Final Report Statistics for Biology 2

Random Forest and SVM for Cancer Classification

Maximilian Joas

2020-04-03

Contents

Data Loading and Preparation

I will not go into detail in this part, since it is basiacally the same code that we did together in class. I will only comment on things that I have changed in comparison to the work in class. The main difference in this section is that I registered a prallel backend for multithreading and that I stored the class Variable "sample.labels" in an extra variable to make the code more reusable in case we use a different dataset where the class variable has another name, we just need to change this variable once.

```r
library(knitr)
library(gdata)
library(ranger)
library(caret)
library(foreach)
library(doMC)
library(e1071)
library(pROC)

## register a paralell backend and define number of Threads
## Note the the numberThreads variable gets passed later to the ranger function
## this is neccessary, because ranger uses all available threads as a default and I
## did not intend to overload the cluster.
numberThreads <- 4
doMC::registerDoMC(8)
```

```r
####  Define Local directories and files ####
## Find the home directory
myHome <- Sys.getenv("HOME")

## Define the main directory for the  data and results
mainDir <- file.path(myHome, "uni/4_sem/biostatistics/report")
dir.create(path = mainDir, showWarnings = FALSE)
message("Main dir: ", mainDir)

## Define a file where we wills store the memory image
memImageFile <- file.path(mainDir, "DenBoerData_loaded.Rdata")

## Define the dir where we will download the data
destDir <- file.path(mainDir, "data")
message("Local data dir: ", destDir)

## Define a file where we will store all results
resultDir <- file.path(mainDir, "resultDir")
message("Result direcotry", resultDir)


#### Define variables for the location and name of the input files ####
## URL fo the folder containing the data fle
## Maybe refactor and pass via command line arguments later
dataURL <- "https://github.com/jvanheld/stat1/raw/master/data/DenBoer_2009/"

files <- c(
  expr = "GSE13425_Norm_Whole.tsv.gz",
  pheno = "phenoData_GSE13425.tsv.gz",
  groups = "GSE13425_group_descriptions.tsv.gz"
)



## Function that downloads input files in case they don"t exist locally yet
DownloadMyFiles <- function(files,
                            dataURL,
                            destDir) {

  ## Create local directory
  dir.create(destDir, recursive = TRUE, showWarnings = FALSE)

  for (f in files) {

    ## Destination file
    destFile <- file.path(destDir, f)

    ## Check if file exists
    if (file.exists(destFile)) {
      message("skipping download because file exists: \n", destFile)
    } else {
      sourceURL <- file.path(dataURL, f)
      download.file(url = sourceURL, destfile = destFile)
    }
  }
}




## Call the function to download the files
DownloadMyFiles(files = files, dataURL = dataURL, destDir = destDir)

kable(data.frame(list.files(destDir)),
      caption = "Content of the destination directory after file download. ")



## ---------------------------------------------------------------------
#### Load data tables ####

## Load expression table
exprTable <- read.table(file.path(destDir, files["expr"]),
                        sep = "\t",
                        header = TRUE,
                        quote = "",
                        row.names = 1)
dim(exprTable)
head(exprTable)

## Load metadescriptions (pheno table)
phenoTable <- read.table(file.path(destDir, files["pheno"]),
                         sep = "\t",
                         header = TRUE,
                         quote = "",
                         row.names = 1
                         )
head(phenoTable)

## for reusability provide the name of the class variable for this dataset
classVariable <- "sample.labels"

## Load group descriptions
groupDescriptions <- read.table(file.path(destDir, files["groups"]),
                                sep = "\t",
                                header = TRUE,
                                quote = "",
                                row.names = 1)
dim(groupDescriptions)
print(groupDescriptions)
kable(groupDescriptions)
```

Prepare Data specifically for the classification task

Since I am working with two data tables, one with the gene expression data and one with the pheno data, I check if the patients have the same order in each of the tables. This should be the case, but a mistake here would be crucial. I also transposed the expression table to have all the patient as rows and the gene expression values in columns. This format is required by many packages for classification.

```r
#### Prepare data for classification task ####
## check if pheno data and expression data have same order of patients
check <- colnames(exprTable) == rownames(phenoTable)
message("Do Pheno and Expressiondata have the same order of patients")
message(all(check = TRUE))

## Transposing the exprTable do that it has the right format for the package
exprTableTransposed <- t(exprTable)

## this prevents an error in the ranger function due to ilegal names
colnames(exprTableTransposed) <- make.names(colnames(exprTableTransposed))
```

Functions for Feature Selection and Classification

I order to have as many training and test patients as possible I decided to use leave one out cross validation as an approach. The standard way to do this would be to use the caret packages. I decided, however, to implement the functionality myself. The reason for this is that I also wanted to use the LOOCV approach for feature selection to avoid any bias. With my own implementation I could be sure that I used the exact same method for feature selection and the training of the model. For my Implementation I used the foreach package for parallelization, which is highly optimized. Thus, my implementation is also highly performant.

My functions are structured as follows: I have one function "SelectFeaturesWithWu

```r
FilterClass <- function(minimumNumberofCases){
  releventClasses <- names(which(table(
                  phenoTable[, classVariable]) > minimumNumberofCases))
   helper <- phenoTable[, classVariable] %in% releventClasses
   phenoTable <- phenoTable[helper, ]
   exprTableTransposed <- exprTableTransposed[helper, ]
}
#### Feature Selection ####
## Function that selects genes based on the random forest variable importance
## It takes a vector as index variable that indicates which patients should be used
SelectFeaturesWithRandomForest <- function(trainIndex,
                                           numFeatures,
                                           verbose = TRUE
                                           ) {
    if(verbose) message("Fitting a Random Forest for feature selection")
    fit <- ranger(y = phenoTable[trainIndex, classVariable],
                  x = exprTableTransposed[trainIndex,],
                  importance = "impurity",
                  mtry = tunedMtry,
                  num.threads = numberThreads,
                  )

    if(verbose) message("Finished fitting, now extracting variable importance")
    varImportance <- fit$variable.importance
    selectedGenes <- sort(varImportance, na.last = TRUE, decreasing = TRUE)
    return(selectedGenes[1:numFeatures])
}



## Selection with leave one out cross validation
SelectRandomForestLOOCV <- function(numFeatures,
                                    verbose = FALSE) {
    res <- foreach(i = 1:nrow(exprTableTransposed)) %dopar% {
            curVariables <- SelectFeaturesWithRandomForest(-i,
                                                           numFeatures,
                                                           verbose)
            return(curVariables)

    }
    names(res) <- rownames(phenoTable)
    return(res)
}



## Function that classifies patients with random forest
RandomForestClassifier <- function(numberTrees,
                                   testIndex,
                                   trainIndex,
                                   selectedCovariates,
                                   verbose = TRUE) {
    if(verbose) message("Fitting the Random Forest")
    if(verbose) message(paste0("Using ",numberTrees," trees"))
    rf.fit <- ranger(y = phenoTable[trainIndex, classVariable],
                     x = exprTableTransposed[trainIndex, selectedCovariates],
                     num.trees = numberTrees,
                     num.threads = numberThreads,
                     mtry = tunedMtry)

    testData <- t(as.data.frame(exprTableTransposed[testIndex,
                                selectedCovariates]))
    if(length(testIndex) !=1) {
        testData <- exprTableTransposed[testIndex, selectedCovariates]
        }
    if(verbose) message("Starting prediction based on the fitted model")
    predicted <- predict(rf.fit, testData)
    if(verbose) message("Finished predicting, now returning the predictions")
    predicted$predictions
    return(predicted$predictions)
}


## Function that classifies patients with SVM
SvmClassifier <- function(myKernel,
                          testIndex,
                          trainIndex,
                          selectedCovariates,
                          verbose = TRUE) {
    if(verbose) message("Starting fitting a SVM Mode")
    svm.fit <- svm(y = phenoTable[trainIndex, classVariable],
                   x = exprTableTransposed[trainIndex, selectedCovariates],
                   kernel = myKernel,
                   gamma = 0.1,
                   cost = 10,
                   type = "C-classification")
    # neccessary in case the test index has length one (loocv)
    testData <- t(as.data.frame(exprTableTransposed[testIndex,
                                                    selectedCovariates]))
    if(length(testIndex) !=1) {
        testData <- exprTableTransposed[testIndex, selectedCovariates]
    }
    predicted <- predict(svm.fit, newdata = testData)
    return(predicted)
}



#### Function for LOOCV ####
## -------------------------------------------------------------------

LOOCV <- function(FUN,
                  parameter,
                  selection,
                  verbose = FALSE) {
    res <- foreach(i = 1:nrow(exprTableTransposed)) %dopar% {
        curVariables <- names(selection[
                                rownames(exprTableTransposed)[i]][[1]])
        trainIndex <- c(1:nrow(exprTableTransposed))[-i]
        if(verbose) message("Fitting Loocv")
        res <- FUN(parameter,
                   i, -i,
                   curVariables,
                   verbose)
        res <- droplevels(res)
        names(res) <- NULL
        return(res)
    }
    if(verbose) message('returning')
    names(res) <- rownames(exprTableTransposed)
    return(unlist(res))
}
```

Execution of code

TODO

“‘{r} execution_mtry, eval=FALSE} #### EXECUTION #### ## ——————————————————————

first we tune the mtry parameter of the random forest model with cared

10 folds repeat 3 times

at first we need to perform class filtering

FilterClass(30) control <- trainControl(method = ‘repeatedcv’, number = 10, repeats = 3, search = ‘random’) set.seed(123) rf.random <- train(y = phenoTable[,classVariable], x = exprTableTransposed, method = “ranger”, metric = “Accuracy”, trControl = control) tunedMtry <- rf.random$finalModel$mtry tunedMtryAllFeatures <- rf.random$finalModel$mtry

“‘

Feature Selection Execution

TODO

```r
## now we select how many covariates we want to select
x <- SelectFeaturesWithRandomForest(c(1:nrow(exprTableTransposed)),
                                    nrow(exprTableTransposed))
plot(sort(x, na.last = TRUE, decreasing = TRUE))
lo <- loess(x ~ c(1:nrow(exprTableTransposed)))
out = predict(lo)
secondDer <- diff(diff(out))
maximalChangePoint <- max(secondDer)
maximalChangeIndex <- match(maximalChangePoint, secondDer)
pointHelper <- (1:nrow(phenoTable) == maximalChangeIndex)
lines(out, col = 'red', lwd = 2)
abline(v =maximalChangeIndex)
points(out[pointHelper], x = maximalChangeIndex, col = "red", pch = 22, cex = 2)
numberFeatures <- maximalChangeIndex



## finally we can perform the acutal selection
loocvSelections <- SelectRandomForestLOOCV(numberFeatures)
write.csv(loocvSelections, 'loocvSelections.csv')
```

Tune Mtry after the Feature Selection

TOD `{r} tuned_mtry2, eval= FALSE} control <- trainControl(method = 'repeatedcv',`
`= 10,                         repeats = 3,                         search = 'random')`
`set.seed(123) rf_random <- train(y = phenoTable[,classVariable],                   x =`
`exprTableTransposed[,names(x)],                    method = "ranger",                    metric`
`= "Accuracy",                    trControl = control) tunedMtryFeatureSelection <- rf_random$finalModel`

Classification Execution

TODO

```
numberOfTrees <- c(200, 500, 1000)
kernels <- c("radial", "linear", "polynomial", "sigmoid")
allGenesSelected <- rep(list(x), nrow(exprTableTransposed))

names(allGenesSelected) <- rownames(exprTableTransposed)
selections <- list(allGenes = allGenesSelected,
                   rfSelection = loocvSelections)
resultList <- foreach(selection = names(selections), .combine = "c") %do% {
    if(selection == "rfSelection") tunedMtry = tunedMtryFeatureSelection else tunedMtry = ceiling(sqrt(
    message(tunedMtry)
    selData <- selections[[selection]]
    rf.comb <- foreach(numTree = numberOfTrees, .combine = "c") %do% {
        rf.loocv <- LOOCV(RandomForestClassifier, numTree, selData)
        helperLoocvFile <- paste0("rf_loocv_Selection_", selection,
                                  "_numTrees_", numTree, ".csv")
        curLoocvFile <- file.path(resultDir, helperLoocvFile)
        write.csv(rf.loocv, curLoocvFile)
        res <- list(numTree = rf.loocv)
        names(res) <- paste0(numTree)
        return(res)
    }
    svm.comb <- foreach (kern = kernels, .combine = "c") %do% {
        svm.loocv <- LOOCV(SvmClassifier, kern, selData)
        helperLoocvFile <- paste0("SVM_loocv_Selection_",
                                  selection, "_kernel_", kern, ".csv")
        curLoocvFile <- file.path(resultDir, helperLoocvFile)
        write.csv(svm.loocv, curLoocvFile)
        res <- list(kern = svm.loocv)
        names(res) <- paste0(kern)
        return(res)
    }
    svm.name <- paste0("svm ", selection)
    rf.name <- paste0("rf ", selection)
    res <- c(rfRes = rf.comb, svmRes = svm.comb)
    names(res) <- paste0(selection, names(res))
    return(res)
}
```

Evaluation

TODO

```
response <- phenoTable[, classVariable]
response <- factor(response, levels = rev(levels(response)))
names(response) <- rownames(phenoTable)
evaluationResults <- foreach(res = resultList, .combine = ’rbind’) %do% {
  check <- names(response) == names(res)
  message("Patients in Same Order:")
  print(all(check,TRUE))

  tab <- table(res, response)
  misclError <- 1-sum(diag(tab))/sum(tab)
  ci.upper <- misclError + 1.96 * sqrt( (misclError * (1 - misclError)) / 190)
  ci.lower <- misclError - 1.96 * sqrt( (misclError * (1 - misclError)) / 190)
  ci.interval <- paste0(round(ci.lower,4), "-", round(ci.upper,4))
  message("Misclassifiction Error for current predictions:")
  message(round(misclError,4))
  return(data.frame(MisclassificationError = round(misclError,4), CI = ci.interval))
}

## make resulttable more beautiful

helper <- evaluationResults[8:14,]
parameters <- c("RF 200 Trees",
                "RF 500 Trees",
                "RF 1000 Trees",
                "SVM Radial Kernel",
                "SVM Linear Kernel",
                "SVM Ploynomial Kernel",
                "SVM Sigmoid Kernel")

evaluationResults <-  evaluationResults[-(8:14),]
evaluationResults <- cbind(evaluationResults, helper)
rownames(evaluationResults) <- parameters
colnames(evaluationResults) <- c("All Genes",
                                 "95% CI",
                                 "48 Selected Genes",
                                 "95% CI")

kable(evaluationResults)
```

message("Saving Image file") message(paste(memImageFile)) save.image(memImageFile);

Interpretation

TODO