compgen2021: Week 3 exercises

Irem YUCEL

Exercises for Week 3

Classification

For this set of exercises we will be using the gene expression and patient annotation data from the glioblastoma patient. You can read the data as shown below:

```
library(compGenomRData)
library("caretEnsemble")
library("caret")
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:caretEnsemble':
##
##     autoplot
```

```
## Loading required package: lattice
```

```
library("mlbench")
library("pROC")
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# get file paths
fileLGGexp=system.file("extdata",
                       "LGGrnaseq.rds",
                       package="compGenomRData")
fileLGGann=system.file("extdata",
                       "patient2LGGsubtypes.rds",
```

```
                      package="compGenomRData")
# gene expression values
gexp=readRDS(fileLGGexp)

# patient annotation
patient=readRDS(fileLGGann)
```

1. Our first task is to not use any data transformation and do classification. Run the k-NN classifier on the data without any transformation or scaling. What is the effect on classification accuracy for k-NN predicting the CIMP and noCIMP status of the patient? [Difficulty: **Beginner**]

The accuracy increased after scaling and transforming the data:

```
# get file paths
fileLGGexp=system.file("extdata",
                       "LGGrnaseq.rds",
                       package="compGenomRData")
fileLGGann=system.file("extdata",
                       "patient2LGGsubtypes.rds",
                       package="compGenomRData")
# gene expression values
gexp=readRDS(fileLGGexp)

# patient annotation
patient=readRDS(fileLGGann)

#head(gexp)
#Need to transpose matrix
tgexp <- t(gexp)
dim(tgexp)
```

```
## [1]   184 20501
```

```
# I use the most variable 5000 genes.
SDs=apply(tgexp,2,sd )
topPreds=order(SDs,decreasing = TRUE)[1:5000]
tgexp=tgexp[,topPreds]

# merge for class groups
tgexp=merge(patient,tgexp,by="row.names")

# push sample ids back to the row names
rownames(tgexp)=tgexp[,1]
tgexp=tgexp[,-1]


# train the k-NN model on the data without any transformation or scaling
intrain <- createDataPartition(y = tgexp[,1], p= 0.7)[[1]]

# separate test and training sets
training <- tgexp[intrain,]
testing <- tgexp[-intrain,]
```

```
library(caret)
knnFit=knn3(x=training[,-1], # training set
            y=training[,1], # training set class labels
            k=5)

testPred=predict(knnFit,testing[,-1],type="class")
confusionMatrix(data=testing[,1],reference=testPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     20      7
##     noCIMP    7     20
##
##                Accuracy : 0.7407
##                  95% CI : (0.6035, 0.8504)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.0002677
##
##                   Kappa : 0.4815
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.7407
##             Specificity : 0.7407
##          Pos Pred Value : 0.7407
##          Neg Pred Value : 0.7407
##              Prevalence : 0.5000
##          Detection Rate : 0.3704
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7407
##
##        'Positive' Class : CIMP
##
```

```
#Apply the same on the transformed and scaled data
trans_gexp=t(log10(gexp+1))
st_gexp=scale(trans_gexp)

SDs_scaled=apply(st_gexp,2,sd )
topPreds_st=order(SDs_scaled,decreasing = TRUE)[1:5000]
st_gexp=st_gexp[,topPreds_st]

# merge for class groups
st_gexp=merge(patient,st_gexp,by="row.names")

# push sample ids back to the row names
rownames(st_gexp)=st_gexp[,1]
st_gexp=st_gexp[,-1]

st_intrain <- createDataPartition(y = st_gexp[,1], p= 0.7)[[1]]
```

```r
# separate test and training sets
st_training <- st_gexp[intrain,]
st_testing <- st_gexp[-intrain,]

st_knnFit=knn3(x=st_training[,-1], # training set
            y=st_training[,1], # training set class labels
            k=5)

st_testPred=predict(st_knnFit,st_testing[,-1],type="class")
confusionMatrix(data=st_testing[,1],reference=st_testPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIMP noCIMP
##     CIMP     25      2
##     noCIMP    3     24
##
##                Accuracy : 0.9074
##                  95% CI : (0.797, 0.9692)
##     No Information Rate : 0.5185
##     P-Value [Acc > NIR] : 9.651e-10
##
##                   Kappa : 0.8148
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8929
##             Specificity : 0.9231
##          Pos Pred Value : 0.9259
##          Neg Pred Value : 0.8889
##              Prevalence : 0.5185
##          Detection Rate : 0.4630
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9080
##
##        'Positive' Class : CIMP
##
```

2. Bootstrap resampling can be used to measure the variability of the prediction error. Use bootstrap resampling with k-NN for the prediction accuracy. How different is it from cross-validation for different ks? [Difficulty: **Intermediate**]

The results are different for bootstrapping and cross validation but they are very similar

```r
# 5-fold cross validation
trctrl <- trainControl(method = "cv",number=5)

#bootstrap:
trctrl_boot <- trainControl(method = "boot", number=20, returnResamp="all")

# we will now train k-NN model
```

```
#bootstrap
knnBoot <- train(subtype~.,
                 data = st_gexp,
                 method = "knn",
                trControl=trctrl_boot,
                tuneGrid = data.frame(k=1:8))
#cv
knn_cv <- train(subtype~.,
                data = st_gexp,
                 method = "knn",
                 trControl=trctrl,
                 tuneGrid = data.frame(k=1:8))

knnBoot$results
```

```
##   k  Accuracy     Kappa AccuracySD    KappaSD
## 1 1 0.9075187 0.8127776 0.02053021 0.04131751
## 2 2 0.9002070 0.7978352 0.02682655 0.05356707
## 3 3 0.8949595 0.7870442 0.02140994 0.04327270
## 4 4 0.8873146 0.7717796 0.02305525 0.04573102
## 5 5 0.8887190 0.7748057 0.02847809 0.05640376
## 6 6 0.8842539 0.7659097 0.03548405 0.07057184
## 7 7 0.8854471 0.7680840 0.03021846 0.06010015
## 8 8 0.8844205 0.7661759 0.03676356 0.07275225
```

```
knn_cv$results
```

```
##   k  Accuracy     Kappa AccuracySD    KappaSD
## 1 1 0.9021021 0.8043128 0.05936535 0.11811638
## 2 2 0.8968468 0.7939766 0.05176262 0.10288532
## 3 3 0.8965465 0.7931388 0.04900166 0.09767266
## 4 4 0.8966967 0.7933836 0.03537940 0.07022570
## 5 5 0.8963964 0.7928623 0.06558658 0.13096111
## 6 6 0.8855856 0.7710678 0.05306910 0.10602801
## 7 7 0.8801802 0.7603890 0.05694235 0.11356824
## 8 8 0.8803303 0.7608865 0.04943453 0.09806956
```

3. There are a number of ways to get variable importance for a classification problem. Run random forests on the classification problem above. Compare the variable importance metrics from random forest and the one obtained from DALEX applied on the random forests model. How many variables are the same in the top 10? [Difficulty: **Advanced**]

solution:

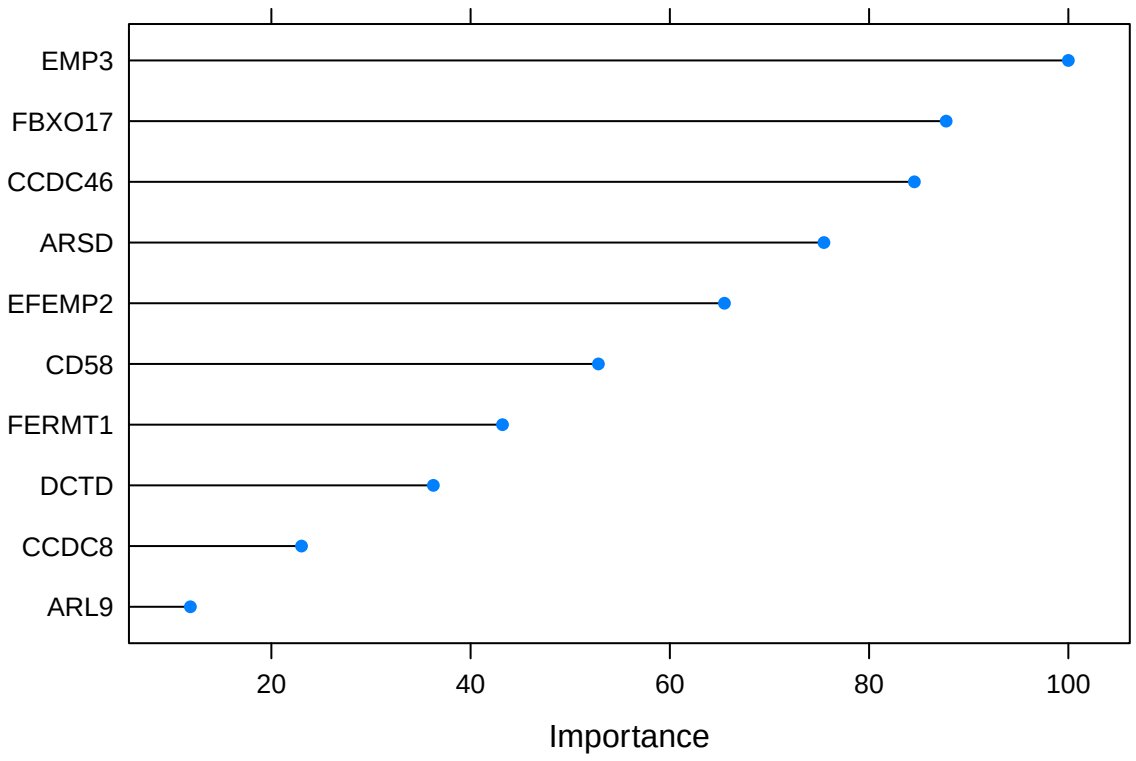
```
trctrl <- trainControl(method = "cv",number=5,classProb=TRUE)

rfFit <- train(subtype~.,
               data = st_gexp,
               method = "ranger",
               trControl=trctrl,
               importance="impurity")
library(DALEX)
```

```
## Welcome to DALEX (version: 2.4.2).
## Find examples and detailed introduction at: http://ema.drwhy.ai/
```

```
explainer_rf<- DALEX::explain(rfFit,
                              label="random forest",
                              data =st_gexp[,-1],
                              y = st_gexp[,1]=="CIMP")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  random forest
##   -> data              :  184  rows  5000  cols
##   -> target variable   :  184  values
##   -> predict function  :  yhat.train  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package caret , ver. 6.0.93 , task classification (  default  )
##   -> model_info        :  Model info detected classification task but 'y' is a logical . Converted t
##   -> predicted values  :  numerical, min =  0 , mean =  0.4963587 , max =  1
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -1 , mean =  0.003641304 , max =  1
##   A new explainer has been created!
```

```
#The next line took too long to execute.
#virf=model_parts(explainer_rf,B = 5,type="ratio")
plot(varImp(rfFit), top=10)
```




```
#plot(virf, max_vars=10)
```

4. Come up with a unified importance score by normalizing importance scores from random forests and DALEX, followed by taking the average of those scores. [Difficulty: **Advanced**]

solution:

Regression

For this set of problems we will use the regression data set where we tried to predict the age of the sample from the methylation values. The data can be loaded as shown below:

```
# file path for CpG methylation and age
fileMethAge=system.file("extdata",
                        "CpGmeth2Age.rds",
                        package="compGenomRData")

# read methylation-age table
ameth=readRDS(fileMethAge)
```

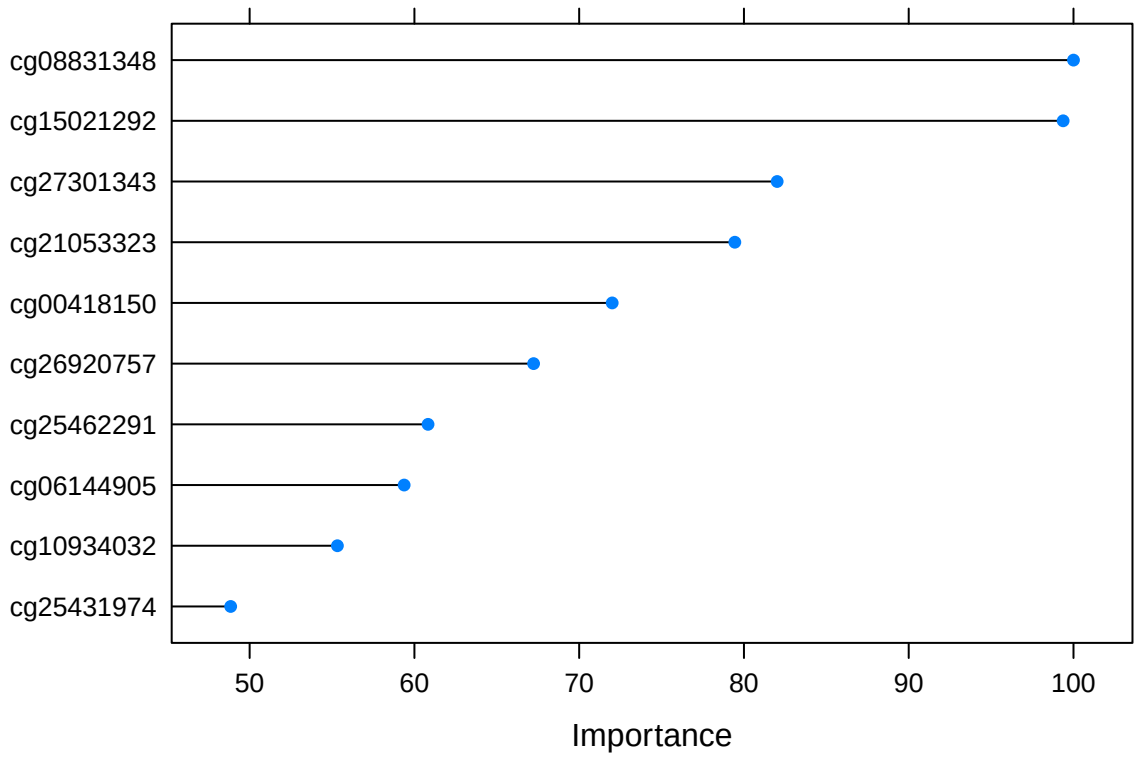
1. Run random forest regression and plot the importance metrics. [Difficulty: **Beginner**]

solution:

```
# filter based on variance
ameth=ameth[,c(TRUE,matrixStats::colSds(as.matrix(ameth[,-1]))>0.1)]
dim(ameth)
```

```
## [1]  108 2290
```

```
# get indices for 80% of the data set
intrain <- createDataPartition(y = ameth[,1], p= 0.7)[[1]]

training <- ameth[intrain,]
testing <- ameth[-intrain,]

trctrl <- trainControl(method = "none" )

rfregFit <- train(Age~.,
                  data = training,
                  method = "ranger",
                  trControl=trctrl,
                  importance="permutation",
                  tuneGrid = data.frame(mtry=50,
                                        min.node.size = 5,                                splitr
)

testPred=predict(rfregFit,testing[,-1])

# plot variable importance for top 10 variables
plot(varImp(rfregFit),top=10)
```




```
rfregFit$finalModel$r.squared
```

```
## [1] 0.8244249
```

2. Split 20% of the methylation-age data as test data and run elastic net regression on the training portion to tune parameters and test it on the test portion. [Difficulty: **Intermediate**]

solution:

```
trctrl <- trainControl(method = "cv",number=5)

gbFit <- train(Age~., data = training,
                  method = "xgbTree",
                  trControl=trctrl,
                  tuneGrid = data.frame(nrounds=200,
                                        eta=c(0.05,0.1,0.3),
                                        max_depth=4,
                                        gamma=0,
                                        colsample_bytree=1,
                                        subsample=0.5,
                                        min_child_weight=1))

gbFit$bestTune
```

```
##   nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 1     200         4 0.05     0                1                1       0.5
```

```
testPred=predict(gbFit,testing[,-1])
```

3. Run an ensemble model for regression using the **caretEnsemble** or **mlr** package and compare the results with the elastic net and random forest model. Did the test accuracy increase? **HINT:** You need to install these extra packages and learn how to use them in the context of ensemble models. [Difficulty: **Advanced**]

solution:

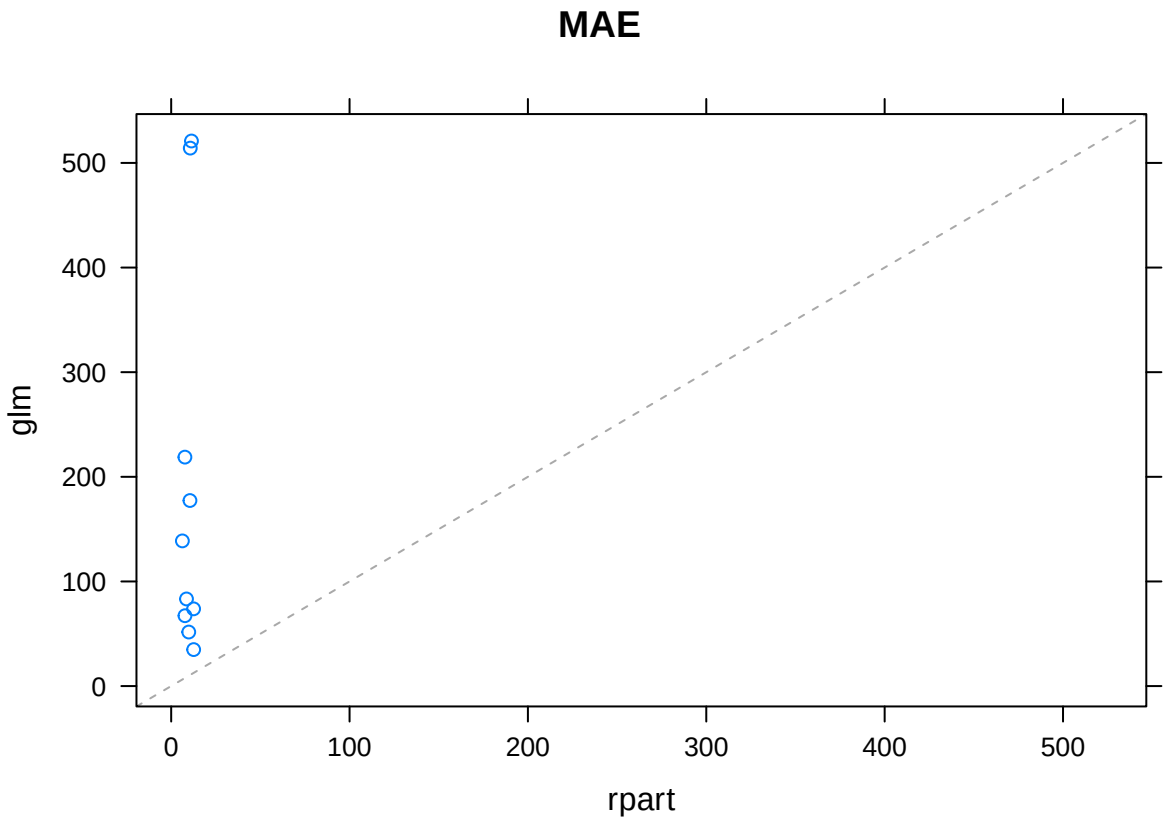
```
training$Age = as.numeric(training$Age)

my_control <- trainControl(
  method="cv",
  number=10,
  savePredictions="final",
  index=createResample(training$Age, 10),
  summaryFunction=defaultSummary
  )

model_list <- caretList(
  Age~.,
  data=training,
  trControl=my_control,
  methodList=c("glm", "rpart")
  )
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
xyplot(resamples(model_list))
```




```
ens <- caretEnsemble(model_list)
summary(ens)
```

```
## The following models were ensembled: glm, rpart
## They were weighted:
## 4.6803 -0.0046 0.8457
## The resulting RMSE is: 16.0476
## The fit for each individual model on the RMSE is:
##  method      RMSE     RMSESD
##     glm 244.90475 219.214567
##   rpart  15.27114   3.740233
```