SDS example workflow

October 30, 2019

Here we present an example to use the package using a the public available TCGA lung cancer data set (TCGA-LUAD).

Prerequisites

R libraries

```
library(survival)
#install.packages("glmnet")
library(glmnet)
#if (!requireNamespace("BiocManager", quietly = TRUE))
#  install.packages("BiocManager")
#BiocManager::install("survcomp", version = "3.8")
library(survcomp)
library(matrixStats)
library(SDS)
```

Data sets (order of subjects have to match in all 3 datasets)

- survival data (data.frame) in form of: OVERALL.SURVIVAL, overall.survival.indicator
- gene expression data (matrix) already filtered and normalized
- clinical data (data.frame) with dummy variables for categorical variables

```
filteredExp_lung = readRDS(file = paste0(pathL,"filteredExp_lung"))
filteredClin_lung = readRDS(file = paste0(pathL,"filteredClin_dummy_lung"))
surv_lung = readRDS(file = paste0(pathL,"survival_lung"))
```

Variables

Here we set up variables to run the code later. It is avaisable to split the data into training and testing data set and validate the builded model later on the testing data. Here we split them up 2:1 for training:testing

```
corrT = 0.6 # threshold for pairwise correlation
top = 75 # number of selected best performing attributes
weight = 1.3 # weight for scoreC in combination with scoreS

set.seed(10) # setting a random seed
trainingSize = floor((dim(surv_lung)[1]/3)*2) # number of subjects in training set
training <- sample(rownames(surv_lung), size = trainingSize) # subject names in training set
test <- rownames(surv_lung)[!(rownames(surv_lung) %in% training)] # subject names in test set

# survival object for training set
survObjectTraining <- Surv(surv_lung[training,]$OVERALL.SURVIVAL,
                           surv_lung[training,]$overall.survival.indicator)
```

Feature Selection

Deriving the scores for feature selection

The survival distance score (scoreS) and the clinical distance score (scoreC) are both build on the training set.

```r
scoreS = Score_S_Function(attrib = filteredExp_lung[,training],
                          surv = surv_lung[training,])
scoreC = Score_C_Function(attrib = filteredExp_lung[,training],
                          clin = filteredClin_lung[training,])

scoreS[1:5]
```

```
##  ENSG00000000005.5  ENSG00000001084.9 ENSG00000001626.13
##           25854.31           25654.90           26073.02
## ENSG00000002079.11  ENSG00000002587.8
##           27176.07           27662.63
```

```r
scoreC[1:5]
```

```
##  ENSG00000000005.5  ENSG00000001084.9 ENSG00000001626.13
##          0.9788137          0.9754445          0.9710743
## ENSG00000002079.11  ENSG00000002587.8
##          0.9280503          0.9957948
```

picking the best performing attributes

```r
score_combo = scoreCombination(scoreS = scoreS, scoreC = scoreC, weight = weight)
topGenes = names(sort(score_combo, decreasing = TRUE))[1:top]
score_combo[1:5]
```

```
##  ENSG00000000005.5  ENSG00000001084.9 ENSG00000001626.13
##         0.25512393         0.08017585         0.08272691
## ENSG00000002079.11  ENSG00000002587.8
##        -0.91386724         1.40925570
```

Feature Reduction

To do the feature selection we reduce the attribute matrix to only the training or testing set and the top selected genes. The returned list entails the meta feature for the training set and the test set.

```r
at_train = t(filteredExp_lung[topGenes,training])
at_test = t(filteredExp_lung[topGenes,test])

meta_features = corrGroupFilter(attrib_training = at_train, attrib_test = at_test,
                                corrT = corrT, top = top, scores = score_combo)
meta_features$training[1:5,1:5]
```

```
##                   [,1]      [,2]      [,3]      [,4]      [,5]
## TCGA-NJ-A4YG -5.419433 -7.004395 -7.004395 -7.004395 -2.249508
## TCGA-55-8614 -4.618377 -5.355342 -6.940305 -6.940305 -4.618377
## TCGA-49-AARR -3.830739 -3.830739 -7.290170 -5.705208 -2.160887
## TCGA-55-6543 -3.155191 -5.270668 -6.855631 -6.855631 -4.533703
## TCGA-L9-A8F4 -1.456992 -5.363882 -5.363882 -6.100848 -2.328258
```

```r
meta_features$test[1:5,1:5]
```

```
##                   [,1]      [,2]      [,3]      [,4]      [,5]
## TCGA-44-6147 -5.815183 -2.645258 -4.592791 -3.007828 -3.493255
## TCGA-55-7283 -4.747766 -3.899769 -7.069694 -7.069694 -5.484732
## TCGA-86-A456 -1.757029 -4.307226 -7.114581 -3.944656 -4.307226
## TCGA-95-7039 -6.605204 -4.283276 -6.605204 -6.605204 -6.605204
```

```
## TCGA-44-A47B -7.329524 -5.744561 -7.329524 -7.329524 -5.744561
```

Risk Prediction

To do the risk prediction we use the ridge regression provided by the package glmnet. The return value of the risk score function is a list for the training set and the test set with one meta feature per subject.

```
dataTraining <- meta_features$training
dataTest <- meta_features$test
riskScoreList <- meanRiskScoreFunction(features_training = dataTraining,
                                       features_test = dataTest,
                                       survObject_training = survObjectTraining)
riskScoreList$training[1:5]
```

```
## [1] 0.1029184 0.1653081 0.1551549 0.1546220 0.1412030
```

```
riskScoreList$test[1:5]
```

```
## [1] 0.2113638 0.1045205 0.1218013 0.1167533 0.1023968
```

Model building and validation

Here, we create a new training data set with the clinical variables and the risk score derived in the previous step. With those data we fit a cox proportional hazards model. We validate the model with the test set and concordance index (C-index) as metric.

```
data_int_Training <- data.frame(filteredClin_lung[training,],
                                riskScore = riskScoreList$training)
data_int_Test <- data.frame(filteredClin_lung[test,],
                            riskScore = riskScoreList$test)



cfit <- coxph(survObjectTraining ~ ., data_int_Training)

predTest <- predict(object = cfit, newdata = data_int_Test, type = "lp")
cindex_validation = concordance.index(predTest,
                                      surv.time = surv_lung[test,]$OVERALL.SURVIVAL,
                                      surv.event = surv_lung[test,]$overall.survival.indicator,
                                      method = "noether")
cIndex = cindex_validation$c.index

cIndex
```

```
## [1] 0.6993773
```