sigFeature: Significant feature selection using SVM-RFE & t-statistic

Pijush Das, Dr. Susanta Roychudhury, Dr. Sucheta Tripathy

April 26, 2018

Contents

1 Introduction

sigFeature is an R package which is able to find out the significant features using support vector machine recursive feature elimination method (SVM-RFE) (Guyon, I., et al. 2002) and t-statistic. Feature selection is an important part dealing with machine learning technology. SVM-RFE is recognized as one of the most effective filtering methods, which is based on a greedy algorithm that only finds the best possible combination for classification without considering the differentially significant features between the classes. To overcome this limitation of SVM-RFE, the proposed approach is tuned to find differentially significant features along with notable classification accuracy. This package is able to enumerate the feature selection of any two-dimensional (for binary classification) data such as a microarray etc. This vignette explains the use of the package in a publicly available microarray dataset.

2 Data

Example dataset in this package is provided from patients with squamous cell carcinoma of the oral cavity (OSCC)(ACCNO: GSE2280 O'Donnell RK et al. (2005)). Affymetrix Human Genome U133A Array is used for genome-wide transcription analysis in this dataset. In O'Donnell RK et al. (2005), the gene expression profiles obtained from primary squamous cell carcinoma of the oral cavity (OSCC). Samples having metastasis to lymph nodes (N+) is compared with non-metastatic samples (N-). A total of 18 OSCCs were characterized for gene expression. In their analysis, a predictive rule was built using a support vector machine, and the accuracy of the rule was evaluated using cross-validation on the original dataset. This was tested against four independent patient samples. A signature gene set is produced which is able to predict correctly the four independent patients. Using this method lymph node metastasis can be predicted by gene expression profiles. The details of the study can be found at: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE2280.

For "sigFeature" package evaluation, the microarray dataset (GSE2280) has been classified into two classes such as lymph node metastatic (N+) and no lymph node metastatic (N-) according to the TNM staging, provided in the dataset. After downloading the dataset from Gene Expression Omnibus (GEO) database firstly, it was normalized using "quantile" normalization method by using the Bioconductor package "limma". There are altogether 27 samples (19 N+ and 8 N-). In order to reduce the runtime, t statistics was computed between the N+ and N- samples for each feature (gene). The p values less than 0.07 was used as a subset for testing. Now the expression value of the sub-dataset is considered here as "x". The patients without lymph

node metastasis are represented -1, and the patients with lymph node metastasis are represented as 1. Those -1 and 1 value is incorporated with "y" as sample labels.

3 Example

In this section, we show the R script necessary to conduct the significant feature selection with "sigFeature" package on the example dataset. First, we load the "sigFeature" package and the example dataset (as mentioned in the data section) into the memory of the machine. After that, step by step execution takes place. Backward feature elimination procedure is a time-consuming procedure. If we are dealing with the full dataset, we may have to wait for a long time to process the dataset. Alternatively, we can take advantage of parallel processing using the R package "parallel" to reduce the processing time. To reduce the build-up time of this package sometime, we use to process data in this vignette.

```
library(sigFeature)
library(SummarizedExperiment)
```

```
## Loading required package: MatrixGenerics
```

```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'MatrixGenerics'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: stats4
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
```

```
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, basename, cbind, colnames, dirname, do.call,
##     duplicated, eval, evalq, get, grep, grepl, intersect, is.unsorted,
##     lapply, mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, rank, rbind, rownames, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which.max, which.min
## Loading required package: S4Vectors
##
## Attaching package: 'S4Vectors'
## The following objects are masked from 'package:base':
##
##     I, expand.grid, unname
## Loading required package: IRanges
##
## Attaching package: 'IRanges'
## The following object is masked from 'package:grDevices':
##
##     windows
## Loading required package: GenomeInfoDb
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
##
## Attaching package: 'Biobase'
## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians
## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians
data(ExampleRawData, package="sigFeature")
ExampleRawData
## class: SummarizedExperiment
## dim: 2204 27
## metadata(0):
## assays(1): counts
## rownames(2204): 1494_f_at 179_at ... AFFX-M27830_M_at
##   AFFX-r2-Hs28SrRNA-5_at
## rowData names(0):
## colnames(27): GSM42246 GSM42248 ... GSM42270 GSM42271
```

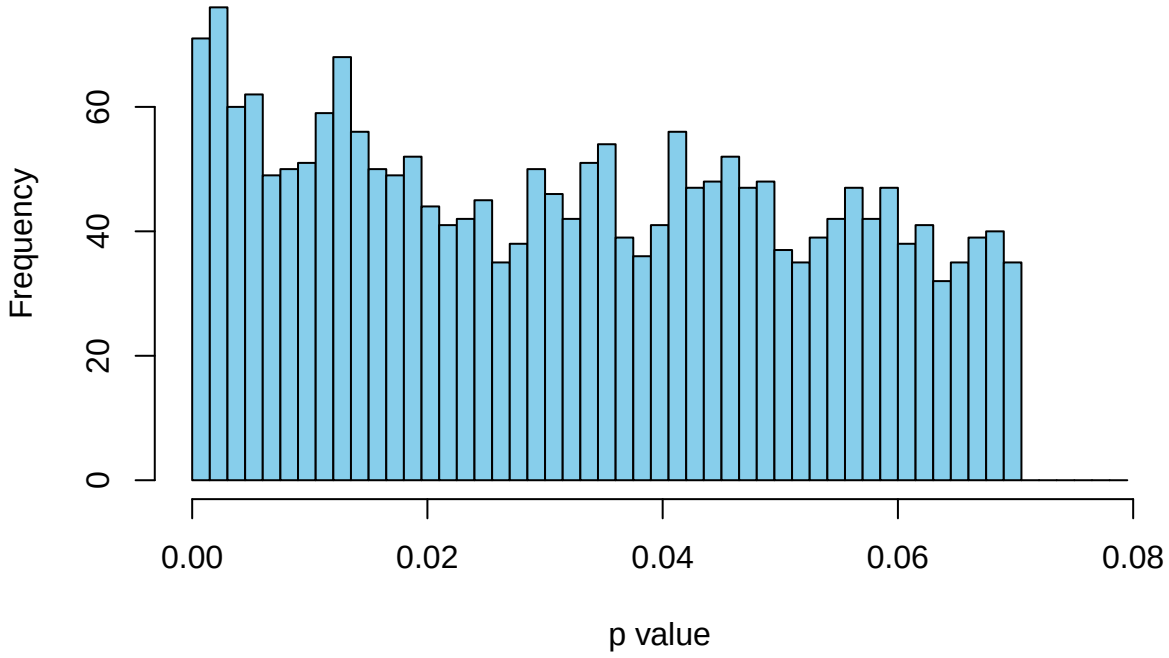
```
## colData names(1): sampleLabels
```

In this example, the variable "x" is containing the microarray expression data, and the "y" contains the sample labels. The variable "x" is a two-dimensional matrix in which the row contains the sample names, and the column contains the gene/feature names. The variable "y" contains the sample labels as -1 and 1.

```
x  <- t(assays(ExampleRawData)$counts)
y  <- colData(ExampleRawData)$sampleLabels
```

Before going into details of the vignette, let us plot the unadjusted p-value for each features which are estimated by using the expression values of the two classes. The function named "sigFeaturePvalue()" is used here to calculate the p-value using t-statistic. In this case, we assume unequal variances and using the Welch approximation to estimate the appropriate degrees of freedom.

```
pvals <- sigFeaturePvalue(x,y)
hist(unlist(pvals),breaks=seq(0,0.08,0.0015),col="skyblue",
    xlab="p value",ylab="Frequency",main="")
```



The idea of "sigFeature()" (Significant Feature Selection): The recursive feature elimination that uses support vector machine can rank better classifiers, but those classifiers may or may not be differently significant between the classes. Features with notable classification accuracy and also differentially significant between the classes have a predominant role in a biological aspect. The algorithm, introduced here is very efficient in ranking classifiers, as well as in determining differently significant classes among them.

In data mining and optimization, feature selection is a very active field of research where the SVM-RFE is distinguished as one of the most effective methods. SVM-RFE is a greedy method that only hopes to find the best possible combination for classification. In this proposed algorithm, we have included a t-statistics along with SVM-RFE. The primary objective of this algorithm is to enumerate the ranking weights for all features and sort the features according to weight vectors as the classification basis. The coefficient and the expression mean difference between two compared groups of each individual feature is used to calculate the weight value of that particular gene or the feature. The iteration process is followed by backward removal of the feature. The iteration process is continued until there is only one feature remaining in the dataset. The smallest ranking weight will be removed by the algorithm and stored in a stack, while remaining the feature variables have a significant impact for producing weight. Finally, the feature variables which are stored in the stack will be listed in the descending order of descriptive different degree.

```
#system.time(sigfeatureRankedList <- sigFeature(x, y))
```

The variable "sigfeatureRankedList" will return the address of the genes/features after execution. To visualize the output, print command can be used that will print the address of top 10 ranked genes/features.

```
data(sigfeatureRankedList)
print(sigfeatureRankedList[1:10])
```

```
##  [1] 2064  370 2032 2035 1519 1573 1446 2105  997  611
```

For producing the model vector for training the data set by using the top 1000 genes/features, the R script is given as below. In this case the package "e1071" is used. For producing the model vector, linear kernel is chosen to get better performance.

```
library(e1071)
sigFeature.model=svm(x[ ,sigfeatureRankedList[1:1000]], y,
                     type="C-classification", kernel="linear")
summary(sigFeature.model)
```

```
##
## Call:
## svm.default(x = x[, sigfeatureRankedList[1:1000]], y = y, type = "C-classification",
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  16
##
##  ( 10 6 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

The R script below shows the prediction result of the same dataset that was trained before. The user can also separate the dataset into two parts e.g. training dataset and testing dataset and can implement the algorithm.

```
pred <- predict(sigFeature.model, x[ ,sigfeatureRankedList[1:1000]])
table(pred,y)
```

```
##     y
## pred -1  1
##   -1  8  0
##    1  0 19
```

To solve the classification problem with the help of ranking the features, another algorithm was proposed by Guyon et. al 2002 named as SVM-RFE. In this algorithm, the dataset has been trained with SVM linear kernel model to produce a weight vector for each feature. After that, the feature with smallest rank is recursively removed and stored in a stack data structure. The iteration process continues till the last feature remains. This criterion is the w value of the decision hyperplane given by the SVM. The weight vector is calculated by squiring the variable w. Finally the features are selected in a list in the descending

order. The function "svmrfeFeatureRanking()" is included in this package to compare the performance with "sigFeature()".

```
#system.time(featureRankedList <- svmrfeFeatureRanking(x, y))
data(featureRankedList)
print("Top 10 features are printed below:")
```

```
## [1] "Top 10 features are printed below:"
```

```
print(featureRankedList[1:10])
```

```
##  [1] 1073 1404 1152    5 1253 1557  105 1207  792   57
```

For producing the model vector of the training dataset by using the top 1000 genes/features, the R script is given below. The linear kernel is chosen for producing the model vector.

```
RFE.model=svm(x[ ,featureRankedList[1:1000]], y,
            type="C-classification", kernel="linear")
summary(RFE.model)
```

```
##
## Call:
## svm.default(x = x[, featureRankedList[1:1000]], y = y, type = "C-classification",
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  17
##
##  ( 10 7 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

The R script below will show the prediction result of the same dataset which is trained before. The user can also separate the dataset into two parts e.g. training dataset and testing dataset and can implement the algorithm.

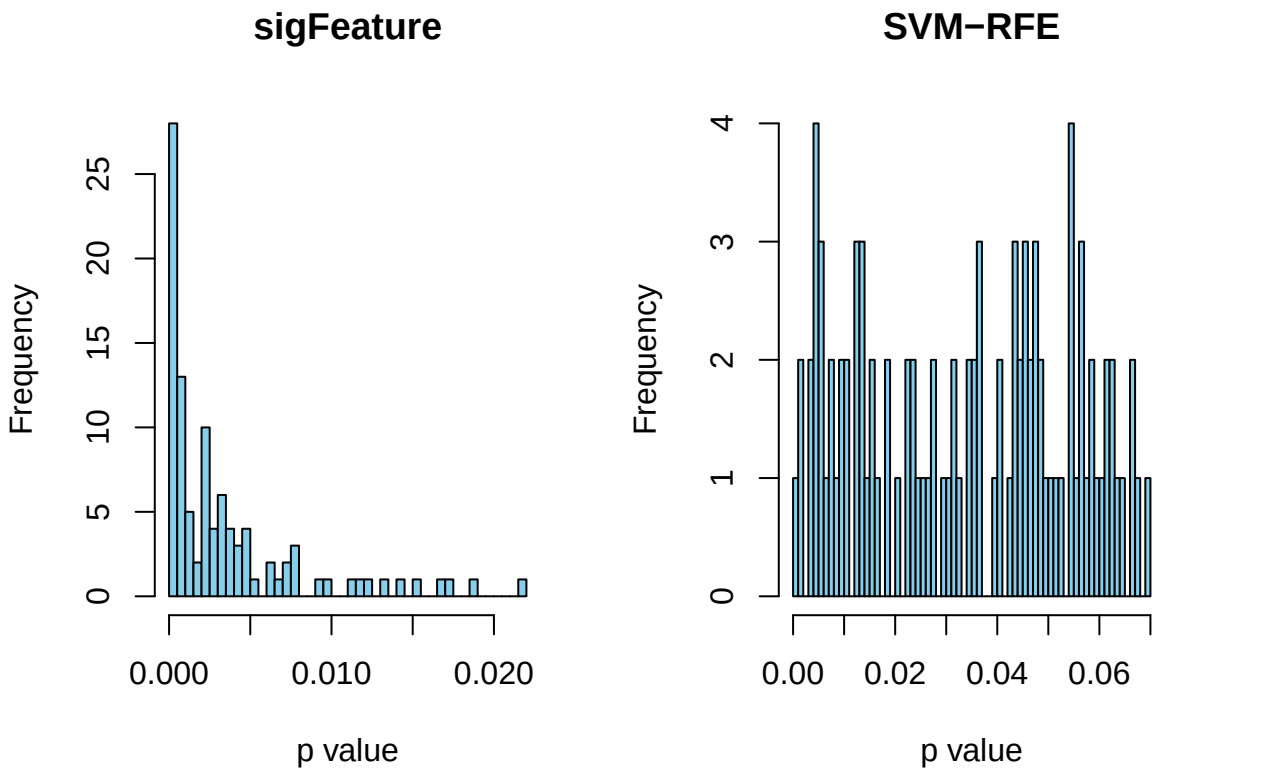
```
pred <- predict(RFE.model, x[ ,featureRankedList[1:1000]])
table(pred,y)
```

```
##     y
## pred -1  1
##   -1  8  0
##   1   0 19
```

In this example section we will observe the p value of the selected top 100 features. The function named "sigFeaturePvalue()" will calculate the p value using t-statistic. The expression value will be taken from the dataset of top 100 features.

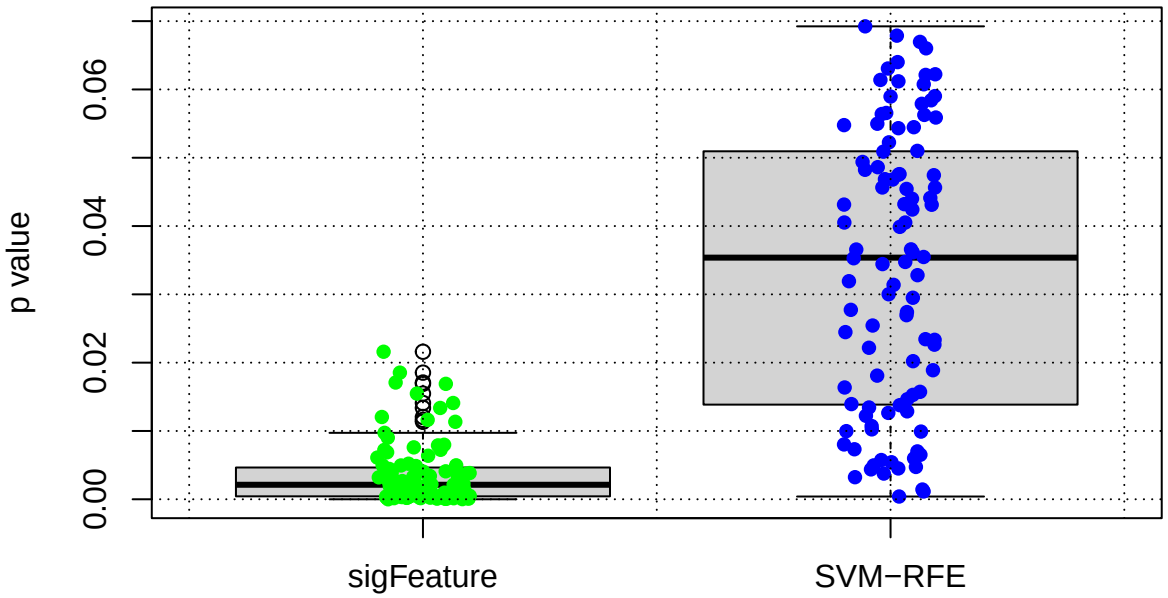
```
pvalsigFe <- sigFeaturePvalue(x, y, 100, sigfeatureRankedList)
pvalRFE   <- sigFeaturePvalue(x, y, 100, featureRankedList)
```

```
par(mfrow=c(1,2))
hist(unlist(pvalsigFe),breaks=50, col="skyblue", main=paste("sigFeature"),
    xlab="p value")
hist(unlist(pvalRFE),breaks=50, col="skyblue",
    main=paste("SVM-RFE"), xlab="p value")
```



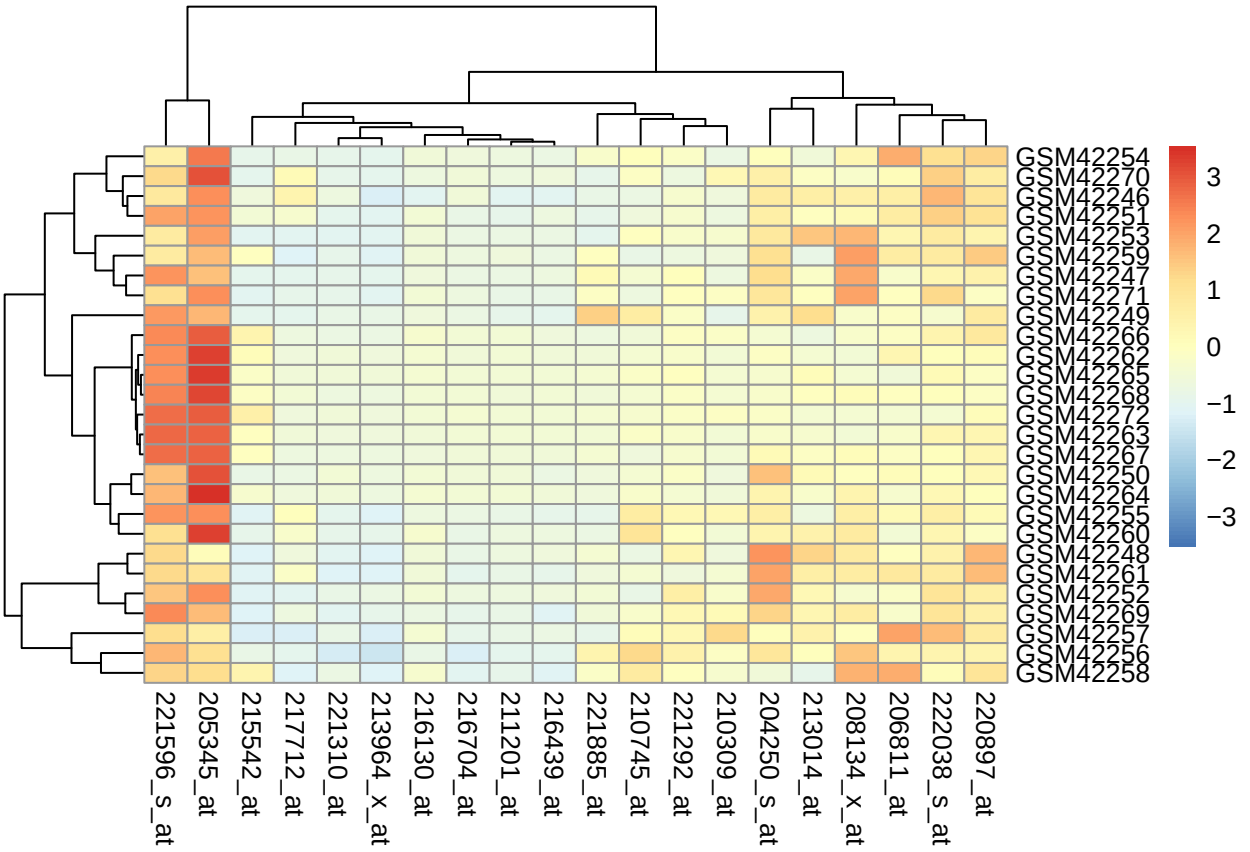
In this section a comparison of p values produced by using top 100 features (using "sigFeature" and "SVM-RFE" algorithm) is shown by using scatter box plot.

```
mytitle<-'Box Plot'
boxplot(unlist(pvalsigFe), unlist(pvalRFE), main=mytitle,
        names=c("sigFeature", "SVM-RFE"),
        ylab="p value", ylim=c(min(unlist(pvalsigFe)), max(unlist(pvalRFE))))
stripchart(unlist(pvalsigFe), vertical=TRUE, method="jitter", add=TRUE, pch=16,
        col=c('green'))
stripchart(unlist(pvalRFE), vertical=TRUE, at=2, method="jitter", add=TRUE,
        pch=16, col=c('blue'))
grid(nx=NULL, ny=NULL, col="black", lty="dotted")
```



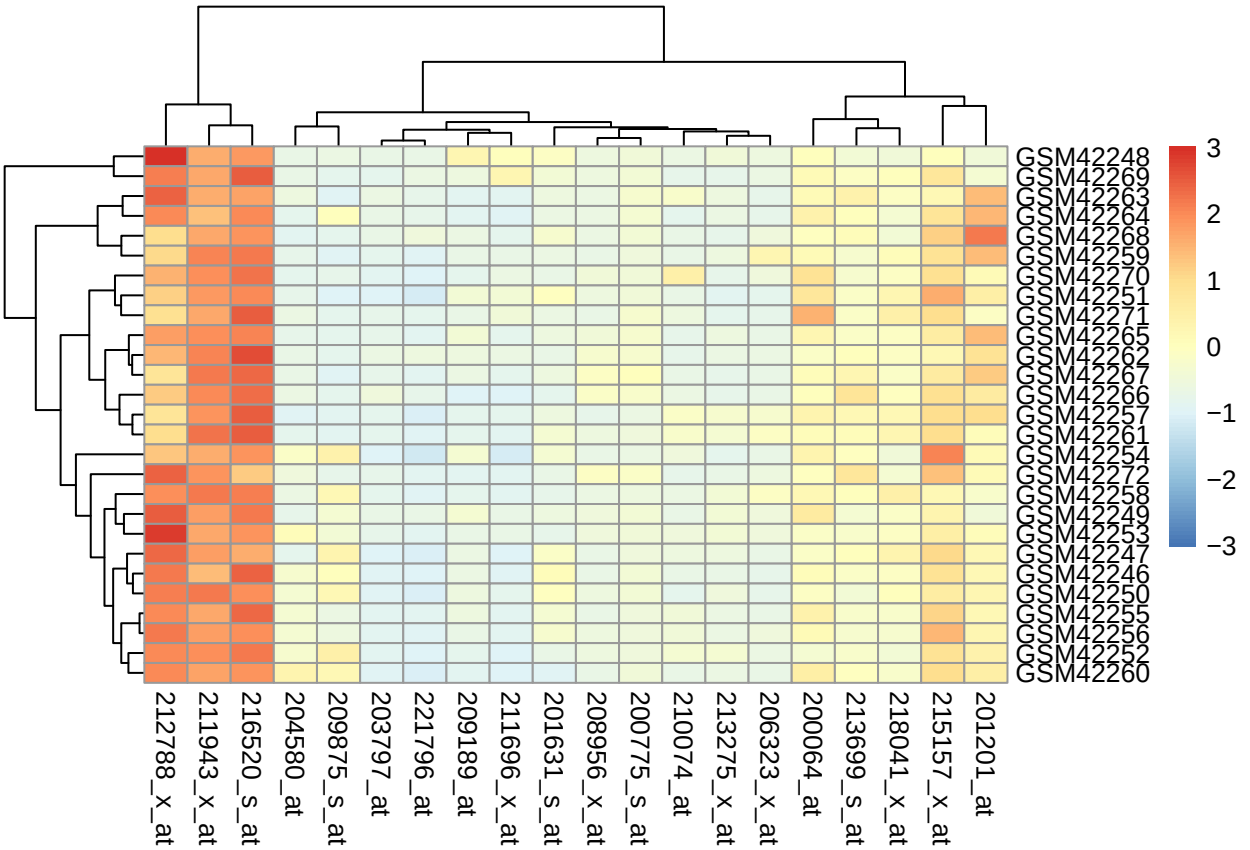
In this section a heatmap is shown below with the expression value of top 20 features generated by using "sigFeature()" function.

```
library("pheatmap")
library("RColorBrewer")
pheatmap(x[ ,sigfeatureRankedList[1:20]], scale="row",
        clustering_distance_rows="correlation")
```



In this section a heatmap is shown below with the expression value of top 20 features generated by using "svmrfeFeatureRanking()" function.

```
pheatmap(x[ ,featureRankedList[1:20]], scale="row",
        clustering_distance_rows="correlation")
```



For estimation of feature selection algorithms, Ambroise and McLachlan (2002) suggested external cross-validation, in which the feature selection and validation are performed on different parts of the sample set, to obtain an unbiased performance estimate. The stratified cross-validation (Breiman et al.,1984) is slightly different from regular cross-validation. In k-fold stratified cross-validation, a data set is randomly partitioned into k equally sized folds such that the class distribution in each fold is approximately the same as that in the entire data set. In contrast, regular cross-validation randomly partitions the data set into k-folds without considering class distributions. Kohavi (1995) reported that stratified cross-validation has smaller bias and variance than regular cross-validation. The "sigFeature()" function thus farther enhances by incorporating external cross-validation method. In this external k-fold cross-validation procedure, k-1 folds are used for selecting the feature, and one fold remains untouched which will later use as a test sample set.

Detail description of the input parameters is given in the help file. The set.seed function is used here only for the purpose of getting the exact results when regenerating this vignette from its source files. The function is given below.

```
#set.seed(1234)
#results = sigFeature.enfold(x, y, "kfold", 10)
data("results")
str(results[1])
```

```
## List of 1
##  $ :List of 5
##   ..$ feature.ids     : int [1:2204] 2064 2031 2035 1573 370 2023 1605 2032 997 1069 ...
##   ..$ train.data.ids  : chr [1:24] "GSM42246" "GSM42248" "GSM42250" "GSM42252" ...
##   ..$ test.data.ids   : chr [1:3] "GSM42263" "GSM42251" "GSM42260"
##   ..$ train.data.level: Named num [1:24] 1 1 1 1 1 -1 -1 -1 -1 -1 ...
##   .. ..- attr(*, "names")= chr [1:24] "GSM42246" "GSM42248" "GSM42250" "GSM42252" ...
##   ..$ test.data.level : Named num [1:3] -1 1 1
```

```
##   .. ..- attr(*, "names")= chr [1:3] "GSM42263" "GSM42251" "GSM42260"
```

In this example section a new function is introduced named as "sigFeatureFrequency()". The main purpose of this function is to arrange the feature on the basis of its frequency. In the previous section the dataset is divided into k-folds. Out of which k-1 folds are randomly taken and used for selecting the feature. This procedure is repeated k times and the frequency values are calculated for top 400 genes from the k number of lists. The "sigFeatureFrequency()" function is used here to rank further all the k'th feature list according to its frequency. Details description of this function is given in the help file.

```
FeatureBasedonFrequency <- sigFeatureFrequency(x, results, 400, 400, pf=FALSE)
str(FeatureBasedonFrequency[1])
```

```
## List of 1
##  $ :List of 5
##   ..$ feature.ids     : num [1:400] 187 225 246 303 313 370 394 469 479 492 ...
##   ..$ train.data.ids  : chr [1:24] "GSM42246" "GSM42248" "GSM42250" "GSM42252" ...
##   ..$ test.data.ids   : chr [1:3] "GSM42263" "GSM42251" "GSM42260"
##   ..$ train.data.level: Named num [1:24] 1 1 1 1 1 -1 -1 -1 -1 -1 ...
##   .. ..- attr(*, "names")= chr [1:24] "GSM42246" "GSM42248" "GSM42250" "GSM42252" ...
##   ..$ test.data.level : Named num [1:3] -1 1 1
##   .. ..- attr(*, "names")= chr [1:3] "GSM42263" "GSM42251" "GSM42260"
```

In this section external cross-validation is computed and finally mean cross validation errors and the standard deviation of the errors is returnd. The features, which are produced on the basis of frequency is used for producing the mean cross validation error. Training and test samples are same which were initially split from the main sample set in to k-fold. In each iteration k-1 fold are considered as training sample and remaining one fold is considered as testing sample. In this external cross validation procedure feature number is increased one by one and using that feature training sample is trained. The number of miss classification is averaged out for each fold change which finally represented as error rate.

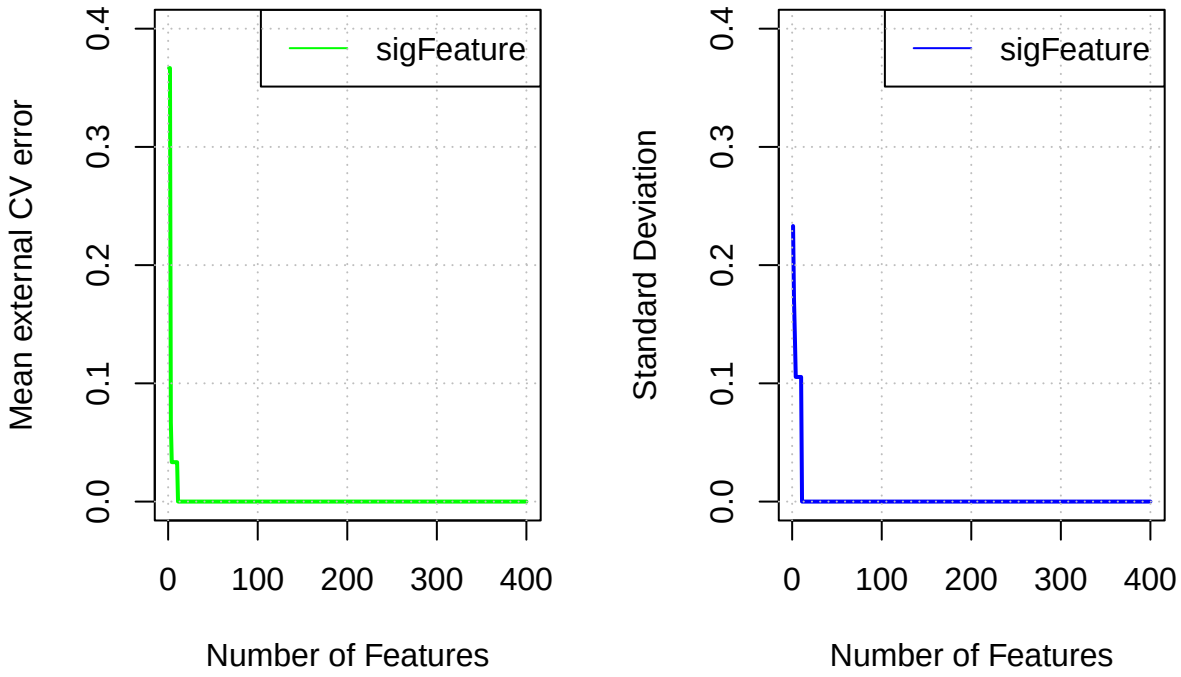
```
#inputdata <- data.frame(y=as.factor(y) ,x=x)
#To run the code given bellow will take huge time. Thus the process
#data is given below.
#featsweepSigFe = lapply(1:400, sigCVError, FeatureBasedonFrequency, inputdata)
data("featsweepSigFe")
str(featsweepSigFe[1])
```

```
## List of 1
##  $ :List of 3
##   ..$ svm.list:List of 10
##   .. ..$ :'data.frame':  1 obs. of  4 variables:
##   .. .. ..$ gamma     : num 0.000244
##   .. .. ..$ cost      : num 0.001
##   .. .. ..$ error     : num 0.333
##   .. .. ..$ dispersion: num NA
##   .. ..$ :'data.frame':  1 obs. of  4 variables:
##   .. .. ..$ gamma     : num 0.000244
##   .. .. ..$ cost      : num 0.001
##   .. .. ..$ error     : num 0.333
##   .. .. ..$ dispersion: num NA
##   .. ..$ :'data.frame':  1 obs. of  4 variables:
##   .. .. ..$ gamma     : num 0.000244
##   .. .. ..$ cost      : num 0.001
##   .. .. ..$ error     : num 0.667
##   .. .. ..$ dispersion: num NA
##   .. ..$ :'data.frame':  1 obs. of  4 variables:
```

```
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0.333
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0.333
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 1
##    .. .. ..$ cost      : num 1.78
##    .. .. ..$ error     : num 0.667
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0.5
##    .. .. ..$ dispersion: num NA
##    .. ..$ :'data.frame':  1 obs. of  4 variables:
##    .. .. ..$ gamma     : num 0.000244
##    .. .. ..$ cost      : num 0.001
##    .. .. ..$ error     : num 0.5
##    .. .. ..$ dispersion: num NA
##    ..$ error   : num 0.367
##    ..$ errorSD : num 0.233
```

In this section the "PlotErrors()" function draw two plots. The first plot is representing the average external cross validation error and the second plot is representing the standard deviation of the of the miss classifications.

```
PlotErrors(featsweepSigFe, 0, 0.4)
```



Writing the results can be achieved with the "WritesigFeature()" function. Please refer to the help file of "WritesigFeature()" for more details on how to save to results to a specific file.

```
#WritesigFeature(results, x)
```

4 SessionInfo

As last part of this document, we call the function "sessionInfo()", which reports the version numbers of R and all the packages used in this session. It is good practice to always keep such a record as it will help to trace down what has happened in case that an R script ceases to work because the functions have been changed in a newer version of a package.

```
sessionInfo()
```

```
## R version 4.1.0 RC (2021-05-10 r80283)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server x64 (build 17763)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] RColorBrewer_1.1-2          pheatmap_1.0.12
```

```
##  [3] e1071_1.7-6                SummarizedExperiment_1.22.0
##  [5] Biobase_2.52.0             GenomicRanges_1.44.0
##  [7] GenomeInfoDb_1.28.0        IRanges_2.26.0
##  [9] S4Vectors_0.30.0           BiocGenerics_0.38.0
## [11] MatrixGenerics_1.4.0       matrixStats_0.58.0
## [13] sigFeature_1.10.0
##
## loaded via a namespace (and not attached):
##  [1] xfun_0.23               lattice_0.20-44        colorspace_2.0-1
##  [4] htmltools_0.5.1.1       yaml_2.2.1             RBGL_1.68.0
##  [7] XML_3.99-0.6            rlang_0.4.11           BiocParallel_1.26.0
## [10] GenomeInfoDbData_1.2.6  lifecycle_1.0.0        stringr_1.4.0
## [13] zlibbioc_1.38.0         munsell_0.5.0          gtable_0.3.0
## [16] zip_2.1.1               biocViews_1.60.0       evaluate_0.14
## [19] knitr_1.33              SparseM_1.81           RUnit_0.4.32
## [22] class_7.3-19            highr_0.9              Rcpp_1.0.6
## [25] scales_1.1.1            BiocManager_1.30.15    DelayedArray_0.18.0
## [28] graph_1.70.0            XVector_0.32.0         digest_0.6.27
## [31] stringi_1.6.2           openxlsx_4.2.3         grid_4.1.0
## [34] tools_4.1.0             bitops_1.0-7           magrittr_2.0.1
## [37] RCurl_1.98-1.3          proxy_0.4-25           Matrix_1.3-3
## [40] rmarkdown_2.8           R6_2.5.0               nlme_3.1-152
## [43] compiler_4.1.0
```

5 References

1. Meyer, D., et al., e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. 2012.
2. O'Donnell RK, Kupferman M, Wei SJ, Singhal S et al. Gene expression signature predicts lymphatic metastasis in squamous cell carcinoma of the oral cavity. Oncogene 2005 Feb 10;24(7):1244-51.
3. Guyon, I., et al. (2002) Gene selection for cancer classification using support vector machines, Machine learning, 46, 389-422.
4. Ambroise, C., & McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the national academy of sciences, 99(10), 6562-6566.
5. Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984) Classiflcation and Regression Trees. Wadsworth and Brooks, Monterey, CA.
6. Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai (Vol. 14, No. 2, pp. 1137-1145).