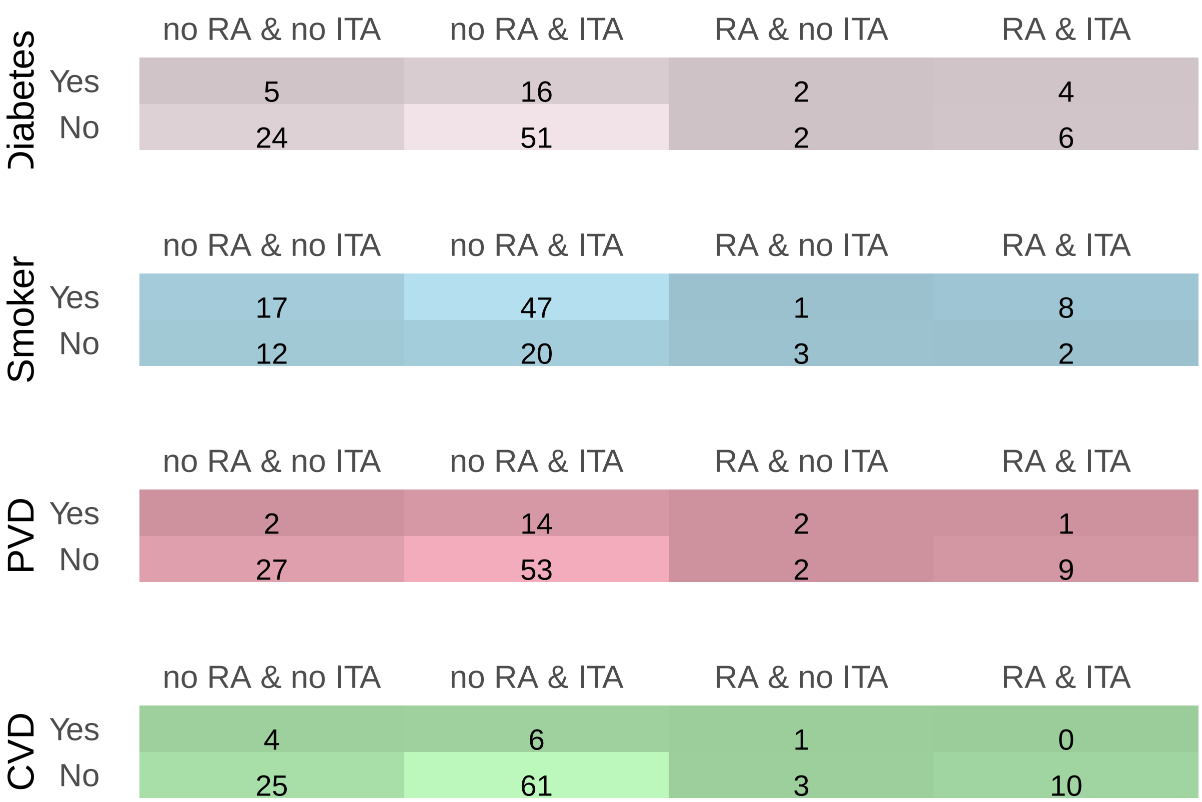
```
  theme_light()
```




```
## Confusion Matrix
# Diabetes
confusion_matrix <- as.data.frame(table(all_data$Diabetes,
                                        all_data$y_categories))
confusion_matrix %>%
  mutate(Var1 = ifelse(Var1 == 1, "Yes", "No")) %>%
  ggplot(mapping = aes(x = Var2, y = Var1)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(high = "lavenderblush1", low = "lavenderblush3",
                      name="Frequency", limits=c(0,70)) +
  ylab("Diabetes") +
  xlab("") +
  theme_classic() +
  theme(legend.position = "none",
        axis.title.x = element_text(vjust=-6),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.x = element_text(size = 12, vjust = 0.5),
        axis.text.y = element_text(size = 12),
        text = element_text(size = 14)) +
  scale_x_discrete(position="top",
                   labels = c("no RA & no ITA", "no RA & ITA",
```

```r
                              "RA & no ITA", "RA & ITA")) -> plot_diabetes

# Smoking
confusion_matrix <- as.data.frame(table(all_data$Ever_smoked,
                                        all_data$y_categories))
confusion_matrix %>%
  mutate(Var1 = ifelse(Var1 == 1, "Yes", "No")) %>%
  ggplot(mapping = aes(x = Var2, y = Var1)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(high = "lightblue1", low = "lightblue3",
                      name="Frequency", limits=c(0,70)) +
  ylab("Smoker") + xlab("") + theme_classic() +
  theme(legend.position = "none",
        axis.title.x = element_text(vjust=-6),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.x = element_text(size = 12, vjust = 0.5),
        axis.text.y = element_text(size = 12),
        text = element_text(size = 14)) +
  scale_x_discrete(position="top", labels = c("no RA & no ITA",
                                              "no RA & ITA", "RA & no ITA",
                                              "RA & ITA")) -> plot_smoker

# PVD
confusion_matrix <- as.data.frame(table(all_data$PVD, all_data$y_categories))
confusion_matrix %>%
  mutate(Var1 = ifelse(Var1 == 1, "Yes", "No")) %>%
  ggplot(mapping = aes(x = Var2, y = Var1)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(high = "pink1", low = "pink3", name="Frequency",
                      limits=c(0,70)) +
  ylab("PVD") + xlab("") + theme_classic() +
  theme(legend.position = "none",
        axis.title.x = element_text(vjust=-6),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.x = element_text(size = 12, vjust = 0.5),
        axis.text.y = element_text(size = 12),
        text = element_text(size = 14)) +
  scale_x_discrete(position="top",
                   labels = c("no RA & no ITA", "no RA & ITA",
                              "RA & no ITA", "RA & ITA")) -> plot_pvd

# CVD
confusion_matrix <- as.data.frame(table(all_data$CVD, all_data$y_categories))
confusion_matrix %>%
  mutate(Var1 = ifelse(Var1 == 1, "Yes", "No")) %>%
```

```
  ggplot(mapping = aes(x = Var2, y = Var1)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(high = "darkseagreen1", low = "darkseagreen3",
                      name="Frequency", limits=c(0,70)) +
  ylab("CVD") + xlab("") + theme_classic() +
  theme(legend.position = "none",
        axis.title.x = element_text(vjust=-6),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.x = element_text(size = 12, vjust = 0.5),
        axis.text.y = element_text(size = 12),
        text = element_text(size = 14)) +
  scale_x_discrete(position="top",
                   labels = c("no RA & no ITA", "no RA & ITA", "RA & no ITA",
                              "RA & ITA")) -> plot_cvd

ggarrange(plot_diabetes, plot_smoker, plot_pvd, plot_cvd, ncol=1, nrow=4)
```




```
# Data wrangling

# Before asking the questions we need to find a proper model that fits our
# data well
# Here, we are facing multiple problems because of the low total frequencies
# in the categories, there are no reliable predictions possible.
# I use multinomial logistic regression as a baseline model for the whole data set:
```

```r
set.seed(50)
lr = multinom(y_categories ~ ., data = all_data)
```

```
## # weights:  32 (21 variable)
## initial  value 152.492380
## iter  10 value 96.158528
## iter  20 value 91.865226
## iter  30 value 91.808704
## iter  40 value 91.805634
## final  value 91.805568
## converged
```

```r
# Predict
lr_predict = predict(lr, newdata = all_data)

# Building classification table
tab = table(pred = lr_predict, true = y_categories)
tab
```

```
##     true
## pred  1  2  3  4
##    1  7  5  1  0
##    2 22 61  2 10
##    3  0  0  1  0
##    4  0  1  0  0
```

```r
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab))/sum(tab)),3)
```

```
## [1] 0.627
```

```r
# Accuracy per Category
accuracy_category = numeric(4)
for (i in 1:length(accuracy_category))
{
  accuracy_category[i] = (tab[i,i])/67
}
round((accuracy_category),3)
```

```
## [1] 0.104 0.910 0.015 0.000
```

```r
# Only the target variables that have the highest amount in total
# frequencies were predicted from the model
# Because the imbalance of the target variable,
# we try to balance the data set by upsampling method from caret
upsample_data <- upSample(x, as.factor(y_categories))
table(upsample_data$Class)
```

```
##
##  1  2  3  4
## 67 67 67 67
```

```r
# Multinomial logistic regression with upsampled data
set.seed(50)
lr = multinom(Class ~ ., data = upsample_data)
```

```
## # weights:  32 (21 variable)
## initial  value 371.526889
```

```
## iter  10 value 281.311411
## iter  20 value 269.469910
## iter  30 value 268.741798
## iter  40 value 268.728561
## final  value 268.728480
## converged
```

```r
# Predict
lr_predict = predict(lr, newdata = upsample_data)

# Building classification table
tab = table(pred = lr_predict, true = upsample_data$Class)
tab
```

```
##     true
## pred  1  2  3  4
##    1 28 16 22  6
##    2  9 19  0  7
##    3 19 12 45  0
##    4 11 20  0 54
```

```r
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab))/sum(tab)),3)
```

```
## [1] 0.545
```

```r
# Accuracy per Category
accuracy_category = numeric(4)
for (i in 1:length(accuracy_category))
{
  accuracy_category[i] = (tab[i,i])/67
}
round((accuracy_category),3)
```

```
## [1] 0.418 0.284 0.672 0.806
```

```r
# Since we see in both methods an increased accuracy rate in the last two categories,
# the accuracy rates of the first two categories are decreasing a lot
# To prevent this prediction imbalance between the categories,
# I decided to perform two binary classifications for RA and ITA seperately instead
bin_y = y[-3]

# Ratio bin_y
table(bin_y$RA)
```

```
##
##  0  1
## 96 14
```

```r
table(bin_y$ITA)
```

```
##
##  0  1
## 33 77
```

```r
# Upsample RA and ITA
up_RA = upSample(x, as.factor(bin_y$RA))
up_ITA = upSample(x, as.factor(bin_y$ITA))
```

```r
# Show total frequencies
table(up_RA$Class)
```

```
##
##  0  1
## 96 96
```

```r
table(up_ITA$Class)
```

```
##
##  0  1
## 77 77
```

```r
# Now we have two balanced binary target variables and we perform our model fitting again

# Logistic regression
set.seed(50)
lr_RA = glm(Class ~.,family=binomial(link='logit'),data=up_RA)
lr_ITA = glm(Class ~.,family=binomial(link='logit'),data=up_ITA)

# Summary
summary(lr_RA)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(link = "logit"), data = up_RA)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9348  -0.9960   0.2307   0.9211   2.0243
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.94595    1.82870  -4.345 1.39e-05 ***
## Age          0.12137    0.02589   4.689 2.75e-06 ***
## Gender      -0.58592    0.53920  -1.087 0.277191
## Diabetes     1.30941    0.39552   3.311 0.000931 ***
## Ever_smoked -0.17583    0.37148  -0.473 0.635987
## PVD         -0.41410    0.49541  -0.836 0.403218
## CVD         -0.94504    0.68986  -1.370 0.170714
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 266.17  on 191  degrees of freedom
## Residual deviance: 225.08  on 185  degrees of freedom
## AIC: 239.08
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(lr_ITA)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(link = "logit"), data = up_ITA)
```

```
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.76053  -1.15321  -0.04015   1.04032   1.85630
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.42310    1.49334  -2.292   0.0219 *
## Age          0.03161    0.02000   1.581   0.1139
## Gender       1.04906    0.57912   1.811   0.0701 .
## Diabetes     0.15003    0.43481   0.345   0.7301
## Ever_smoked  0.64931    0.38797   1.674   0.0942 .
## PVD          1.29049    0.61078   2.113   0.0346 *
## CVD         -1.49916    0.60204  -2.490   0.0128 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 213.49  on 153  degrees of freedom
## Residual deviance: 191.74  on 147  degrees of freedom
## AIC: 205.74
##
## Number of Fisher Scoring iterations: 4
```

```
# Predict
lr_predict_RA = predict(lr_RA, newdata = up_RA, type = "response")
lr_predict_ITA = predict(lr_ITA, newdata = up_ITA, type = "response")

# Building classification table
tab_RA = table(pred = round(lr_predict_RA, 0), true = up_RA$Class)
tab_RA
```

```
##     true
## pred  0  1
##    0 58 11
##    1 38 85
```

```
tab_ITA = table(pred = round(lr_predict_ITA, 0), true = up_ITA$Class)
tab_ITA
```

```
##     true
## pred  0  1
##    0 45 28
##    1 32 49
```

```
# Calculating accuracy - sum of diagonal elements divided by total obs
# Precision
precision = function(confusion_matrix)
{
  TP = confusion_matrix[2,2] # True positive
  FP = confusion_matrix[1,2] # False positive
  return (TP/(TP + FP))
}

# Recall
```

```r
recall = function(confusion_matrix)
{
  TP = confusion_matrix[2,2] # True positive
  FN = confusion_matrix[2,1] # False negative
  return (TP/(TP + FN))
}

# Metrics
acc_RA = (sum(diag(tab_RA))/sum(tab_RA)) # Accuracy RA
acc_ITA = (sum(diag(tab_ITA))/sum(tab_ITA)) # Accuracy ITA
prec_RA = precision(tab_RA) # Precision RA
prec_ITA = precision(tab_ITA) # Precision ITA
re_RA = recall(tab_RA) # Recall RA
re_ITA = recall(tab_ITA) # Recall ITA

# Metric data frame
metric = as.data.frame(matrix(
  c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA), nrow = 3, ncol = 2))
colnames(metric) = c("RA", "ITA")
rownames(metric) = c("Accuracy", "Precision", "Recall")
metric
```

```
##                  RA       ITA
## Accuracy  0.7447917 0.6103896
## Precision 0.8854167 0.6363636
## Recall    0.6910569 0.6049383
```

```r
# Model fitting
# Building randomly generated train and test data set for x and y
set.seed(50)
ratio = 0.75

# RA
n = nrow(up_RA)
sample_number = (ratio * n)
train_ind = sample(n, sample_number)
xtrain = up_RA[-7][train_ind,]
ytrain = up_RA$Class[train_ind]
xtest = up_RA[-7][-train_ind,]
ytest = up_RA$Class[-train_ind]

# Building data frames
train_RA = data.frame(xtrain, ytrain)
test_RA = data.frame(xtest, ytest)

# ITA
n = nrow(up_ITA)
sample_number = (ratio * n)
train_ind = sample(n, as.integer(sample_number))
xtrain = up_ITA[-7][train_ind,]
ytrain = up_ITA$Class[train_ind]
xtest = up_ITA[-7][-train_ind,]
ytest = up_ITA$Class[-train_ind]
```

```r
# Building data frames
train_ITA = data.frame(xtrain, ytrain)
test_ITA = data.frame(xtest, ytest)

# The data will be trained on Random Forest and Support Vector Machines
# Random Forest
rf_RA = randomForest(ytrain~., data=train_RA, ntree=300)
rf_ITA = randomForest(ytrain~., data=train_ITA, ntree=300)

# Summary
print(rf_RA)
```

```
##
## Call:
##  randomForest(formula = ytrain ~ ., data = train_RA, ntree = 300)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 22.22%
## Confusion matrix:
##    0  1 class.error
## 0 51 24    0.320000
## 1  8 61    0.115942
```

```r
print(rf_ITA)
```

```
##
## Call:
##  randomForest(formula = ytrain ~ ., data = train_ITA, ntree = 300)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 39.13%
## Confusion matrix:
##    0  1 class.error
## 0 39 17   0.3035714
## 1 28 31   0.4745763
```

```r
# Metrics
acc_RA = (sum(diag(rf_RA$confusion))/nrow(train_RA)) # Accuracy RA
acc_ITA = (sum(diag(rf_ITA$confusion))/nrow(train_ITA)) # Accuracy ITA
prec_RA = precision(rf_RA$confusion) # Precision RA
prec_ITA = precision(rf_ITA$confusion) # Precision ITA
re_RA = recall(rf_RA$confusion) # Recall RA
re_ITA = recall(rf_ITA$confusion) # Recall ITA

# Metric data frame
rf_metric_df1 = as.data.frame(
  matrix(c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA),
         nrow = 3, ncol = 2))
colnames(rf_metric_df1) = c("RA", "ITA")
rownames(rf_metric_df1) = c("Accuracy", "Precision", "Recall")
rf_metric_df1
```

```
##                   RA       ITA
## Accuracy  0.7777778 0.6086957
## Precision 0.7176471 0.6458333
## Recall    0.8840580 0.5254237
```

```r
# Parameter tuning
set.seed(50)
trees = c(50, 100, 200, 300, 400, 500)
nodes = c(2, 4, 5, 6, 8, 10)
tuned_rf_RA = tune(randomForest, ytrain~., data = train_RA,
                   ranges = list(nodesize = nodes, ntree = trees))
tuned_rf_ITA = tune(randomForest, ytrain~., data = train_ITA,
                    ranges = list(nodesize = nodes, ntree = trees))

# Evaluating the best size of trees
summary(tuned_rf_RA)
```

```
##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  nodesize ntree
##         2   100
##
## - best performance: 0.2285714
##
## - Detailed performance results:
##    nodesize ntree     error dispersion
## 1         2    50 0.2633333  0.1007418
## 2         4    50 0.2709524  0.1395282
## 3         5    50 0.2700000  0.1098580
## 4         6    50 0.2700000  0.1619367
## 5         8    50 0.2695238  0.1175504
## 6        10    50 0.2980952  0.1168194
## 7         2   100 0.2285714  0.1084256
## 8         4   100 0.2842857  0.1342007
## 9         5   100 0.2419048  0.1081650
## 10        6   100 0.2700000  0.1142648
## 11        8   100 0.3114286  0.1667528
## 12       10   100 0.3252381  0.1318330
## 13        2   200 0.2485714  0.1118524
## 14        4   200 0.2766667  0.1382767
## 15        5   200 0.2628571  0.1341584
## 16        6   200 0.2561905  0.1263835
## 17        8   200 0.2990476  0.1429594
## 18       10   200 0.2904762  0.1591898
## 19        2   300 0.2352381  0.1254109
## 20        4   300 0.2700000  0.1331830
## 21        5   300 0.2633333  0.1386333
## 22        6   300 0.2628571  0.1208186
## 23        8   300 0.2842857  0.1110896
## 24       10   300 0.3047619  0.1304874
## 25        2   400 0.2485714  0.1118524
```

```
## 26        4   400 0.2766667  0.1382767
## 27        5   400 0.2628571  0.1208186
## 28        6   400 0.2700000  0.1331830
## 29        8   400 0.2842857  0.1538790
## 30       10   400 0.2980952  0.1479038
## 31        2   500 0.2419048  0.1169401
## 32        4   500 0.2561905  0.1302323
## 33        5   500 0.2628571  0.1208186
## 34        6   500 0.2628571  0.1208186
## 35        8   500 0.2914286  0.1275740
## 36       10   500 0.2771429  0.1479821
```

```
summary(tuned_rf_ITA)
```

```
##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  nodesize ntree
##         5    50
##
## - best performance: 0.3840909
##
## - Detailed performance results:
##    nodesize ntree     error dispersion
## 1         2    50 0.4265152  0.1827295
## 2         4    50 0.4189394  0.1307803
## 3         5    50 0.3840909  0.1893451
## 4         6    50 0.3931818  0.1976307
## 5         8    50 0.3946970  0.2184438
## 6        10    50 0.4007576  0.1564618
## 7         2   100 0.4189394  0.1282197
## 8         4   100 0.4098485  0.1951303
## 9         5   100 0.3848485  0.2096355
## 10        6   100 0.3939394  0.1691731
## 11        8   100 0.4106061  0.1577747
## 12       10   100 0.3939394  0.1823435
## 13        2   200 0.4189394  0.2026206
## 14        4   200 0.4280303  0.1971072
## 15        5   200 0.4106061  0.1795524
## 16        6   200 0.4181818  0.1534307
## 17        8   200 0.4196970  0.1954813
## 18       10   200 0.3946970  0.2184438
## 19        2   300 0.3931818  0.1736908
## 20        4   300 0.4189394  0.1730876
## 21        5   300 0.4098485  0.1699797
## 22        6   300 0.4196970  0.1876247
## 23        8   300 0.4363636  0.1597589
## 24       10   300 0.4106061  0.1707414
## 25        2   400 0.3939394  0.1823435
## 26        4   400 0.4272727  0.1749619
## 27        5   400 0.4098485  0.1401208
## 28        6   400 0.4272727  0.1611895
```

```
## 29        8   400 0.4106061  0.1920122
## 30       10   400 0.4272727  0.1611895
## 31        2   500 0.3939394  0.1906189
## 32        4   500 0.3931818  0.1936870
## 33        5   500 0.4439394  0.1729457
## 34        6   500 0.4363636  0.1862945
## 35        8   500 0.4106061  0.1707414
## 36       10   500 0.4106061  0.1707414
```

```
# Prediction with tuned parameters
best_rf_RA = randomForest(ytrain~., data=train_RA,
                          nodesize = tuned_rf_RA$best.parameters[1,1],
                          ntree=tuned_rf_RA$best.parameters[1,2])
best_rf_ITA = randomForest(ytrain~., data=train_ITA,
                           nodesize = tuned_rf_ITA$best.parameters[1,1],
                          ntree=tuned_rf_ITA$best.parameters[1,2])

# Summary
print(best_rf_RA)
```

```
##
## Call:
##  randomForest(formula = ytrain ~ ., data = train_RA, nodesize = tuned_rf_RA$best.parameters[1,
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 24.31%
## Confusion matrix:
##    0  1 class.error
## 0 51 24   0.3200000
## 1 11 58   0.1594203
```

```
print(best_rf_ITA)
```

```
##
## Call:
##  randomForest(formula = ytrain ~ ., data = train_ITA, nodesize = tuned_rf_ITA$best.parameters[1,
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 38.26%
## Confusion matrix:
##    0  1 class.error
## 0 33 23   0.4107143
## 1 21 38   0.3559322
```

```
# Metrics
acc_RA = (sum(diag(best_rf_RA$confusion))/nrow(train_RA)) # Accuracy RA
acc_ITA = (sum(diag(best_rf_ITA$confusion))/nrow(train_ITA)) # Accuracy ITA
prec_RA = precision(best_rf_RA$confusion) # Precision RA
prec_ITA = precision(best_rf_ITA$confusion) # Precision ITA
re_RA = recall(best_rf_RA$confusion) # Recall RA
re_ITA = recall(best_rf_ITA$confusion) # Recall ITA
```

```r
# Metric data frame
rf_metric_df2 = as.data.frame(
  matrix(c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA),
         nrow = 3, ncol = 2))
colnames(rf_metric_df2) = c("RA", "ITA")
rownames(rf_metric_df2) = c("Accuracy", "Precision", "Recall")
rf_metric_df2
```

```
##                  RA       ITA
## Accuracy  0.7569444 0.6173913
## Precision 0.7073171 0.6229508
## Recall    0.8405797 0.6440678
```

```r
# SVM
set.seed(50)
svmfit_RA = svm(ytrain~., data=train_RA, cost = 10, kernel = "radial")
svmfit_ITA = svm(ytrain~., data=train_ITA, cost = 10, kernel = "radial")

# Prediction
pred_RA = predict(svmfit_RA, new_data = train_RA$ytrain)
pred_ITA = predict(svmfit_ITA, new_data = train_ITA$ytrain)

# Confusion matrix
conf_RA = table(pred = pred_RA, true = train_RA$ytrain)
conf_ITA = table(pred = pred_ITA, true = train_ITA$ytrain)

# Metrics
acc_RA = (sum(diag(conf_RA))/nrow(train_RA)) # Accuracy RA
acc_ITA = (sum(diag(conf_ITA))/nrow(train_ITA)) # Accuracy ITA
prec_RA = precision(conf_RA) # Precision RA
prec_ITA = precision(conf_ITA) # Precision ITA
re_RA = recall(conf_RA) # Recall RA
re_ITA = recall(conf_ITA) # Recall ITA

# Metric data frame
svm_metric_df1 = as.data.frame(
  matrix(c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA),
         nrow = 3, ncol = 2))
colnames(svm_metric_df1) = c("RA", "ITA")
rownames(svm_metric_df1) = c("Accuracy", "Precision", "Recall")
svm_metric_df1
```

```
##                  RA       ITA
## Accuracy  0.7361111 0.7391304
## Precision 0.9275362 0.7796610
## Recall    0.6597938 0.7301587
```

```r
# SVM tuning
# Gamma and cost values
set.seed(50)
gamma_values = c(0.1, 1, 10, 100, 1000)
cost_values = c(0.5, 1, 2, 3, 4)

# Tuning
tune_RA = tune(svm, ytrain~., data = train_RA,
```

```r
                ranges = list(gamma = gamma_values, cost = cost_values),
                kernel = 'radial')
tune_ITA = tune(svm, ytrain~., data = train_ITA,
                 ranges = list(gamma = gamma_values, cost = cost_values),
                 kernel = 'radial')

# Summary
summary(tune_RA)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    100    1
##
## - best performance: 0.07619048
##
## - Detailed performance results:
##    gamma cost      error dispersion
## 1  1e-01  0.5 0.35285714 0.08119944
## 2  1e+00  0.5 0.31809524 0.13445859
## 3  1e+01  0.5 0.22095238 0.09465962
## 4  1e+02  0.5 0.11190476 0.10184120
## 5  1e+03  0.5 0.11190476 0.10184120
## 6  1e-01  1.0 0.33238095 0.07231548
## 7  1e+00  1.0 0.29095238 0.14150061
## 8  1e+01  1.0 0.20809524 0.07920778
## 9  1e+02  1.0 0.07619048 0.05029517
## 10 1e+03  1.0 0.07619048 0.05029517
## 11 1e-01  2.0 0.35333333 0.06256466
## 12 1e+00  2.0 0.28952381 0.12536266
## 13 1e+01  2.0 0.18047619 0.05911287
## 14 1e+02  2.0 0.07619048 0.05029517
## 15 1e+03  2.0 0.07619048 0.05029517
## 16 1e-01  3.0 0.35333333 0.06256466
## 17 1e+00  3.0 0.27571429 0.12242716
## 18 1e+01  3.0 0.18761905 0.06738838
## 19 1e+02  3.0 0.07619048 0.05029517
## 20 1e+03  3.0 0.07619048 0.05029517
## 21 1e-01  4.0 0.35333333 0.06256466
## 22 1e+00  4.0 0.28285714 0.12051372
## 23 1e+01  4.0 0.18000000 0.06563065
## 24 1e+02  4.0 0.07619048 0.05029517
## 25 1e+03  4.0 0.07619048 0.05029517
```

```r
summary(tune_ITA)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  gamma cost
##   1000    1
##
## - best performance: 0.1325758
##
## - Detailed performance results:
##    gamma cost     error dispersion
## 1  1e-01  0.5 0.4166667 0.14204405
## 2  1e+00  0.5 0.3659091 0.11282256
## 3  1e+01  0.5 0.3318182 0.08350915
## 4  1e+02  0.5 0.2204545 0.12668868
## 5  1e+03  0.5 0.2045455 0.13918701
## 6  1e-01  1.0 0.4083333 0.16444998
## 7  1e+00  1.0 0.3825758 0.12587060
## 8  1e+01  1.0 0.3045455 0.09228753
## 9  1e+02  1.0 0.1750000 0.08522119
## 10 1e+03  1.0 0.1325758 0.10020194
## 11 1e-01  2.0 0.4083333 0.14971362
## 12 1e+00  2.0 0.3560606 0.11605177
## 13 1e+01  2.0 0.3053030 0.12448496
## 14 1e+02  2.0 0.2015152 0.07636123
## 15 1e+03  2.0 0.1325758 0.10020194
## 16 1e-01  3.0 0.4000000 0.14632526
## 17 1e+00  3.0 0.3568182 0.10396253
## 18 1e+01  3.0 0.2962121 0.12306299
## 19 1e+02  3.0 0.2015152 0.07636123
## 20 1e+03  3.0 0.1325758 0.10020194
## 21 1e-01  4.0 0.4000000 0.14632526
## 22 1e+00  4.0 0.3659091 0.08999612
## 23 1e+01  4.0 0.2871212 0.12086704
## 24 1e+02  4.0 0.2015152 0.07636123
## 25 1e+03  4.0 0.1325758 0.10020194
```

```r
# SVM with best parameter
best_RA = svm(ytrain~., data=train_RA,
              gamma = tune_RA$best.parameters[1],
              cost = tune_RA$best.parameters[2], kernel = "radial")
best_ITA = svm(ytrain~., data=train_ITA,
               gamma = tune_ITA$best.parameters[1],
               cost = tune_ITA$best.parameters[2], kernel = "radial")

# Prediction
pred_best_RA = predict(best_RA, new_data = train_RA$ytrain)
pred_best_ITA = predict(best_ITA, new_data = train_ITA$ytrain)

# Confusion matrix
conf_RA = table(pred = pred_best_RA, true = train_RA$ytrain)
conf_ITA = table(pred = pred_best_ITA, true = train_ITA$ytrain)

# Metrics
acc_RA = (sum(diag(conf_RA))/nrow(train_RA)) # Accuracy RA
acc_ITA = (sum(diag(conf_ITA))/nrow(train_ITA)) # Accuracy ITA
```

```r
prec_RA = precision(conf_RA) # Precision RA
prec_ITA = precision(conf_ITA) # Precision ITA
re_RA = recall(conf_RA) # Recall RA
re_ITA = recall(conf_ITA) # Recall ITA

# Metric data frame
best_svm_metric_df = as.data.frame(
  matrix(c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA),
         nrow = 3, ncol = 2))
colnames(best_svm_metric_df) = c("RA", "ITA")
rownames(best_svm_metric_df) = c("Accuracy", "Precision", "Recall")
best_svm_metric_df
```

```
##                  RA       ITA
## Accuracy  0.9444444 0.9565217
## Precision 0.9565217 0.9491525
## Recall    0.9295775 0.9655172
```

```r
# Model Selection

# SVM outperforms Random Forest on the training sample,
# but a high training accuracy doesnt indicate a high testing accuracy
# Performing Cross-Validation for Model Selection with best models
# Random Forest CV
set.seed(50)
trControl <- trainControl(method = "cv", number = 5, search = "grid")

# RA
mtry <- sqrt(ncol(train_RA))
tunegrid <- expand.grid(.mtry=mtry)
rf_RA_cv <- train(ytrain~., data = train_RA, method = "rf",
                  metric = "Accuracy", tuneGrid = tunegrid,
                  trControl = trControl,
                  nodesize = tuned_rf_RA$best.parameters[1,1],
                  ntree = tuned_rf_RA$best.parameters[1,2])

# ITA
mtry <- sqrt(ncol(train_ITA))
tunegrid <- expand.grid(.mtry=mtry)
rf_ITA_cv = train(ytrain~., data = train_ITA, method = "rf",
                  metric = "Accuracy", tuneGrid = tunegrid,
                  trControl = trControl,
                  nodesize = tuned_rf_ITA$best.parameters[1,1],
                  ntree = tuned_rf_ITA$best.parameters[1,2])

# Average accuracy
print(rf_RA_cv)
```

```
## Random Forest
##
## 144 samples
##   6 predictor
##   2 classes: '0', '1'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 115, 115, 115, 115, 116
## Resampling results:
##
##   Accuracy   Kappa
##   0.8054187  0.6151238
##
## Tuning parameter 'mtry' was held constant at a value of 2.645751
```

```
print(rf_ITA_cv)
```

```
## Random Forest
##
## 115 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 92, 92, 91, 93, 92
## Resampling results:
##
##   Accuracy   Kappa
##   0.5735507  0.1479874
##
## Tuning parameter 'mtry' was held constant at a value of 2.645751
```

```
# SVM CV
set.seed(50)
n1 = dim(train_RA)[1]
n2 = dim(train_ITA)[1]
cvvec_RA <- sample(rep(1:5, length.out = n1, replace = TRUE))
cvvec_ITA = sample(rep(1:5, length.out = n2, replace = TRUE))
pred_y_RA = numeric(n1)
pred_y_ITA = numeric(n2)
mat = matrix(0, 6, 2)

# Self build cross validation
for (i in 1:5)
{
  # Model
  best_RA = svm(ytrain[cvvec_RA != i]~., data=train_RA[cvvec_RA != i,],
                gamma = tune_RA$best.parameters[1],
                cost = tune_RA$best.parameters[2], kernel = "radial")
  best_ITA = svm(ytrain[cvvec_ITA != i]~., data=train_ITA[cvvec_ITA != i,],
                gamma = tune_ITA$best.parameters[1],
                cost = tune_ITA$best.parameters[2], kernel = "radial")
  # Predict
  pred_y_RA[cvvec_RA == i] <- predict(best_RA,
                                      newdata = train_RA[cvvec_RA == i,])
  pred_y_ITA[cvvec_ITA == i] <- predict(best_ITA,
                                      newdata = train_ITA[cvvec_ITA == i,])
  # Confusion Matrix
  tab_RA = table(pred = as.integer(pred_y_RA[cvvec_RA == i]),
```

```
                  true = train_RA$ytrain[cvvec_RA == i])
  tab_ITA = table(pred = as.integer(pred_y_ITA[cvvec_ITA == i]),
                   true = train_ITA$ytrain[cvvec_ITA == i])
  # Accuracy matrix RA & ITA
  mat[i, 1] = round((sum(diag(tab_RA))/sum(tab_RA)),2)
  mat[i, 2] = round((sum(diag(tab_ITA))/sum(tab_ITA)),2)
}

# Accuracy result as data frame
mat = as.data.frame(mat)
colnames(mat) = c("RA", "ITA")
rownames(mat) = c(1:5, "Average")
mat[6,1] = mean(mat[1:5,1])
mat[6,2] = mean(mat[1:5,2])
mat
```

```
##           RA   ITA
## 1       0.34 0.780
## 2       0.62 0.350
## 3       0.38 0.300
## 4       0.62 0.520
## 5       0.64 0.570
## Average 0.52 0.504
```

```
# Fitting on test data
# Prediction with tuned parameters
set.seed(50)
pred_RA = predict(best_rf_RA, newdata = test_RA[-7])
pred_ITA = predict(best_rf_ITA, newdata = test_ITA[-7])

# Confusion matrix
conf_RA = table(pred = pred_RA, true = test_RA$ytest)
conf_RA
```

```
##     true
## pred  0  1
##    0 18  5
##    1  3 22
```

```
conf_ITA = table(pred = pred_ITA, true = test_ITA$ytest)
conf_ITA
```

```
##     true
## pred  0  1
##    0 18 10
##    1  3  8
```

```
# Metrics
acc_RA = (sum(diag(conf_RA))/nrow(test_RA)) # Accuracy RA
acc_ITA = (sum(diag(conf_ITA))/nrow(test_ITA)) # Accuracy ITA
prec_RA = precision(conf_RA) # Precision RA
prec_ITA = precision(conf_ITA) # Precision ITA
re_RA = recall(conf_RA) # Recall RA
re_ITA = recall(conf_ITA) # Recall ITA

# Metric data frame
```

```
final_model = as.data.frame(
  matrix(c(acc_RA, prec_RA, re_RA, acc_ITA, prec_ITA, re_ITA),
         nrow = 3, ncol = 2))
colnames(final_model) = c("RA", "ITA")
rownames(final_model) = c("Accuracy", "Precision", "Recall")
final_model
```

```
##                  RA       ITA
## Accuracy  0.8333333 0.6666667
## Precision 0.8148148 0.4444444
## Recall    0.8800000 0.7272727
```