2017195113_intro

12/14/2020

```r
library(tidyverse)
library(magrittr)
library(outliers)
library(caret)
library(doParallel)

seed = 323
```

#1. Data Preprocessing —-

```r
# data.raw = read.csv("intro_extravert.csv")
#  #1250 obs, 94 vars
#
# #1.1 Data overview
# str(data.raw) #all int
# data.raw %>% view()
#
# #1.2 check NA / Impute NA
# data.raw %>% anyNA #no NAs
#
# #1.2.A. Setting NAs: converting IE, gender, engnat = 0 to NA
# data.raw %<>%
#   mutate(IE = replace(IE, IE == 0, NA))
#
# data.raw %<>%
#   mutate(gender = replace(gender, gender == 0, NA))
#
# data.raw %<>%
#   mutate(engnat = replace(engnat, engnat == 0, NA))
#
# #Check NAs
# data.raw %>% map_dbl(~sum(is.na(.))) #NA is set
#
# #Dropping the NAs
# data.raw %<>% na.omit()
# data.raw %>% anyNA #no NA
#
# #1.2.B. Setting NAs: convert outliers to Nas, impute NAs
# outlier(data.raw)
# data.raw %>% Hmisc::hist.data.frame() #check Q17
# data.raw$Q17A #-77 seems suspicious, impute it to median
```

```
# data.raw %<>% mutate(Q17A = replace(Q17A, Q17A == -77, NA))
#
# #check NA
# data.raw %>% anyNA
# data.raw %>% map_dbl(~sum(is.na(.))) #Q17A, 131
#
# #impute NA
# set.seed(seed)
# data.raw %<>%
#   # preProcess(method = "knnImpute") %>%
#   preProcess(method = "medianImpute") %>%
#   # preProcess(method = "bagImpute") %>%
#   predict(data.raw) %T>% print
#
# #check imputation
# data.raw %>% anyNA
# data.raw %>% summary()
# data.raw$Q17A
#
#
# #1.3 changing data types to relevant data
# data.raw %>% str()
# #IE, gender, engnat to factors
# data.raw %<>% mutate_at(vars(IE: engnat), as.factor)
# #checking
# str(data.raw)
# data.raw$IE %>% levels()
# data.raw$gender %>% levels()
# data.raw$engnat %>% levels()
#
# #1.4 Feature Selection
# #There is no particular feature to be removed. However, if we want to classify the responda
# #into introverts or extroverts, level 3 (neither), should be removed
# #from the target variable. Also, naming should be changed to Y, N
# data.raw$IE %>% table()
# dataset = data.raw %>% mutate(IE = replace(IE, IE == "3", NA))
# dataset %<>% drop_na()
# #check, making 2 leveled factor
# dataset$IE %>% table()
# dataset$IE %<>% as.numeric
# dataset$IE %<>% as.factor
# dataset$IE %>% levels()
# dataset$IE %>% class()
# dataset %>% anyNA
# str(dataset)
# dataset %>% view()
# levels(dataset$IE)[levels(dataset$IE)=="1"] <- "I"
# levels(dataset$IE)[levels(dataset$IE)=="2"] <- "E"

#final preprocessed data
```

```r
str(dataset)
```

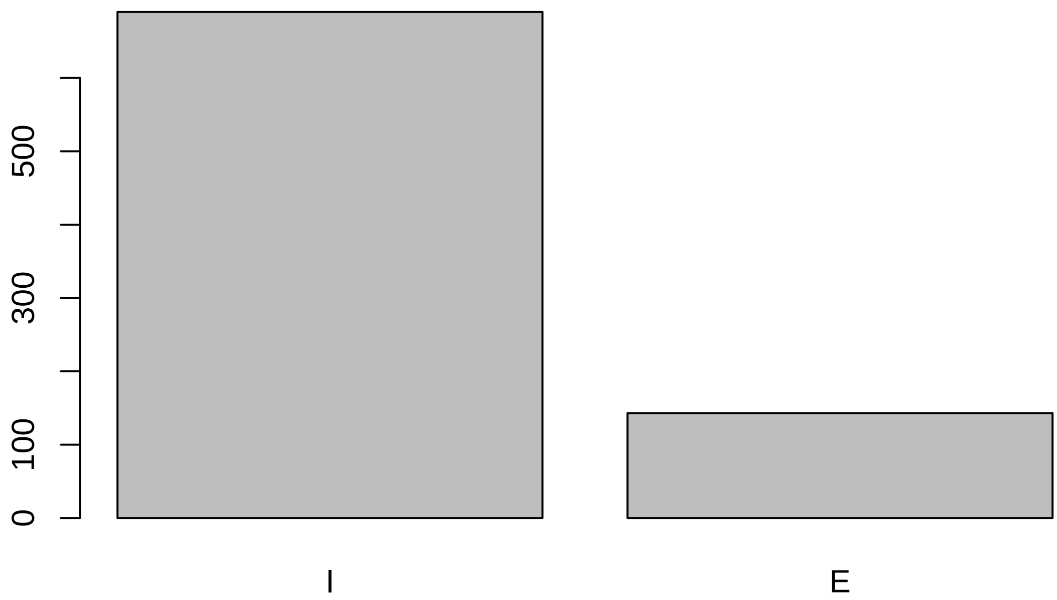
```
## 'data.frame':    924 obs. of  94 variables:
##  $ IE    : Factor w/ 2 levels "I","E": 2 1 1 1 1 1 1 1 1 1 ...
##  $ gender: Factor w/ 3 levels "1","2","3": 1 1 1 1 2 2 2 1 2 3 ...
##  $ engnat: Factor w/ 2 levels "1","2": 2 1 1 2 1 1 1 1 1 2 ...
##  $ Q1A   : int  3 5 5 5 5 4 5 5 5 4 ...
##  $ Q2A   : int  1 5 5 5 5 1 2 5 5 5 ...
##  $ Q3A   : int  1 4 1 2 5 5 5 4 4 4 ...
##  $ Q4A   : int  5 4 2 2 5 3 2 5 5 2 ...
##  $ Q5A   : int  3 5 4 4 1 5 4 4 3 4 ...
##  $ Q6A   : int  5 4 1 5 3 4 4 5 1 4 ...
##  $ Q7A   : int  4 5 5 2 5 5 5 5 5 5 ...
##  $ Q8A   : int  3 3 5 2 4 3 2 2 3 4 ...
##  $ Q9A   : int  3 5 1 5 5 4 2 5 5 5 ...
##  $ Q10A  : int  3 5 5 5 5 5 5 5 5 5 ...
##  $ Q11A  : int  1 4 2 4 1 3 4 1 1 3 ...
##  $ Q12A  : int  5 5 4 5 5 5 5 5 5 4 ...
##  $ Q13A  : int  5 3 5 2 4 1 1 2 1 1 ...
##  $ Q14A  : int  5 1 1 2 1 1 2 1 1 1 ...
##  $ Q15A  : int  1 1 1 2 4 3 1 1 4 1 ...
##  $ Q16A  : int  5 1 4 3 2 3 3 3 3 2 ...
##  $ Q17A  : num  2 1 1 2 2 1 4 1 1 2 ...
##  $ Q18A  : int  4 1 5 4 4 4 3 5 5 4 ...
##  $ Q19A  : int  4 1 1 1 3 2 1 1 1 1 ...
##  $ Q20A  : int  2 2 1 1 4 2 1 1 1 1 ...
##  $ Q21A  : int  5 3 4 1 1 3 1 4 1 3 ...
##  $ Q22A  : int  5 3 1 2 3 5 2 2 4 3 ...
##  $ Q23A  : int  5 1 5 3 3 3 2 4 4 3 ...
##  $ Q24A  : int  4 1 1 2 1 4 1 1 1 1 ...
##  $ Q25A  : int  5 2 5 4 5 4 2 4 5 5 ...
##  $ Q26A  : int  5 1 5 4 5 4 4 5 5 4 ...
##  $ Q27A  : int  1 5 5 1 5 4 4 2 5 5 ...
##  $ Q28A  : int  2 1 1 4 4 3 4 4 2 2 ...
##  $ Q29A  : int  1 4 1 2 2 3 4 5 1 3 ...
##  $ Q30A  : int  4 4 1 3 1 2 2 5 5 2 ...
##  $ Q31A  : int  3 1 5 2 5 5 4 2 5 1 ...
##  $ Q32A  : int  5 2 5 1 2 3 5 2 1 1 ...
##  $ Q33A  : int  5 3 1 2 4 3 2 4 4 3 ...
##  $ Q34A  : int  2 1 1 2 2 3 4 4 3 3 ...
##  $ Q35A  : int  5 2 1 3 4 5 2 1 5 4 ...
##  $ Q36A  : int  3 1 2 2 2 3 2 1 1 1 ...
##  $ Q37A  : int  5 1 1 1 5 5 2 1 1 1 ...
##  $ Q38A  : int  1 3 4 1 1 1 5 1 4 5 ...
##  $ Q39A  : int  1 5 1 1 2 4 1 1 1 1 ...
##  $ Q40A  : int  3 4 5 5 1 4 5 5 1 2 ...
##  $ Q41A  : int  2 5 5 2 5 3 4 5 3 5 ...
##  $ Q42A  : int  2 3 5 5 5 3 3 5 3 5 ...
##  $ Q43A  : int  1 4 5 5 1 2 5 5 5 5 ...
##  $ Q44A  : int  5 5 4 4 1 5 4 5 3 2 ...
```

```
##  $ Q45A  : int  1 5 1 4 4 3 5 5 1 5 ...
##  $ Q46A  : int  5 4 1 3 4 2 4 4 1 5 ...
##  $ Q47A  : int  1 1 1 1 4 3 1 1 3 3 ...
##  $ Q48A  : int  1 1 1 1 5 2 4 2 1 2 ...
##  $ Q49A  : int  1 1 1 1 1 3 4 1 5 3 ...
##  $ Q50A  : int  1 5 1 3 2 3 4 5 5 1 ...
##  $ Q51A  : int  1 4 5 5 1 4 2 1 1 2 ...
##  $ Q52A  : int  2 2 1 4 1 2 4 4 1 2 ...
##  $ Q53A  : int  2 5 4 2 4 5 5 4 1 3 ...
##  $ Q54A  : int  4 4 1 4 4 2 4 5 3 2 ...
##  $ Q55A  : int  1 4 1 2 4 5 4 5 1 1 ...
##  $ Q56A  : int  4 3 1 5 3 4 5 5 5 2 ...
##  $ Q57A  : int  1 2 5 2 5 4 2 5 5 5 ...
##  $ Q58A  : int  4 4 5 1 3 4 2 1 5 1 ...
##  $ Q59A  : int  4 4 5 4 3 5 5 5 1 4 ...
##  $ Q60A  : int  4 4 5 5 1 3 2 5 3 3 ...
##  $ Q61A  : int  4 5 5 3 4 5 5 5 1 3 ...
##  $ Q62A  : int  5 4 1 2 1 4 1 4 1 1 ...
##  $ Q63A  : int  4 2 1 1 3 2 1 1 1 1 ...
##  $ Q64A  : int  2 4 5 4 3 2 2 2 2 5 ...
##  $ Q65A  : int  1 4 1 5 1 5 2 5 2 3 ...
##  $ Q66A  : int  5 4 1 1 1 5 1 4 4 3 ...
##  $ Q67A  : int  1 3 5 2 2 5 2 5 1 5 ...
##  $ Q68A  : int  4 4 5 1 4 4 4 5 1 2 ...
##  $ Q69A  : int  5 1 1 5 1 5 2 1 3 1 ...
##  $ Q70A  : int  5 4 2 4 4 5 2 1 5 4 ...
##  $ Q71A  : int  1 1 1 1 1 4 4 4 5 2 ...
##  $ Q72A  : int  2 3 1 1 1 1 1 4 1 1 ...
##  $ Q73A  : int  5 2 1 3 2 3 1 1 1 2 ...
##  $ Q74A  : int  2 1 2 4 5 3 2 4 5 1 ...
##  $ Q75A  : int  2 3 1 3 4 4 5 4 5 3 ...
##  $ Q76A  : int  1 2 1 4 3 3 3 4 1 1 ...
##  $ Q77A  : int  3 4 1 4 2 4 5 1 3 4 ...
##  $ Q78A  : int  5 4 5 2 4 2 2 1 4 2 ...
##  $ Q79A  : int  4 1 1 4 3 3 4 1 3 2 ...
##  $ Q80A  : int  4 1 1 1 1 1 1 1 1 1 ...
##  $ Q81A  : int  1 5 5 4 4 3 5 5 3 4 ...
##  $ Q82A  : int  3 4 4 4 1 3 4 5 3 4 ...
##  $ Q83A  : int  5 5 5 4 1 4 5 5 5 4 ...
##  $ Q84A  : int  2 5 5 5 2 3 5 5 4 1 ...
##  $ Q85A  : int  3 4 2 4 1 3 4 4 3 1 ...
##  $ Q86A  : int  4 5 5 3 5 4 4 4 5 2 ...
##  $ Q87A  : int  5 4 5 4 2 2 1 2 3 3 ...
##  $ Q88A  : int  5 3 4 1 3 3 4 2 3 3 ...
##  $ Q89A  : int  5 1 2 2 3 1 3 2 1 5 ...
##  $ Q90A  : int  3 1 2 2 5 3 1 2 3 2 ...
##  $ Q91A  : int  4 1 1 2 1 2 1 1 1 1 ...
##  - attr(*, "na.action")= 'omit' Named int [1:23] 73 77 89 119 139 254 326 353 412 472 ...
##   ..- attr(*, "names")= chr [1:23] "73" "77" "89" "119" ...
```

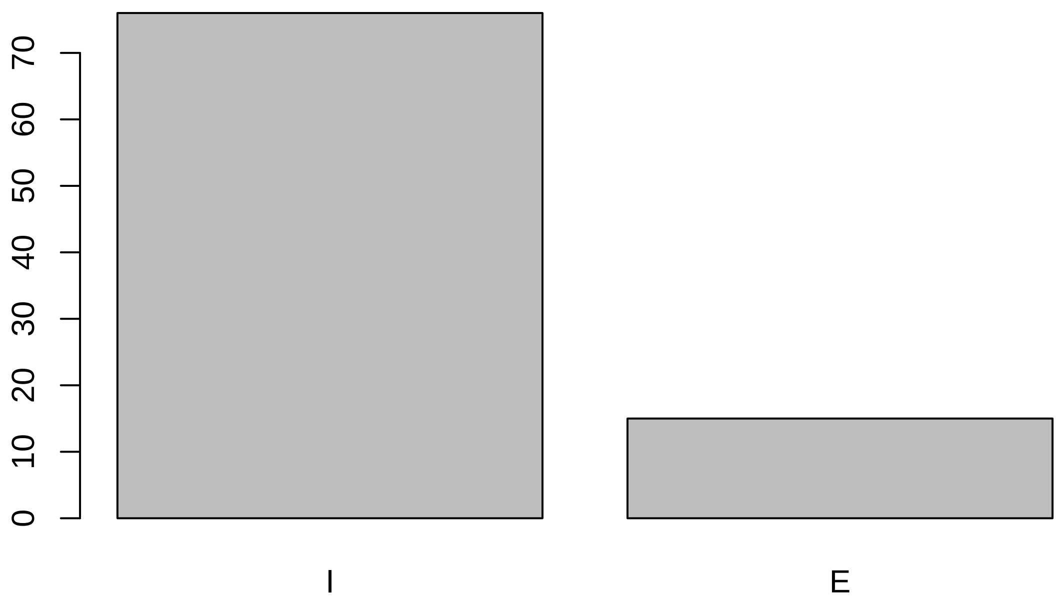
#2. Splitting Data & Configurations —-

```r
#2.1 Splitting Data
target.label = "IE"
set.seed(seed)
train.index = createDataPartition(dataset[[target.label]], p = 0.9, list = F)
trainset = dataset[train.index,]
testset = dataset[-train.index,]

#checking training & testing data formation
trainset[[target.label]] %>% plot()
```




```r
testset[[target.label]] %>% plot() #similar enough
```




```
#2.2 Target Selection & Formula Creation
target = trainset[[target.label]]
features.label = trainset %>% select(-target.label) %>% names() %T>% print()
```

```
##  [1] "gender" "engnat" "Q1A"    "Q2A"    "Q3A"    "Q4A"    "Q5A"    "Q6A"
##  [9] "Q7A"    "Q8A"    "Q9A"    "Q10A"   "Q11A"   "Q12A"   "Q13A"   "Q14A"
## [17] "Q15A"   "Q16A"   "Q17A"   "Q18A"   "Q19A"   "Q20A"   "Q21A"   "Q22A"
## [25] "Q23A"   "Q24A"   "Q25A"   "Q26A"   "Q27A"   "Q28A"   "Q29A"   "Q30A"
## [33] "Q31A"   "Q32A"   "Q33A"   "Q34A"   "Q35A"   "Q36A"   "Q37A"   "Q38A"
## [41] "Q39A"   "Q40A"   "Q41A"   "Q42A"   "Q43A"   "Q44A"   "Q45A"   "Q46A"
## [49] "Q47A"   "Q48A"   "Q49A"   "Q50A"   "Q51A"   "Q52A"   "Q53A"   "Q54A"
## [57] "Q55A"   "Q56A"   "Q57A"   "Q58A"   "Q59A"   "Q60A"   "Q61A"   "Q62A"
## [65] "Q63A"   "Q64A"   "Q65A"   "Q66A"   "Q67A"   "Q68A"   "Q69A"   "Q70A"
## [73] "Q71A"   "Q72A"   "Q73A"   "Q74A"   "Q75A"   "Q76A"   "Q77A"   "Q78A"
## [81] "Q79A"   "Q80A"   "Q81A"   "Q82A"   "Q83A"   "Q84A"   "Q85A"   "Q86A"
## [89] "Q87A"   "Q88A"   "Q89A"   "Q90A"   "Q91A"
```

```
features = trainset %>% select(features.label) %>% as.data.frame()

formula = features %>%
  names() %>%
  paste(., collapse = " + ") %>%
  paste(target.label, "~ ", .) %>%
  as.formula(env = .GlobalEnv) %T>% print
```

```
## IE ~ gender + engnat + Q1A + Q2A + Q3A + Q4A + Q5A + Q6A + Q7A +
```

```
##     Q8A + Q9A + Q10A + Q11A + Q12A + Q13A + Q14A + Q15A + Q16A +
##     Q17A + Q18A + Q19A + Q20A + Q21A + Q22A + Q23A + Q24A + Q25A +
##     Q26A + Q27A + Q28A + Q29A + Q30A + Q31A + Q32A + Q33A + Q34A +
##     Q35A + Q36A + Q37A + Q38A + Q39A + Q40A + Q41A + Q42A + Q43A +
##     Q44A + Q45A + Q46A + Q47A + Q48A + Q49A + Q50A + Q51A + Q52A +
##     Q53A + Q54A + Q55A + Q56A + Q57A + Q58A + Q59A + Q60A + Q61A +
##     Q62A + Q63A + Q64A + Q65A + Q66A + Q67A + Q68A + Q69A + Q70A +
##     Q71A + Q72A + Q73A + Q74A + Q75A + Q76A + Q77A + Q78A + Q79A +
##     Q80A + Q81A + Q82A + Q83A + Q84A + Q85A + Q86A + Q87A + Q88A +
##     Q89A + Q90A + Q91A
```

```r
#2.3 trControl configurations
trControl = trainControl(method = "repeatedcv",
                         number = 5,
                         classProbs = T,
                         search = "random",
                         allowParallel = T)

#2.4 preProc configurations
preProc = c("scale", "center")

#2.5 Metric configuration
metric = "Accuracy"
```

#3. 8 Training Models on trainset —-

```r
# cl = makePSOCKcluster(5)
# registerDoParallel(cl)
#
# #knn
# set.seed(seed)
# fit.knn = train(formula,
#                 data = trainset,
#                 method = "knn",
#                 preProc = preProc, metric = metric)
#
# #logistic regression
# set.seed(seed)
# fit.lg = train(formula,
#                data = trainset,
#                method = "glm",
#                family = "binomial",
#                preProc = preProc, metric = metric)
#
# #gbm
# set.seed(seed)
# fit.gbm = train(formula,
#                 data = trainset,
#                 method = "gbm",
#                 preProc = preProc, metric = metric)
```

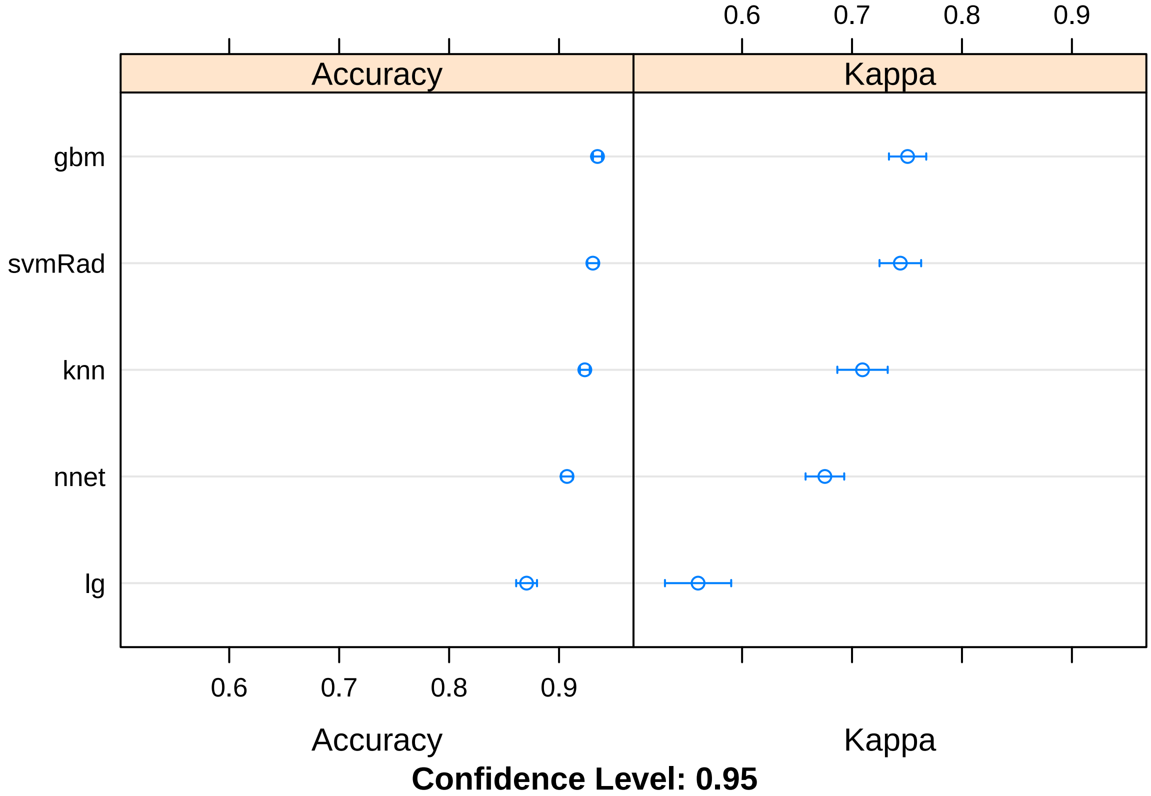
```r
# #svm
# set.seed(seed)
# fit.svm = train(formula,
#                  data = trainset,
#                  method = "svmRadial",
#                  preProc = preProc, metric = metric)
# #nnet
# set.seed(seed)
# fit.nnet = train(formula,
#                   data = trainset,
#                   method = "nnet",
#                   preProc = preProc, metric = metric)
#
# stopCluster(cl)
```

#4. comparing training performance of all 8 models —-

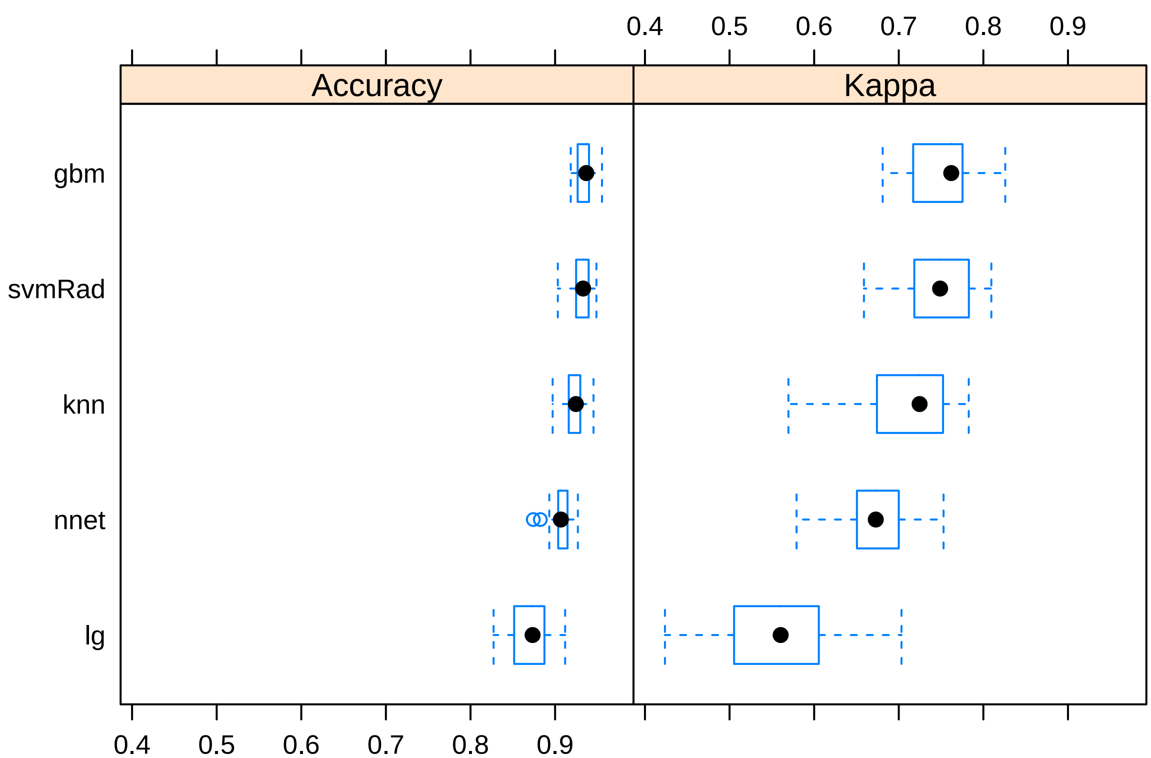
```r
results = resamples(list(knn = fit.knn, lg = fit.lg,
                          gbm = fit.gbm, svmRad = fit.svm, nnet = fit.nnet))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: knn, lg, gbm, svmRad, nnet
## Number of resamples: 25
##
## Accuracy
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn    0.8971061 0.9161290 0.9245902 0.9234471 0.9297659 0.9453925    0
## lg     0.8272425 0.8516129 0.8733766 0.8704822 0.8873720 0.9118644    0
## gbm    0.9184953 0.9266667 0.9368771 0.9349621 0.9400631 0.9554140    0
## svmRad 0.9032258 0.9248366 0.9331104 0.9307510 0.9396825 0.9488055    0
## nnet   0.8741935 0.9036545 0.9068323 0.9072389 0.9146758 0.9269103    0
##
## Kappa
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn    0.5695502 0.6742118 0.7246190 0.7095353 0.7523245 0.7827563    0
## lg     0.4236689 0.5053435 0.5604039 0.5599705 0.6055159 0.7032207    0
## gbm    0.6809584 0.7170308 0.7620514 0.7505340 0.7753738 0.8258413    0
## svmRad 0.6588408 0.7183532 0.7488547 0.7439309 0.7828611 0.8093977    0
## nnet   0.5791855 0.6505571 0.6728487 0.6753122 0.7000074 0.7529266    0
```

```r
dot.plot=dotplot(results) %T>% print()
```




```
bwplot = bwplot(results) %T>% print()
```



```r
# I will tune gbm and svmRad, as they have the highest accuracy and Kappa stats.
```

#5. Tuning 2 models —-

```r
#5.2 gbm
#Tune grid
print(fit.gbm)
```

```
## Stochastic Gradient Boosting
##
## 833 samples
##  93 predictor
##   2 classes: 'I', 'E'
##
## Pre-processing: scaled (94), centered (94)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 833, 833, 833, 833, 833, 833, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9332481  0.7454391
##   1                  100      0.9342377  0.7553013
##   1                  150      0.9319534  0.7485902
##   2                   50      0.9349621  0.7505340
##   2                  100      0.9325655  0.7466645
##   2                  150      0.9314266  0.7431896
##   3                   50      0.9347449  0.7518210
##   3                  100      0.9347071  0.7529483
##   3                  150      0.9349480  0.7554112
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 10.
```

```r
#used n.trees = 50, interaction.depth =2, shrinkage = 0.1 and n.minobsinnode = 10
# getModelInfo("gbm")
# tunegrid.gbm = expand.grid(
#   .n.trees = c(50,100),
#   .interaction.depth = 2,
#   .shrinkage = 0.1,
#   .n.minobsinnode= c(5,10,15)
# )
#Training tuned model
# set.seed(seed)
# cl = makePSOCKcluster(5)
# registerDoParallel(cl)
```

```r
#
# set.seed(seed)
# tune.gbm = train(formula,
#                   data = trainset,
#                   method = "gbm",
#                   metric = metric, preProc = preProc,
#                   trControl = trControl, tuneGrid = tunegrid.gbm)
#
# stopCluster(cl)

#Checking train performance
# print(tune.gbm) #n.trees = 50, interaction.depth = 2, shrinkage = 0.1 and n.minobsinnode =
# tune.gbm %>% getTrainPerf()
# #confusion matrix
# tune.gbm %>% attributes()
tune.gbm %>% confusionMatrix.train() # Accuracy (average) : 0.9424
```

```
## Cross-Validated (5 fold, repeated 1 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    I    E
##          I 81.5  4.4
##          E  1.3 12.7
##
##  Accuracy (average) : 0.9424
```

```r
#5.3 svm
#Tune grid
# print(fit.svm) #used sigma = sigma = 0.006111922 and C = 1
# getModelInfo("svmRadial")
# tunegrid.svm = expand.grid(
#   .sigma = c(0.05 : 0.07),
#   .C = c(0.5, 1, 1.5)
# )
#Training tuned model
# set.seed(seed)
# cl = makePSOCKcluster(5)
# registerDoParallel(cl)
# tune.svm = train(formula,
#                   data = trainset,
#                   method = "svmRadial",
#                   metric = metric,  preProc = preProc,
#                   trControl = trControl, tuneGrid = tunegrid.svm)
# stopCluster(cl)
# #Checking train performance
# print(tune.svm) #sigma = 0.05 and C = 0.5
# tune.svm %>% getTrainPerf()
```

```r
#confusion matrix
tune.svm %>% confusionMatrix.train() # Accuracy (average) : 0.9172
```

```
## Cross-Validated (5 fold, repeated 1 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    I    E
##          I 80.3  5.9
##          E  2.5 11.3
##
##  Accuracy (average) : 0.916
```

#6. Tuned model performance on training set —-

```r
#6.1 confusion matrix (%)
tune.gbm %>% confusionMatrix.train() %>% .$table
```

```
##           Reference
## Prediction         I         E
##          I 81.512605  4.441777
##          E  1.320528 12.725090
```

```r
tune.svm %>% confusionMatrix.train() %>% .$table
```

```
##           Reference
## Prediction         I         E
##          I 80.312125  5.882353
##          E  2.521008 11.284514
```

```r
#6.2 train performance
tune.gbm %>% getTrainPerf()
```

```
##   TrainAccuracy TrainKappa method
## 1     0.9424212  0.7800414    gbm
```

```r
tune.svm %>% getTrainPerf()
```

```
##   TrainAccuracy TrainKappa    method
## 1     0.9160089  0.6803541 svmRadial
```

#7. Tuned model performance on testing set —-

```r
#7.1 confusion matrix (counts)
cm_counts = function(model) {
  a= predict(model, testset)
  b= confusionMatrix(a, testset[[target.label]])
  print(b)
}

test.con.gbm = cm_counts(tune.gbm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  I  E
##          I 73  2
##          E  3 13
##
##                Accuracy : 0.9451
##                  95% CI : (0.8764, 0.9819)
##     No Information Rate : 0.8352
##     P-Value [Acc > NIR] : 0.001453
##
##                   Kappa : 0.8056
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.9605
##             Specificity : 0.8667
##          Pos Pred Value : 0.9733
##          Neg Pred Value : 0.8125
##              Prevalence : 0.8352
##          Detection Rate : 0.8022
##    Detection Prevalence : 0.8242
##       Balanced Accuracy : 0.9136
##
##        'Positive' Class : I
##
```

```r
test.con.svm = cm_counts(tune.svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  I  E
##          I 73  5
##          E  3 10
##
##                Accuracy : 0.9121
##                  95% CI : (0.8341, 0.9613)
##     No Information Rate : 0.8352
```

```
##     P-Value [Acc > NIR] : 0.02652
##
##                   Kappa : 0.6627
##
##  Mcnemar's Test P-Value : 0.72367
##
##             Sensitivity : 0.9605
##             Specificity : 0.6667
##          Pos Pred Value : 0.9359
##          Neg Pred Value : 0.7692
##              Prevalence : 0.8352
##          Detection Rate : 0.8022
##    Detection Prevalence : 0.8571
##       Balanced Accuracy : 0.8136
##
##        'Positive' Class : I
##
```

```r
test.con.gbm %>% .$table
```

```
##           Reference
## Prediction  I  E
##          I 73  2
##          E  3 13
```

```r
test.con.svm %>% .$table
```

```
##           Reference
## Prediction  I  E
##          I 73  5
##          E  3 10
```

```r
#7.2 Confusion Matrix (%)
cm_perc = function(cm) {
  (prop.table(cm$table))*100
}

cm_perc(test.con.gbm)
```

```
##           Reference
## Prediction         I         E
##          I 80.219780  2.197802
##          E  3.296703 14.285714
```

```r
cm_perc(test.con.svm)
```

```
##           Reference
## Prediction         I         E
##          I 80.219780  5.494505
##          E  3.296703 10.989011
```

#8. Comparing trainset performance & testset performance —-

```r
#8.1 Trainset performance
trainperf = function(model) {
  a=select(getTrainPerf(model), -c(TrainKappa,method))
  print(a)
}

train.gbm = trainperf(tune.gbm)
```

```
##   TrainAccuracy
## 1     0.9424212
```

```r
train.svm = trainperf(tune.svm)
```

```
##   TrainAccuracy
## 1     0.9160089
```

```r
#8.2 Testset performance
testperf  = function(cm) {
  print(cm$overall[c("Accuracy")])
}

test.gbm = testperf(test.con.gbm)
```

```
##  Accuracy
## 0.9450549
```

```r
test.svm = testperf(test.con.svm)
```

```
##  Accuracy
## 0.9120879
```

```r
result.table = bind_rows(
  c(train.gbm, test.gbm),
  c(train.svm, test.svm)
) %>% data.frame() %>%
  set_rownames(c("gbm", "svm")) %>%
  set_colnames(c("accuracy.train", "accuracy.test"))
```

#9. Final Analysis —-

```r
print(result.table)
```

```
##     accuracy.train accuracy.test
## gbm      0.9424212     0.9450549
## svm      0.9160089     0.9120879
```

#9.1 Analysis on finding the best model #In this analysis, we are only going to take accuracy into account for determining the best model. #From intial models prior to tuning, gbm and svmRad performed the best. However, after tuning the #two models, gbm performed better than svm both on training and testing sets.

#9.2 Analysis on the data

```r
test.con.gbm %>% .$table
```

```
##           Reference
## Prediction  I  E
##          I 73  2
##          E  3 13
```

```r
cm_perc(test.con.gbm)
```

```
##           Reference
## Prediction         I         E
##          I 80.219780  2.197802
##          E  3.296703 14.285714
```

#What percentage of introverts from all introverts are correctly predicted? #(73/76)*100 = 96.05263 (%) #What percentage of extroverts from all extroverts are incorrectly predicted? # (2/15)*100 = 13.33333 (%)