Cancer methylome analysis CLL vs. B-Cells

Leona Brandl, Carlotta Brueggen, Tim Kuehn, Violetta Schaaf

17th July 2019

The aim of this project is to find genes which differ in their methylation level for cancer cells derived from chronic lymphocytic leukemia (CLL) patients compared to B-cells derived from healthy patients. Therefore, we analyze methylation data of five CLL patients and five healthy patients, looking for differentially methylated regions which we then examine for statistical and biological significance in carcinogenesis and especially inspect genes which according to literature are connected to CLL pathogenesis. Chronic lymphocytic leukemia is the most common form of leukemia in adults and can be characterized as the uncontrolled proliferation of B-lymphocytes. Since their ability to undergo apoptosis is inhibited, these functionally incompetent B-cells accumulate in the blood vessels and displace healthy blood cells. Consequences of this development are increased vulnerability for infections, a lower oxygen level in the blood and longer bleeding periods in case of injury. As CLL originates from B-cells, we use these for comparison in our analysis. It is proved that aberrant methylation has an effect on oncogenesis and there is evidence that this might also be the case for CLL. The most common form of methylation is the conversion of a cytosine to 5-methylcytosine by the addition of a methyl group in fifth position of the nucleobase by the DNA methyltransferase. Methylation is highly relevant in epigenetics since it has an effect on the transcription of genes and may lead to gene silencing. In CLL, a global hypomethylation compared to healthy cells has been observed, with possible consequences being activation of gene expression as well as genomic instability. Moreover, hypermethylation of gene promotors was frequently detected, which might lead to inhibition of promotor activity and affect the transcription of tumor suppressor genes. The methylation data we work with was obtained by bisulfite sequencing, which is a library strategy providing information on the degree of DNA methylation. In this technique DNA is treated with bisulfite upon amplification with PCR, resulting in the conversion of unmethylated cytosines into thymines while methylated cytosines are protected from modification. Afterwards, it is possible to calculate the beta value, which is defined as the ratio of methylated cytosines to the total number of cytosines in the genome, ranging between 0 and 1.

The analysis will be done in several steps:

1. **Data processing**

* Reorganisation of the data
* Quality control concerning NAs, coverage values and unusefull DNA-regions

1. **Normalisation**

* Preparing beta-values
* Transformation of beta-values into M-values

1. **Dimensionality reduction**

* PCA
* Batch effect analysis

1. **Clustering**

* Via K-means clustering

1. **Identifying DMRs**

* t-test
* p-value correction
* Statistical and biological relevance
* Hypo- and hypermethylation

1. **Logistic regression**
2. **Data interpretation**

## 1) Data processing

Before we can start analysing our data set it has to be tidied up to make it easy for us to work with it by making it as clear as possible. Datasets have to be split up and reorganized, coloums removed or renamed and a quality control concering coverage values, beta values, NAs and specific DNA segments showing unusual values, needs to be performed.

### Reading in the datasets

We start by **reading in our CLL-Bcell-data and the annotations** which is an extra document explaining where our data is coming from.

input\_data <- readRDS(file ="CLL-Bcells\_list.RDS.gz")  
annotation <- read.csv("sample\_annotation.csv")

### Reorganisation of the data

Our main dataset, the one containing the data we are working with, is input\_data. It contains methylation data in form of beta values and coverage values of B-cells of five healthy patients and five patients suffering from CLL. It is divided into four subgroups: tiling, genes, promoters and CpG islands. We have to analyse those subgroups separatly, because methylation in promoters, genes and CpG islands can have different biological reasons and also this way the data the computer deals with is not that much while running the algorithm. Therefor **the dataset gets divided into four subdatasets**.

genes <- input\_data$genes  
promoters <- input\_data$promoters  
cpgislands <- input\_data$cpgislands  
tiling <- input\_data$tiling

Those subgroups datasets contain methylation data for different regions on the genome: genes, promoters and cpgislandscontain genome segments with specific functions and methylation features. Genes, especially highly expressed genes, sometimes show methylation in their gene body. Unfortunately the biological function behind this is not fully clear yet. Still it can be interesting investigating in their methylation pattern in terms of cancer. In promoters we already know that methylation plays an important role, because as promoters are the starting point for transcription, their methylation can silence genes laying downstream of these promoters. This way for example tumor suppresor genes don’t get expressed any more, which can lead to cancer. Here we expect to find crucial differences in methylation between sick and healthy cells. CpG islands also are important regions on the DNA with a high density of cytosines in neighbourhood to guanines. These CpG islands are often in promoter regions of housekeeping genes and in healthy beeings only rarely methylated, because their methylation usually leads to mistakes in replication and transcription and quickly in death just because functioning promoters are so essential in those genes. Also in those CpG islands a loss of function can lead to cancer, so we expect significant methylation differences. The fourth subgroup is tiling which describes a specific section on the genome in steps of 5000 nucleotides without a biological context.

As we want to analyse those subgroups separatly, we first just choose one subgroup, genes, to work on our code. Later we can transfer the code on the other subgroups, too. In this genesdataset we **rename the columns**, which stand for different patients (patient 1 to patient 10). They get short precise names to let the reader get the meaning on the first sight.

colnames(genes)[11:30] <- c("P1\_healthy\_beta","P2\_healthy\_beta","P3\_healthy\_beta","P4\_healthy\_beta","P5\_healthy\_beta","P6\_CLL\_beta","P7\_CLL\_beta","P8\_CLL\_beta","P9\_CLL\_beta","P10\_CLL\_beta","P1\_healthy\_coverage","P2\_healthy\_coverage","P3\_healthy\_coverage","P4\_healthy\_coverage","P5\_healthy\_coverage","P6\_CLL\_coverage","P7\_CLL\_coverage","P8\_CLL\_coverage","P9\_CLL\_coverage","P10\_CLL\_coverage")

### Quality control

It is important to remove values that are not needed for or even hinder further analysis. Among other values those are also coverage values, which seem to be unrealistically high due to PCR dublicates or repetetive regions. Also they can be or that low, so the belonging beta-value of the DNA section has no meaning, because there was no fitting allignment when mapping back PCR fragments onto the genome. A **data frame only containing coverage values** is created.

cov\_genes <- genes[,c(1,21:30)]

At this point we already delete some genes, which are the genes on the X- and Y-chromosome. Somehow there must have been problems in obtaining the beta and coverage values for those chromosomes, because the beta-values are either not a number (NaN) or mainly 1.000, which must definitely be a mistake.The belonging coverage is in most cases 0, so we can not use those beta-values values anyway. Following code **removes the X- and Y-chromosome**.

#### Removing genes on X- and Y-chsomosome

cov\_genes\_new <- cov\_genes[-which(cov\_genes =="chrX"),]  
cov\_genes <- cov\_genes\_new[-which(cov\_genes\_new =="chrY"),]  
rm(cov\_genes\_new)   
cov\_genes <- cov\_genes[ ,c(2:11)]

#### Finding a coverage value threshold

But back to the very high or low coverage values: There need to be a threshold defining a senseful coverage value range. To determine a threshold we have a look at the coverage value distribution among all patients using a logarithmic density plot of the coverage means of all patients for each genome segment. Therefor a **matrix containing the means** in generated.

cov\_genes\_means <- rowMeans(data.matrix(cov\_genes))

A **logarithmic density plot** using this matrix values is generated. To **find an upper threshold** for deleting coverage values we try quantile values we read about in literature and draw the lines for those quantiles in the logarithmic plot to see wether one of those coincidences with a kink in the curve. For the **lower threshold** we draw lines at the values 10 and 15, which are common threshold values as we read in litearture.

plot(density(log10(cov\_genes\_means)), xlab = "Coverage means", main = "Coverage distribution")  
quantile(cov\_genes\_means, probs = c(.95))

## 95%   
## 53811.9

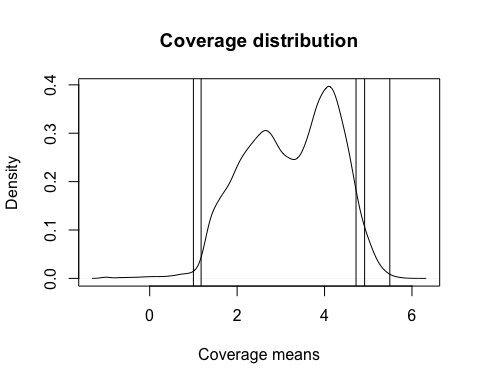
abline(v=log10(52529.75))  
quantile(cov\_genes\_means, probs = c(.975))

## 97.5%   
## 84118.43

abline(v=log10(82556.55))  
quantile(cov\_genes\_means, probs = c(.999))

## 99.9%   
## 313215.5

abline(v=log10(311230.4))  
abline(v=log10(10))  
abline(v=log10(15))



Both lower thresholds and the highest upper threshold seem to be fitting looking at the diagram. But which threshold we will chose eventually also depends on the percentage of deleted rows at the end of quality control. We decided not to delete more than 15 % of the rows, because we don’t want to risk to lose reliable information.

The procedure will now be used to set all the coverage values in cov\_genes that are below or above the chosen threshold to NA. Then all the beta-values associated to those NA coverage values are also turned into NA. Eventually we will set another threshold for the beta-values, defining the maximum number of NA that each row can have in order not to be deleted. In order to do this, we used two nested for-loops.

#### Preparing the first nested for-loop

Before applying a threshold to the coverage values, the few **NAs among the coverage values are set to 0** in order to allow the following for loops to work. These values will be set to NAs again after the loop, since 0 is below the lower threshold.

cov\_genes[is.na(cov\_genes)] <- 0

As the lower threshold we choose the coverage value 15, which is positioned at a kink of the previously showed coverage distribution diagram. As the upper coverage threshold we choose the 90 % quantile. Although it is not positioned at a kink in the diagram we consider it to be a good threshold, because it still only leads to a removal of 7 % of the genes at the end of the quality control, which is far underneath the chosen upper limit of 15 %.

threshold1 <- 15  
threshold2 <- quantile(cov\_genes\_means, probs = seq(0.90,0.90,0.05))

#### The first nested for-loop

Now a **nested for-loop** is used to set the coverage values underneath the lower threshold and above the upper threshold to NA. This will be done for each row and each column of the dataframe cov\_genes.

for(i in 1:nrow(cov\_genes)){  
 for(j in 1:ncol(cov\_genes)){  
 if(isTRUE(cov\_genes[i,j] <= threshold1)){  
 cov\_genes[i,j] <- NA  
 }   
 if(isTRUE(cov\_genes[i,j] >= threshold2)){  
 cov\_genes[i,j] <- NA  
 }  
 }  
}  
rm(i,j,threshold1,threshold2)

#### Preparing the second nested for-loop

A second nested for-loop is used to set the beta-values to NA that correspond to the NA coverage values. A **dataframe containing only beta-values** is created and the **genes located on the X- or Y-chromosome are removed**.

beta\_genes <- genes[,c(1,11:20)]  
beta\_genes\_new <- beta\_genes[-which(beta\_genes =="chrX"),]  
beta\_genes <- beta\_genes\_new[-which(beta\_genes\_new =="chrY"),]  
rm(beta\_genes\_new)  
beta\_genes <- beta\_genes [ ,c(2:11)]

#### The second nested for-loop

This **second nested for-loop** checks every row and every column of the dataframe cov\_genes. If there is an NA, it sets the belonging beta-value in beta\_genes to NA, too.

for(k in 1:ncol(beta\_genes)){  
 for(l in 1:nrow(beta\_genes)){  
 if(isTRUE(is.na(cov\_genes[k,l] == TRUE))){  
 beta\_genes[k,l] <- NA  
 }   
 }  
}  
rm(k,l)

#### Finding a threshold for NAs among the beta-values

Now the second threshold in quality control needs to be chosen. The threshold establishes how many NA are allowed per gene in order for the gene not be completely deleted. Since some beta-values were set to NA in the last step, we want to get insight in **how many NAs we actually have now**.

rmv.rows\_beta\_genes = apply(beta\_genes,1, function(x){sum(is.na(x))})

A new dataframe containing only beta-values for genes which are below the NA threshold is created. We chose a threshold of 3 allowed NAs per row (gene), so **every row containing more than 3 NAs gets removed**.

beta\_genes\_cleaned <- beta\_genes[-which(rmv.rows\_beta\_genes >2),]

To see **how many rows are deleted** due to quality control the numer of rows before and after quality control is compared. We additionally have a look at the **percantage of rows deleted** and see whether it is beneath 15 %.

row\_difference = nrow(genes)-nrow(beta\_genes\_cleaned)  
sum(row\_difference)

## [1] 3915

genes\_deleted\_percentage = row\_difference/nrow(genes)\*100  
sum(genes\_deleted\_percentage)

## [1] 6.961856

With 7% we are beneath the recommended upper limit of 15 % of deleted rows.

## 2) Normalisation

The tidy beta-values we have now are an approximation of the percentage of DNA-methylation. Biologically beta-values are easy to understand. They range from 0 to 1, where 0 means unmethylated and 1 means fully methylated. But their bounded range is also a disadvantage. It leads to the problem that they can not be used for statistical tests like the t-test, because they violate the Gaussian distribution. For highly methylated gene regions and unmethylated gene regions beta-values are very heteroscedastic, which means, that the variability of a variable is unequal across the range of values. Therefore we have to normalise them. Beta-values need to be transformed into M-values via a logit transformation. M-values do not have a bounded range, they can be infinite big and small. While the middle methylation range (beta-value range of 0,2-0,8) is linear to M-values, outside this range the beta-values are compressed and the M-values more accurate and homoscedastic. This is why M-values can be used for statistical tests and are absolutely necessary for further analysis.

### Preparing beta-values for logit transformation

Logit transformation turns the extreme beta-values 0 and 1 into -infinite and infinite. Further tests cannot be computed if some values are equal to -infinite and infinite. Because of this, we **set 0 to a very small number higher than 0, and 1 to a very big number smaller than 1**.

beta\_genes\_cleaned[beta\_genes\_cleaned==0]<-0.00000001  
beta\_genes\_cleaned[beta\_genes\_cleaned==1]<-0.99999999

#### Creating two seperate data frames for sick and healthy patients

In order to perform our normalisation, we **split our data into beta-values of healthy and diseased patients.**

beta\_genes\_healthy <- beta\_genes\_cleaned[,c(1:5)]  
beta\_genes\_cancer <- beta\_genes\_cleaned[,c(6:10)]

Now we are able to **replace the remaining NA’s by the row means of the different genes.**The rowmeans is different between healthy and sick patients so it is important to work with the two separated dataframes.

k <- which(is.na(beta\_genes\_healthy), arr.ind=TRUE)  
beta\_genes\_healthy[k] <- rowMeans(beta\_genes\_healthy, na.rm=TRUE)[k[,1]]  
l <- which(is.na(beta\_genes\_cancer), arr.ind=TRUE)  
beta\_genes\_cancer[l] <- rowMeans(beta\_genes\_cancer, na.rm=TRUE)[l[,1]]

### Transforming beta-values into M-values

In the next step we **transform our beta-values into M-values** so we are able to do statistical tests with the data. We calculate the M-values with the following equation:

M\_genes\_healthy <- log2(beta\_genes\_healthy/(1-(beta\_genes\_healthy)))  
M\_genes\_cancer <- log2(beta\_genes\_cancer/(1-(beta\_genes\_cancer)))

For applying the PCA on our data we have to build a **dataset which contains sick and healthy patients** with **short column names**.

M\_genes <- cbind(M\_genes\_healthy, M\_genes\_cancer)  
colnames(M\_genes) <- c("1H","2H","3H","4H","5H","6CLL","7CLL","8CLL","9CLL","10CLL")

## 3) Dimensionality reduction

To **reduce our data** we use Principal Component Analysis. PCA reduces a large set of variables into a smaller set which still contains most information of the large dataset. Through the PCA, correlated variables get transformed into a much smaller number of uncorrelated variables which are called principal components. We want to have less variables, because it is easier to work with them.

### PCA

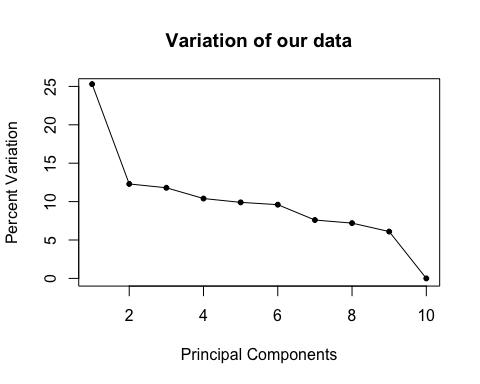
The code for the **PCA** expects the patients to be rows and the samples to be columns. In our data the patients are columns and the genes are rows so we have to **transpose the matrix** with the t() function.

pca\_M <- prcomp(t(M\_genes))

#### How much variation does each component account for?

To decide, how much principal components we want to work with, we have to know how much variation each principal component accounts for in percentage. We plot the percentage for each component and decide with the “elbow method” with which number of components we have to work with.

var\_pca <- pca\_M$sdev^2  
var\_pca\_per <- round(var\_pca/sum(var\_pca)\*100, 1)  
plot(var\_pca\_per, main="Variation of our data", xlab="Principal Components", ylab="Percent Variation", ylim=c(0,25),type = "o", pch=20)



In the plot we can not see an elbow so we would actually have to work on with ten principal components. But as the last PCs don’t explain much variation anyway and are therefor useless for us, we investigate on the first five PCs.

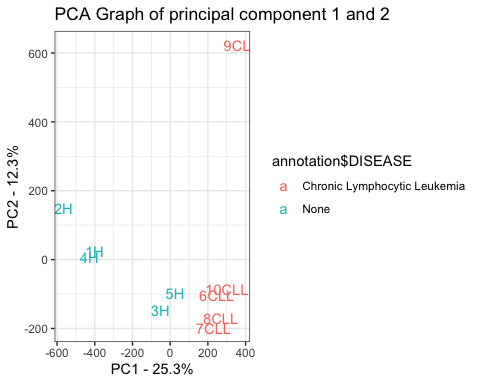
#### Drawing a 2D plot of component 1 and 2

To check whether we have a batch effect, we plot principal component 1 against principle component 2.

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.5.2

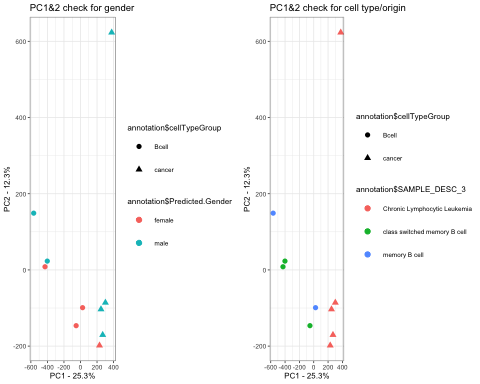
pca\_values <- data.frame(Sample=rownames(pca\_M$x),  
 X=pca\_M$x[,1],  
 Y=pca\_M$x[,2])  
ggplot(data=pca\_values, aes(x=X, y=Y, label=Sample)) +  
 geom\_text(aes(colour = annotation$DISEASE)) +  
 xlab(paste("PC1 - ", var\_pca\_per[1], "%", sep="")) +  
 ylab(paste("PC2 - ", var\_pca\_per[2], "%", sep="")) +  
 theme\_bw() +  
 ggtitle("PCA Graph of principal component 1 and 2")



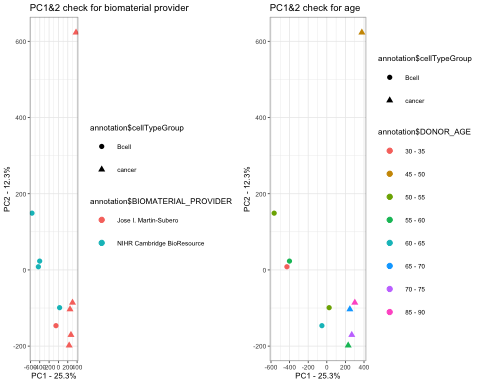
### Investigation for batch effect

Now we use different categories of the annotation table which possibly might cause **batch effect** and divide them by colour and shape in our graph of principal component 1 and 2. Batch effect is a **technical source of error**, which could have been added to the samples by e.g. gender. It is important that the technical variation does not confound with the biology to not falsify the interpreted data. To avoid errors, we used three tests to see whether there is a batch effect in a principal component or not.

ggplot\_1 <- ggplot(data=pca\_values, aes(x=X, y=Y, label=Sample)) +  
 geom\_point(aes(shape=annotation$cellTypeGroup, color=annotation$Predicted.Gender)) +  
 xlab(paste("PC1 - ", var\_pca\_per[1], "%", sep="")) +  
 ylab(paste("PC2 - ", var\_pca\_per[2], "%", sep="")) +  
 theme\_bw(base\_size = 6) +  
 ggtitle("PC1&2 check for gender")  
  
ggplot\_2 <- ggplot(data=pca\_values, aes(x=X, y=Y, label=Sample)) +  
 geom\_point(aes(shape=annotation$cellTypeGroup, color=annotation$SAMPLE\_DESC\_3)) +  
 xlab(paste("PC1 - ", var\_pca\_per[1], "%", sep="")) +  
 ylab(paste("PC2 - ", var\_pca\_per[2], "%", sep="")) +  
 theme\_bw(base\_size = 6) +  
 ggtitle("PC1&2 check for cell type/origin")  
  
ggplot\_4 <- ggplot(data=pca\_values, aes(x=X, y=Y, label=Sample)) +  
 geom\_point(aes(shape=annotation$cellTypeGroup, color=annotation$DONOR\_AGE)) +  
 xlab(paste("PC1 - ", var\_pca\_per[1], "%", sep="")) +  
 ylab(paste("PC2 - ", var\_pca\_per[2], "%", sep="")) +  
 theme\_bw(base\_size = 6) +  
 ggtitle("PC1&2 check for age")  
  
ggplot\_3 <- ggplot(data=pca\_values, aes(x=X, y=Y, label=Sample)) +  
 geom\_point(aes(shape=annotation$cellTypeGroup, color=annotation$BIOMATERIAL\_PROVIDER)) +  
 xlab(paste("PC1 - ", var\_pca\_per[1], "%", sep="")) +  
 ylab(paste("PC2 - ", var\_pca\_per[2], "%", sep="")) +  
 theme\_bw(base\_size = 6) +  
 ggtitle("PC1&2 check for biomaterial provider")  
  
library(gridExtra)  
library(ggplot2)  
grid.arrange(ggplot\_1,ggplot\_2, ncol=2)



grid.arrange(ggplot\_3,ggplot\_4, ncol=2)



#### Checking our PC 1 to 5 for significant batch effect per category

To verify our observations, we want to test if batch effect of the different categories is significant for PC 1-5. Therefore, we first create three dataframes containing PC 1-5 and the category of the annotation table we want to investigate, depending on the properties of the data, because we need to perform different tests for different data types. For numbers we use the **permutation test** (of Pearson’s r correlation coefficient), for two categories the **Wilcoxon test** and for more than two different categories the **Kruskal-Wallis test**.

x <- pca\_M[["x"]]  
pca1\_5 <- x[,1:5]  
batch\_kruskal <- data.frame(pca1\_5, annotation$SAMPLE\_DESC\_3)  
batch\_wilcoxon <- data.frame(pca1\_5, annotation$BIOMATERIAL\_PROVIDER, annotation$DISEASE, annotation$Predicted.Gender)  
batch\_permutation <- data.frame (pca1\_5, annotation$SEQ\_RUNS\_COUNT)

#### Permutation test for numbers

The **permutation test** is a non-parametric test to check, whether two not connected spot checks derive from the same basic total unit. The H0 hypothesis assumes that both spot checks derive from the same basic total unit. The permutation test in the usual form is tailored to the counterhypothesis that the distributions of the two samples emerge by a shift apart. Often the question is not only if there is a shift, but also how big is this shift. Basically it is possible to check for a batch effect with numbers.

cor.perm <- function (x, y, nperm = 1000)  
{  
 r.obs <- cor (x = x, y = y)  
 P.par <- cor.test (x = x, y = y)$p.value  
 # r.per <- replicate (nperm, expr = cor (x = x, y = sample (y)))  
 r.per <- sapply (1:nperm, FUN = function (i) cor (x = x, y = sample (y)))  
 r.per <- c(r.per, r.obs)  
   
 P.per <- sum (abs (r.per) >= abs (r.obs))/(nperm + 1)   
 return(list(r.obs = r.obs, P.par = P.par, P.per = P.per))  
}  
  
x <-batch\_permutation$PC1  
y <-batch\_permutation$annotation.SEQ\_RUNS\_COUNT  
  
p\_PC1\_SEQ <- cor.perm(x,y)  
  
x <-batch\_permutation$PC2  
p\_PC2\_SEQ <- cor.perm(x,y)  
  
x <-batch\_permutation$PC3  
p\_PC3\_SEQ <- cor.perm(x,y)  
  
x <-batch\_permutation$PC4  
p\_PC4\_SEQ <- cor.perm(x,y)  
  
x <-batch\_permutation$PC5  
p\_PC5\_SEQ <- cor.perm(x,y)  
  
p\_SEQ\_RUNS\_COUNT <- data.frame(p\_PC1\_SEQ$P.per, p\_PC2\_SEQ$P.per, p\_PC3\_SEQ$P.per, p\_PC4\_SEQ$P.per, p\_PC5\_SEQ$P.per)

#### Wilcoxon test for 2 categories

The **Wilcoxon test** for dependent spot checks tests, whether the central tendencies are different. “Dependent spot checks” is used if one measurement in a sample and one measurement in another sample influence each other. In three situations this is the case: 1) Repeated measures 2) Natural pairs 3) Matching Basically it is possible to check for a batch effect with two categories only.

pc\_1\_BIOMATERIAL\_PROVIDER <- wilcox.test(batch\_wilcoxon$PC1 ~ batch\_wilcoxon$annotation.BIOMATERIAL\_PROVIDER, data = batch\_wilcoxon, exact = FALSE)  
pc\_2\_BIOMATERIAL\_PROVIDER <- wilcox.test(batch\_wilcoxon$PC2 ~ batch\_wilcoxon$annotation.BIOMATERIAL\_PROVIDER, data = batch\_wilcoxon, exact = FALSE)  
pc\_3\_BIOMATERIAL\_PROVIDER <- wilcox.test(batch\_wilcoxon$PC3 ~ batch\_wilcoxon$annotation.BIOMATERIAL\_PROVIDER, data = batch\_wilcoxon, exact = FALSE)  
pc\_4\_BIOMATERIAL\_PROVIDER <- wilcox.test(batch\_wilcoxon$PC4 ~ batch\_wilcoxon$annotation.BIOMATERIAL\_PROVIDER, data = batch\_wilcoxon, exact = FALSE)  
pc\_5\_BIOMATERIAL\_PROVIDER <- wilcox.test(batch\_wilcoxon$PC5 ~ batch\_wilcoxon$annotation.BIOMATERIAL\_PROVIDER, data = batch\_wilcoxon, exact = FALSE)  
  
pc\_1\_BIOMATERIAL\_PROVIDER <- pc\_1\_BIOMATERIAL\_PROVIDER$p.value  
pc\_2\_BIOMATERIAL\_PROVIDER <- pc\_2\_BIOMATERIAL\_PROVIDER$p.value  
pc\_3\_BIOMATERIAL\_PROVIDER <- pc\_3\_BIOMATERIAL\_PROVIDER$p.value  
pc\_4\_BIOMATERIAL\_PROVIDER <- pc\_4\_BIOMATERIAL\_PROVIDER$p.value  
pc\_5\_BIOMATERIAL\_PROVIDER <- pc\_5\_BIOMATERIAL\_PROVIDER$p.value  
  
pc\_1\_DISEASE <- wilcox.test(batch\_wilcoxon$PC1 ~ batch\_wilcoxon$annotation.DISEASE, data = batch\_wilcoxon, exact = FALSE)  
pc\_2\_DISEASE <- wilcox.test(batch\_wilcoxon$PC2 ~ batch\_wilcoxon$annotation.DISEASE, data = batch\_wilcoxon, exact = FALSE)  
pc\_3\_DISEASE <- wilcox.test(batch\_wilcoxon$PC3 ~ batch\_wilcoxon$annotation.DISEASE, data = batch\_wilcoxon, exact = FALSE)  
pc\_4\_DISEASE <- wilcox.test(batch\_wilcoxon$PC4 ~ batch\_wilcoxon$annotation.DISEASE, data = batch\_wilcoxon, exact = FALSE)  
pc\_5\_DISEASE <- wilcox.test(batch\_wilcoxon$PC5 ~ batch\_wilcoxon$annotation.DISEASE, data = batch\_wilcoxon, exact = FALSE)  
  
pc\_1\_DISEASE <- pc\_1\_DISEASE$p.value  
pc\_2\_DISEASE <- pc\_2\_DISEASE$p.value  
pc\_3\_DISEASE <- pc\_3\_DISEASE$p.value  
pc\_4\_DISEASE <- pc\_4\_DISEASE$p.value  
pc\_5\_DISEASE <- pc\_5\_DISEASE$p.value  
  
pc\_1\_Predicted.Gender <- wilcox.test(batch\_wilcoxon$PC1 ~ batch\_wilcoxon$annotation.Predicted.Gender, data = batch\_wilcoxon, exact = FALSE)  
pc\_2\_Predicted.Gender <- wilcox.test(batch\_wilcoxon$PC2 ~ batch\_wilcoxon$annotation.Predicted.Gender, data = batch\_wilcoxon, exact = FALSE)  
pc\_3\_Predicted.Gender <- wilcox.test(batch\_wilcoxon$PC3 ~ batch\_wilcoxon$annotation.Predicted.Gender, data = batch\_wilcoxon, exact = FALSE)  
pc\_4\_Predicted.Gender <- wilcox.test(batch\_wilcoxon$PC4 ~ batch\_wilcoxon$annotation.Predicted.Gender, data = batch\_wilcoxon, exact = FALSE)  
pc\_5\_Predicted.Gender <- wilcox.test(batch\_wilcoxon$PC5 ~ batch\_wilcoxon$annotation.Predicted.Gender, data = batch\_wilcoxon, exact = FALSE)  
  
pc\_1\_Predicted.Gender <- pc\_1\_Predicted.Gender$p.value  
pc\_2\_Predicted.Gender <- pc\_2\_Predicted.Gender$p.value  
pc\_3\_Predicted.Gender <- pc\_3\_Predicted.Gender$p.value  
pc\_4\_Predicted.Gender <- pc\_4\_Predicted.Gender$p.value  
pc\_5\_Predicted.Gender <- pc\_5\_Predicted.Gender$p.value  
  
p\_DISEASE <- data.frame(pc\_1\_BIOMATERIAL\_PROVIDER, pc\_2\_BIOMATERIAL\_PROVIDER, pc\_3\_BIOMATERIAL\_PROVIDER, pc\_4\_BIOMATERIAL\_PROVIDER, pc\_5\_BIOMATERIAL\_PROVIDER)  
p\_BIOMATERIAL\_PROVIDER <- data.frame(pc\_1\_BIOMATERIAL\_PROVIDER, pc\_2\_BIOMATERIAL\_PROVIDER, pc\_3\_BIOMATERIAL\_PROVIDER, pc\_4\_BIOMATERIAL\_PROVIDER, pc\_5\_BIOMATERIAL\_PROVIDER)  
p\_Predicted.Gender <- data.frame(pc\_1\_Predicted.Gender, pc\_2\_Predicted.Gender, pc\_3\_Predicted.Gender, pc\_4\_Predicted.Gender, pc\_5\_Predicted.Gender)

#### Kruskal-Wallis for several categories

The **Kurskal-Wallis test** checks, whether there is a difference in central tendencies between several independent spot checks. The test is used, if the prerequisites for a parametric method is not fulfilled. One of the benefits of the test is that the data does not have to be normally distributed. On the other hand, the test can also be calculated for small samples and outliers. Basically it is possible to check for a batch effect with several categories.

batch\_kruskal <- within(batch\_kruskal, {  
 PC1 <- as.numeric(as.character(PC1))  
})  
  
batch\_kruskal <- within(batch\_kruskal, {  
 PC2 <- as.numeric(as.character(PC2))  
})  
  
batch\_kruskal <- within(batch\_kruskal, {  
 PC3 <- as.numeric(as.character(PC3))  
})  
  
batch\_kruskal <- within(batch\_kruskal, {  
 PC4 <- as.numeric(as.character(PC4))  
})  
  
batch\_kruskal <- within(batch\_kruskal, {  
 PC5 <- as.numeric(as.character(PC5))  
})  
  
sample\_desc\_3\_pc1 <- kruskal.test(batch\_kruskal$PC1 ~ batch\_kruskal$annotation.SAMPLE\_DESC\_3, data = batch\_kruskal)  
sample\_desc\_3\_pc2 <- kruskal.test(batch\_kruskal$PC2 ~ batch\_kruskal$annotation.SAMPLE\_DESC\_3, data = batch\_kruskal)  
sample\_desc\_3\_pc3 <- kruskal.test(batch\_kruskal$PC3 ~ batch\_kruskal$annotation.SAMPLE\_DESC\_3, data = batch\_kruskal)  
sample\_desc\_3\_pc4 <- kruskal.test(batch\_kruskal$PC4 ~ batch\_kruskal$annotation.SAMPLE\_DESC\_3, data = batch\_kruskal)  
sample\_desc\_3\_pc5 <- kruskal.test(batch\_kruskal$PC5 ~ batch\_kruskal$annotation.SAMPLE\_DESC\_3, data = batch\_kruskal)  
  
p\_SAMPLE\_DESC\_3 <- data.frame(sample\_desc\_3\_pc1$p.value, sample\_desc\_3\_pc2$p.value, sample\_desc\_3\_pc3$p.value, sample\_desc\_3\_pc4$p.value, sample\_desc\_3\_pc5$p.value)

We create a data frame containing **all p values of the categories** we investigate for a batch effect.

p\_DISEASE\_t <- as.data.frame(t(p\_DISEASE))  
p\_BIOMATERIAL\_PROVIDER\_t <- as.data.frame(t(p\_BIOMATERIAL\_PROVIDER))  
p\_Predicted.Gender\_t <- as.data.frame(t(p\_Predicted.Gender))  
p\_SEQ\_RUNS\_COUNT\_t <- as.data.frame(t(p\_SEQ\_RUNS\_COUNT))  
p\_SAMPLE\_DESC\_3\_t <- as.data.frame(t( p\_SAMPLE\_DESC\_3))  
  
total\_pvalue <- cbind(p\_DISEASE\_t,p\_BIOMATERIAL\_PROVIDER\_t,p\_Predicted.Gender\_t,p\_SEQ\_RUNS\_COUNT\_t,p\_SAMPLE\_DESC\_3\_t)

We give columns and rows informative names.

names(total\_pvalue)[1] <- "p\_DISEASE"  
names(total\_pvalue)[2] <- "p\_BIOMATERIAL"  
names(total\_pvalue)[3] <- "p\_GENDER"  
names(total\_pvalue)[4] <- "p\_SEQ\_RUNS\_COUNT"  
names(total\_pvalue)[5] <- "p\_SAMPLE\_DESC"  
  
rownames(total\_pvalue)[rownames(total\_pvalue) == "pc\_1\_BIOMATERIAL\_PROVIDER"] <- "PC1"  
rownames(total\_pvalue)[rownames(total\_pvalue) == "pc\_2\_BIOMATERIAL\_PROVIDER"] <- "PC2"  
rownames(total\_pvalue)[rownames(total\_pvalue) == "pc\_3\_BIOMATERIAL\_PROVIDER"] <- "PC3"  
rownames(total\_pvalue)[rownames(total\_pvalue) == "pc\_4\_BIOMATERIAL\_PROVIDER"] <- "PC4"  
rownames(total\_pvalue)[rownames(total\_pvalue) == "pc\_5\_BIOMATERIAL\_PROVIDER"] <- "PC5"

#### Preparation for heatmap

To **decide which PC we will use for further analysis**, we draw a heatmap showing which principle component is affected significantly by a batch effect. All significant p-values are shown in red and non-significant ones in blue. For analyzing and comparing all p-values we state all p-values under 0.1 as significant. To make visualization easier we set values above the significance threshold to 10 and values equal or below to 1, so we just have to find colours for 2 values.

total\_pvalue <- data.matrix(total\_pvalue)  
  
for (i in 1:ncol(total\_pvalue)){  
 for (j in 1:nrow(total\_pvalue)){  
 if(isTRUE(total\_pvalue[i,j] > 0.1)){  
 total\_pvalue[i,j] <- 10  
 }  
 if (isTRUE(total\_pvalue[i,j] <= 0.1)){  
 total\_pvalue [i,j] <- 1  
 }  
 }  
}

#### Heatmap

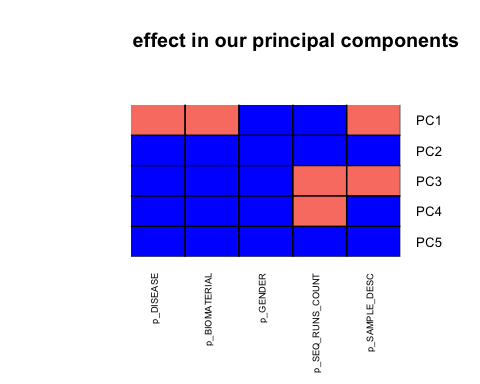
library(gplots)

## Warning: package 'gplots' was built under R version 3.5.2

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

par(cex.main = 1)  
   
 heatmap.2(  
 total\_pvalue, main = "Batch effect in our principal components",  
 title(main, cex.main = 1 \* op[["cex.main"]]),  
 margins = c(8,5),  
 Colv = NA,   
 Rowv = NA,  
 dendrogram = "none",   
 sepwidth = c(0.01, 0.01),   
 sepcolor = "black",   
 trace= "none",   
 col = colorRampPalette(c("salmon","blue")),   
 breaks = c(seq(0, 1, length = 2),  
 seq(1.1, 10, length = 2)),  
 colsep = 1:ncol(total\_pvalue),   
 rowsep = 1:nrow(total\_pvalue),   
 cexCol = 0.7,  
 cexRow = 1,  
 key = FALSE)



By looking at the heatmap or at the p-values one can see that principal component 1 has a batch effect. Therefore **we go on with our analysis using principal component 2** which doesn’t show a batch effect at all.

#### create a dataframe with loading scores of the 10000 most importat genes, which have the most influence for principal component 2.

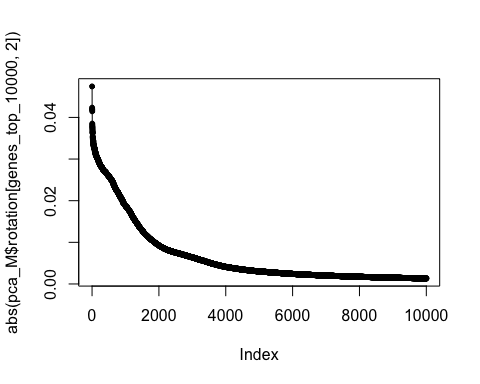
Now we want to **find the most important genes** which means that we want to find the genes which have the most influence on the principal component two. For that we use the loading scores which we can find in rotation of the PCA. The loadings describe the weight of each gene in the principal component. If a gene has a high loading it has a high weight in the principal component. A loading can be positive or negative but we will first have a look on the absolut values. We create a dataframe containing the 2000 and 10000 most important genes in principal component 2 in ranked order.

loading\_scores <- pca\_M$rotation[,2]  
gene\_scores <- abs(loading\_scores)   
ranked\_gene\_score <- sort(gene\_scores, decreasing=TRUE)  
genes\_top\_10000 <- names(ranked\_gene\_score[1:10000])  
genes\_top\_2000 <- names(ranked\_gene\_score[1:2000])

#### Look for elbow in top gene variance

Now we want to **decide how many genes we want to keep for our further analysis** by looking for an elbow in the plot of loading scores.

plot(abs(pca\_M$rotation[genes\_top\_10000,2]),type = "o", pch=20)



Actually there is a kink in the elbow plot of loading scores of the genes at round about 2000 genes. K-means clustering already worked with 2000 genes but by using only 2000 genes we get only 3 significant different methylated genes, which is not sufficient. By using 10000 genes we get 24 differentially methylated genes, which is an amount which we can work with.

## 4) Clustering with k-means

K-means is a method to find groups in the data. It uses an iterative algorithm which means that there is a repitition of steps with the aim of finding the right cluster for each datapoint. First step is the clustering which means that the datapoints get split up into k clusters. Each datapoint gets alloted to its next clusterpoint. Next step is an updating of the clusters by calculating center of each cluster and define them as new clusterpoints. These two steps will be repeated as long as set in the code. By performing k-means clustering we want to **check whether sick and healthy patients are separated in two different clusters.** Therefor we first create two dataframes containing all M-values from the 2000 and 10000 most important genes for principal component 2.

k\_means\_data2 <- M\_genes[genes\_top\_2000,]  
k\_means\_data <- M\_genes[genes\_top\_10000,]  
  
k <- kmeans(x = t(k\_means\_data2), centers = 2, iter.max = 1000)  
k <-  
 kmeans(  
 x = t(k\_means\_data),  
 centers = 2,  
 iter.max = 1000  
 )  
  
cluster <- data.frame(k[["cluster"]])  
cluster

## k...cluster...  
## 1H 1  
## 2H 1  
## 3H 2  
## 4H 1  
## 5H 2  
## 6CLL 2  
## 7CLL 2  
## 8CLL 2  
## 9CLL 2  
## 10CLL 2

One can see that our sick and healthy patients belong to two different clusters so we can use principal component two for further testing.

## 5) Identifying differetially methylated regions (DMRs)

### Student’s t-test

To see whether the methylation differences, so differences in M-values, between the two Sample groups are significant, **a student t-test is performed**. It can be used, because due to normalisation the varaiance of the values are equal now. The test compares the mean M-value of sick patients to the mean M-value of healthy patients and gives a p-value, which is significant or not depending on how strict is the threshold you set.

cluster <- k[["cluster"]]  
p\_value = NULL  
for(i in 1:nrow(k\_means\_data)){  
 x = k\_means\_data[i, 1:5]  
 y = k\_means\_data[i, 6:10]  
 t\_test = t.test(x, y)  
 p\_value[i] = t\_test$p.value  
}

We **create a dataframe wich contains the p-values** from t-test.

pvalues <- data.frame(p\_value)

To get the gene ID belonging to those p-values we **change the rownames**.

row\_names <- row.names(k\_means\_data)  
p\_combined <- pvalues  
rownames(p\_combined) <- row\_names

### Correction of p values by using the holm method

To reduce the family-wise error rate, which is the propability of getting one ore more type 1 errors (also called false positive error), the *holm method* is used. Besides a *dataframe* containing the uncorrected p-values and the corrected p-values in rising order and the belonging gene ID and symbol is created.

p\_combined$p\_adjusted = p.adjust(p\_combined$p\_value, method = "holm")  
p\_holm <- data.frame(p\_combined)  
symbols <- genes[rownames(p\_holm),]  
holm <- cbind(p\_holm, "Symbols" = symbols$symbol)  
holm\_new <- holm[order(holm$p\_value),]

### Finding DMRs

#### Significant different methylated genes

The p-values from the t-test were corrected with the holm method. 24 genes had a p-value under 0.1. Because of that we defined this genes of healthy patients as significantly differently methylated than the ones of sick patients. The most differently methylated gene is MIR-941-1. It is a gene for a microRNA (non-coding and short RNA) that plays a role in the transcriptional regulation of gene expression. MicroRNAs have an influence on stability and translation of mRNAs by being a part of the RNA induced silencing complex (RISC). RISC can bind mRNAs complementary to its miRNA or siRNA. This can either inhibit or destabilize the mRNA, so that less or no proteins coded by the mRNA get produced. MIR-941-1 is known to be transcriptionally silenced in gastric cancer cells which fits to our results. Other differently methylated genes in our dataset which are coding for miRNAs were MIR3687-1, MIR1915, MIR8075, MIR941-2, MIR941-3, MIR941-4 and MIR647. Another group of differently methylated genes play a role in splicing. Three of the 24 tested genes coded for the U6 subunit of the spliceosome, namely RNU6-481P, RNU6-638P and RNU6-1237P. Splicing is part of the processing of the mRNA in eukaryotes which makes it very important for the protein activity. Also, we found out that RNU7-116P is differently methylated. RNU7-116P codes for a U7 small nuclear ribonucleoprotein which is used for histone RNA processing. Both the U6 and U7 subunit both play a part in transcriptional regulation. Due to this regulation, proteins can be differently expressed which might lead to cancer development.

#### Genes we found in literature which are not significant different nethylated

Besides there have been genes, often correlated to CLL, which we found in literature before. **Literature genes**, which we also found in our initial dataset genes are those ones.

#grep("symbol", genes$symbol, value = TRUE)

* TWIST2
* VHL
* ABI3
* ADORA3
* PRF1
* CLLU1
* LPL
* NOTCH1
* SCARF1
* TNF
* CTBP2
* EGR1

The following genes from the literature can be found in our dataset heaped, as they probably belong to a family:

* NFAT
* EBF
* ODC

The **literature genes still existing in the dataframe after the t-test** are those ones.

#grep("gene\_name", holm\_new$Symbols, value = TRUE)

* ENSG00000048462 -> TNFRSF17
* ENSG00000159958 -> TNFRSF13C
* ENSG00000232810 -> TNF
* ENSG00000233125 -> ACTBP12 (Maybe not that important, because original gene out of literature was named “CTBP2”)

Those ones still existing in the dataframe after the t-test and the 24 significant genes are the relevant genes which may have an impact on CLL forming. We **create a dataframe of relevant genes** and their p-valus to have an overview of genes we should do some research about.

relevant\_genes1 <- holm\_new[c(1:24),]  
relevant\_genes2 <- holm\_new[c("ENSG00000048462","ENSG00000159958","ENSG00000232810","ENSG00000233125"),]  
relevant\_genes <- rbind(relevant\_genes1,relevant\_genes2)  
print(relevant\_genes)

## p\_value p\_adjusted Symbols  
## ENSG00000215946 1.175067e-09 1.175067e-05 MIR941-1  
## ENSG00000204802 1.015632e-08 1.015530e-04 <NA>  
## ENSG00000264890 1.376601e-08 1.376326e-04 <NA>  
## ENSG00000199643 1.475274e-08 1.474831e-04 RNU4-90P  
## ENSG00000206921 1.722710e-08 1.722021e-04 RNU6-481P  
## ENSG00000279920 2.285155e-08 2.284012e-04 <NA>  
## ENSG00000252391 3.152707e-08 3.150815e-04 RNU6-638P  
## ENSG00000277437 4.763914e-08 4.760580e-04 MIR3687-1  
## ENSG00000222071 6.322396e-08 6.317338e-04 MIR1915  
## ENSG00000273587 1.331822e-07 1.330624e-03 SNORA78  
## ENSG00000275908 1.661780e-07 1.660119e-03 <NA>  
## ENSG00000277942 2.189029e-07 2.186621e-03 MIR8075  
## ENSG00000207554 3.582260e-07 3.577961e-03 MIR647  
## ENSG00000211837 6.980247e-07 6.971173e-03 TRAJ53  
## ENSG00000278903 7.360642e-07 7.350337e-03 <NA>  
## ENSG00000273981 8.751361e-07 8.738234e-03 <NA>  
## ENSG00000274450 9.378005e-07 9.363000e-03 <NA>  
## ENSG00000216141 1.213785e-06 1.211721e-02 MIR941-2;MIR941-3;MIR941-4  
## ENSG00000248979 3.518207e-06 3.511874e-02 LAMTOR3P2  
## ENSG00000278639 4.424737e-06 4.416330e-02 <NA>  
## ENSG00000258577 4.986456e-06 4.976483e-02 SNRPGP1  
## ENSG00000277532 5.631739e-06 5.619912e-02 <NA>  
## ENSG00000207508 9.249911e-06 9.229562e-02 RNU6-1237P  
## ENSG00000252749 9.339817e-06 9.318335e-02 RNU7-116P  
## ENSG00000048462 3.398150e-02 1.000000e+00 TNFRSF17  
## ENSG00000159958 9.172109e-02 1.000000e+00 TNFRSF13C  
## ENSG00000232810 2.391774e-01 1.000000e+00 TNF  
## ENSG00000233125 5.847920e-01 1.000000e+00 ACTBP12

### Some additional information to our genes out of the literature

#### Gene number one: ACTBP12

ACTBP12 is just a pseudogene with no specific function. As the gene out of our literature analysis was named âCTBP2â and not âACTBP12â, this is not surprising. Still we wanted to see, whether there is a relation between these genes.

#### Gene number two: TNF

This gene belongs to the tumor necrosis factor (TNF) superfamily and encodes a multifunctional proinflammatory cytokine. This cytokine is mainly secreted by macrophages, which are also important for the immune system. By binding to specific receptors, this cytokine is involved in the regulation of a wide spectrum of biological processes including cell proliferation, differentiation, apoptosis, lipid metabolism, and coagulation. It is linked with a variety of diseases for example autoimmune diseases but also cancer, which is important in our case. These specific TNF receptors are expressed in a wide variety of tissues in mammals, especially in leucocytes which include B-cells. This signalling pathways, as already mentioned, is able to promote B-cell survival and proliferation. Sources to TNF: (Rickert et al., 2011) (Gravestein and Borst, 1998)

#### Gene number three: TNFRSF13C

Tumor necrosis factor receptor superfamily member 13C (TNFRSF13C) is, as the name suggests, a gene which encodes for a receptor of the TNF. It is also known as B-cell-activating factor receptor (BAFF-R). In general, BAFF promotes the proliferation and differentiation of B-cells.

Sources to TNFRSF13C: (Mackay et al., 2010)

#### Gene number four: TNFRSF17

The protein encoded by the gene tumor necrosis factor receptor superfamily member 17 (TNFRSF17) gene is a member of the TNF-receptor superfamily. It is a receptor, especially expressed in mature B lymphocytes and might play a role in B cell development. It leads to NF-kappaB and MAPK8/JNK activation and therefore it binds to different TRAF family members. It might transduce signals for cell survival and proliferation.

Sources to TNFRSF17: RefSeq, Jul 2008

#### Sources in general:

Gravestein, L.A., and Borst, J. (1998). Tumor necrosis factor receptor family members in the immune system. Seminars in Immunology 10, 423-434.

Rickert, R.C., Jellusova, J., and Miletic, A.V. (2011). Signaling by the tumor necrosis factor receptor superfamily in B-cell biology and disease. Immunological reviews 244, 115-133.

Mackay, F., Figgett, W.A., Saulep, D., Lepage, M., and Hibbs, M.L. (2010). B-cell stage and context-dependent requirements for survival signals from BAFF and the B-cell receptor. Immunological reviews 237, 205-225.

RefSeq, Jul 2008

### Statistical and biological relevant methylation differences in genes

A vulcano plot can help visualising whether a gene is claimed to be differently methylated in healthy and sick patients because of statistical reasons, which means a significant p-value or because of biological reasons, which would be a big difference in the mean of the M-values or beta-values. Only the genes which show both characteristics, really have a significant methylation difference.

#### Fold change calculation

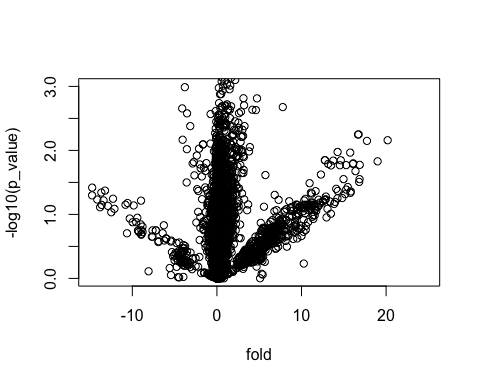
On the x-axis of the volcano plot there is the **difference of the beta-value means of each sample group**. This is the axis showing the biological relevance. Usually for better visualization you use the log2 fold change of the M-values. Due to the negative M-values that exist, this is mathematically not possible. Therefor we do not use the log2 fold change of the M-values, but the one of the beta-values.

beta\_10000 <- beta\_genes\_cleaned[genes\_top\_10000,]  
beta\_10000\_log <- log2(beta\_10000)  
control <- beta\_10000\_log[, 1:5]  
tumor <- beta\_10000\_log[, 6:10]  
control\_mean <- apply(control, 1, mean)  
tumor\_mean <- apply(tumor, 1, mean)  
fold = control\_mean - tumor\_mean

#### Visualisation including information about hyper- and hypomethylation

Also on the y-axis we have to improvise: The y-axis shows the p-values and therefor the statistical relevance. Actually the corrected p-values should be used, but somehow a lot of p-values were corrected to the value 1, which distorts the plot. Therefor we use the uncorrected p-values for the y-axis and **create a volcano plot**

plot(fold, -log10(p\_value), ylim = c(0,3))



To visualise genes that are statistically and biologically relevant and subdivide them into upmethylated and hypomethylated genes in case of cancer, we first **create thresholds for significance** in p-values and beta-value-difference (fold).

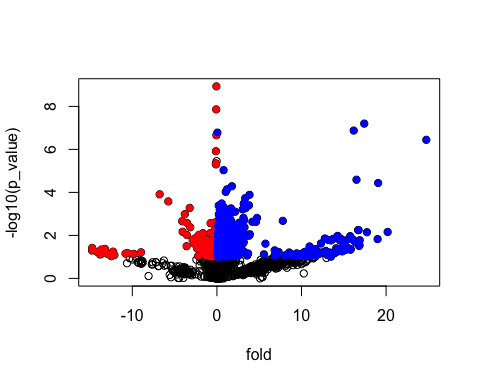
threshold\_significant = 0.1  
threshold\_fold = 0.05

We create a **filter for biological relevant genes** using the threshold for beta-value-difference, one for **statistically relevant genes** and one that **combines both in one filter**.

filter\_by\_fold = abs(fold) >= threshold\_fold  
filter\_by\_pvalue = abs(p\_value) <= threshold\_significant  
filter\_combined = filter\_by\_fold & filter\_by\_pvalue

The genes which fit this filter and are **upmethylated** in cancer cells are coloured in red, while the **hypomethylated** genes are coloured in blue.

plot(fold, -log10(p\_value))  
points (fold[filter\_combined & fold < 0],  
 -log10(p\_value[filter\_combined & fold < 0]),  
 pch = 16, col = "red")  
  
points (fold[filter\_combined & fold > 0],  
 -log10(p\_value[filter\_combined & fold > 0]),  
 pch = 16, col = "blue")



#### Hyper- and hypomethylation in relevant genes

We have had created a dataframe relevant\_genes containing the genes which show significant differences in methylation and four genes, which do not show significant methylation differences between sick and healthy, but seem to be worth a look, because we read about them in literature before. But to know wether those genes in cancer patients are hyper or hypomethylated we need to have a look at the fold value. Therefor we **add the fold values to the relevant genes**. Genes in cancer patients are **upmethylated** if the **fold is negative** and the absolute fold higher than 0.05. And Genes are **downmethylated** if the **fold is positiv** and the absolute fold higher than 0.05.

fold <- data.frame(fold)  
fold2 <- as.data.frame(fold[rownames(relevant\_genes),])  
relevant\_genes <- cbind(relevant\_genes, "fold" = fold2$fold)  
print(relevant\_genes)

## p\_value p\_adjusted Symbols  
## ENSG00000215946 1.175067e-09 1.175067e-05 MIR941-1  
## ENSG00000204802 1.015632e-08 1.015530e-04 <NA>  
## ENSG00000264890 1.376601e-08 1.376326e-04 <NA>  
## ENSG00000199643 1.475274e-08 1.474831e-04 RNU4-90P  
## ENSG00000206921 1.722710e-08 1.722021e-04 RNU6-481P  
## ENSG00000279920 2.285155e-08 2.284012e-04 <NA>  
## ENSG00000252391 3.152707e-08 3.150815e-04 RNU6-638P  
## ENSG00000277437 4.763914e-08 4.760580e-04 MIR3687-1  
## ENSG00000222071 6.322396e-08 6.317338e-04 MIR1915  
## ENSG00000273587 1.331822e-07 1.330624e-03 SNORA78  
## ENSG00000275908 1.661780e-07 1.660119e-03 <NA>  
## ENSG00000277942 2.189029e-07 2.186621e-03 MIR8075  
## ENSG00000207554 3.582260e-07 3.577961e-03 MIR647  
## ENSG00000211837 6.980247e-07 6.971173e-03 TRAJ53  
## ENSG00000278903 7.360642e-07 7.350337e-03 <NA>  
## ENSG00000273981 8.751361e-07 8.738234e-03 <NA>  
## ENSG00000274450 9.378005e-07 9.363000e-03 <NA>  
## ENSG00000216141 1.213785e-06 1.211721e-02 MIR941-2;MIR941-3;MIR941-4  
## ENSG00000248979 3.518207e-06 3.511874e-02 LAMTOR3P2  
## ENSG00000278639 4.424737e-06 4.416330e-02 <NA>  
## ENSG00000258577 4.986456e-06 4.976483e-02 SNRPGP1  
## ENSG00000277532 5.631739e-06 5.619912e-02 <NA>  
## ENSG00000207508 9.249911e-06 9.229562e-02 RNU6-1237P  
## ENSG00000252749 9.339817e-06 9.318335e-02 RNU7-116P  
## ENSG00000048462 3.398150e-02 1.000000e+00 TNFRSF17  
## ENSG00000159958 9.172109e-02 1.000000e+00 TNFRSF13C  
## ENSG00000232810 2.391774e-01 1.000000e+00 TNF  
## ENSG00000233125 5.847920e-01 1.000000e+00 ACTBP12  
## fold  
## ENSG00000215946 -0.062098540  
## ENSG00000204802 NaN  
## ENSG00000264890 -0.087534616  
## ENSG00000199643 NaN  
## ENSG00000206921 NaN  
## ENSG00000279920 NaN  
## ENSG00000252391 NaN  
## ENSG00000277437 NaN  
## ENSG00000222071 17.415912725  
## ENSG00000273587 16.170359452  
## ENSG00000275908 0.056731496  
## ENSG00000277942 -0.065590372  
## ENSG00000207554 24.754040645  
## ENSG00000211837 NaN  
## ENSG00000278903 NaN  
## ENSG00000273981 NaN  
## ENSG00000274450 NaN  
## ENSG00000216141 -0.113837067  
## ENSG00000248979 -0.035460336  
## ENSG00000278639 -0.122955439  
## ENSG00000258577 -0.107881498  
## ENSG00000277532 NaN  
## ENSG00000207508 0.787502101  
## ENSG00000252749 NaN  
## ENSG00000048462 1.307070457  
## ENSG00000159958 -1.316244737  
## ENSG00000232810 -0.978602969  
## ENSG00000233125 -0.006033483

## 6) Logistic regression

Logistic regression is used to model the relationship between a binary outcome variable and one or more predictor variables. In this case, the predictor variables are methylation data of the genes and the outcome is either healthy or sick. Hence we want to **predict a patient’s health status by the methylation status of their genes**.

We create a dataframe, which contains M values and health status of 7 random patients. Therefore, the M value dataframe needs to be transformed first. We then add the patients’ health status. This data frame is used for generating the logistic regression model. We then do cross validation by applying the regression model to the test set data frame and by checking whether the predicted and actual health status match, we can see if the regression model makes works.

log\_regression <- k\_means\_data[,c(1,2,4,5,8,9,10)]  
log\_regression <- t(log\_regression)  
log\_regression <- data.frame(log\_regression)  
test\_set <- k\_means\_data[,c(3,6,7)]  
test\_set <- data.frame(t(test\_set))  
  
healthstatus <- annotation$DISEASE  
healthstatus <- data.frame(healthstatus)  
healthstatus\_regression <- healthstatus[c(1, 2, 4, 5, 8, 9, 10),]  
  
train\_set <- cbind(healthstatus\_regression, log\_regression)  
  
regression\_model <- glm(healthstatus\_regression ~ ., family = "binomial", data = train\_set)  
summary\_regression\_model <- summary(regression\_model)  
summary\_regression\_model$coefficients

## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -68.3042527 272581.35 -0.0002505830 0.9998001  
## ENSG00000275217 4.6430226 17166.33 0.0002704727 0.9997842  
## ENSG00000280145 25.3715551 98344.01 0.0002579878 0.9997942  
## ENSG00000227979 -31.7522839 157443.93 -0.0002016736 0.9998391  
## ENSG00000276242 2.3408763 14049.26 0.0001666192 0.9998671  
## ENSG00000271650 -0.8263456 43984.03 -0.0000187874 0.9999850  
## ENSG00000272312 1.4226377 4825.18 0.0002948362 0.9997648

We only get estimate values for the genes shown above whereas the values for the remainig genes are NA and therefore not shown in the table. This is because of multicollinearity, which means that one or more predictor variables are highly correlated and consequently do not contribute to the regression model. The estimate value explains the influence of the predictor variable on the outcome variable. Positive values indicate that a gene contributes to health whereas genes with negative values promote disease. It is notable that we get high standard errors, which implies that the regression model is not very reliable. These high standard error values result from mulitcollinearity.

prediction <- predict(regression\_model, newdata = test\_set, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =  
## ifelse(type == : prediction from a rank-deficient fit may be misleading

prediction

## 3H 6CLL 7CLL   
## 3.012131e-10 2.220446e-16 2.220446e-16

levels(train\_set$healthstatus\_regression)

## [1] "Chronic Lymphocytic Leukemia" "None"

Since we only have very few data available, the prediction of the logistic regression model based on the train set containing 70% of our data does not correspond with the actual health status. Even though the predicted probability is 10^6 times higher for the healthy patient than for the ones with CLL, it is still very close to 0, which implies that the patient is probably sick. Therefore we use all data available to model logistic regression without performing cross validation.

k\_means\_data\_no\_cv <- t(k\_means\_data)  
k\_means\_data\_no\_cv <- data.frame(k\_means\_data\_no\_cv)  
full\_data <- cbind(healthstatus, k\_means\_data\_no\_cv)  
model\_no\_cv <- glm(healthstatus ~ ., family = "binomial", data = full\_data)  
prediction\_no\_cv <- predict(model\_no\_cv, newdata = full\_data, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =  
## ifelse(type == : prediction from a rank-deficient fit may be misleading

summary\_model\_no\_cv <- summary(model\_no\_cv)  
summary\_model\_no\_cv$coefficients

## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -62.3090250 123717.301 -5.036404e-04 0.9995982  
## ENSG00000275217 5.0612636 10153.185 4.984902e-04 0.9996023  
## ENSG00000280145 54.1513644 143983.605 3.760940e-04 0.9996999  
## ENSG00000227979 -50.0417969 141092.708 -3.546732e-04 0.9997170  
## ENSG00000276242 2.1658625 8097.999 2.674565e-04 0.9997866  
## ENSG00000271650 -9.8198190 42406.585 -2.315635e-04 0.9998152  
## ENSG00000272312 -0.3410775 7096.218 -4.806469e-05 0.9999616  
## ENSG00000265127 -2.1800901 9128.043 -2.388343e-04 0.9998094  
## ENSG00000271876 0.1846260 5218.514 3.537904e-05 0.9999718  
## ENSG00000278927 -0.9134562 5588.323 -1.634580e-04 0.9998696

We now apply the second model to our data set.

prediction\_no\_cv

## 1H 2H 3H 4H 5H   
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00   
## 6CLL 7CLL 8CLL 9CLL 10CLL   
## 2.143345e-11 2.143345e-11 2.143345e-11 2.143345e-11 2.143345e-11

The predicted probability to be healthy is 1 for the healthy patients and very close to 0 for the sick patients, so it corresponds with the actual health status. This is the result we expected, since we made the prediction for the data we used to create the model.