Package PRPS

Type: Package

Title: Calculate classification scores and classify samples into 2 to 3 groups

Version: 0.1.0

Author: Aixiang Jiang, …, David Scott, Ryan Morin

Maintainer: Aixiang Jiang [aijiang@bccrc.ca](mailto:aijiang@bccrc.ca)

Depends: R (>= 3.3.1), lattice, caret, limma, e1071

Suggests: knitr

VignetteBuilder: knir

# I. Introduction

This package calculates classification prediction score with three method choices:

## 1. LPS (Linear Prediction Score);

In the classification step, if LPS is chosen, Empirical Bayes’ probabilities are calcualted and classification is based on cutoff on probabilities;

#### References:

Wright G, Tan B, Rosenwald A, Hurt EH, Wiestner A, Staudt LM. A trait expression-based method to diagnose clinically distinct subgroups of diffuse large B cell lymphoma. Proc Natl Acad Sci U S A. 2003 Aug 19;100(17):9991-6.

## 2. PRPS (Probability ratio based classification predication score);

if PRPS is chosen, two types of outputs are given: one is based on cutoff on Empirical Bayes’ probabilities, the other one is based on natural cutoff 0 on PRPS scores;

#### References:

Ennishi D, Jiang A, Boyle M, Collinge B, Grande BM, Ben-Neriah S, Rushton C, Tang J, Thomas N, Slack GW, Farinha P, Takata K, Miyata-Takata T, Craig J, Mottok A, Meissner B, Saberi S, Bashashati A, Villa D, Savage KJ, Sehn LH, Kridel R, Mungall AJ, Marra MA, Shah SP, Steidl C, Connors JM, Gascoyne RD, Morin RD, Scott DW. Double-Hit Trait Expression Signature Defines a Distinct Subgroup of Germinal Center B-Cell-Like Diffuse Large B-Cell Lymphoma. J Clin Oncol. 2018 Dec 3:JCO1801583. doi: 10.1200/JCO.18.01583.

## 3. PS (Prediction Strength).

when PS is selected, by default, classification is based on natural cutoff 0 on PS scores, however, a separate function can alos issues classification based on cutoff on Empirical Bayes’ probabilities when necessary.

#### References:

TR Golub, DK Slonim, P Tamayo, C Huard, M Gaasenbeek, JP Mesirov, H Coller, ML Loh, JR Downing, MA Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring Science, 286 (1999), pp. 531-537

# II. Typical path: training + testing

## 1. Typical workflow when training and testing data sets are comparable

When training and testing data sets are comparable, the typical workflow is:

1). select the algorithm you would like to use, there are three choices: LPS, PRPS, and PS

2). have your training data set ready, and make your decisions on the parameters

3). run LPStraining, or PRPStraining or PStraining

4). run LPStesting, or PRPStesting or PStesting

## 2. Example data

In the data folder, there are four data files.

### 1). rosenwald.cli

This data frame contains subset clinic information about rosenwald dataset, wihch is downloaded from LPS R package: <https://cran.r-project.org/web/packages/LPS/LPS.pdf>. The original rosenwald data set contains 240 Diffuse Large B-Cell Lymphomas samples (<https://llmpp.nih.gov/DLBCL/>), in this subset, however, 40 samples were randomly selected from the training set that are not in Type III sub-types, and 20 samples ere randomly selected from the validation set that are not in Type III sub-types, together there are 60 Diffuse Large B-Cell Lymphomas samples in rosenwald.cli. The column “set” indicates if a sample was in training or validation data set in the Rosenwald paper, column group is for COO (cell of origin) classification, which could be GCB (germinal center), ABC (activated B cell) , and UNC (un-classified) in the paper, however, in this subset data, we only have GCB and ABC types.

library(PRPS)  
#> Loading required package: lattice  
#> Loading required package: caret  
#> Loading required package: limma  
#> Loading required package: e1071  
data("rosenwald.cli")  
str(rosenwald.cli)  
#> 'data.frame': 60 obs. of 4 variables:  
#> $ set : Factor w/ 2 levels "Training","Validation": 1 1 1 1 1 1 1 1 1 1 ...  
#> $ group : Factor w/ 2 levels "ABC","GCB": 2 1 1 2 2 2 1 2 1 1 ...  
#> $ follow.up: num 2.4 1 10.5 1 1.6 0.2 10.8 2.3 8.4 1 ...  
#> $ status : Factor w/ 2 levels "Alive","Dead": 2 2 1 2 2 2 1 2 1 2 ...

#### References:

Rosenwald A, Wright G, Chan WC, et al. The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma. N Engl J Med 2002;346(25):1937-1947

### 2). rosenwald.expr

This data matrix contains Lymphochip microarrays expression data for the 60 Diffuse Large B-Cell Lymphomas samples as described in rosenwald.cli.

data("rosenwald.expr")  
str(rosenwald.expr)  
#> num [1:7399, 1:60] 0.607 0.239 0.828 0.932 0.86 0 0.476 0.814 0.81 0.87 ...  
#> - attr(\*, "dimnames")=List of 2  
#> ..$ : chr [1:7399] "27481" "17013" "24751" "27498" ...  
#> ..$ : chr [1:60] "LYM018" "LYM120" "LYM180" "LYM285" ...

#### References:

Rosenwald A, Wright G, Chan WC, et al. The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma. N Engl J Med 2002;346(25):1937-1947

### 3). WrightCOO

This data frame contains 158 COO (cell of origin) related genes with both Ensembl annotation ID (row names) and gene symbols (column “Gene”). These genes are used to classfy DLBCL samples into GCB, ABC, or UNC.

data("WrightCOO")  
str(WrightCOO)  
#> 'data.frame': 158 obs. of 1 variable:  
#> $ Gene: chr "NR3C1" "PMM2" "STS" "BCL2" ...  
head(WrightCOO)  
#> Gene  
#> ENSG00000113580 NR3C1  
#> ENSG00000140650 PMM2  
#> ENSG00000101846 STS  
#> ENSG00000171791 BCL2  
#> ENSG00000168811 IL12A  
#> ENSG00000196549 MME

#### References:

Wright G, Tan B, Rosenwald A, Hurt EH, Wiestner A, Staudt LM. A trait expression-based method to diagnose clinically distinct subgroups of diffuse large B cell lymphoma. Proc Natl Acad Sci U S A. 2003 Aug 19;100(17):9991-6.

### 4). DHITsig

This is the double-hit signature (DHITsig) signature gene list with their importance scores (weights) that publised on JCO. Double-hit originally refers to presence of concurrent MYC and BCL2 translocations identified with Fluorescence in situ hybridization (FISH), however, DHITsig here is based on RNAseq data although the the DHIT information was involved in DHITsig development.

data("DHITsig")  
str(DHITsig)  
#> Named num [1:104] 0.674 0.666 0.618 0.597 0.582 ...  
#> - attr(\*, "names")= chr [1:104] "OR13A1" "FAM216A" "MYC" "SLC25A27" ...  
head(DHITsig)  
#> OR13A1 FAM216A MYC SLC25A27 ALOX5 UQCRH   
#> 0.6742184 0.6662736 0.6180968 0.5973289 0.5822841 0.5545504

#### References:

Ennishi D, Jiang A, Boyle M, Collinge B, Grande BM, Ben-Neriah S, Rushton C, Tang J, Thomas N, Slack GW, Farinha P, Takata K, Miyata-Takata T, Craig J, Mottok A, Meissner B, Saberi S, Bashashati A, Villa D, Savage KJ, Sehn LH, Kridel R, Mungall AJ, Marra MA, Shah SP, Steidl C, Connors JM, Gascoyne RD, Morin RD, Scott DW. Double-Hit Trait Expression Signature Defines a Distinct Subgroup of Germinal Center B-Cell-Like Diffuse Large B-Cell Lymphoma. J Clin Oncol. 2018 Dec 3:JCO1801583. doi: 10.1200/JCO.18.01583.

## 3. Example code

In this section, we are providing example code to get classification score when training and testing data are comparable

### 1) LPS

#### LPStraining

Get the data ready:

group = rosenwald.cli$group  
dat = rosenwald.expr

Use GCB as a reference group as in LPS package, then LPStraining result can be got as following

nin = 100   
trainLPS = LPStraining (trainDat = dat, groupInfo = group, refGroup = "GCB", topN = nin,  
 weightMethod = "ttest")  
str(trainLPS)  
#> List of 4  
#> $ LPS\_pars :List of 2  
#> ..$ weights:'data.frame': 100 obs. of 3 variables:  
#> .. ..$ tValue: num [1:100] 8.33 8.37 8 7.86 -8.01 ...  
#> .. ..$ pValue: num [1:100] 2.24e-11 1.86e-11 6.22e-11 1.35e-10 1.76e-10 ...  
#> .. ..$ FDR : num [1:100] 8.29e-08 8.29e-08 1.53e-07 2.49e-07 2.60e-07 ...  
#> ..$ meansds: Named num [1:4] 454 -294 125 187  
#> .. ..- attr(\*, "names")= chr [1:4] "testLPSmean" "refLPSmean" "testLPSsd" "refLPSsd"  
#> $ LPS\_train :'data.frame': 60 obs. of 4 variables:  
#> ..$ LPS\_score : num [1:60] -398.71 384.84 468.91 -5.58 -208.42 ...  
#> ..$ LPS\_class : chr [1:60] "GCB" "ABC" "ABC" "GCB" ...  
#> ..$ LPS\_prob\_test: num [1:60] 1.32e-10 9.99e-01 1.00 5.55e-03 1.29e-06 ...  
#> ..$ LPS\_prob\_ref : num [1:60] 1 0.0011 0.000169 0.994446 0.999999 ...  
#> $ classCompare:List of 6  
#> ..$ positive: chr "ABC"  
#> ..$ table : 'table' int [1:2, 1:2] 37 0 0 22  
#> .. ..- attr(\*, "dimnames")=List of 2  
#> .. .. ..$ Prediction: chr [1:2] "GCB" "ABC"  
#> .. .. ..$ Reference : chr [1:2] "GCB" "ABC"  
#> ..$ overall : Named num [1:7] 1 1 0.939 1 0.627 ...  
#> .. ..- attr(\*, "names")= chr [1:7] "Accuracy" "Kappa" "AccuracyLower" "AccuracyUpper" ...  
#> ..$ byClass : Named num [1:11] 1 1 1 1 1 ...  
#> .. ..- attr(\*, "names")= chr [1:11] "Sensitivity" "Specificity" "Pos Pred Value" "Neg Pred Value" ...  
#> ..$ mode : chr "sens\_spec"  
#> ..$ dots : list()  
#> ..- attr(\*, "class")= chr "confusionMatrix"  
#> $ classTable : 'table' int [1:2, 1:3] 0 22 37 0 0 1  
#> ..- attr(\*, "dimnames")=List of 2  
#> .. ..$ groupInfo: chr [1:2] "GCB" "ABC"  
#> .. ..$ LPS\_class: chr [1:3] "ABC" "GCB" "UNCLASS"

The last item in the output of LPStraining provides information on how our LPS prediction model is doing

trainLPS$classTable  
#> LPS\_class  
#> groupInfo ABC GCB UNCLASS  
#> GCB 0 37 0  
#> ABC 22 0 1

This means with given COO info, and with top 100 significant genes based on t test, LPS classification can put all GCB samples back to GCB samples, for ABC samples, however, one of them is put into UNCLASS, all other samples are put back into ABC.

More detail information about LPStraining function can be found with the two choices:

?(LPStraining)  
help(LPStraining)

#### LPStesting

Now, we need to have a testing data set that is comparable to the training data set. As an example, we get a bootstrap sample set with a given N to build our psuedo testing data set.

set.seed(1027562)  
N = 200 ### sample size for testing  
testdat = sample(1:dim(dat)[2], size = N, replace = TRUE)  
testdat = dat[, testdat]  
str(testdat)  
#> num [1:7399, 1:200] -1.46 -1.51 -1.48 -1.43 -1.65 ...  
#> - attr(\*, "dimnames")=List of 2  
#> ..$ : chr [1:7399] "27481" "17013" "24751" "27498" ...  
#> ..$ : chr [1:200] "LYM112" "LYM197" "LYM142" "LYM430" ...

Now, we can use the output from LPStraining and the testdat for LPStesting

testLPS = LPStesting(LPStrainObj = trainLPS, newdat = testdat)  
str(testLPS)  
#> 'data.frame': 200 obs. of 4 variables:  
#> $ LPS\_score : num -393.5 -625.8 -289.8 -279.4 -22.9 ...  
#> $ LPS\_class : chr "GCB" "GCB" "GCB" "GCB" ...  
#> $ LPS\_prob\_test: num 1.72e-10 4.20e-16 2.96e-08 4.85e-08 2.88e-03 ...  
#> $ LPS\_prob\_ref : num 1 1 1 1 0.997 ...  
table(testLPS$LPS\_class)  
#>   
#> ABC GCB UNCLASS   
#> 74 124 2

We can also combine training and testing data sets together for testing

fullLPS = LPStesting(LPStrainObj = trainLPS, newdat = cbind(testdat, dat))  
str(fullLPS)  
#> 'data.frame': 260 obs. of 4 variables:  
#> $ LPS\_score : num -393.5 -625.8 -289.8 -279.4 -22.9 ...  
#> $ LPS\_class : chr "GCB" "GCB" "GCB" "GCB" ...  
#> $ LPS\_prob\_test: num 1.72e-10 4.20e-16 2.96e-08 4.85e-08 2.88e-03 ...  
#> $ LPS\_prob\_ref : num 1 1 1 1 0.997 ...  
table(fullLPS$LPS\_class)  
#>   
#> ABC GCB UNCLASS   
#> 96 161 3

Now, we are wondering if the testLPS and fulltest give us consistent results.

table(testLPS$LPS\_class, fullLPS$LPS\_class[1:N])  
#>   
#> ABC GCB UNCLASS  
#> ABC 74 0 0  
#> GCB 0 124 0  
#> UNCLASS 0 0 2

More detail about LPStesting can be found with:

?(LPStesting)  
help(LPStesting)

### 2) PRPS

#### PRPStraining

Use the same training data set: *dat* as above, and use GCB as a reference group as in LPS package, then PRPStraining result can be got as following. Notice that now we are calling PRPStraining but all setting is the same as for LPStraining

nin = 100   
trainPRPS = PRPStraining (trainDat = dat, groupInfo = group, refGroup = "GCB", topN = nin,  
 weightMethod = "ttest")  
str(trainPRPS)  
#> List of 4  
#> $ PRPS\_pars :List of 3  
#> ..$ weights :'data.frame': 100 obs. of 3 variables:  
#> .. ..$ tValue: num [1:100] 8.33 8.37 8 7.86 -8.01 ...  
#> .. ..$ pValue: num [1:100] 2.24e-11 1.86e-11 6.22e-11 1.35e-10 1.76e-10 ...  
#> .. ..$ FDR : num [1:100] 8.29e-08 8.29e-08 1.53e-07 2.49e-07 2.60e-07 ...  
#> ..$ meansds : Named num [1:4] 113.6 -151.6 42.5 66.3  
#> .. ..- attr(\*, "names")= chr [1:4] "testPRPSmean" "refPRPSmean" "testPRPSsd" "refPRPSsd"  
#> ..$ traitsmeansds: num [1:100, 1:4] 0.328 0.437 0.388 0.403 -1.258 ...  
#> .. ..- attr(\*, "dimnames")=List of 2  
#> .. .. ..$ : chr [1:100] "27562" "17708" "24796" "27458" ...  
#> .. .. ..$ : chr [1:4] "testmean" "refmean" "testsd" "refsd"  
#> $ PRPS\_train :'data.frame': 60 obs. of 5 variables:  
#> ..$ PRPS\_score : num [1:60] -158.6 99.5 149.9 -78.7 -106.5 ...  
#> ..$ PRPS\_class : chr [1:60] "GCB" "ABC" "ABC" "GCB" ...  
#> ..$ PRPS\_prob\_test: num [1:60] 1.85e-09 9.99e-01 1.00 9.99e-05 2.87e-06 ...  
#> ..$ PRPS\_prob\_ref : num [1:60] 1.00 5.15e-04 2.95e-05 1.00 1.00 ...  
#> ..$ PRPS\_class0 : chr [1:60] "GCB" "ABC" "ABC" "GCB" ...  
#> $ classCompare:List of 6  
#> ..$ positive: chr "ABC"  
#> ..$ table : 'table' int [1:2, 1:2] 37 0 0 23  
#> .. ..- attr(\*, "dimnames")=List of 2  
#> .. .. ..$ Prediction: chr [1:2] "GCB" "ABC"  
#> .. .. ..$ Reference : chr [1:2] "GCB" "ABC"  
#> ..$ overall : Named num [1:7] 1 1 0.94 1 0.617 ...  
#> .. ..- attr(\*, "names")= chr [1:7] "Accuracy" "Kappa" "AccuracyLower" "AccuracyUpper" ...  
#> ..$ byClass : Named num [1:11] 1 1 1 1 1 ...  
#> .. ..- attr(\*, "names")= chr [1:11] "Sensitivity" "Specificity" "Pos Pred Value" "Neg Pred Value" ...  
#> ..$ mode : chr "sens\_spec"  
#> ..$ dots : list()  
#> ..- attr(\*, "class")= chr "confusionMatrix"  
#> $ classTable : 'table' int [1:2, 1:2] 0 23 37 0  
#> ..- attr(\*, "dimnames")=List of 2  
#> .. ..$ groupInfo : chr [1:2] "GCB" "ABC"  
#> .. ..$ PRPS\_class: chr [1:2] "ABC" "GCB"

The last item in the output of PRPStraining provides information on how our PRPS prediction model is doing

trainPRPS$classTable  
#> PRPS\_class  
#> groupInfo ABC GCB  
#> GCB 0 37  
#> ABC 23 0

This means with given COO info, and with top 100 significant genes based on t test, PRPS classification is 100% matching to given COO.

More detail information about LPStraining function can be found with the two choices:

?(PRPStraining)  
help(PRPStraining)

#### PRPStesting

With the exact same testing data set: *testdat*, use the output from PRPStraining, we can apply PRPStesting as following:

testPRPS = PRPStesting(PRPStrainObj = trainPRPS, newdat = testdat)  
str(testPRPS)  
#> 'data.frame': 200 obs. of 5 variables:  
#> $ PRPS\_score : num -162 -223 -112 -190 -48 ...  
#> $ PRPS\_class : chr "GCB" "GCB" "GCB" "GCB" ...  
#> $ PRPS\_prob\_test: num 1.18e-09 6.66e-14 1.32e-06 1.38e-11 3.77e-03 ...  
#> $ PRPS\_prob\_ref : num 1 1 1 1 0.996 ...  
#> $ PRPS\_class0 : chr "GCB" "GCB" "GCB" "GCB" ...  
table(testPRPS$PRPS\_class)  
#>   
#> ABC GCB   
#> 76 124

We can also combine training and testing data sets together for testing

fullPRPS = PRPStesting(PRPStrainObj = trainPRPS, newdat = cbind(testdat, dat))  
str(fullPRPS)  
#> 'data.frame': 260 obs. of 5 variables:  
#> $ PRPS\_score : num -162 -223 -112 -190 -48 ...  
#> $ PRPS\_class : chr "GCB" "GCB" "GCB" "GCB" ...  
#> $ PRPS\_prob\_test: num 1.18e-09 6.66e-14 1.32e-06 1.38e-11 3.77e-03 ...  
#> $ PRPS\_prob\_ref : num 1 1 1 1 0.996 ...  
#> $ PRPS\_class0 : chr "GCB" "GCB" "GCB" "GCB" ...  
table(fullPRPS$PRPS\_class)  
#>   
#> ABC GCB   
#> 99 161

Now, we are wondering if the testLPS and fulltest give us consistent results.

table(testPRPS$PRPS\_class, fullPRPS$PRPS\_class[1:N])  
#>   
#> ABC GCB  
#> ABC 76 0  
#> GCB 0 124

If we compare PRPS results with LPS results, we can find that there is 1 UNCLASS in LPStraining output and there are 2 UNCLASS in LPStesting output, however, there is no UNCLASS in either PRPStraining or PRPStesting output. Remember that our *dat* contains no UNCLASS samples, and our *testdat* are bootstrapped from *dat*, therefore, there should be no UNCLASS either in *testdat*. Together, we think that PRPS might be a slightly better approach than LPS.

### 3) PS

#### PStraining

Use the same training data set: *dat* as above, and use GCB as a reference group as in LPS package, then PStraining result can be got as following. Notice that now we are calling PStraining but all setting is the same as for LPStraining

nin = 100   
trainPS = PStraining (trainDat = dat, groupInfo = group, refGroup = "GCB", topN = nin,  
 weightMethod = "ttest")  
str(trainPS)  
#> List of 4  
#> $ PS\_pars :'data.frame': 100 obs. of 6 variables:  
#> ..$ meanOfGroupMeans: num [1:100] 0.157 0.161 0.162 0.165 -0.186 ...  
#> ..$ tValue : num [1:100] 8.33 8.37 8 7.86 -8.01 ...  
#> ..$ refGroupMean : num [1:100] -0.554 -0.569 -0.533 -0.543 0.546 ...  
#> ..$ testGroupMean : num [1:100] 0.867 0.89 0.858 0.873 -0.919 ...  
#> ..$ pValue : num [1:100] 2.24e-11 1.86e-11 6.22e-11 1.35e-10 1.76e-10 ...  
#> ..$ FDR : num [1:100] 8.29e-08 8.29e-08 1.53e-07 2.49e-07 2.60e-07 ...  
#> $ PS\_train :'data.frame': 60 obs. of 2 variables:  
#> ..$ PS\_score: num [1:60] -0.726 0.632 0.889 -0.454 -0.563 ...  
#> ..$ PS\_class: chr [1:60] "GCB" "ABC" "ABC" "GCB" ...  
#> $ classCompare:List of 6  
#> ..$ positive: chr "ABC"  
#> ..$ table : 'table' int [1:2, 1:2] 37 0 0 23  
#> .. ..- attr(\*, "dimnames")=List of 2  
#> .. .. ..$ Prediction: chr [1:2] "GCB" "ABC"  
#> .. .. ..$ Reference : chr [1:2] "GCB" "ABC"  
#> ..$ overall : Named num [1:7] 1 1 0.94 1 0.617 ...  
#> .. ..- attr(\*, "names")= chr [1:7] "Accuracy" "Kappa" "AccuracyLower" "AccuracyUpper" ...  
#> ..$ byClass : Named num [1:11] 1 1 1 1 1 ...  
#> .. ..- attr(\*, "names")= chr [1:11] "Sensitivity" "Specificity" "Pos Pred Value" "Neg Pred Value" ...  
#> ..$ mode : chr "sens\_spec"  
#> ..$ dots : list()  
#> ..- attr(\*, "class")= chr "confusionMatrix"  
#> $ classTable : 'table' int [1:2, 1:2] 37 0 0 23  
#> ..- attr(\*, "dimnames")=List of 2  
#> .. ..$ groupInfo: chr [1:2] "GCB" "ABC"  
#> .. ..$ PS\_class : chr [1:2] "GCB" "ABC"

The last item in the output of PStraining provides information on how our PS prediction model is doing

trainPS$classTable  
#> PS\_class  
#> groupInfo GCB ABC  
#> GCB 37 0  
#> ABC 0 23

This means with given COO info, and with top 100 significant genes based on t test, PS classification is 100% matching to given COO.

More detail information about LPStraining function can be found with the two choices:

?(PStraining)  
help(PStraining)

#### PStesting

With the exact same testing data set: *testdat*, use the output from PStraining, we can apply PStesting as following:

testPS = PStesting(PStrainObj = trainPS, newdat = testdat)  
str(testPS)  
#> 'data.frame': 200 obs. of 2 variables:  
#> $ PS\_score: num -0.857 -0.935 -0.696 -0.9 -0.146 ...  
#> $ PS\_class: chr "GCB" "GCB" "GCB" "GCB" ...  
table(testPS$PS\_class)  
#>   
#> ABC GCB   
#> 76 124

We can also combine training and testing data sets together for testing

fullPS = PStesting(PStrainObj = trainPS, newdat = cbind(testdat, dat))  
str(fullPS)  
#> 'data.frame': 260 obs. of 2 variables:  
#> $ PS\_score: num -0.859 -0.935 -0.699 -0.9 -0.149 ...  
#> $ PS\_class: chr "GCB" "GCB" "GCB" "GCB" ...  
table(fullPS$PS\_class)  
#>   
#> ABC GCB   
#> 99 161

Now, we are wondering if the testLPS and fulltest give us consistent results.

table(testPS$PS\_class, fullPS$PS\_class[1:N])  
#>   
#> ABC GCB  
#> ABC 76 0  
#> GCB 0 124

PS performs exactly the same as PRPS for the above training and testing data sets (*dat* and *testdat*), and both of them might be better approaches than LPS

# III. Special path: classification score calculation without training

If we already have selected features/traits and their parameters needed for classification score calculation, we might also calculate classification scores for a given data set without training data sets.

## 1. Weights for selected features/traits + testing are available

If we only know selected features/traits, but we do not know any further information. In this case, if we would like to calculate classification score for a given tesing data set, what can we do?

Without any further information and without any assumption, the only method we can use to calculate classification score is to apply LPS approach. In this case, we can call LPSClassWithPriors for classification with selected features and their weights and more info.

Assuming that we have features and their weights from some where, for example, we can get this information from trainLPS:

lpswts = trainLPS$LPS\_pars$weights  
str(lpswts)  
#> 'data.frame': 100 obs. of 3 variables:  
#> $ tValue: num 8.33 8.37 8 7.86 -8.01 ...  
#> $ pValue: num 2.24e-11 1.86e-11 6.22e-11 1.35e-10 1.76e-10 ...  
#> $ FDR : num 8.29e-08 8.29e-08 1.53e-07 2.49e-07 2.60e-07 ...

Here, the tValue is the weight for the top 100 genes selected with LPStraining.

genes = rownames(lpswts)  
lpswts = lpswts[,1]  
names(lpswts) = genes  
head(lpswts)  
#> 27562 17708 24796 27458 17496 27631   
#> 8.329309 8.367778 8.002261 7.856705 -8.007235 7.745951

While LPS score calculation only requires weights, in order to get classification, we need more information to help us. LPS classification is based on Empirical Bayes probabilities, which requires LPS score mean and sd for two groups (for the example data, GCB or ABC). In order to get these information, we might have two ways to achieve:

### 1) Previous LPS score distribution parameters are available

In this case, if the current testing is comparable to previous training data that already have LPS score distribution parameters, we can directly use for our testing data classification.

As an illustration example, assume that the parameters in trainLPS are what we got prevously, since *dat* and *testdat* are comparable, we can directly use these parameters to get final classification.

lpspars = trainLPS$LPS\_pars  
lps\_means\_sds = lpspars$meansds  
lps\_means\_sds  
#> testLPSmean refLPSmean testLPSsd refLPSsd   
#> 454.2384 -293.6822 124.9089 187.3133

Notice that refGroup = “GCB” in the previous training, and testGroup = “ABC”, so, with the default probability cutoff = 0.8, we can have:

lpsouts = LPSClassWithPriors(testdat, weights = lpswts, testGroup = "ABC", refGroup = "GCB", LPSMeanSds = lps\_means\_sds)  
str(lpsouts)  
#> 'data.frame': 200 obs. of 4 variables:  
#> $ LPS\_score : num -393.5 -625.8 -289.8 -279.4 -22.9 ...  
#> $ LPS\_class : chr "GCB" "GCB" "GCB" "GCB" ...  
#> $ LPS\_prob\_test: num 1.72e-10 4.20e-16 2.96e-08 4.85e-08 2.88e-03 ...  
#> $ LPS\_prob\_ref : num 1 1 1 1 0.997 ...  
table(lpsouts$LPS\_class)  
#>   
#> ABC GCB UNCLASS   
#> 74 124 2

On the other hand, if the current testing is **not** comparable to previous training data that already have LPS score distribution parameters, then calibration will be needed before calling function *LPSClassWithPriors* in this package.

start from here on Feb 4, write some example code to do calibration or standardization (in this case, maybe change the wording in the above as well)

### 2) Priors for two groups’ perecentages are reasonable

In this case, assuming that there are p1 ratio cases can be clustered into one group, and p2 ratio cases can be clustered into the other group, and p1+p2 < 1, then, we can calculate LPS score means and sds for the two group, and then calcualte Empirical Bayes probabilities, in the end, with given probability cutoff (default value is 0.8 in this package) to get final classification. If you are not comforatable for the percentage priors, multiple sets of priors might be a good idea to try, in the end, a user could clasify the overlapping two groups across different priors, and leave other uncertainty samples into UNCLASS.

## 2. Weights are not available

### 1) use equal weights

without weights, which means that these all features will be treated equally just like in heatmap/clustering, and the weight for all genes are 1.

### 2) use priors to calculate weights