Presentation Project Head and Neck Cancer

Nils Mechtel, Tobias Hub, Niklas Urbanek, Pascal Poc

24. July 2019

Predicting Cancer-Specific Vulnerability via DAISY System

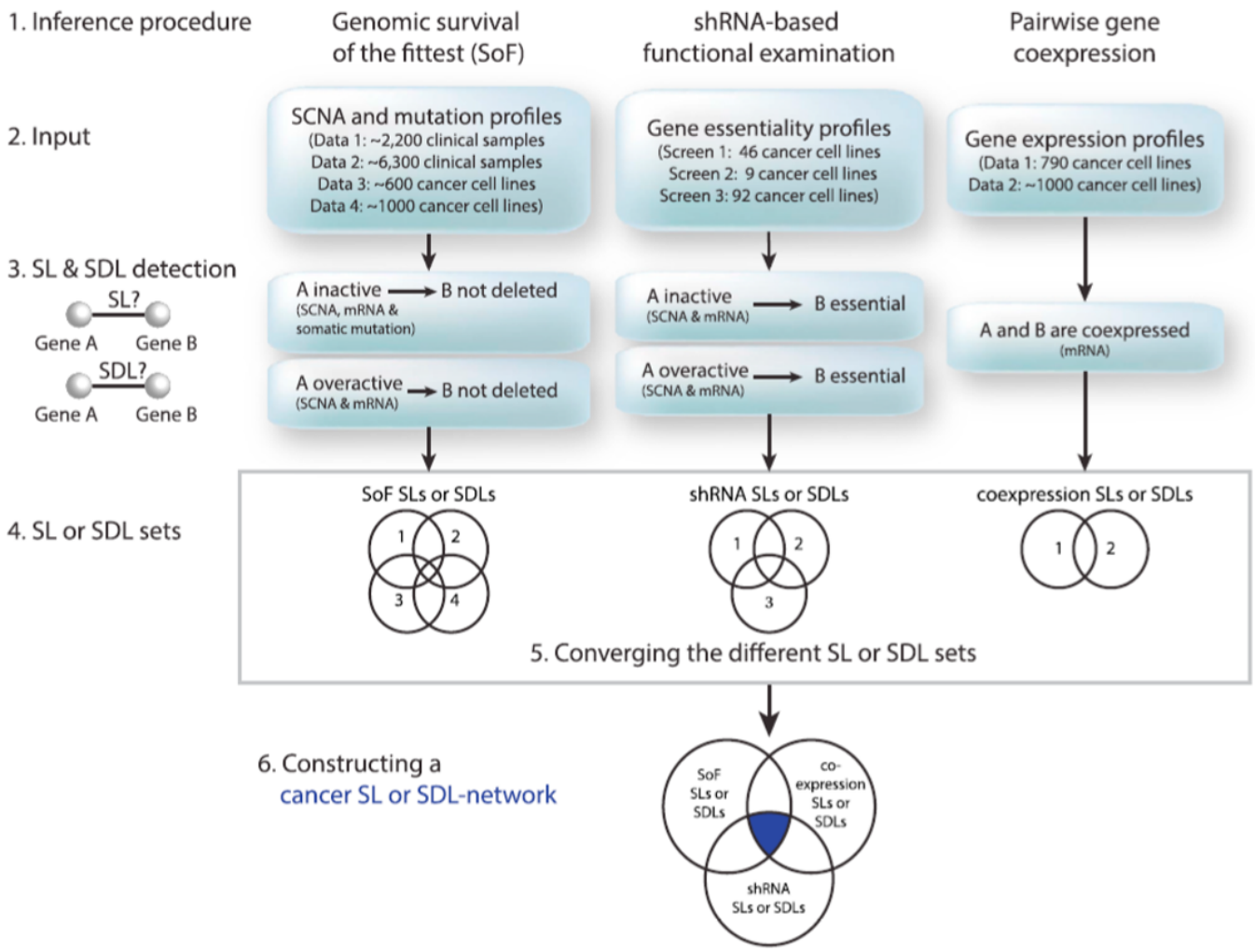


Figure 1: Daisy-Workflow

Milestones

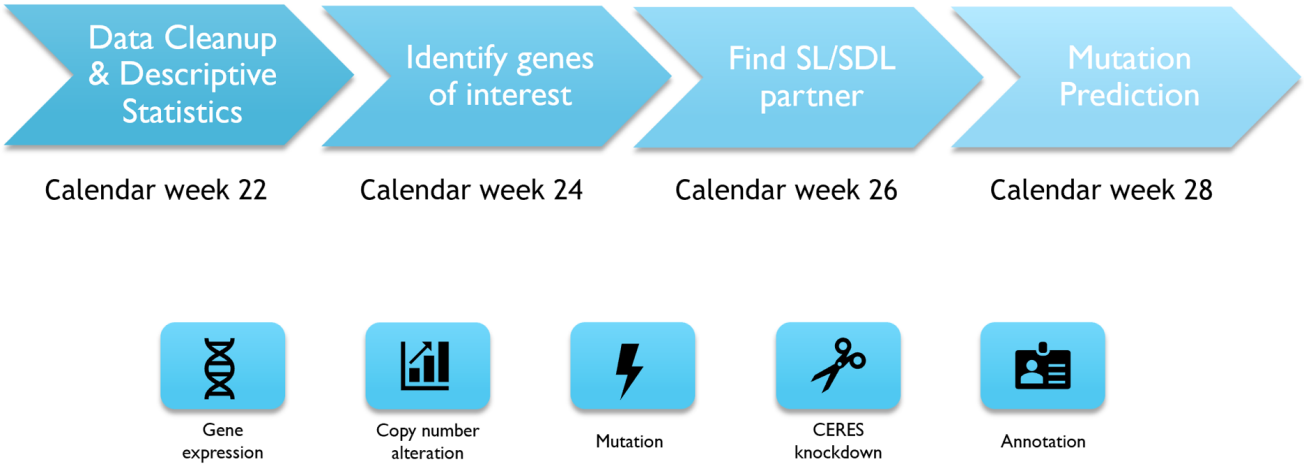


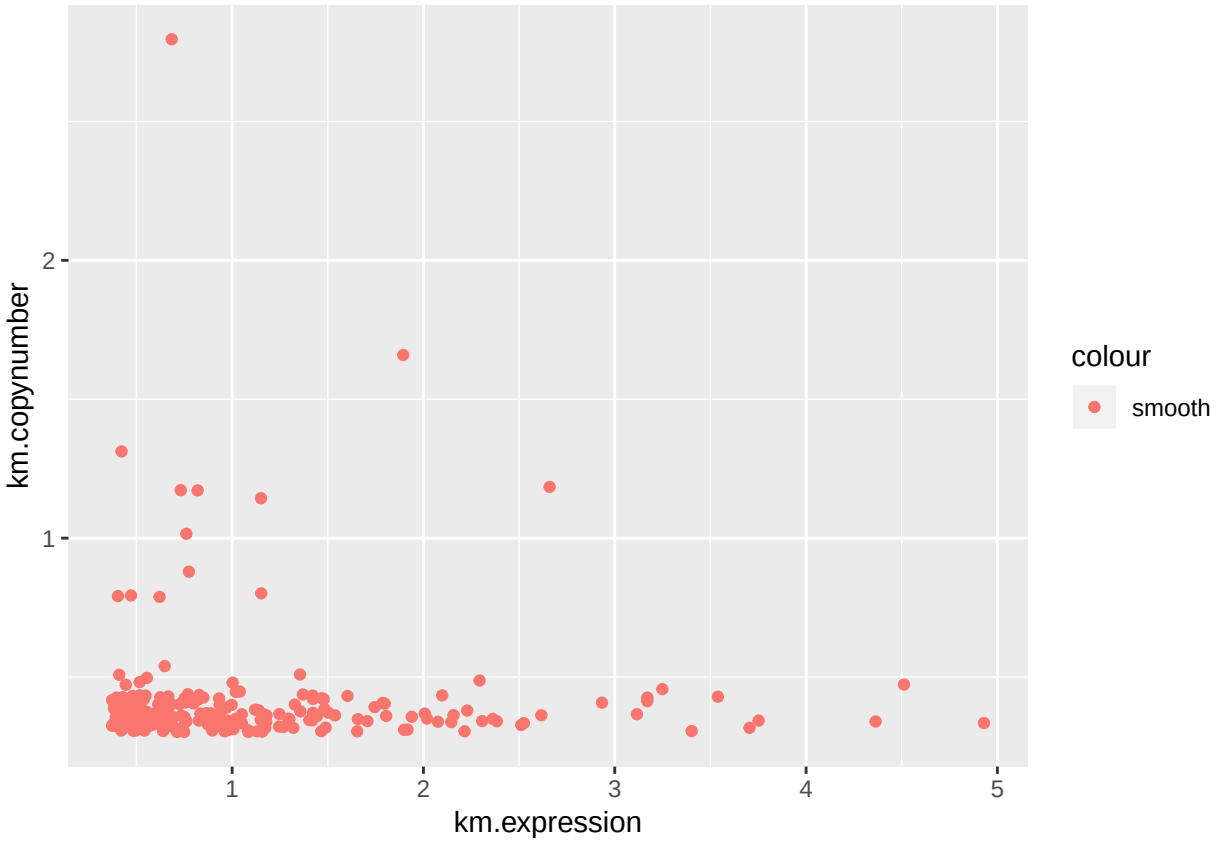
Figure 2: Milestones of our project

Identify genes of interest

K-means clustering of CNA and expression data with normalized values for SDL GOI

To find potential genes for SDL pair detection, clustering through k-means was executed.

```
# Intersect set of expression and CNA
km.genes = Reduce(intersect, list(rownames(expression.HNC), rownames(copynumber.HNC)))

# Create dataframe with row means of expression and CNA values
km.expression = rowMeans(expr.HNC.norm[km.genes,])
km.copynumber = rowMeans(cna.HNC.norm[km.genes,])

# Combine expression and CNA valuzes for k mean clustering
comb.ex.cna = cbind(km.expression, km.copynumber)
# Converting Matrix into a data frame
comb.ex.cna = as.data.frame(comb.ex.cna, row.names = NULL, optional = FALSE,
              cut.names = FALSE, fix.empty.names = TRUE,
              stringsAsFactors = default.stringsAsFactors())

comb.ex.cna = comb.ex.cna[which(comb.ex.cna$km.copynumber >= 0.3 &
                                comb.ex.cna$km.expression >= quantile(comb.ex.cna$km.expression,0.9)),]

# Plotting of normalized expression valuess against CNA data for a short overview
ggplot(comb.ex.cna[,1:2],
       aes(x = km.expression, y = km.copynumber, color = "smooth")) +
  geom_point()
```



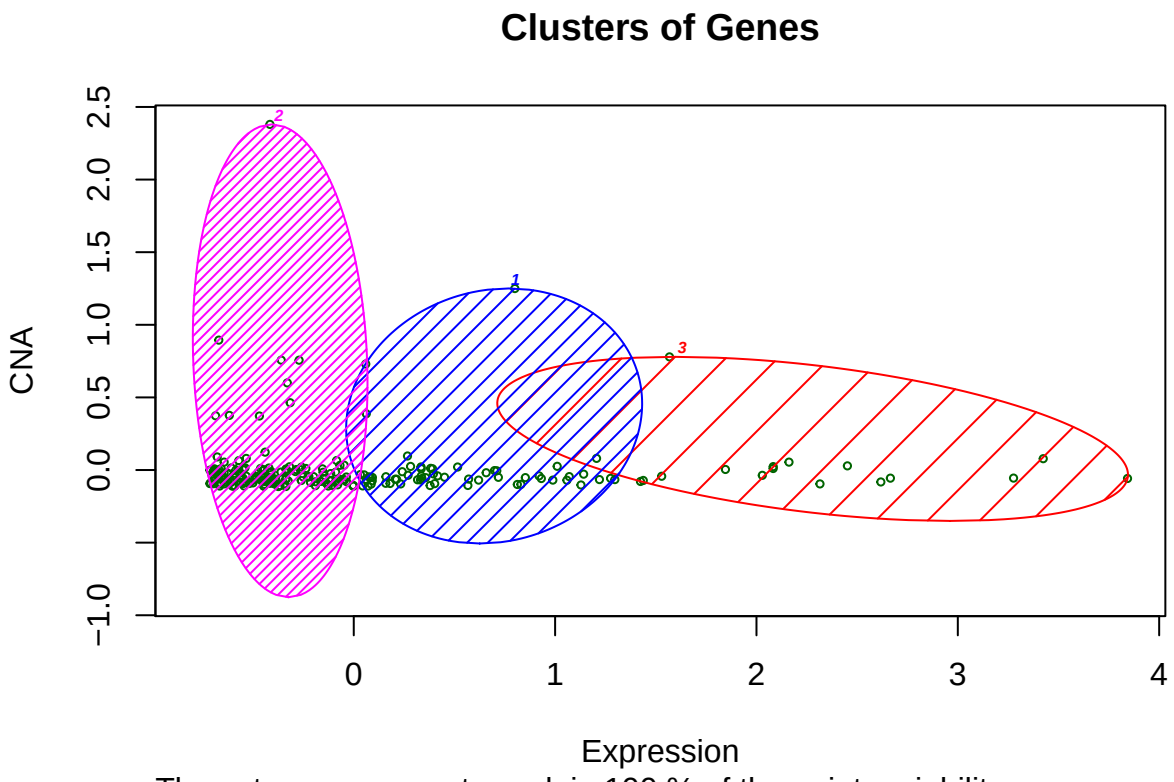
To check the perfect ammount of clusters in k-means, an ellbow test was implemented.

```
# Ellbow method: to find optimal ammount of clusters
emp = vector() # Create an empty vector
# Calculate the sum within squared values for 10 clusters
for (i in 1:10) emp[i] = sum(kmeans(comb.ex.cna[1:2], i)$withinss)

# Plotting of Ellbow method
plot(1:10,
     emp,
     type = 'b',
     main = paste('The Elbow Method'),
     xlab = 'Number of clusters',
     ylab = 'Sum of squares')
```

Result: The ellbow test shows that 3 clusters are sufficient.

K-means were executed and plotted. Afterwards the potential SDL candidates were identified through higher expression values.

```
# Fitting k-means to the dataset - elbow method (k = 3)
kmeans = kmeans(x = comb.ex.cna[,1:2], centers = 3,iter.max = 1000, nstart = 1)
y_kmeans = kmeans$cluster

# Visualising the clusters
clusplot(comb.ex.cna,
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 4,
         plotchar = FALSE,
         span = TRUE,
         stand = FALSE,
         cex =0.5,
         main = paste('Clusters of Genes'),
         xlab = 'Expression',
         ylab = 'CNA')
```



The Plot "Clusters of Genes" features the 3 clusters of values from expression and CNA data. Due to little variance between the data pairs, the intersection set is comparatively big.

In order to define SDL candidates, the cluster with highest expression values was chosen and filtered again according to expression numbers.

```
comb.ex.cna = cbind(comb.ex.cna, y_kmeans)# Combine kmeans with genes
# Sort comb.ex.cna data decresing for selecting adequate number of cluster
comb.ex.cna = comb.ex.cna[order(comb.ex.cna$km.expression,decreasing =TRUE),]

# Filtering of the most expressed and alterated genes
goi.sdl = comb.ex.cna[which(comb.ex.cna$y_kmeans == comb.ex.cna[1,3]),]

# Sort dataframe
goi.sdl = goi.sdl[order(goi.sdl$km.expression,decreasing=TRUE),]
kable(goi.sdl[1:5,])
```

	km.expression	km.copynumber	y_kmeans
LCN2	4.929810	0.3341841	3
TP63	4.511925	0.4728786	3
GPR87	4.363817	0.3399337	3
IFI27	3.752208	0.3430111	3
SOX15	3.704685	0.3169922	3

```r
goi.SDL = rownames(goi.sdl)[1:5]
```

Investigate high frequencies of deleterious mutations SL GOI

Create a dataframe with data for the heatmap.

```r
# All mutation data of HNC samples in one dataframe
mutation.HNC.df = do.call(rbind, mutation.HNC)
# Dataframe with all mutated genes in HNC samples just once
heatmap.genes = as.data.frame(table(mutation.HNC.df$Hugo_Symbol))

heatmap.data = as.data.frame(sapply(1:(length(ID)), function(a) {
  # Per HNC sample: every gene which is mutated gets a 1, otherwise a 0
  out = ifelse(heatmap.genes$Var1 %in% mutation.HNC[[a]]$Hugo_Symbol, 1, 0)
  return(out)
}))

# Define row and column names
rownames(heatmap.data) = heatmap.genes$Var1
colnames(heatmap.data) = ID

# Order from high to low frequency
heatmap.data$sum = apply(heatmap.data, 1, sum)
heatmap.data = heatmap.data[order(-heatmap.data$sum), 1:27]

# Heatmap of the most frequently deleterious mutated genes in HNC samples
heatmap(data.matrix(heatmap.data)[1:40,],
        Rowv=NA,
        Colv=NA,
        col = c("lightgrey", "tomato2"),
        scale="column",
        cexRow = 0.8,
        cexCol = 0.8,
        main = "Most frequently deleterious mutated genes in HNC samples")
```

Most frequently deleterious mutated genes in HNC samples

Result: TP53, FAT1, CDKN2A, KRT3 and CASP8 are the 5 most frequently deleterious mutated genes in our HNC samples.

According to Leemans et al. (2018) these are frequent mutations in HNC:

Cellular process	Gene	Protein	Type of gene	Mutation frequency (%)	CNA frequency (%)
Cell cycle	*CDKN2A*	$p16^{INK4A}$	Tumour suppressor	22	32
	TP53	p53	Tumour suppressor	72	1.4
	CCND1	G1–S-specific cyclin D1	Oncogene	0.6	25
Growth signals	*EGFR*	EGFR	Oncogene	4	11
Survival	*PIK3CA*	Catalytic p110α subunit of class 1 PI3Ks	Oncogene	18	21
	PTEN	PTEN	Tumour suppressor	3	4
WNT signalling	*FAT1*	Protocadherin FAT1	Tumour suppressor	23	8
	AJUBA	LIM domain-containing protein AJUBA	Tumour suppressor	7*	1
	NOTCH1	NOTCH1	Tumour suppressor	18	4
Epigenetic regulation	*KMT2D*	Histone-lysine *N*-methyltransferase KMT2D	Tumour suppressor	16	0.4
	NSD1	Histone-lysine *N*-methyltransferase NSD1	Tumour suppressor	12*	0.8

Data from REF. 3. Mutation data were taken from The Cancer Genome Atlas (TCGA) ($n = 504$) using the cBioPortal. CNA, copy number alteration; EGFR, epidermal growth factor receptor. *Putative passenger mutation that requires further functional studies.

Figure 3: Genes with frequent and highly significant somatic genetic changes in HNSCC

Compare our SL GOI with genes from literature.

```
# create dataframe with goi from literature
goi.lit = data.frame(Gene = c("CDKN2A", "TP53", "CCND1", "EGFR", "PIK3CA", "PTEN", "FAT1",
                              "AJUBA", "NOTCH1", "KMT2D", "NSD1"),
                     "Mutation.frequency" = c(22, 72, 0.6, 4, 18, 3, 23, 7, 18, 16, 12),
                     "CNA.frequency" = c(32, 1.4, 25, 11, 21, 4, 8, 1, 4, 0.4, 0.8))

# Search for the position of goi.lit in mutated genes of HNC samples
heatmap_position = sapply(1:nrow(goi.lit), function(a) {
  position = which(rownames(heatmap.data) == goi.lit$Gene[a])
  # If gene is not mutated in HNC samples its position is replaced with NA
  if(length(position) == 0) {position = NA}
  return(position)
})

goi.lit$"Heatmap.position" = heatmap_position # Add heatmap position to goi.lit
rownames(goi.lit) = goi.lit$Gene
goi.lit$Gene = NULL
goi.lit = goi.lit[order(goi.lit$"Heatmap.position"),]
kable(goi.lit) # Show a table of goi.lit
```

	Mutation.frequency	CNA.frequency	Heatmap.position
TP53	72.0	1.4	1
FAT1	23.0	8.0	2
CDKN2A	22.0	32.0	3
AJUBA	7.0	1.0	28
KMT2D	16.0	0.4	66
NOTCH1	18.0	4.0	82
NSD1	12.0	0.8	715
CCND1	0.6	25.0	NA
EGFR	4.0	11.0	NA
PIK3CA	18.0	21.0	NA

	Mutation.frequency	CNA.frequency	Heatmap.position
PTEN	3.0	4.0	NA

Because our most frequent mutated genes aproximately match with frequent mutations from literature we choose the 5 genes with most frequent mutations in HNC samples.

```
goi.SL = rownames(heatmap.data)[1:5]
goi.SL
```

```
[1] "TP53"   "FAT1"   "CDKN2A" "KRT3"   "CASP8"
```

Find SL/SDL partner

SL/SDL partner search following the daisy model

In further steps all test sets get converged and only genes which pass each test significantly are chosen as a SL/SDL partner. Therefore, we only need to apply the tests on those genes which are present in all three data inputs.

```
Daisy.genes = Reduce(intersect,
                          list(rownames(copynumber), # For SoF
                               rownames(kd.ceres), # For functional examination
                               rownames(expression) # For gene coexpression
                          ))
```

To examine gene pairs A and B which fulfill the criteria of SoF and functional examination we performe a Wilcoxon rank sum test on copynumber or CERES knockdown data. We have 5 inactive genes (goi.SL) and 5 overactive genes (goi.SDL) which are defined as genes A. The Wilcoxon rank sum test now returns all genes B which pass the test in a significant manner ($p<0.05$).

```
'Daisy.Wilcox' = function(input_genes,input_data, inactive_overactive, less_greater){
  input_data = input_data[Daisy.genes,]
  Daisy_Wilcox = lapply(1:length(input_genes), function(m) {
  goi = input_genes[m] # Set 1 of 5 goi
  # Depending on SL or SDL partner search
  # (1) gene A is overactive (SDL)
  if(inactive_overactive == 1) { # inactive_overactive == 1 chooses overactive
    mean.exp.goi = mean(t(expression[goi,]))
    exp.goi.norm = expression[goi,] - mean.exp.goi # Normalised expression values of goi
    mut_overa.ID = lapply(1:ncol(expression), function(a) {
      # Filter all IDs which are overactive: overexpressed and CNA > 0,3
      out = if(exp.goi.norm[goi, a] > 0 & copynumber[goi,a] > 0.3) {a} else {NA}
      return(out)
    })
  # (2) gene A is inactive (SL)
  } else { # inactive_overactive == 0 chooses inactive
    mut_overa.ID = lapply(1:length(mutation), function(a) {
      # Filter all IDs which have a mutation of goi
      out = ifelse(goi %in% mutation[[a]]$Hugo_Symbol, a, NA)
      return(out)
    })
  }

  # ID of samples which have NO inactivity/overactivity of goi
  not_mut_overa.ID = grep("NA", mut_overa.ID)
  # ID of samples which HAVE inactivity/overactivity of goi
  mut_overa.ID = as.integer(mut_overa.ID[is.na(mut_overa.ID) == FALSE])

  p.value = sapply(1:nrow(input_data), function(b) {
    mut_overa.data = t(input_data[b, mut_overa.ID])
    not_mut_overa.data = t(input_data[b, not_mut_overa.ID])
    if (less_greater == 1) {
      # One sided (greater) Wilcoxon rank sum test
      p = wilcox.test(mut_overa.data, not_mut_overa.data, alternative = "greater")$p.value
```

```
    } else {
      # One sided (less) Wilcoxon rank sum test
      p = wilcox.test(mut_overa.data, not_mut_overa.data, alternative = "less")$p.value
    }

  })

  out = data.frame(genes = rownames(input_data), p_value = p.value)
  out = out[which(out$p_value < 0.05),]
  return(out)
  })
  names(Daisy_Wilcox) = input_genes
  return(Daisy_Wilcox)
}
```

To examine gene pairs A and B which fulfill the criteria of gene co expresssion we performe a Spearman correlation test on the expression data. We have 5 inactive genes (goi.SL) and 5 overactive genes (goi.SDL) which are defined as genes A The correlation test now returns all genes B that correlate to gene A in a significant manner ($p<0.05$).

```
'Daisy.Spearman' = function(input_genes){
  Daisy.expression = expression[Daisy.genes,]
  # Build up a list with the 5 goi; lapply: goes through the code for each gene A
  Daisy_Spearman = lapply(1:length(input_genes), function(m) {
  goi = input_genes[m]

  # Creation of a list with the length of the expression data
  # Go through every gene and calculate cor.test
  p.value = sapply(1:nrow(Daisy.expression), function(a){
    expression.goi =t(Daisy.expression[goi,]) # Expressiondata of the goi (gene A)
    expression.GeneB =t(Daisy.expression[a,]) # Expressiondata of all other genes (gene B)
    # One sided spearman correlation test
    p = cor.test(expression.goi,expression.GeneB, alternative = "greater",
                 method = "spearman", exact = FALSE)$p.value
  })

  # Return a dataframe with the columns genes(HugoSymbols) and p_values (p.values)
  out = data.frame(genes = rownames(Daisy.expression), p_value = p.value)
  out = out[which(out$p_value < 0.05),] # Filter for significant p.values
  # Only keep genes wich are not goi (would be a corr of 1)
  out = out[which(out$genes != goi),]
  return(out)
  })
  names(Daisy_Spearman) = input_genes
  return(Daisy_Spearman)
}
```

Both functions return a list, which contains the five genes of interest, each containing a dataframe with all genes examined as potential SL/SDL partners plus their calculated p-values.

SL-parter

SoF

```
# SoF, inactive, one sided Wilcoxon rank test, alternative = "greater"
SL.SoF = Daisy.Wilcox(goi.SL, copynumber,
                      inactive_overactive = 0,
                      less_greater = 1)
# Example of SL SoF for first GOI (TP53)
head(SL.SoF[[1]])
```

```
      genes    p_value
7   NAALAD2 0.01270171
10 NAALADL1 0.01787355
36    TCF24 0.03545790
41  TMEM262 0.01434325
46  C6orf99 0.04298062
96   TIMM23 0.04783701
```

Functional examination

```
# Functional examination, inactive, one sided Wilcoxon rank test, alternative = "less"
SL.functional.examination = Daisy.Wilcox(goi.SL, kd.ceres,
                                         inactive_overactive = 0,
                                         less_greater = 0)
# Example of SL functional examination for first GOI (TP53)
head(SL.functional.examination[[1]])
```

```
   genes      p_value
6   MED6 0.0242023283
16  CDH3 0.0007747717
22 ZGLP1 0.0027229296
70 SRRM5 0.0249260666
78   GNE 0.0336798132
82 TROAP 0.0042240416
```

Gene coexpression

```
SL.coexpression = Daisy.Spearman(goi.SL)
# Example of SL coexpression for first GOI (TP53)
head(SL.coexpression[[1]])
```

```
      genes      p_value
12    ACOT8 1.579807e-02
13     ABI1 1.074688e-03
18  POM121C 7.980707e-04
19  ZSCAN30 1.057778e-03
21 TMEM170B 1.179551e-03
22    ZGLP1 9.669063e-05
```

SDL-parter

SoF

```
# SoF, overactive, one sided Wilcoxon rank test, alternative = "greater"
SDL.SoF = Daisy.Wilcox(goi.SDL, copynumber,
                       inactive_overactive = 1,
                       less_greater = 1)
```

Functional examination

```
# Functional examination, overactive, one sided Wilcoxon rank test, alternative = "less"
SDL.functional.examination = Daisy.Wilcox(goi.SDL, kd.ceres,
                                          inactive_overactive = 1,
                                          less_greater = 0)
```

Gene coexpression

```
SDL.coexpression = Daisy.Spearman(goi.SDL)
```

Converging the different SL and SDL sets

Define function to converge SL/SDL sets.

```
'partner' = function(input_genes, SoF, functional.examination, coexpression){
  partner = lapply(1:length(input_genes), function(a) {
    dat_picker.SoF = SoF[[a]]
    dat_picker.func.exam = functional.examination[[a]]
    dat_picker.coexpression = coexpression[[a]]
    # Intersect of genes from all three tests
    partnergenes = Reduce(intersect, list(dat_picker.SoF$genes,
                                          dat_picker.func.exam$genes,
                                          dat_picker.coexpression$genes))
    out = data.frame(
      genes = partnergenes,
      # Column with reduced genes
      SoF = dat_picker.SoF[which(dat_picker.SoF$genes %in% partnergenes), "p_value"],
      functional_examination = dat_picker.func.exam[which(dat_picker.func.exam$genes
                                                          %in% partnergenes), "p_value"],
      coexpression = dat_picker.coexpression[which(dat_picker.coexpression$genes
                                                   %in% partnergenes), "p_value"]
    ) # Column with p value from coexpression
    return(out)
  })
  names(partner) = input_genes
  return(partner)
}
```

Converge different sets.

```
SL.partner = partner(goi.SL, SL.SoF, SL.functional.examination, SL.coexpression)
SDL.partner = partner(goi.SDL, SDL.SoF, SDL.functional.examination, SDL.coexpression)
kable(data.frame("SL genes" = names(SL.partner),
           "SL number of partner" = sapply(1:length(SL.partner), function(a){
             nrow(SL.partner[[a]])
             }),
           "SDL genes" = names(SDL.partner),
           "SDL number of partner" = sapply(1:length(SDL.partner), function(a){
             nrow(SDL.partner[[a]])
             })
           ))
```

SL.genes	SL.number.of.partner	SDL.genes	SDL.number.of.partner
TP53	21	LCN2	128
FAT1	54	TP63	175
CDKN2A	60	GPR87	205
KRT3	124	IFI27	68
CASP8	22	SOX15	100

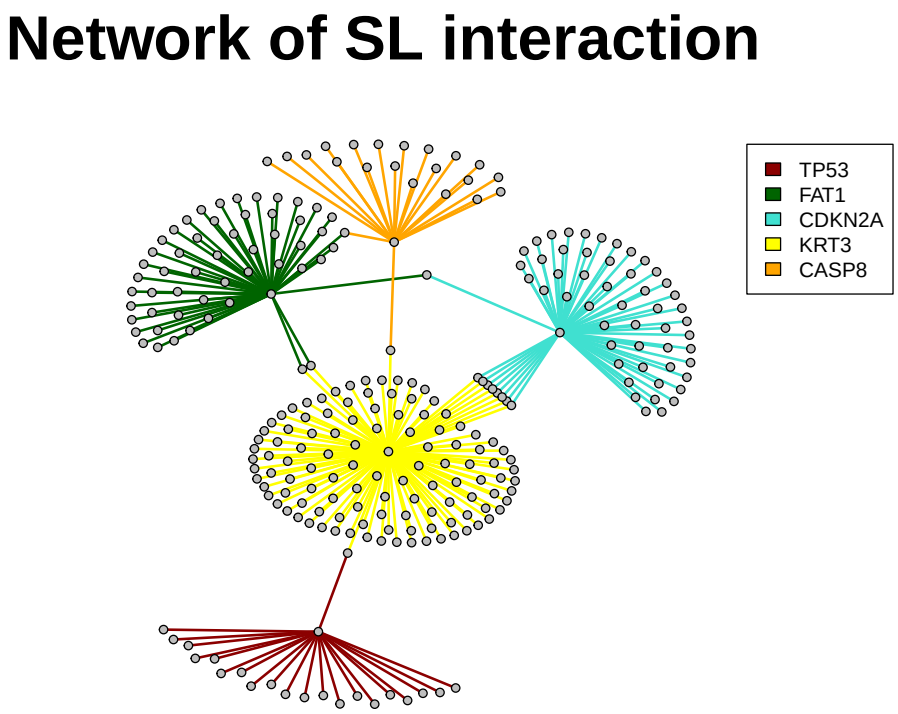
Plotting networks of the Interactions

Creation of a function for transforming given data into a format iGraph can work with and plotting of the network.

```
'network' = function(X.partner,width,node.color,frame.color,node.size,Titel,label,GOI){
  # Input: SL/SDL.partner, number, string, string, number, string, NA or names(plot)

  # Define colours
  color.vector = c("darkred", "darkgreen", "turquoise", "yellow", "orange")
  # Transform data into a dataframe with 3 colums for each GOI
  edgelist = lapply(1:length(X.partner), function(a){
    dat_picker = X.partner[[a]]
    edge.list = data.frame(Gene.A = names(X.partner)[a],
                           Gene.B = dat_picker$genes,
                           colours = color.vector[a])
  })
  edge.list = do.call(rbind, edgelist) # Combine all goi-dataframes to one.

  # Change column colours from factor to chracter for plotting function
  edge.list$colours = as.character(edge.list$colours)

  # Tranform the dataframe into an iGraph object
  net = graph_from_data_frame(edge.list, directed = T)
  layout1 = layout_with_kk(net) # Given layout

  # Plotting parameters are defined. vertex= Genes edges = interactions
  plot(net, layout = layout1, rescale = T,
       ylim = c(-0.9,1),
       xlim = c(-0.1,-0.1),
       edge.color = edge.list$colours,
       edge.width = width,
       edge.arrow.mode = 0,
       vertex.color = node.color,
       vertex.frame.color = frame.color,
       vertex.size = node.size,
       vertex.label.font = 2,
       vertex.label.color = "black",
       vertex.label = label,
       margin = 0.1)

  # Add legend to the graph
  legend (x = 1.2,
          y = 1, GOI,
          fill = c("darkred", "darkgreen", "turquoise", "yellow", "orange"))

  # Add titel to the graph
  title(Titel,cex.main=3,col.main="black ")

}
```

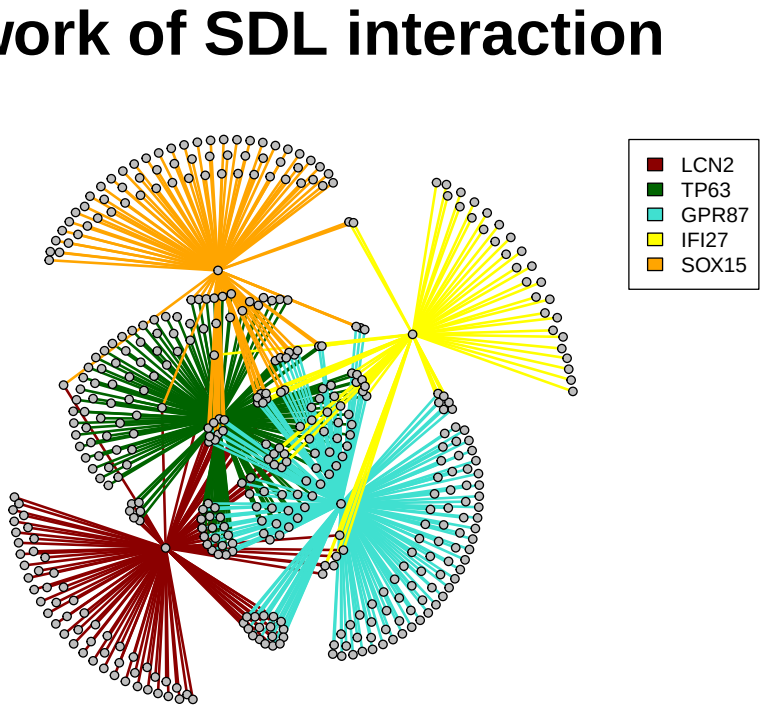
Creating a networkplot for the SL-partner

```
network(SL.partner,2,"grey","black",3,"Network of SL interaction", NA, goi.SL )
```



Creating a network for the SDL-partner

```
network(SDL.partner,2,"grey","black",3,"Network of SDL interaction", NA, goi.SDL )
```



Because there are to many SL/SDL partner to plot a network we reduce SL/SDL partner to the most significants.

To do so, we reduce the number of partner genes for better plotting without removing genes which are connected to more than one GOI.

```
'reduce.partner' = function(list){
  # Go through every element of SL/SDL partner
  reduced_partner = lapply(1:length(list), function(a){
    dat_picker = list[[a]]

    # Create an extra column with the mean p value of all 3 tests
    dat_picker$order = apply(dat_picker[, 2:4], 1, mean)
    # Order the partner genes after this mean value from most to least significant p value
    ordered_data = dat_picker[order(dat_picker$order), ][, 1:4]

    gene_intersect = lapply(1:length(list), function(b) {
      # Search for genes that are partner for more than one GOI
      as.list(Reduce(intersect, list(dat_picker$genes), list[[b]]$genes))
    })
    gene_intersect[a] = NULL # Remove intersect of the GOI partner genes with itself
    keep_genes = c(gene_intersect[[1]],
```

```
                    gene_intersect[[2]],
                    gene_intersect[[3]],
                    gene_intersect[[4]]) # Combine all the genes in a list
    position = sapply(1:length(keep_genes), function(c) {
      which(as.character(ordered_data$genes) == keep_genes[c])
    })
    # Partner genes that are already in the first 20 genes don´t need to be chosen
    position = position[which(position > 20)]

    if (length(position) > 3) {
      # If there are still more than 3 extra genes only choose the first 3
      position = position[1:3]
    }
    out = ordered_data[c(1:20, position), ]
    return(out) # Return reduced data
  })
  names(reduced_partner) = names(list)
  return(reduced_partner)
}
```

Reduce SL.parter/SDL.partner to the most significant

```
SL.partner.reduced = reduce.partner(SL.partner)
SDL.partner.reduced = reduce.partner(SDL.partner)

kable(data.frame("SL genes" = names(SL.partner),
            "SL partner" = sapply(1:length(SL.partner.reduced), function(a){
              nrow(SL.partner.reduced[[a]])
              }),
            "SDL genes" = names(SDL.partner),
            "SDL partner" = sapply(1:length(SDL.partner.reduced), function(a){
              nrow(SDL.partner.reduced[[a]])
              })
            ))
```

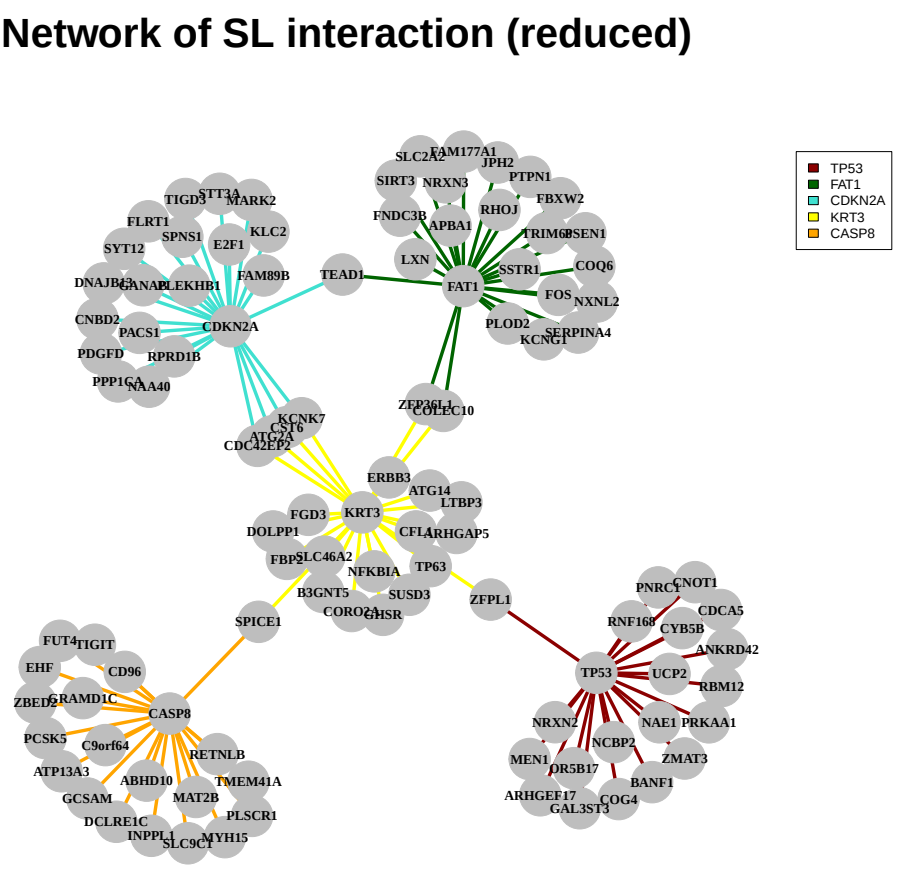
SL.genes	SL.partner	SDL.genes	SDL.partner
TP53	20	LCN2	23
FAT1	23	TP63	23
CDKN2A	23	GPR87	23
KRT3	23	IFI27	23
CASP8	20	SOX15	23

Now the number of SL/SDL partner is much better to plot.

Plotting SL/SDL partner network with reduced number of partner

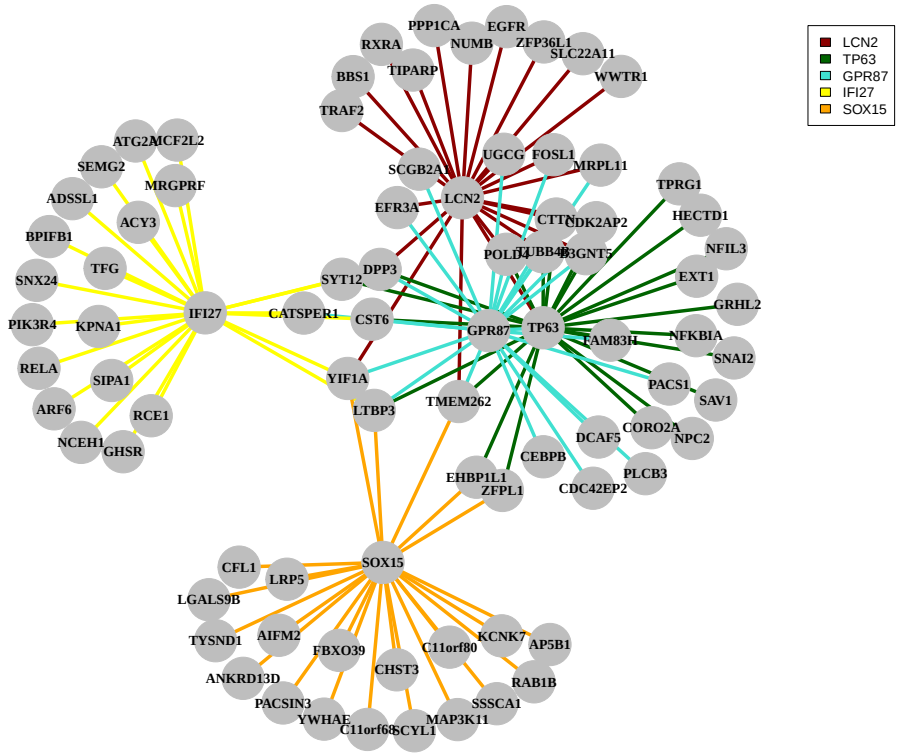
SL interaction plot with reduced partners

```
network(SL.partner.reduced,4,"grey","grey",12,"Network of SL interaction (reduced)",
        names(plot),
        goi.SL)
```



SDL interaction plot with reduced partners

```
network(SDL.partner.reduced,4,"grey","grey",12,"Network of SDL interaction (reduced)",
        names(plot),
        goi.SDL)
```



Mutation prediction via logistic regression

Prepare data for regression model

```
# Reduce all genes to those of which we have expression, copynumber and knockdown data.
regression.genes = Reduce(intersect,
                          list(rownames(expression.HNC),
                               rownames(copynumber.HNC),
                               rownames(kd.ceres.HNC)
                          ))

# Create a dataframe with expression, CNA, knockdown and mutation data from a HNC sample.
regression.data = lapply(1:length(ID), function(a){

  regression.mutation = sapply(1:length(regression.genes), function(b){
    ifelse(regression.genes[b] %in% mutation.HNC[[a]]$Hugo_Symbol, TRUE, FALSE)
  })

  regression.data = data.frame("genes" = paste(regression.genes, ID[a], sep = "_"),
                               "expression" = expression.HNC[regression.genes, a],
                               "copynumber" = copynumber.HNC[regression.genes, a],
                               "knockdown" = kd.ceres.HNC[regression.genes, a],
                               "mutation" = regression.mutation)
})

# Bind these 27 dataframes to one dataframe
regression.data = do.call(rbind, regression.data)

dim(regression.data)
```

```
[1] 458190      5
```

```
head(regression.data)
```

```
                genes expression copynumber   knockdown mutation
1     DPM1_ACH-000188  5.3593103     0.1574 -0.42737583    FALSE
2    SCYL3_ACH-000188  1.2570106     0.1441 -0.01577348    FALSE
3 C1orf112_ACH-000188  3.1093606     0.1441 -0.03149540    FALSE
4      FGR_ACH-000188  0.1634987     0.2019 -0.10972076    FALSE
5      CFH_ACH-000188  4.0285692    -0.0637  0.03189585    FALSE
6    FUCA2_ACH-000188  4.6536333     0.1872  0.02271554    FALSE
```

Reformat the data because strings can not be an input for a machine learning model

```
regression.data$mutation = factor(regression.data$mutation, levels = c("FALSE", "TRUE"))
# Reformat the rownames
rownames(regression.data) = regression.data$genes
# Get rid of the gene names becaue they are now the rownames
regression.data = regression.data[,2:ncol(regression.data)]
head(regression.data)
```

```
                    expression copynumber   knockdown mutation
DPM1_ACH-000188      5.3593103     0.1574 -0.42737583    FALSE
SCYL3_ACH-000188     1.2570106     0.1441 -0.01577348    FALSE
C1orf112_ACH-000188  3.1093606     0.1441 -0.03149540    FALSE
FGR_ACH-000188       0.1634987     0.2019 -0.10972076    FALSE
CFH_ACH-000188       4.0285692    -0.0637  0.03189585    FALSE
FUCA2_ACH-000188     4.6536333     0.1872  0.02271554    FALSE
```

Split the data into train and test-data

```
# 75% of data is used to train, 25% to test the regression model
inTrain = createDataPartition(y = regression.data$mutation, p = .75, list = FALSE)
# Only get training data
train.set = regression.data[inTrain,]
head(train.set)
```

```
                    expression copynumber   knockdown mutation
DPM1_ACH-000188      5.3593103     0.1574 -0.42737583    FALSE
SCYL3_ACH-000188     1.2570106     0.1441 -0.01577348    FALSE
C1orf112_ACH-000188  3.1093606     0.1441 -0.03149540    FALSE
FGR_ACH-000188       0.1634987     0.2019 -0.10972076    FALSE
CFH_ACH-000188       4.0285692    -0.0637  0.03189585    FALSE
NFYA_ACH-000188      3.1874511     0.1712 -0.13549254    FALSE
```

```
# Only get testing data
test.set = regression.data[-inTrain,]
head(test.set)
```

```
                  expression copynumber   knockdown mutation
FUCA2_ACH-000188    4.653633     0.1872  0.02271554    FALSE
GCLC_ACH-000188     4.149747     0.1549  0.06840387    FALSE
KRIT1_ACH-000188    3.524816    -0.2043  0.02438860    FALSE
LAP3_ACH-000188     5.096346    -0.7520 -0.18606702    FALSE
CASP10_ACH-000188   1.937344     0.0374  0.10407772    FALSE
CFLAR_ACH-000188    4.042644     0.0374 -0.08786211    FALSE
```

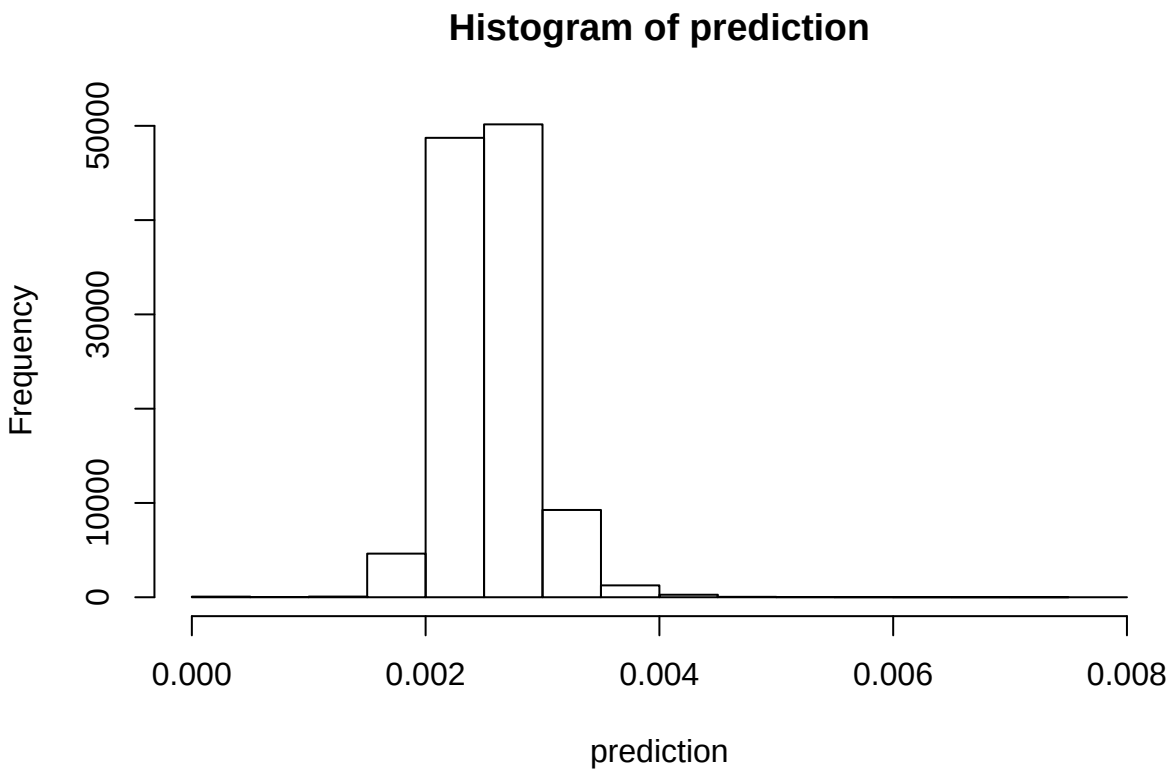
Train the model

```
regression.model = glm(mutation ~ ., data = train.set, family = "binomial")
summary(regression.model)
```

```
Call:
glm(formula = mutation ~ ., family = "binomial", data = train.set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.1389  -0.0742  -0.0711  -0.0684   3.6081

Coefficients:
            Estimate Std. Error  z value Pr(>|z|)
(Intercept) -6.02930    0.05103 -118.142  < 2e-16 ***
expression   0.03178    0.01479    2.148  0.03172 *
copynumber   0.24603    0.08113    3.033  0.00242 **
knockdown    0.24811    0.12285    2.020  0.04342 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 12249  on 343642  degrees of freedom
Residual deviance: 12231  on 343639  degrees of freedom
AIC: 12239

Number of Fisher Scoring iterations: 9
```

Evaluate the precision and the recall of the model

```
prediction = predict(regression.model, newdata = test.set, type="response")
hist(prediction) # Look at the predictions
```



Devide prediction in threshholds

```
threshhold_ls = seq(min(prediction), max(prediction), by = 0.001) # Make a thresshold list
length(threshhold_ls)
```

```
[1] 8
```

```
threshold_list = lapply(1:length(threshhold_ls), function(a){
  # Check for every threshhold:
  table(prediction > threshhold_ls[a], # Returns TRUE or FALSE
        test.set$mutation) # Comparing if return fits actual bimodal value
  # Counting True Negative, False Negative, False Positive, True Positive
  })

threshold_list[[1]]
```

```
       FALSE   TRUE
  FALSE     1      0
  TRUE  114253    293
```

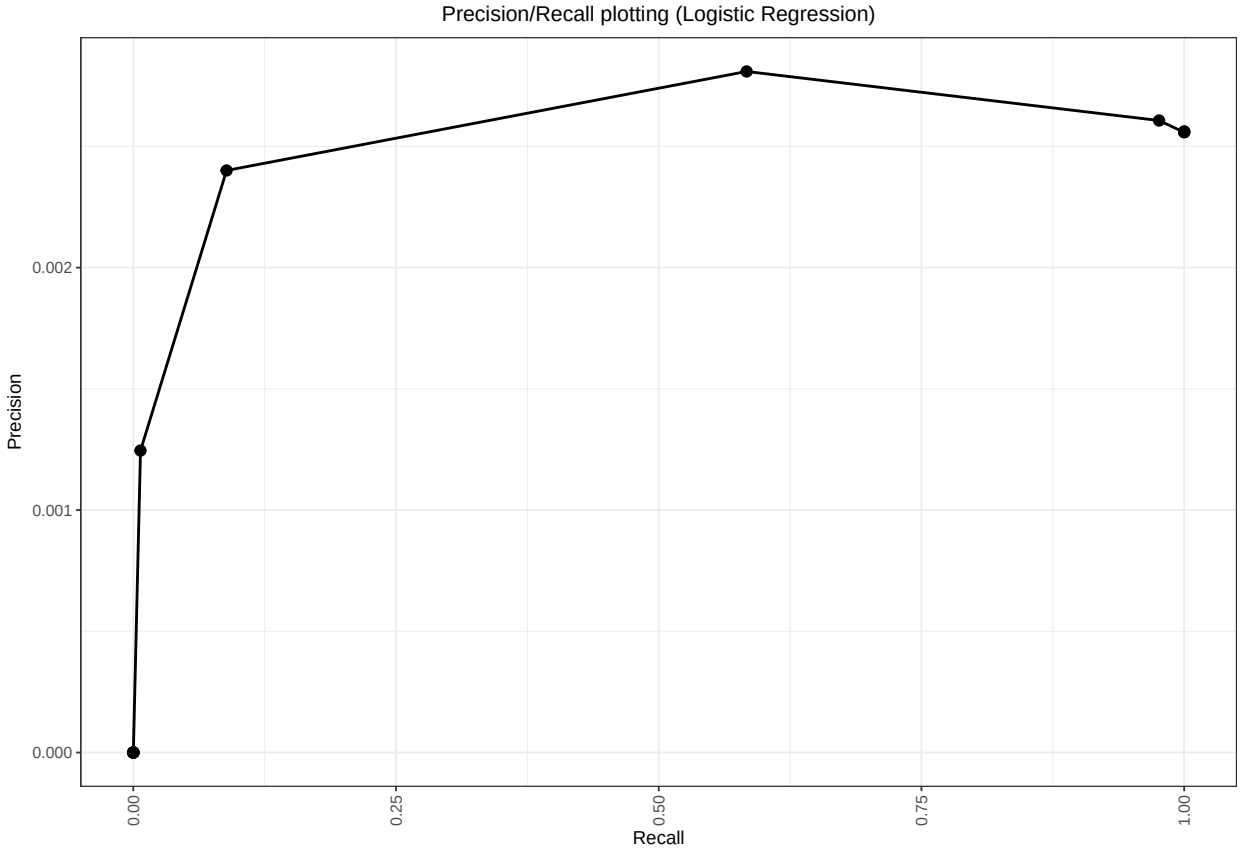
Calculate the recall and precision values

```
# Make a dataframe out of the list of lists
results_df = lapply(1:length(threshold_list), function(a) {
  threshold = threshold_list[[a]]
  threshold_value = threshhold_ls[[a]]

  df_precision_recall = as.data.frame(list(threshold["FALSE", "FALSE"],
                                           threshold["FALSE", "TRUE"],
                                           threshold["TRUE", "FALSE"],
                                           threshold["TRUE", "TRUE"]))
  df_precision_recall$threshold = threshold_value
  colnames(df_precision_recall) = c("TN", "FN", "FP", "TP", "Thresh")
  return(df_precision_recall)
})

# Bind all the lists together
results_df = do.call(rbind, results_df)

# Calculate the recall; recall = True positive / all positive events
recall = apply(results_df, 1, function(x){ x[["TP"]] / (x[["TP"]]+x[["FN"]]) })

# Calculate the precision; precision = True positive / all positive detected events
precision = apply(results_df, 1, function(x){ x[["TP"]] / (x[["TP"]]+x[["FP"]]) })

results_df$recall = recall # Bind recall to the data
results_df$precision = precision # Bind precision to the data
head(results_df) # Look at the data
```

```
     TN  FN     FP  TP       Thresh    recall   precision
1     1   0 114253 293 9.558393e-07 1.0000000 0.002557924
2    64   0 114190 293 5.009558e-04 1.0000000 0.002559332
3    89   0 114165 293 1.000956e-03 1.0000000 0.002559891
4   161   0 114093 293 1.500956e-03 1.0000000 0.002561502
5  4814   7 109440 286 2.000956e-03 0.9761092 0.002606493
6 53545 122  60709 171 2.500956e-03 0.5836177 0.002808804
```

Plot the data

```
ggplot(results_df, aes(recall,precision)) +
  geom_point() +
  geom_line() +
  ggtitle("Precision/Recall plotting (Logistic Regression)") +
  ylab("Precision") +
  xlab("Recall") +
  theme_bw(base_size = 7) +
  theme(legend.position="bottom",
        legend.direction="horizontal",
        plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        legend.title= element_blank(),
        strip.text.y = element_text(angle = 0))
```



References

Articles

1.) Jerby-Arnon, L., et al. (2014). "Predicting Cancer-Specific Vulnerability via Data-Driven Detection of Synthetic Lethality." Cell 158(5): 1199-1209.

2.) Ashworth, A., et al. (2011). "Genetic Interactions in Cancer Progression and Treatment." Cell 145(1): 30-38.

3.) O'Neil, N. J., et al. (2017). "Synthetic lethality and cancer." Nature Reviews Genetics 18: 613.

Figures

Figure 1 - Jerby-Arnon, L., et al. (2014). "Predicting Cancer-Specific Vulnerability via Data-Driven Detection of Synthetic Lethality." Cell 158(5): 1199-1209.

Figure 3 - Leemans, C. R. et al. (2018). The molecular landscape of head and neck cancer. Nat. Rev. Cancer 18, 269-282.