GeneSelectR Application on TCGA-BRCA Dataset

Damir Zhakparov

2023-10-11

Table of Contents

[Introduction 1](#_Toc147912475)

[Biological Question 1](#_Toc147912476)

[Dataset Overview 2](#_Toc147912477)

[Molecular Subtypes Overview 2](#_Toc147912478)

[1. Differential Gene Expression Analysis 2](#_Toc147912479)

[2. Feature Selection with GeneSelectR 4](#_Toc147912480)

[2.1 Data Preparation 4](#_Toc147912481)

[2.2 Running GeneSelectR 4](#_Toc147912482)

[3. Results Inspection 5](#_Toc147912483)

[3.1 Machine Learning Performance Metrics 5](#_Toc147912484)

[3.2 Feature Importance 6](#_Toc147912485)

[3.3 Overlap between Gene Lists and DGE list 14](#_Toc147912486)

[3.4 Gene Lists Annotation 14](#_Toc147912487)

[3.5 Gene Ontology Enrichment 15](#_Toc147912488)

[3.6 Quantification of Children Nodes of Parent Node of Interest 19](#_Toc147912489)

[3.7 Semantic Similarity Analysis 21](#_Toc147912490)

[4. Picking the Winner and Conclusion 23](#_Toc147912491)

# Introduction

This tutorial walks you through the use of GeneSelectR with the **TCGA-BRCA RNA Expression dataset** from The Cancer Genome Atlas. To acquire the dataset, we employed the [TCGAbiolinks](link-to-TCGAbiolinks) R package. For a step-by-step guide on data extraction, consult this [script](link-to-script).

## Biological Question

The primary aim of this tutorial is to identify a transcriptomic signature specific to each molecular subtype of breast cancer as defined by PAM50 markers. By doing so, we hope to shed light on the unique molecular mechanisms underlying each subtype, which could subsequently inform targeted therapeutic strategies and prognostic assessments.

## Dataset Overview

In this guide, we focus on a subset of **380 samples**, all categorized under the **‘Primary Solid Tumor’** type and labeled with **PAM50 markers**: Basal(n = 80), Her2(n = 38), LumA(n = 188), and LumB (n = 74). The analysis encompasses all **60,600 sequenced transcripts**. For an in-depth look at the TCGA-BRCA dataset, please visit the [official documentation](link-to-official-documentation). The data files used in the tutorial can be accessed **Error! Hyperlink reference not valid.**.

## Molecular Subtypes Overview

### Basal-like

* **Characteristics**: Triple Negative (HR-, HER2-), high levels of Ki-67
* **Prognosis**: Poor
* **Common Treatments**: Chemotherapy

### HER2-enriched

* **Characteristics**: HER2 Positive (HER2+), Hormone Receptor Negative (HR-)
* **Prognosis**: Intermediate
* **Common Treatments**: HER2-targeted therapies like trastuzumab

### Luminal A

* **Characteristics**: Hormone Receptor Positive (HR+), low levels of HER2 and Ki-67
* **Prognosis**: Best among the subtypes
* **Common Treatments**: Hormone therapy

### Luminal B

* **Characteristics**: Hormone Receptor Positive (HR+), higher levels of HER2 and Ki-67
* **Prognosis**: Worse than Luminal A but better than Basal-like
* **Common Treatments**: Hormone therapy, may require chemotherapy

By understanding the unique transcriptomic landscape of each subtype, we can better predict disease outcomes and tailor treatment regimens.

# 1. Differential Gene Expression Analysis

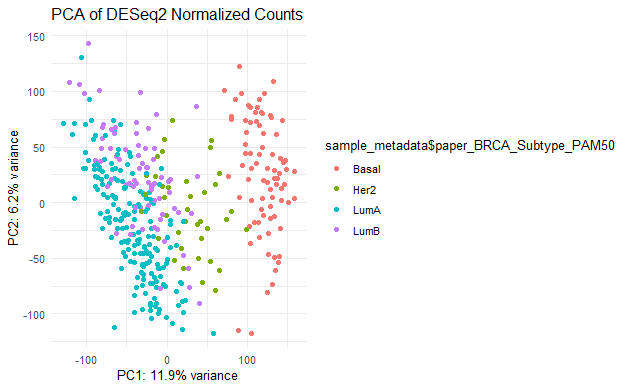
As a baseline differential gene expression analysis (DGE) provides a good starting point. First of all we will load the dataset and metadata files and import necessart packages:

# set up working directories   
data\_dir <- file.path('../raw-data')  
output\_dir <- file.path('../results')  
  
#load the files   
sample\_metadata <- readRDS(file.path(data\_dir, 'sample\_metadata.rds'))  
raw\_counts <- readRDS(file.path(data\_dir, 'raw\_counts.rds'))

After that we will create a DESEq2 object and then filter out genes with low counts:

dds <- DESeqDataSetFromMatrix(countData = raw\_counts,  
 colData = sample\_metadata,  
 design = ~ paper\_BRCA\_Subtype\_PAM50)  
  
# filter the low counts  
table(sample\_metadata$paper\_BRCA\_Subtype\_PAM50)  
#>   
#> Basal Her2 LumA LumB   
#> 80 38 188 74  
smallestGroupSize <- 38 #smallest group is 38 samples  
keep <- rowSums(counts(dds) >= 10) >= smallestGroupSize  
dds <- dds[keep,]

Before performing the DGE analysis it’s strongly advised to perform PCA on normalized counts to see if there any batch effects. We can do it by running the function and then plotting:



And now let’s run the DGE analysis:

# Run DESeq  
dds <- DESeq(dds)

After the calculations are done, let’s filter out differentially expressed genes:

# Get DE results  
res <- results(dds)  
filtered\_res <- subset(res, padj < 0.001 & abs(log2FoldChange) > log2(5))  
filtered\_df <- as.data.frame(filtered\_res)

# 2. Feature Selection with GeneSelectR

## 2.1 Data Preparation

GeneSelectR expects the data to be a matrix/dataframe where rows represent samples and columns are features. Please note that the dataset has to be normalized with between sample normalization prior to any analysis with the package. So now we extract the vst-transformed matrix from our DESeq2 object for the feature selection:

# Extract VST (Variance-Stabilized Transformed) Data  
vsd <- vst(dds, blind = FALSE)   
vsd\_matrix <- assay(vsd)  
vsd\_matrix <- t(vsd\_matrix)

Then we will create a response vector y with sample labels:

sample\_metadata <- sample\_metadata[match(rownames(exp),rownames(sample\_metadata)), ]  
sample\_metadata$paper\_BRCA\_Subtype\_PAM50 <- as.factor(sample\_metadata$paper\_BRCA\_Subtype\_PAM50)  
sample\_metadata$num\_label <- as.integer(sample\_metadata$paper\_BRCA\_Subtype\_PAM50) # NOTE: the labels should be encoded with numeric values  
table(sample\_metadata$num\_label)  
#> < table of extent 0 >

## 2.2 Running GeneSelectR

After preparing the data we are ready to run GeneSelectR:

X <- exp  
y <- sample\_metadata %>% select(num\_label)  
  
selection\_results <- GeneSelectR(X, # gene expression matrix  
 y, # label vector   
 njobs = -1, # number of cores to de deployes (-1 = all)  
 n\_splits = 5, # number of train/test splits to perform   
 max\_features = 250, # max futures to be selected per FS method (only RF, Lasso and Univariate)  
 perform\_test\_split = TRUE, # if partition data into train and test   
 scoring = 'accuracy', # scoring metric for optimization   
 calculate\_permutation\_importance = TRUE, # whether to calculate permutation importance   
 search\_type = 'random', # type of grid search  
 n\_iter = 150L) # number of hyperparameter combinations to be samples

# 3. Results Inspection

After the analysis has been finished, we can look into the PipelineResults objects to inspect the results. For example if we call:

str(selection\_results, max.level = 1)  
#> Formal class 'PipelineResults' [package "GeneSelectR"] with 6 slots

We can see following slots in the Pipeline results object: - best\_pipeline - contains all the parameters for the best performing pipeline; - cv\_results - contains the entire output of the CV procedure during hyperparameter search; - inbuilt\_feature\_importance - inbuilt feature importance with mean, std and rank for every feature across iterations;  
- permutation\_importance - permutation feature importance with mean, std and rank for every feature across iterations;  
- cv\_mean\_scores - CV scores but agreggated into one dataframe with mean and sd of the optimization metrics scores;  
- test\_metrics - metrics (f1, recall, precision and accuracy) for the unseen test set. Let’s inspect some of the most interesting metrics for the evaluation.

## 3.1 Machine Learning Performance Metrics

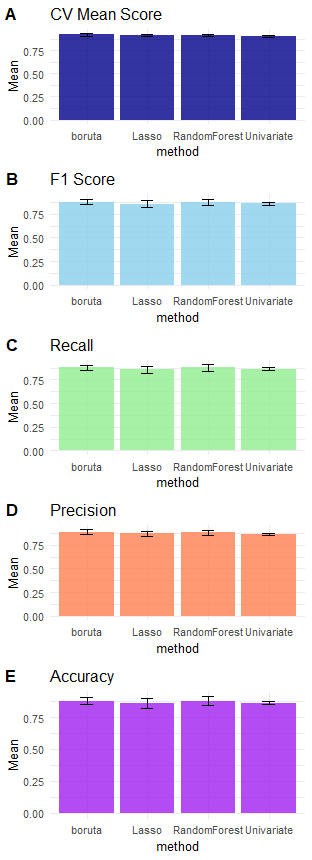
First, let’s explore the cross validation scores for every of our methods. We can inspect the CV mean performance by displaying the dataframe:

selection\_results@cv\_mean\_score  
#> method mean\_score sd\_score  
#> 1 boruta 0.9210374 0.012047342  
#> 2 Lasso 0.9153061 0.007984372  
#> 3 RandomForest 0.9127721 0.007926976  
#> 4 Univariate 0.9045578 0.011999833

We can see that boruta is slightly better in terms of CV performance, but all methods are somewhat comparable. In this case we can inspect the performance on unseen data that is stored in test\_metrics slot:

selection\_results@test\_metrics  
#> # A tibble: 4 × 9  
#> method f1\_mean f1\_sd recall\_mean recall\_sd precision\_mean precision\_sd  
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1 Lasso 0.854 0.0351 0.858 0.0388 0.870 0.0262  
#> 2 RandomForest 0.873 0.0346 0.876 0.0356 0.883 0.0286  
#> 3 Univariate 0.859 0.0127 0.863 0.0150 0.866 0.0136  
#> 4 boruta 0.876 0.0251 0.876 0.0273 0.889 0.0234  
#> # ℹ 2 more variables: accuracy\_mean <dbl>, accuracy\_sd <dbl>

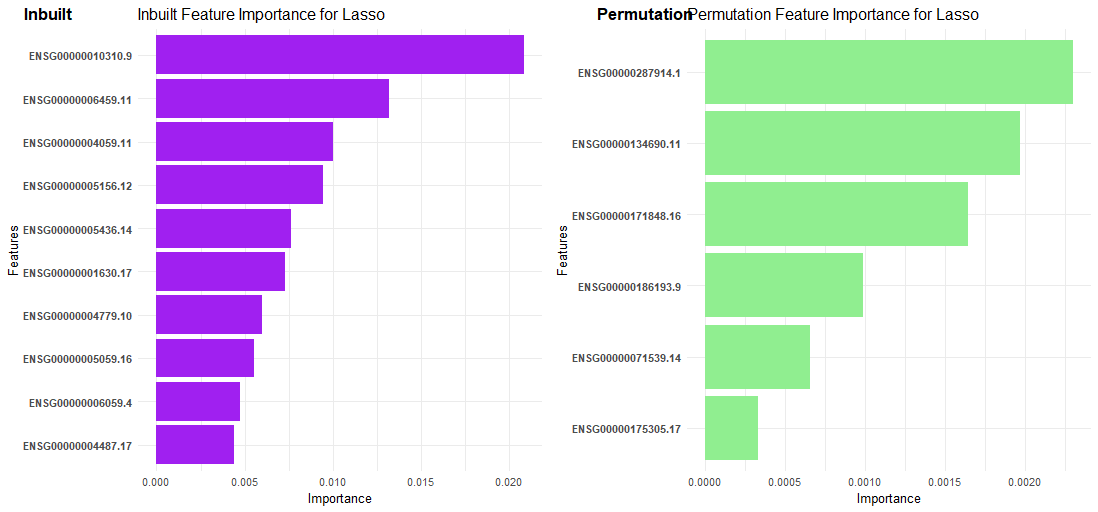
Again, boruta seems to be the best one, although marginally. We can produce a combined plot of all metrics by calling:



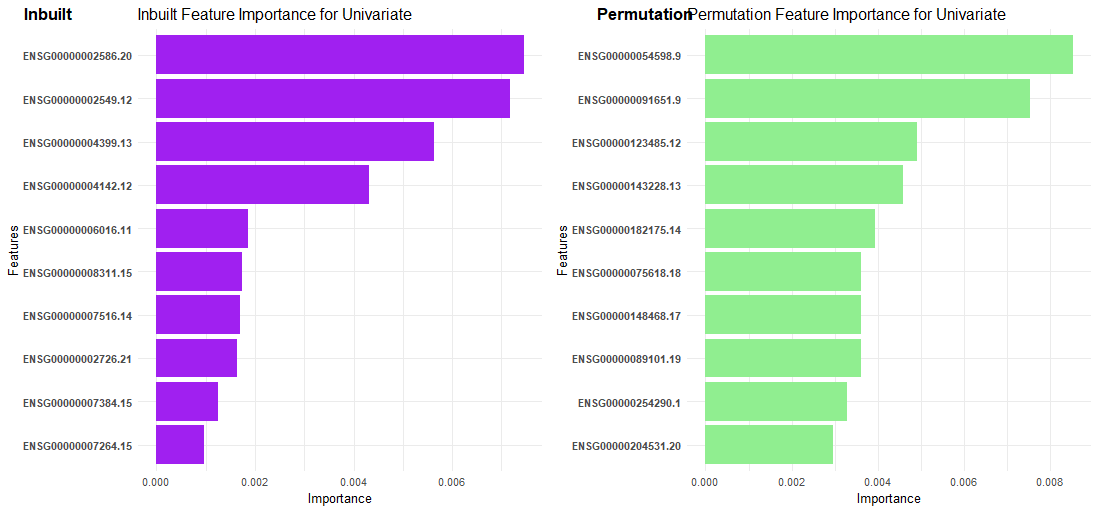
## 3.2 Feature Importance

The next step is to inspect what are the most important features for every feature selection method. To plot the importance scores we will call the plot\_feature\_importance(). The function returns a list of plots demonstrating mean feature importance scores across different data splits, so we will store it in a separate object:

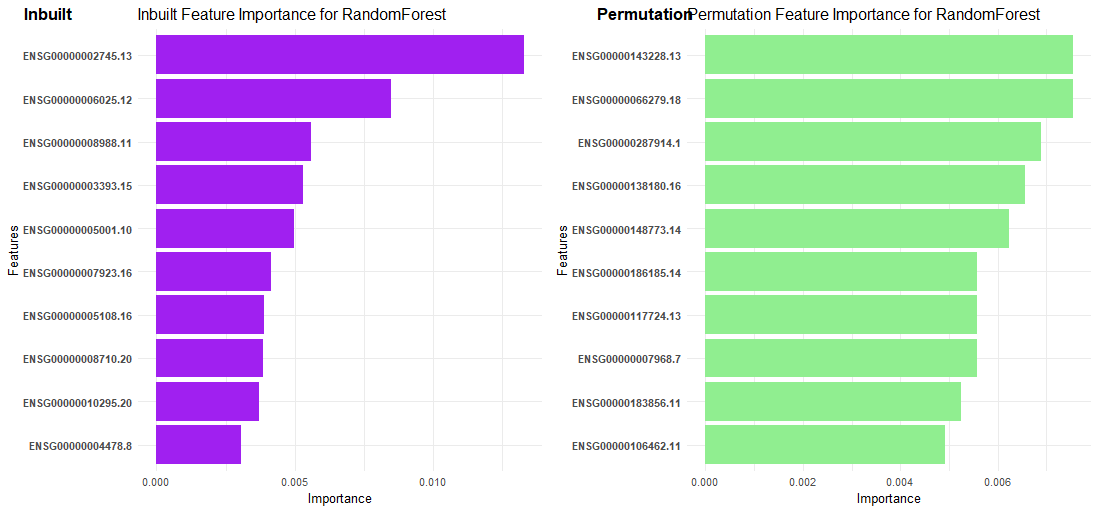
plot\_list <- plot\_feature\_importance(selection\_results)  
plot\_list  
#> $Lasso



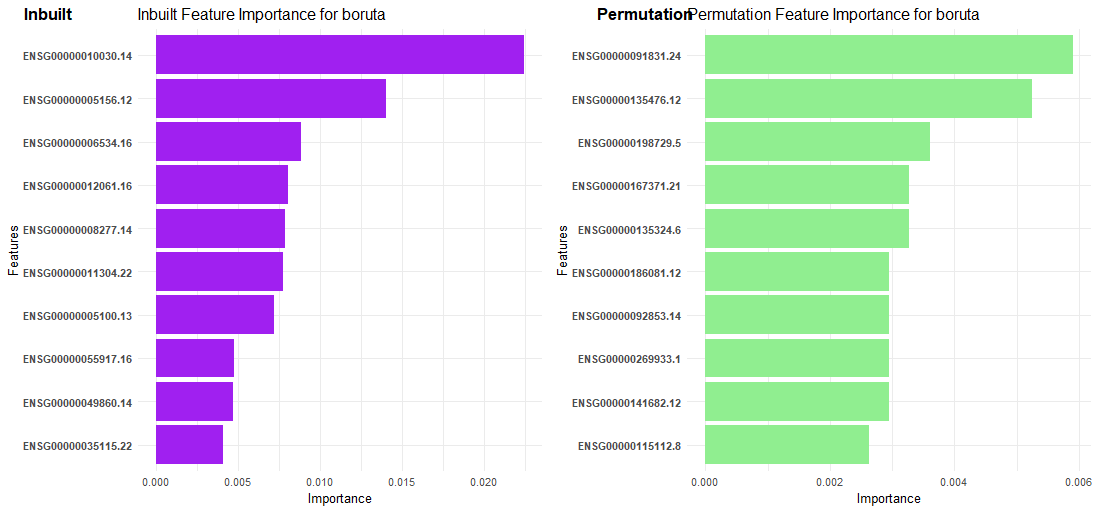
#>   
#> $Univariate



#>   
#> $RandomForest



#>   
#> $boruta



Interestingly, we can see that boruta has a lot of relevant genes to cancer as top features: ENSG00000010030 (ETV7),  
ENSG00000005156 (LIG3), ENSG00000006534 (ALDH3B1), ENSG00000005100 (DHX33).

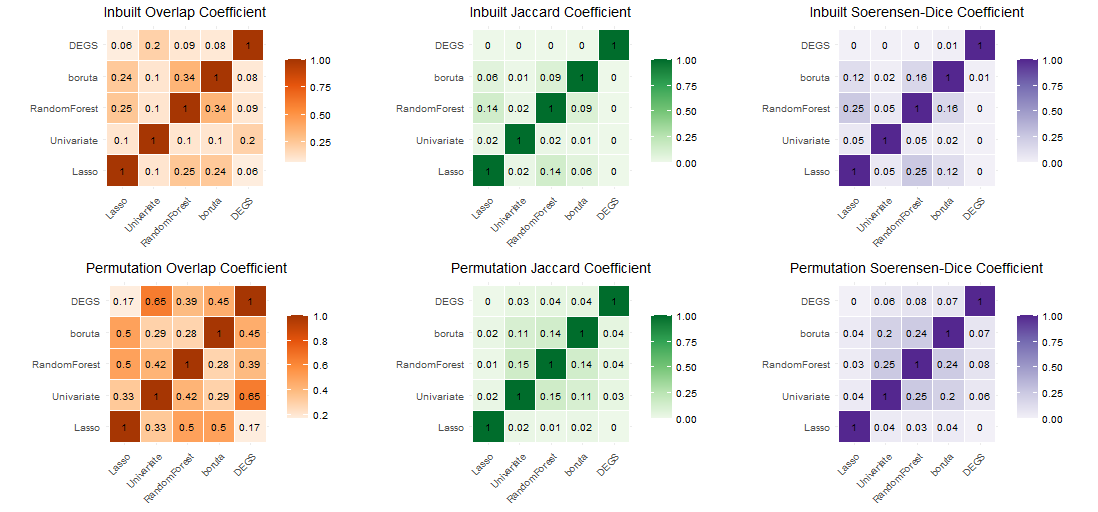
## 3.3 Overlap between Gene Lists and DGE list

It might be interesting to see if there is any overlap between the feature selection lists also including the list of differentially expressed genes. Let’s first extract the list of DEGs:

deg\_list <- rownames(filtered\_df)

Then we can calculate the overlap coefficients and plot them:

#inspect overlap with DEGs  
overlap\_degs <- calculate\_overlap\_coefficients(selection\_results, custom\_lists = list('DEGS' = deg\_list))  
plot\_overlap\_heatmaps(overlap\_degs)



We can see three different coefficients that calculate the list a bit differently. Since our lists are different in sizes, the most relevant one here is Overlap Coefficient. We can see that some of the permutation importance lists have similarities. For example, Lasso is quite similar to boruta and RandomForest. As for the DEGs list it’s quite similar to the Univariate method, although this is somewhat expected.

## 3.4 Gene Lists Annotation

For the enrichment analysis and subsequent biological interpretation it is convenient to convert gene identifiers to others that are appropriate/useful in different situations. To do so we can do the following:

# remove version number from the ensembl ids  
background <- as.character(colnames(vsd\_matrix))  
background <- gsub("\\..\*", "", background)  
custom\_list <- list('background' = background,  
 'DEGs' = deg\_list)  
  
ah <- AnnotationHub::AnnotationHub()  
human\_ens <- AnnotationHub::query(ah, c("Homo sapiens", "EnsDb"))  
human\_ens <- human\_ens[['AH98047']]  
#> loading from cache  
#> require("ensembldb")  
annotations\_ahb <- ensembldb::genes(human\_ens, return.type = "data.frame") %>%  
 dplyr::select(gene\_id,gene\_name,entrezid,gene\_biotype)  
  
annotations\_df <- annotate\_gene\_lists(pipeline\_results = selection\_results,  
 annotations\_ahb = annotations\_ahb,  
 format = 'ensembl\_id',  
 custom\_lists = custom\_list)  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:many mapping between keys and columns  
#> 'select()' returned 1:many mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:1 mapping between keys and columns  
#> 'select()' returned 1:many mapping between keys and columns  
#> 'select()' returned 1:many mapping between keys and columns  
#> 'select()' returned 1:many mapping between keys and columns

This returns an object of class AnnotatedGeneLists containing gene symbol, ENSEMBL ID and ENTREZ ID for the selected features as well as background list and DEGs list.

## 3.5 Gene Ontology Enrichment

Now we will do Gene Ontology Enrichment analysis to mine the pathways relevant to our biological question. To do so we can call the function:

# perform GO Analysis  
annotated\_GO\_inbuilt <- GO\_enrichment\_analysis(annotations\_df,  
 list\_type = 'inbuilt', #run GO enrichment on inbuilt selected features  
 keyType = 'ENSEMBL', # run analysis with ENSEMBLIDs  
 background = background,  
 ont = 'BP') # run BP ontology  
#> Performing GO Enrichment analysis for the:Lasso  
#> Performing GO Enrichment analysis for the:Univariate  
#> Performing GO Enrichment analysis for the:RandomForest  
#> Performing GO Enrichment analysis for the:boruta  
#> Performing GO Enrichment analysis for the:DEGs  
  
annotated\_GO\_permutation <- GO\_enrichment\_analysis(annotations\_df,  
 list\_type = 'permutation', #run GO enrichment on permutation based selected features  
 keyType = 'ENSEMBL', # run analysis with ENSEMBLIDs  
 background = background,  
 ont = 'BP') # run BP ontology  
#> Performing GO Enrichment analysis for the:Lasso  
#> Performing GO Enrichment analysis for the:Univariate  
#> Performing GO Enrichment analysis for the:RandomForest  
#> Performing GO Enrichment analysis for the:boruta  
#> Performing GO Enrichment analysis for the:DEGs

So far inbuilt feature importance looked promising, so now we can inspect what GO terms are enriched in every gene list:

plot\_enrichment <- function (GO\_df) {

ggplot(data = GO\_df@result %>%   
 arrange(pvalue) %>%   
 head(10) %>%   
 mutate(GeneCount = as.numeric(gsub("/.\*$", "", GeneRatio))),  
 aes(x = reorder(Description, -log10(pvalue)), y = -log10(pvalue))) +  
 geom\_point(aes(size = GeneCount, color = pvalue)) +  
 scale\_color\_gradient(low = "red", high = "blue") +  
 scale\_size\_continuous(range = c(3, 9)) +  
 coord\_flip() +  
 xlab("Top 10 GO Term Descriptions") +  
 ylab("-log10(pvalue)") +  
 ggtitle('Univariate List') +  
 theme\_minimal()

**}**

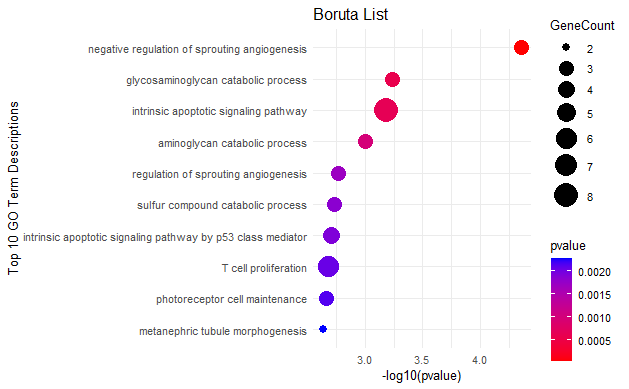
plot\_enrichment(annotated\_GO\_inbuilt$boruta)

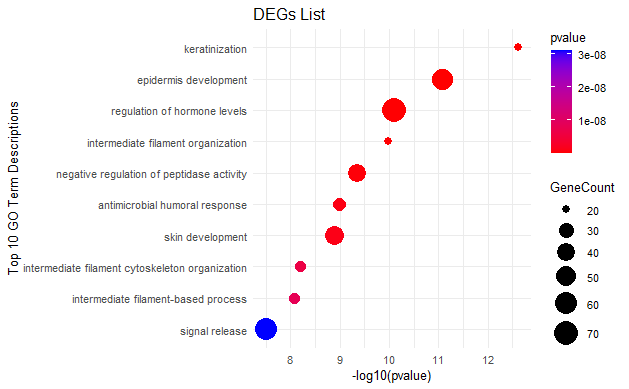
plot\_enrichment(annotated\_GO\_inbuilt$DEGs)

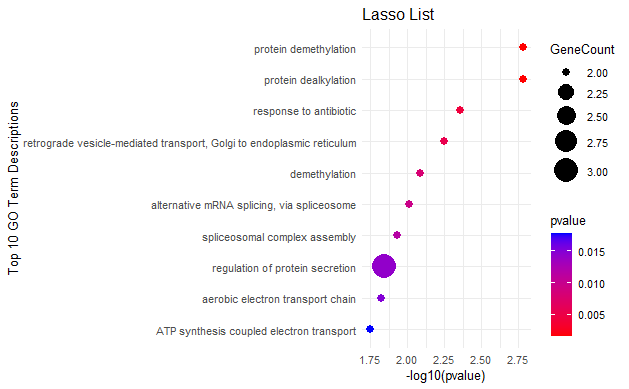
plot\_enrichment(annotated\_GO\_inbuilt$Lasso)

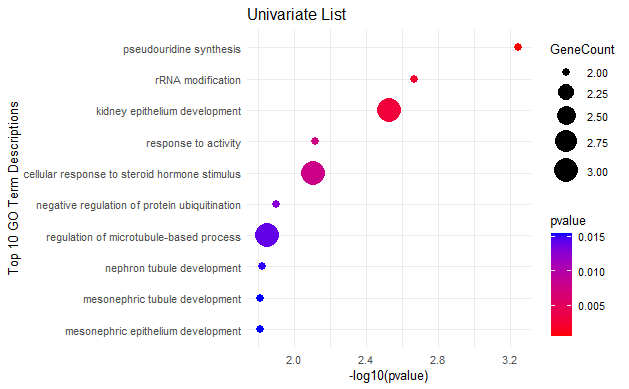
plot\_enrichment(annotated\_GO\_inbuilt$Univariate)

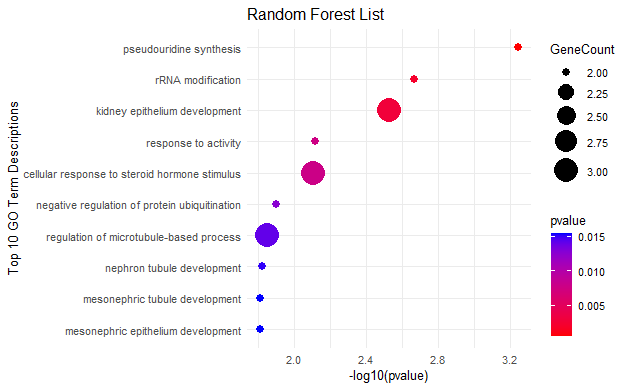
plot\_enrichment(annotated\_GO\_inbuilt$RandomForest)









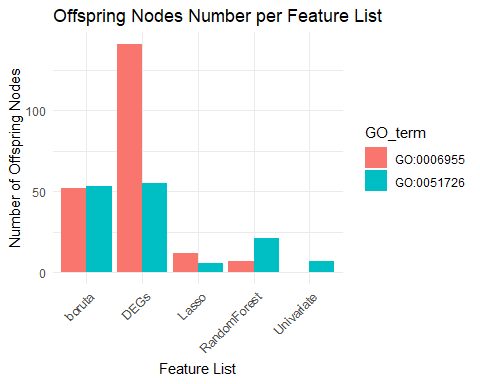
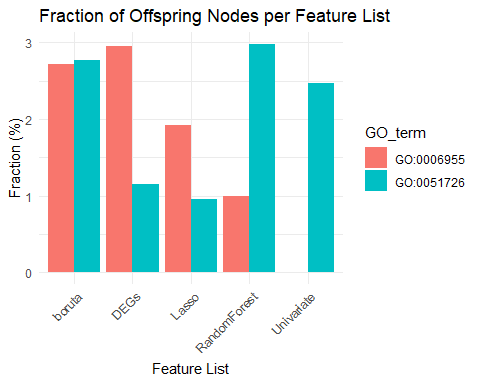


Looking at top 10 enriched pathways, boruta seems to be the one that has the most relevant ones. For example, we can see here sprouting angiogenesis regulation terms that is related to metastasis spread. Additionally, such terms as glycosaminoglycan metabolic process and especially apoptotic p53 pathway are all of high relevance in relation to cancer.

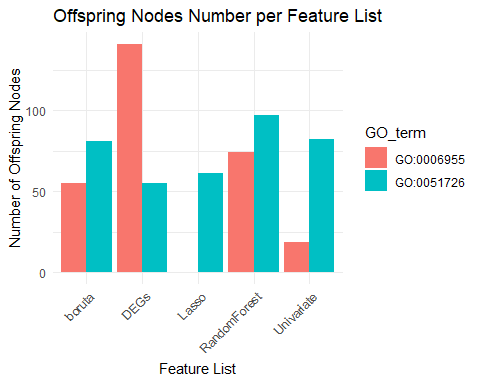
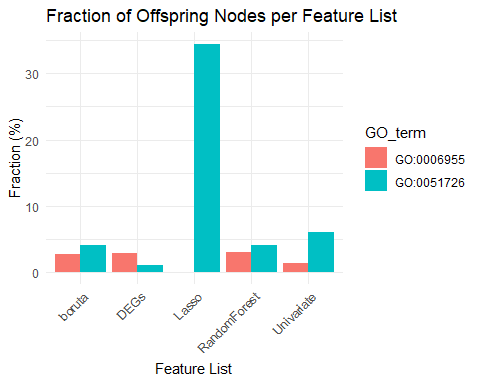
## 3.6 Quantification of Children Nodes of Parent Node of Interest

Additionally, we can quantify how many of the children nodes of a parent node of interest are there in our list. For example, we can take two relevant broad GO Biological Process (BP) terms that are cell cycle regulation (<GO:0051726>) and immune response (<GO:0006955>).

annot\_child\_fractions\_inbuilt <- compute\_GO\_child\_term\_metrics(GO\_data = annotated\_GO\_inbuilt,  
 GO\_terms = c("GO:0051726", "GO:0006955"),  
 plot = TRUE)



annot\_child\_fractions\_permut <- compute\_GO\_child\_term\_metrics(GO\_data = annotated\_GO\_permutation,  
 GO\_terms = c("GO:0051726", "GO:0006955"),  
 plot = TRUE)

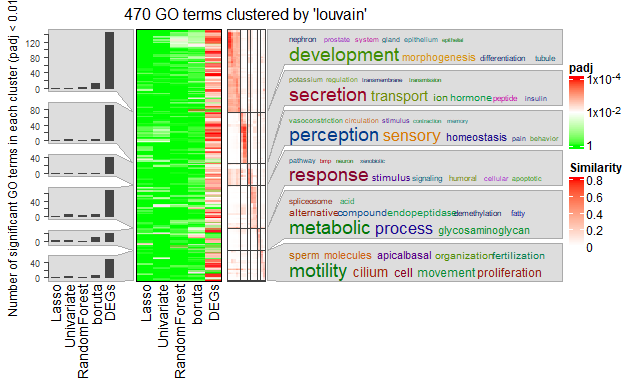


Here, once again we see that in terms of fractions of the parent terms of interest, boruta seems to be performing very well for both ‘immune process’ and ‘cell cycle regulation terms’.

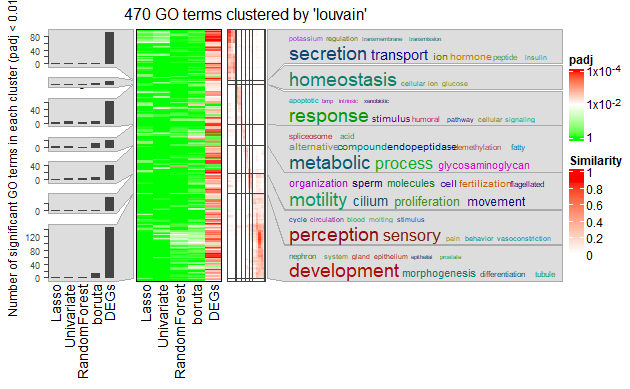
## 3.7 Semantic Similarity Analysis

Finally, we can now perform semantic similarity (SS) analysis of identified GO terms. To do that we will run this:

# perform SS analysis  
hmap\_inbuilt <- run\_simplify\_enrichment(annotated\_GO\_inbuilt,  
 method = 'louvain',  
 measure = 'Rel',  
 ont = 'BP',  
 padj\_column = 'pvalue',  
 padj\_cutoff = 0.01)  
#> Use column 'ID' as `go\_id\_column`.  
#> Loading required namespace: gridtext  
#> 470/5074 GO IDs left for clustering.  
#> Cluster 470 terms by 'louvain'... 7 clusters, used 0.07560682 secs.  
#> Perform keywords enrichment for 6 GO lists...



hmap\_permutation <- run\_simplify\_enrichment(annotated\_GO\_permutation,  
 method = 'louvain',  
 measure = 'Jiang',  
 ont = 'BP',  
 padj\_column = 'pvalue',  
 padj\_cutoff = 0.01)  
#> Use column 'ID' as `go\_id\_column`.  
#> 470/5074 GO IDs left for clustering.  
#> Cluster 470 terms by 'louvain'... 9 clusters, used 0.04021907 secs.  
#> Perform keywords enrichment for 7 GO lists...



After performing the SS analysis, we can observe that there are 6 clusters in total. By examining word clouds there are two clusters of interest: cluster 4 containing humoral immunity and cell signal transduction terms, and cluster 5 that contains terms related to glycosaminoglycan and other metabolism-related terms. If we look at the signifcance and fraction heatmap on the left we see that while DEGs list is significant everywhere, without any specificity, boruta is singificantly enriched in these two clusters.

# 4. Picking the Winner and Conclusion

After having completed the entire analysis we can now pick the winner among the lists. From different points of view boruta seems to be the best because: 1. It has highest ML performance on both CV and test sets; 2. Boruta has more relevant GO BP terms for differentiation between molecular subtypes that are related to cancer cell metabolism, apoptosis and angiogenesis; 3. It has higher fractions of parent terms of interest in comparison to other lists;  
4. In terms of Semantic Similarity analysis it is more specific to immune response-related terms and metabolism terms, as compared to DEGs that result in broad GO BP terms; 5. In comparison to DEGs list it is much shorter and more manageable for the downstream analysis and interpretation With this boruta seems to be the best candidate for downstream analysis and probably targeted experiments to establish biomarkers. With all this in mind, we can clearly say that the most suitable list for the further investigation is boruta.

utils::sessionInfo()  
#> R version 4.3.1 (2023-06-16 ucrt)  
#> Platform: x86\_64-w64-mingw32/x64 (64-bit)  
#> Running under: Windows 11 x64 (build 22000)  
#>   
#> Matrix products: default  
#>   
#>   
#> locale:  
#> [1] LC\_COLLATE=Russian\_Kazakhstan.utf8 LC\_CTYPE=Russian\_Kazakhstan.utf8   
#> [3] LC\_MONETARY=Russian\_Kazakhstan.utf8 LC\_NUMERIC=C   
#> [5] LC\_TIME=Russian\_Kazakhstan.utf8   
#>   
#> time zone: Europe/Paris  
#> tzcode source: internal  
#>   
#> attached base packages:  
#> [1] stats4 stats graphics grDevices utils datasets methods   
#> [8] base   
#>   
#> other attached packages:  
#> [1] ensembldb\_2.24.0 AnnotationFilter\_1.24.0   
#> [3] GenomicFeatures\_1.52.1 AnnotationDbi\_1.62.2   
#> [5] ggplot2\_3.4.2 DESeq2\_1.40.2   
#> [7] SummarizedExperiment\_1.30.2 Biobase\_2.60.0   
#> [9] MatrixGenerics\_1.12.2 matrixStats\_1.0.0   
#> [11] GenomicRanges\_1.52.0 GenomeInfoDb\_1.36.1   
#> [13] IRanges\_2.34.1 S4Vectors\_0.38.1   
#> [15] BiocGenerics\_0.46.0 GeneSelectR\_0.0.0.9000   
#> [17] dplyr\_1.1.2   
#>   
#> loaded via a namespace (and not attached):  
#> [1] ProtGenerics\_1.32.0 bitops\_1.0-7   
#> [3] enrichplot\_1.20.0 HDO.db\_0.99.1   
#> [5] httr\_1.4.6 RColorBrewer\_1.1-3   
#> [7] doParallel\_1.0.17 tools\_4.3.1   
#> [9] utf8\_1.2.3 R6\_2.5.1   
#> [11] lazyeval\_0.2.2 GetoptLong\_1.0.5   
#> [13] withr\_2.5.0 prettyunits\_1.1.1   
#> [15] gridExtra\_2.3 plotwidgets\_0.5.1   
#> [17] cli\_3.6.1 scatterpie\_0.2.1   
#> [19] labeling\_0.4.2 slam\_0.1-50   
#> [21] tm\_0.7-11 commonmark\_1.9.0   
#> [23] Rsamtools\_2.16.0 yulab.utils\_0.0.6   
#> [25] gson\_0.1.0 DOSE\_3.26.1   
#> [27] rstudioapi\_0.15.0 RSQLite\_2.3.1   
#> [29] generics\_0.1.3 gridGraphics\_0.5-1   
#> [31] shape\_1.4.6 BiocIO\_1.10.0   
#> [33] gtools\_3.9.4 GO.db\_3.17.0   
#> [35] Matrix\_1.5-4.1 fansi\_1.0.4   
#> [37] lifecycle\_1.0.3 yaml\_2.3.7   
#> [39] gplots\_3.1.3 qvalue\_2.32.0   
#> [41] BiocFileCache\_2.8.0 grid\_4.3.1   
#> [43] blob\_1.2.4 promises\_1.2.0.1   
#> [45] crayon\_1.5.2 lattice\_0.21-8   
#> [47] cowplot\_1.1.1 KEGGREST\_1.40.0   
#> [49] magick\_2.7.5 pillar\_1.9.0   
#> [51] knitr\_1.43 ComplexHeatmap\_2.16.0   
#> [53] fgsea\_1.26.0 rjson\_0.2.21   
#> [55] codetools\_0.2-19 fastmatch\_1.1-3   
#> [57] glue\_1.6.2 downloader\_0.4   
#> [59] ggfun\_0.1.1 data.table\_1.14.8   
#> [61] vctrs\_0.6.3 png\_0.1-8   
#> [63] treeio\_1.24.1 testthat\_3.1.10   
#> [65] gtable\_0.3.3 cachem\_1.0.8   
#> [67] xfun\_0.39 S4Arrays\_1.0.4   
#> [69] mime\_0.12 tidygraph\_1.2.3   
#> [71] pheatmap\_1.0.12 iterators\_1.0.14   
#> [73] interactiveDisplayBase\_1.38.0 ellipsis\_0.3.2   
#> [75] nlme\_3.1-162 simplifyEnrichment\_1.10.0   
#> [77] ggtree\_3.8.0 bit64\_4.0.5   
#> [79] progress\_1.2.2 filelock\_1.0.2   
#> [81] KernSmooth\_2.23-22 colorspace\_2.1-0   
#> [83] DBI\_1.1.3 tidyselect\_1.2.0   
#> [85] proxyC\_0.3.3 bit\_4.0.5   
#> [87] compiler\_4.3.1 curl\_5.0.1   
#> [89] xml2\_1.3.5 NLP\_0.2-1   
#> [91] DelayedArray\_0.26.6 shadowtext\_0.1.2   
#> [93] rtracklayer\_1.60.0 scales\_1.2.1   
#> [95] caTools\_1.18.2 rappdirs\_0.3.3   
#> [97] stringr\_1.5.0 digest\_0.6.33   
#> [99] tmod\_0.50.13 rmarkdown\_2.23   
#> [101] XVector\_0.40.0 htmltools\_0.5.5   
#> [103] pkgconfig\_2.0.3 highr\_0.10   
#> [105] dbplyr\_2.3.3 fastmap\_1.1.1   
#> [107] rlang\_1.1.1 GlobalOptions\_0.1.2   
#> [109] shiny\_1.7.4.1 farver\_2.1.1   
#> [111] jsonlite\_1.8.7 BiocParallel\_1.34.2   
#> [113] GOSemSim\_2.26.1 RCurl\_1.98-1.12   
#> [115] magrittr\_2.0.3 GenomeInfoDbData\_1.2.10   
#> [117] ggplotify\_0.1.1 patchwork\_1.1.2   
#> [119] munsell\_0.5.0 Rcpp\_1.0.11   
#> [121] ape\_5.7-1 viridis\_0.6.3   
#> [123] reticulate\_1.32.0.9000 stringi\_1.7.12   
#> [125] tagcloud\_0.6 ggraph\_2.1.0   
#> [127] brio\_1.1.3 zlibbioc\_1.46.0   
#> [129] MASS\_7.3-60 AnnotationHub\_3.8.0   
#> [131] plyr\_1.8.8 org.Hs.eg.db\_3.17.0   
#> [133] parallel\_4.3.1 ggrepel\_0.9.3   
#> [135] Biostrings\_2.68.1 graphlayouts\_1.0.0   
#> [137] splines\_4.3.1 gridtext\_0.1.5   
#> [139] hms\_1.1.3 circlize\_0.4.15   
#> [141] locfit\_1.5-9.8 igraph\_1.5.0   
#> [143] markdown\_1.7 reshape2\_1.4.4   
#> [145] biomaRt\_2.56.1 BiocVersion\_3.17.1   
#> [147] XML\_3.99-0.14 evaluate\_0.21   
#> [149] RcppParallel\_5.1.7 BiocManager\_1.30.21   
#> [151] foreach\_1.5.2 tweenr\_2.0.2   
#> [153] httpuv\_1.6.11 tidyr\_1.3.0   
#> [155] purrr\_1.0.1 polyclip\_1.10-4   
#> [157] clue\_0.3-64 ggforce\_0.4.1   
#> [159] xtable\_1.8-4 restfulr\_0.0.15   
#> [161] tidytree\_0.4.2 later\_1.3.1   
#> [163] viridisLite\_0.4.2 tibble\_3.2.1   
#> [165] clusterProfiler\_4.8.2 aplot\_0.1.10   
#> [167] memoise\_2.0.1 beeswarm\_0.4.0   
#> [169] GenomicAlignments\_1.36.0 cluster\_2.1.4