Signature Project

Christina Lee

5/1/2022

Download Bioconductor and install required libraries:

```
# install bioconductor
#if (!require("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")

# install needed libraries from bioconductor
#BiocManager::install("GEOquery")
#BiocManager::install("FCBF")


# install libraries for creating biplots for PCA
#install_github("vqv/ggbiplot")
```

Load libraries (Note: install any that you need to install in order to load):

```
# load needed libraries
library(dplyr)
library(tidyr)
library(GEOquery)
library(ggplot2)
library(gplots)
library(FCBF)
library(pheatmap)
library(devtools)
library(ggbiplot)
library(factoextra)
library(ROCR)
library(limma)
library(psych)
library(caret)
library(foreach)
```

Read in the data set from GEO database using GEOquery:

```
# save the accession number into a variable
geo.id <- "GSE70947"

# query the GEO database with accession id
gse <- getGEO(geo.id)
```

The data set we want to analyse will usually be the first object in the list. So, let's extract it:

```
# extract the first object in the list
gse <- gse[[1]]
```

```
gse
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 62976 features, 296 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM1823702 GSM1823703 ... GSM1823997 (296 total)
##   varLabels: title geo_accession ... tumor_size_cm:ch1 (49 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: 1 2 ... 62976 (62976 total)
##   fvarLabels: ID ProbeName ... SPOT_ID (10 total)
##   fvarMetadata: Column Description labelDescription
## experimentData: use 'experimentData(object)'
## Annotation: GPL13607
```

Here, we can see that there are 62976 features (genes), and 296 samples in this data set.

Data submitted to GEO contain sample labels assigned by the experimenters, and additional info about the processing protocol. All of these info can be extracted by the pData() to gain access to "phenoData":

```
sample.info <- pData(gse)
#sample.info
```

Now, we can select the specific columns of interest. It seems like we will be needing the "characteristics_ch1.1" which tells us whether the sample is normal/tumor sample and "characteristics_ch1.5" which tells us the age of the patients:

```
# extract wanted columns
sample.info.sub <- dplyr::select(sample.info, characteristics_ch1.1,
                                 characteristics_ch1.5)

# rename the columns
sample.info.sub <- dplyr::rename(sample.info.sub, group= characteristics_ch1.1,
                                 age = characteristics_ch1.5)


# remove unwanted strings in each column and save to a new data frame
metadata.modified <- dplyr::mutate(sample.info.sub,
                                   group=gsub("tissue: ", "", group),
                                   age = gsub("age_in_years: ", "", age))


# convert age from chr to numeric
metadata.modified$age <- as.numeric(metadata.modified$age)


# round age up to nearest integer --the "-1" excludes column 1
metadata.modified[,-1] <- round(metadata.modified[,-1],0)


# check new data frame
#metadata.modified
```

To access data about genes such as gene names and IDs from an ExpressionSet class, we can use fData() to

gain access to "featureData":

```
gene.info <- fData(gse)
#gene.info
```

Now, we can select the specific columns of interest. Here, we will be needing the "GeneName" and "ID" columns. Once extracted, I want to save it as a data frame for later use:

```
# extract wanted columns
gene.info.sub <- dplyr::select(gene.info, GeneName)

# transpose
gene.info.final <- as.data.frame(gene.info.sub)

# check new data frame
#gene.info.final
```

Extract gene expression data:

```
# obtain the gene expression data and save it to a variable
gene.expr <- exprs(gse)

# convert to data frame
gene.expr <- as.data.frame(gene.expr)
```

Check the first 6 rows of "gene expr" data frame with sample # and gene ID along with expression values:

```
#head(gene.expr)
```

Next, check if the data set has been log2 transformed & normalized or not by randomly checking some of the summary statistics of the samples–the numbers should be in a small range if they are:

```
# randomly check descriptive stats of 4 samples --seems to be normalized already
summary(gene.expr$GSM1823702)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -1.440   4.825   6.841   7.370   9.504  18.441    3694
```

```
summary(gene.expr$GSM1823726)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -1.440   4.851   6.843   7.376   9.500  18.290    2522
```

```
summary(gene.expr$GSM1823997)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -1.440   4.850   6.859   7.383   9.509  18.441    3405
```

```
summary(gene.expr$GSM1823863)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -1.083   4.870   6.851   7.385   9.512  18.441    2810
```

It looks like the data set has been normalized. Just to make sure, we can double check how expression data was processed by the experimenters using pData():

```
# let's double check -- it is indeed quantile normalized.
pData(gse)$data_processing[1]
```

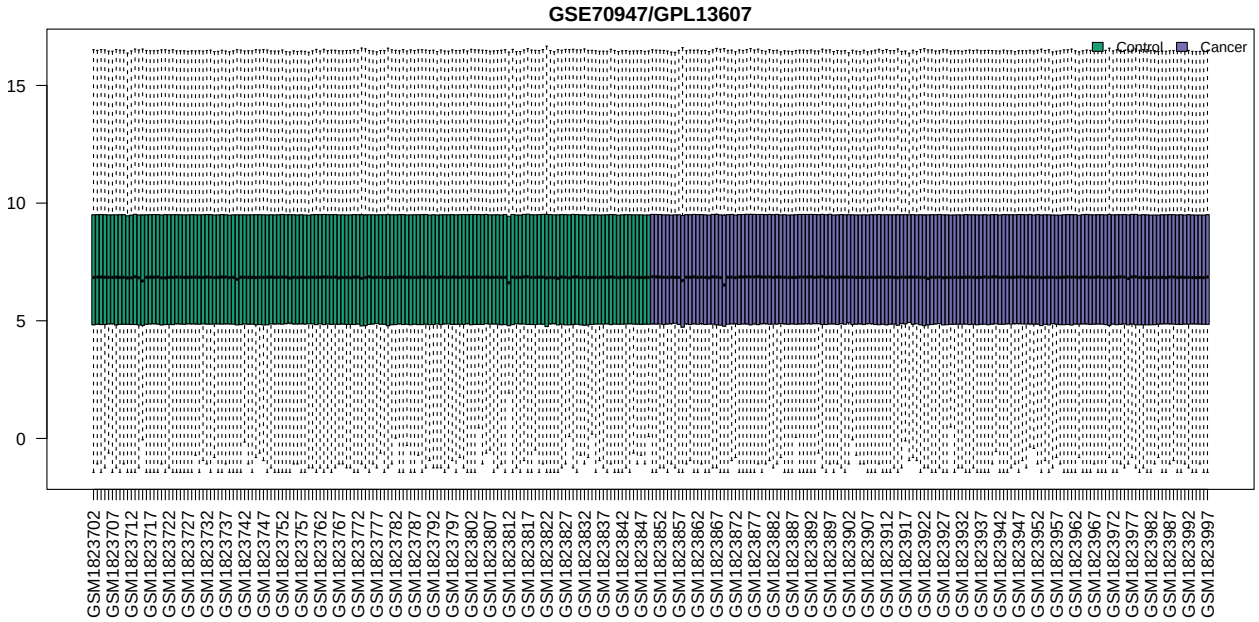
```
## [1] "Expression data were background-corrected and quantile normalized using limma in R. Features fla
```

From this output, we can see that the expression data is already quantile normalized using the limma package from Bioconductor. To double-check, I've also looked on GEO database and examined the experimenter's notes on the expression values in the samples (sample accession number starts with GSM), and the expression values are actually already log2 quantile normalized. Therefore, the data set is already log2 transformed as well as quantile normalized by the experimenters and we're all set to go.

Next, let's check if there are any outliers in the data set with box plots:

```
# group membership for all samples --148 control & 148 cancer samples
gsms <- paste0("00000000000000000000000000000000000000000000000000",
        "00000000000000000000000000000000000000000000000000",
        "00000000000000000000000000000000000000000000000011",
        "11111111111111111111111111111111111111111111111111",
        "11111111111111111111111111111111111111111111111111",
        "11111111111111111111111111111111111111111111111111")

sml <- strsplit(gsms, split="")[[1]]

# assign samples to groups
gs <- factor(sml)
groups <- make.names(c("Control","Cancer"))
levels(gs) <- groups
gse$group <- gs
```

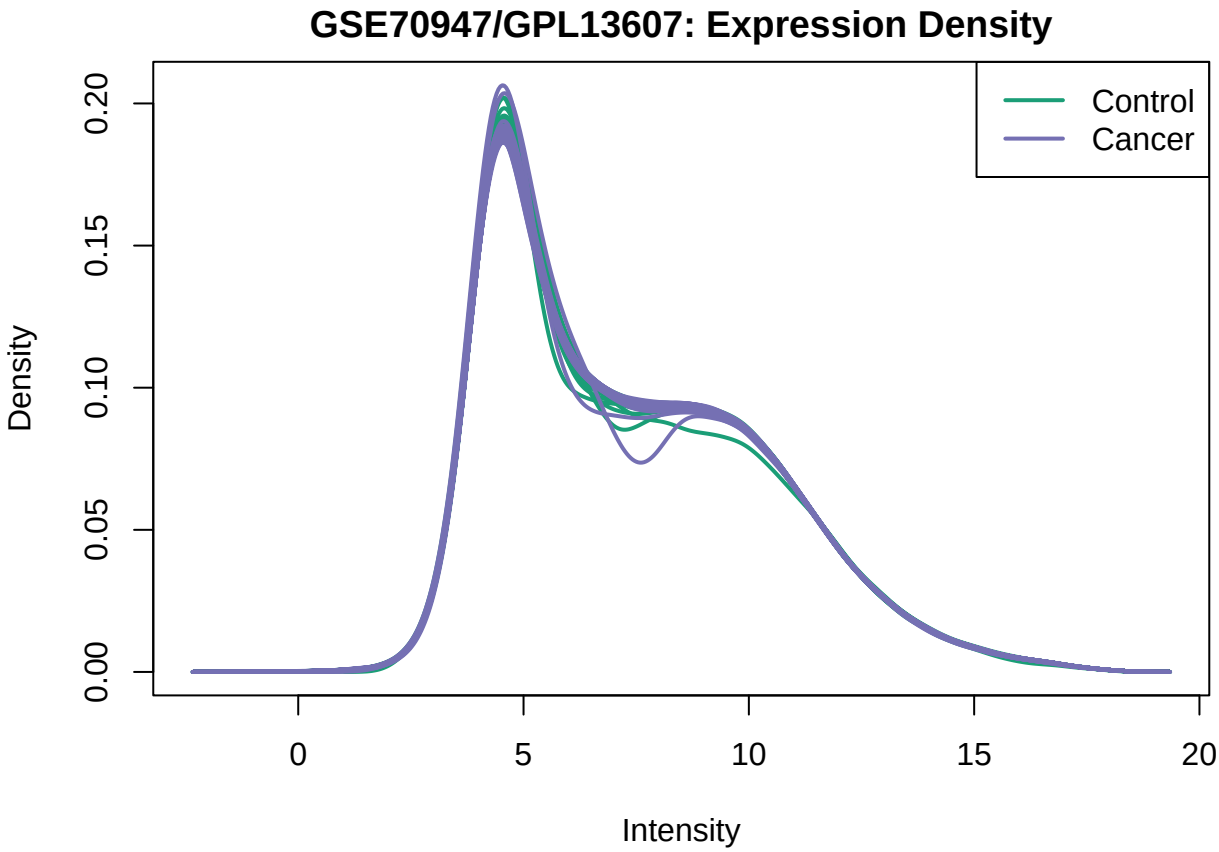
```
# construct box plot
ord <- order(gs)  # order samples by group
palette(c("#1B9E77", "#7570B3", "#E7298A", "#E6AB02", "#D95F02",
          "#66A61E", "#A6761D", "#B32424", "#B324B3", "#666666"))
# set margins
par(mar=c(6.5,2,1.5,0.3))
# add title to plot
title <- paste ("GSE70947", "/", annotation(gse), sep ="")
boxplot(gene.expr[,ord], boxwex=0.9, notch=T, main=title, outline=F,
        las=2, col=gs[ord])
# add legends to plot
legend("topright", groups, fill=palette(), bty="n", horiz = T, cex = 0.85)
```



Here, we can see that there are no outliers and samples (control/cancer) have their median centered which indicates that the data is, again, normalized and is cross-comparable.

Create a expression density plot to view the distribution of the values of the samples:

```
# expression value distribution
par(mar=c(4,4,2,1))
title <- paste ("GSE70947", "/", annotation(gse), ":", " Expression Density",
                sep ="")
plotDensities(gene.expr, group=gs, main=title, legend ="topright")
```



Here, density curves do not differ from sample to sample. Thus, also signifies that the data set is normalized. In addition, there are skewness in the distribution, however, since it has already been log2 transformed I will leave it at that.

Next, let's first check how many total NA's are in the data set:

```
# check number of NA's
sum(is.na(gene.expr))
```

```
## [1] 729494
```

Notice, there are NA's in this data set. That's normal. Let me explain. The expression data without performing log-transformations has a very large range (from 0 up to 1,000,000) with a lot of outliers in the upper range. Therefore, it is a common practice to log2 transform the data to resolve this issue. However, this often introduce another problem, because log-transformation results in data that contains negative infinity (-Inf) values or NAs and are caused by all the 0-values in the data since log2(0) == -Inf. Knowing that these values are really just 0s before the log transform, I will go ahead and impute them from NAs to 0.

```
# make a copy of the original data set
gene.expr.zero <- gene.expr
```

ZEROImpute:

```
# impute all NAs to 0
gene.expr.zero <- gene.expr.zero %>%
                  replace(is.na(gene.expr.zero), 0) %>%
                  mutate_if(is.numeric, round, digits = 3)  # round the values
```

Check if imputation was successful:

```
# check to see if all NAs have been imputed
sum(is.na(gene.expr.zero))
```

```
## [1] 0
```

Check data frame to see if imputation was successful:

```
# check data frame
#head(gene.expr.zero)
```

Transpose data frame so that samples => rows and genes => columns:

```
# transpose data frame
gene.expr.zero<- as.data.frame(t(gene.expr.zero))

# check data frame
#head(gene.expr.zero)
```

Map Gene name to Gene IDs:

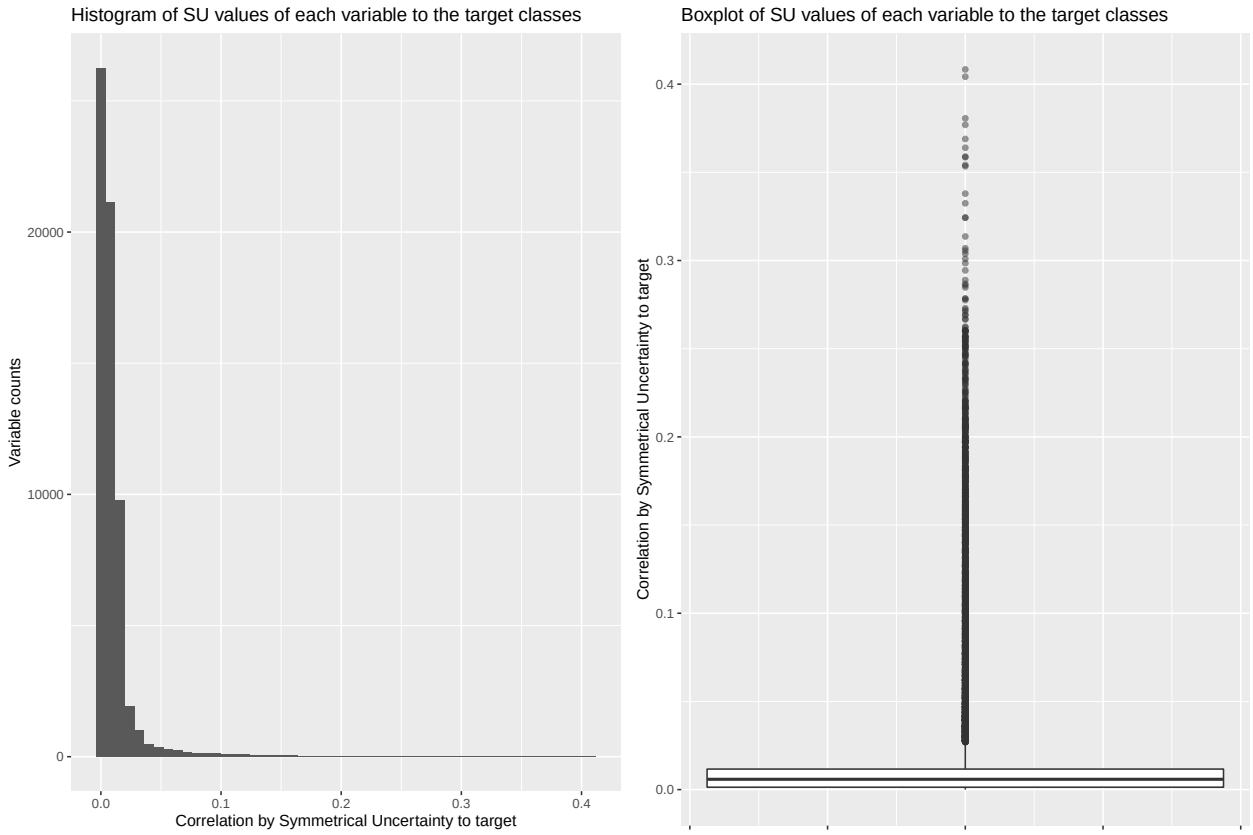
```
# map gene ID to gene names so we have gene names as our column name
colnames(gene.expr.zero) <- gene.info.final[,1]

# check data frame
#head(gene.expr.zero)
```

Once we're done mapping the gene ID to gene name. Here, I will filter out highly correlated genes with by implementing FCBF (Fast Correlation Based Filter for Feature Selection) from Bioconductor. The algorithm uses the idea of "predominant correlation". It selects features with high correlation with the target and little correlation with other variables. It does not use the classical Pearson or Spearman correlations, but a metric called Symmetrical Uncertainty (SU). The algorithm selects features correlated with the target above a given SU threshold. It then detects predominant correlations of features with the target. A predominant correlation occurs when for a feature "X", no other feature is more correlated to "X" than "X" is to the target.The features are ranked, and the iteration starts with X, the best ranked. The features more correlated with X than with the target are removed. Any features less correlated with X than with the target are kept. The algorithm then proceeds to the 2nd best ranked feature on the trimmed list and so on..:

```
# transpose
gene.expr.zero.t <- as.data.frame(t(gene.expr.zero))

# first, for FCBF to work the gene expression has to be discretized.
discrete.gene.expr <- as.data.frame(discretize_exprs(gene.expr.zero.t))


# FCBF also requires a target class variable as input
group <- as.factor(metadata.modified$group)
```

Let's first make a histogram of SU values of each variable to the target classes in order to distinguish the threshold for SU.

```
# plot SU
su <- su_plot(discrete.gene.expr, group)
```




```
su
```

```
## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

According to the plot, seems like 0.1 as a threshold that is reasonable for this data set.

We can now proceed to run FCBF:

```
# run FCBF to get filtered features/genes
fcbf.features <- fcbf(discrete.gene.expr, group, minimum_su = 0.1)
```

```
## [1] "Number of features features =  62976"
## [1] "Number of prospective features =  962"
## [1] "Number of final features =  24"
```

After running FCBF, we went from 62976 features to now a lean set of 24 features/genes.

Examine the genes that were chosen by FCBF along with the SU values:

```
# check genes
fcbf.features
```

```
##                                    index        SU
## EZH2.1                             54725 0.4083618
## COL10A1.1                          16328 0.4042125
## LOC100132724                       18819 0.3069758
## lincRNA.chr2.120459730.120511405_R 28148 0.2469364
## MS4A1.1                            35072 0.1901143
## PTPN1.1                            39613 0.1774414
```

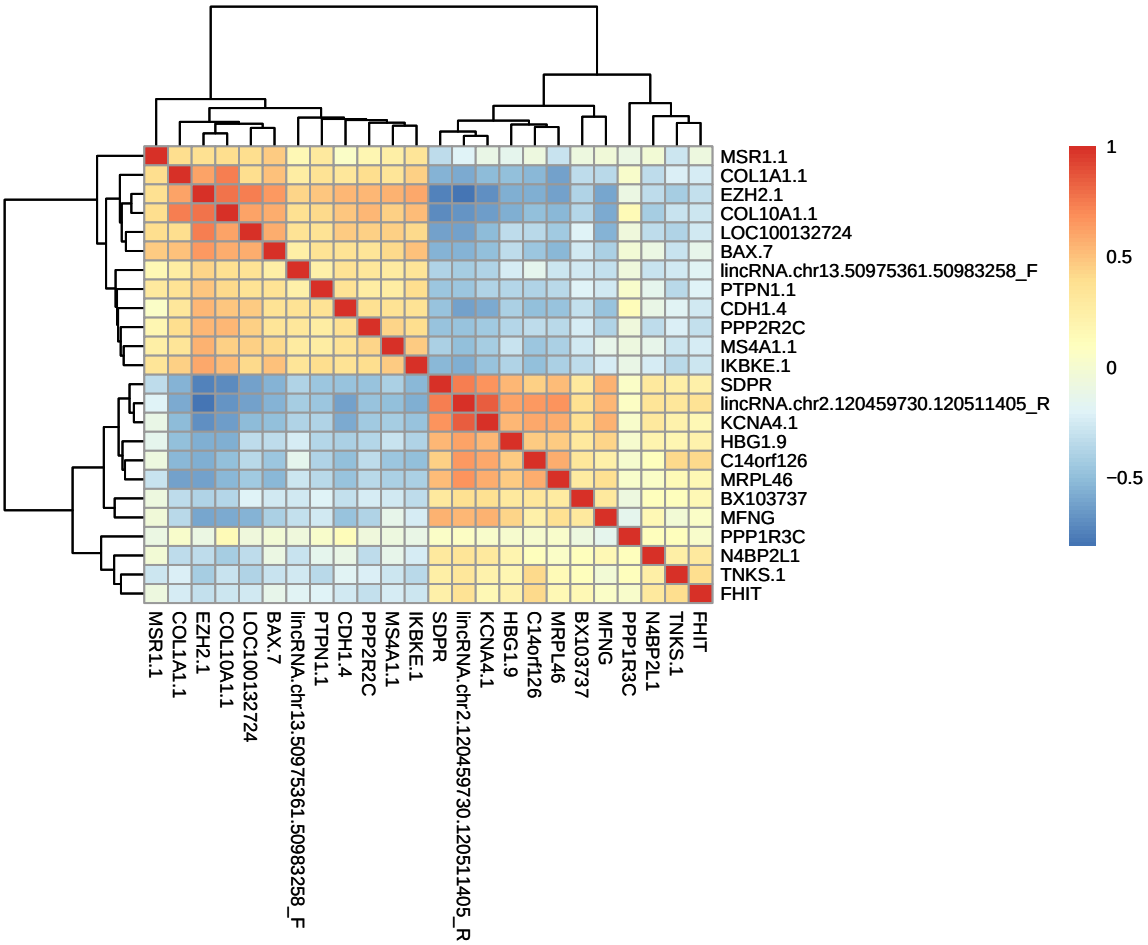
```
## COL1A1.1                                24670 0.1766899
## TNKS.1                                  36588 0.1616661
## N4BP2L1                                 25686 0.1605699
## BAX.7                                   41972 0.1580289
## PPP2R2C                                 18476 0.1565958
## HBG1.9                                  47773 0.1467538
## C14orf126                               41840 0.1394830
## lincRNA.chr13.50975361.50983258_F       17664 0.1311711
## MSR1.1                                  21690 0.1311011
## SDPR                                    59278 0.1268186
## BX103737                                27422 0.1242469
## MFNG                                     8291 0.1226883
## MRPL46                                   2537 0.1093258
## KCNA4.1                                 27130 0.1093258
## CDH1.4                                  28436 0.1055050
## FHIT                                    30230 0.1039724
## IKBKE.1                                 36709 0.1004273
## PPP1R3C                                 24182 0.1002604
```

As can see, EZH2.1 has the strongest correlation to the target class with an SU value of 0.41 and then comes COL10A1.1 with an SU value of 0.40 and so on.

```
# take the new features and create a data frame
fcbf.table <- gene.expr.zero.t[fcbf.features$index,]

# transpose and supply the class label
gene.expr.fcbf <- cbind(as.data.frame(t(fcbf.table)), Class = group)

# check data frame structure
#head(gene.expr.fcbf)
```

Heat map & dendrogram to illustrate gene correlation of the 24 genes:

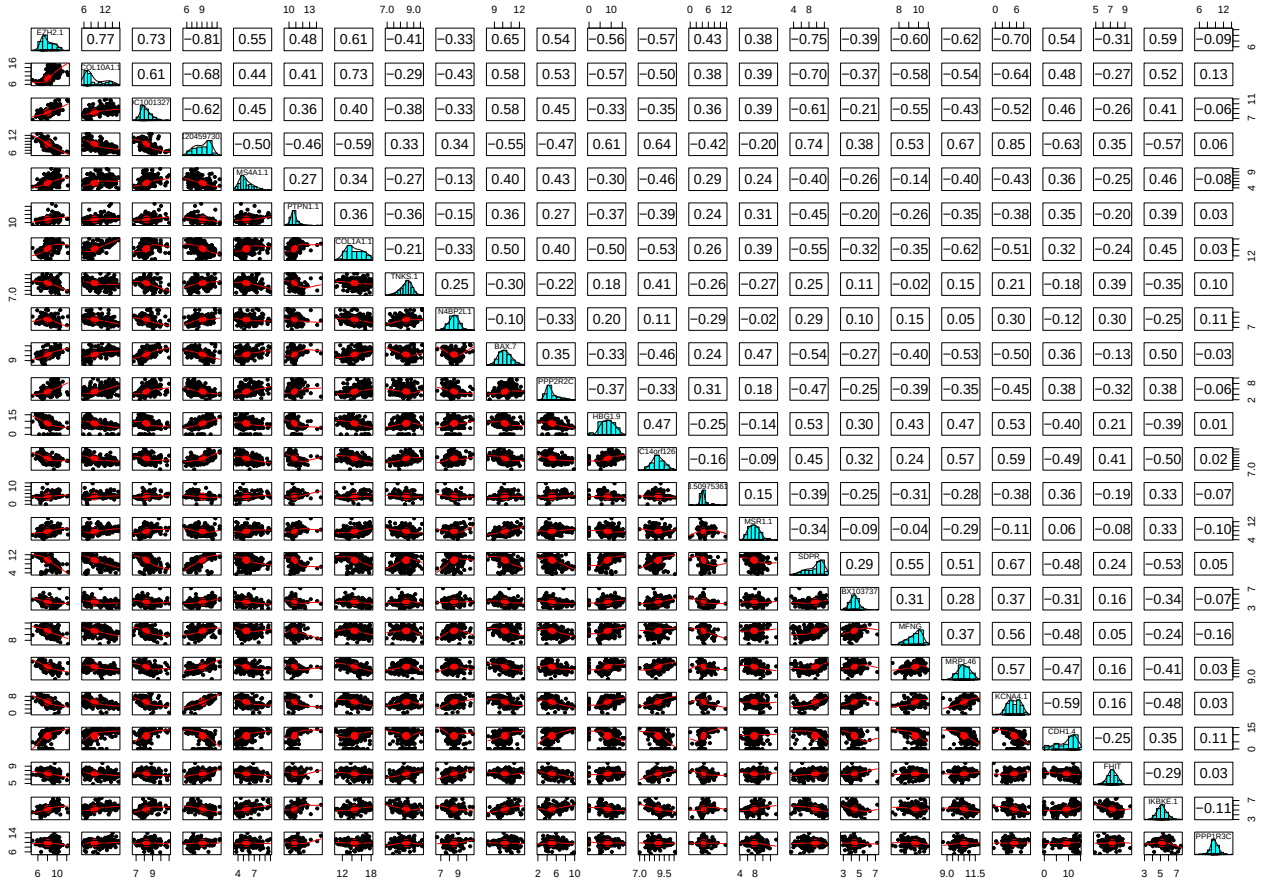
```
## argument use="c" stops an error if there are any missing data points
corMatrix <- cor((gene.expr.fcbf[-25]), use="c")
hmap <- pheatmap(corMatrix, cex = 0.81)
hmap
```



Looking at the heat map, we can see most of the genes are not either positive or negatively correlated with each other as they appear in lighter color (blue = negative correlation, red = positive correlation). However, there are some that seems to be quite correlated with each other. For example, EZH2.1 and COL10A1.1 and LOC100132724 are fairly correlated with each other, SDPR, LincRNA.chr2.120459730.120511405_R, and KCNA4.1 are also quite correlated as well.

We can also take a look at the the distribution of the genes and another view at the correlations between the genes with pairs.panels():

```
pairs.panels(gene.expr.fcbf[-25], cex.cor = 1.5)
```



We can see from the plot that most of the gene's distribution are quite normal with a few (~5) that has some skewness to their distributions. It is also worth noting the fact that some genes are indeed quite correlated to each other as seen on the heat map.

Next, although all samples are in the same range I want to further scale the expression values into a smaller range that is close to zero for my classification models so I will z-score transform them here. I will be z score scaling my FCBF data set with just 24 genes and run PCA analysis on it and train my model with newly derived PCs and the 24 genes that I've obtained from FCBF to see which allows the best performance.

FCBF Data Set Scaling:

```
# z-score scale gene expression values without class label
gene.expr.fcbf.z <- as.data.frame(lapply(gene.expr.fcbf[-25], scale))

# round all values
gene.expr.fcbf.z <- round(gene.expr.fcbf.z, digits = 3)

# add the sample names back to row names
rownames(gene.expr.fcbf.z)  <-  rownames(metadata.modified)

# supply another metadata "age" to data frame
gene.expr.fcbf.z$Age <- as.numeric(metadata.modified$age)

# supply class label back
gene.expr.fcbf.z$Class <- as.character(metadata.modified$group)


# check data frame
```

```
#head(gene.expr.fcbf.z)
```

FCBF Data set: Check mean and sd of 3 random features to see if they are z-score transformed:

```
# now they all have a mean of 0 and sd of 1
round(mean(gene.expr.fcbf.z$EZH2.1))    # mean
```

```
## [1] 0
```

```
round(mean(gene.expr.fcbf.z$FHIT))
```

```
## [1] 0
```

```
round(mean(gene.expr.fcbf.z$PPP2R2C))
```

```
## [1] 0
```

```
round(sd(gene.expr.fcbf.z$EZH2.1))    # sd
```

```
## [1] 1
```

```
round(sd(gene.expr.fcbf.z$FHIT))
```

```
## [1] 1
```

```
round(sd(gene.expr.fcbf.z$PPP2R2C))
```

```
## [1] 1
```

FCBF Data Set

Split into train and test sets:

```
set.seed(345)

# shuffle the order of the samples
rows <- sample(nrow(gene.expr.fcbf.z))
```

```
# use random vector to reorder the FCBF dataset
gene.expr.fcbf.new <- gene.expr.fcbf.z[rows, ]
```

```
set.seed(345)
# random sample 30% of data set into test and the rest 70% into train set
gene.splits.2 <- createDataPartition(gene.expr.fcbf.new$Class, p= 0.70, list = F)


# split --should have 208 observations in train and 88 in test
genes.fcbf.training <- gene.expr.fcbf.new[gene.splits.2, ]
genes.fcbf.testing <- gene.expr.fcbf.new[-gene.splits.2, ]
```

Check class distribution between train and test sets –they are even 50/50 split

```
prop.table(table(genes.fcbf.training$Class))  # train set
```

```
##
## normal  tumor
##    0.5    0.5
```

```
prop.table(table(genes.fcbf.testing$Class))   # test set
```

```
##
## normal  tumor
```

```
##    0.5    0.5
```

Convert class label into binaries of 1 = tumor and 0 = normal:

```
# TRAIN SET: encode target variable
genes.fcbf.training$Class[genes.fcbf.training$Class == "normal"] <- 0   # normal to 0
genes.fcbf.training$Class[genes.fcbf.training$Class == "tumor"] <- 1    # tumor to 1

# covert chr to factor
genes.fcbf.training$Class<- as.factor(genes.fcbf.training$Class)
```

```
# TEST SET: encode target variable
genes.fcbf.testing$Class[genes.fcbf.testing$Class == "normal"] <- 0   # normal to 0
genes.fcbf.testing$Class[genes.fcbf.testing$Class == "tumor"] <- 1    # tumor to 1

# covert chr to factor
genes.fcbf.testing$Class<- as.factor(genes.fcbf.testing$Class)
```

Check FCBF class label

```
#  class labels should be binaries of 0 and 1
str(genes.fcbf.training$Class)  # train
```

```
##  Factor w/ 2 levels "0","1": 1 2 1 2 1 2 1 2 2 1 ...
```

```
str(genes.fcbf.testing$Class)   # test
```

```
##  Factor w/ 2 levels "0","1": 2 1 1 1 1 1 2 1 1 1 ...
```

Check structure of train and test set

```
# TRAIN SET -- should have 208 observations 26 variables
#str(genes.fcbf.training)
```

```
## TEST SET --should have 88 observations 26 variables
#str(genes.fcbf.testing)
```

Conduct Principal Component Analysis (PCA) for FCBF Data Set:

```
# exclude the class label [-26] from PCA analysis --no need to scale or center
# it is already z-score scaled/normalized
pc <- prcomp(genes.fcbf.training[,-26])

# check summary
summary(pc)
```

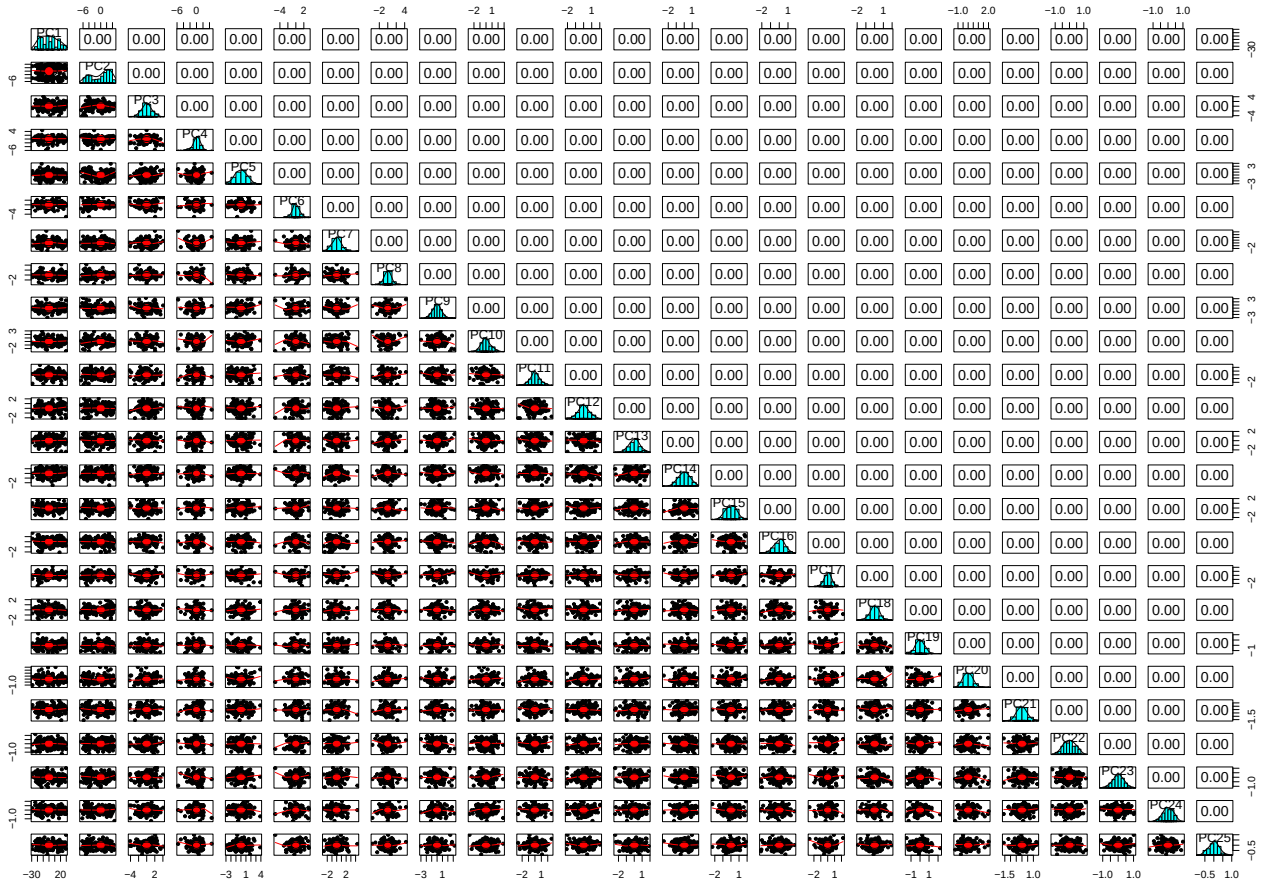
```
## Importance of components:
##                             PC1     PC2     PC3    PC4     PC5     PC6     PC7
## Standard deviation     14.3346 3.19581 1.33529 1.2607 1.16259 1.04718 1.00475
## Proportion of Variance  0.8925 0.04436 0.00774 0.0069 0.00587 0.00476 0.00438
## Cumulative Proportion   0.8925 0.93689 0.94463 0.9515 0.95741 0.96217 0.96655
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.96947 0.90543 0.85904 0.81922 0.75995 0.73220 0.71753
## Proportion of Variance 0.00408 0.00356 0.00321 0.00292 0.00251 0.00233 0.00224
## Cumulative Proportion  0.97064 0.97420 0.97740 0.98032 0.98283 0.98515 0.98739
##                           PC15    PC16    PC17    PC18    PC19   PC20    PC21
## Standard deviation     0.68581 0.64825 0.61365 0.58609 0.53305 0.5030 0.46456
## Proportion of Variance 0.00204 0.00183 0.00164 0.00149 0.00123 0.0011 0.00094
## Cumulative Proportion  0.98943 0.99126 0.99290 0.99439 0.99562 0.9967 0.99766
```

```
##                             PC22    PC23    PC24    PC25
## Standard deviation     0.42435 0.37750 0.35806 0.29741
## Proportion of Variance 0.00078 0.00062 0.00056 0.00038
## Cumulative Proportion  0.99844 0.99906 0.99962 1.00000
```

Looking at the summary of PCA analysis, we can see that PC1 alone explains ~90% of variability and if we take PC2 into consideration, PC1 together with PC2 will give us a total of ~94% of the variability.

Let's check the correlation of the PCs

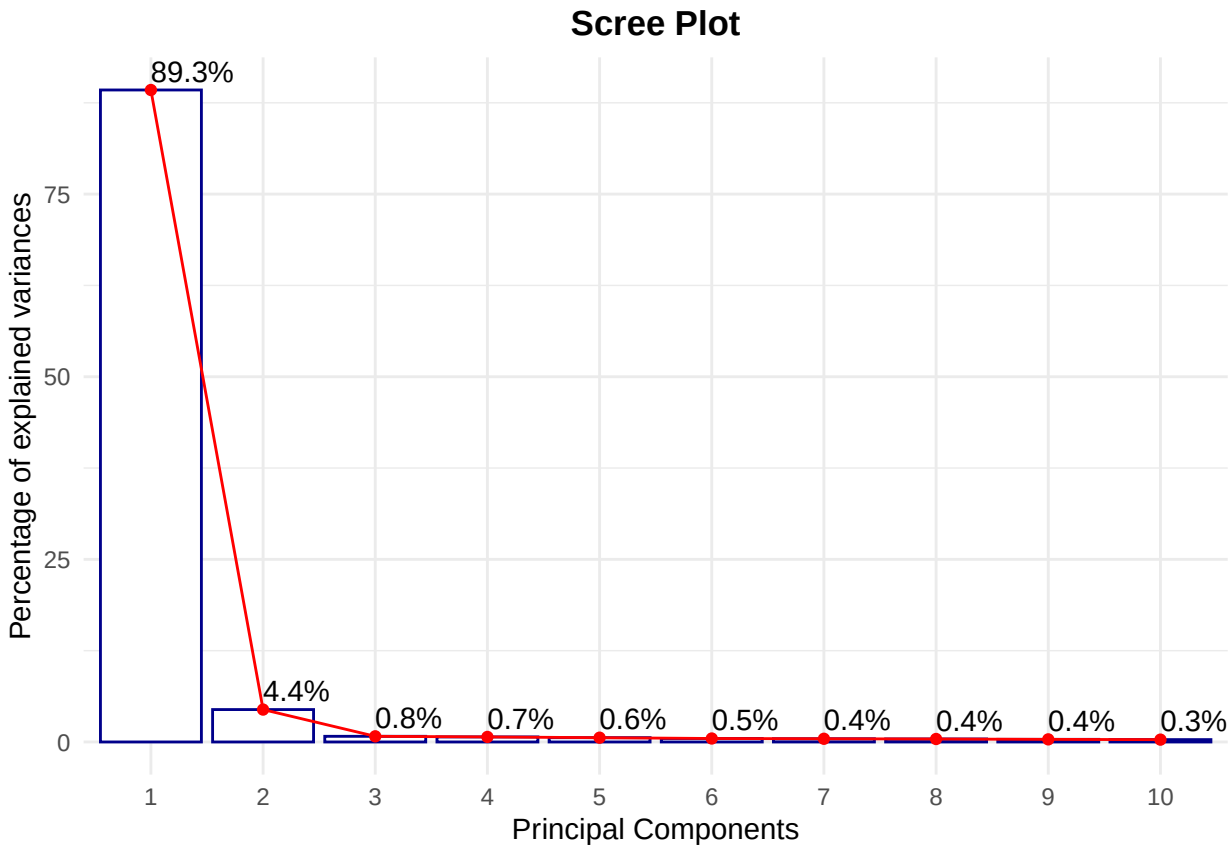
```
pairs.panels(pc$x, cex.cor = 1.5)
```



Here, we can see that there are no correlations between the PCs since PCs are orthogonal to each other, the correlation between them are always 0. Thus, this helps us to get rid of multicollinearity issues that occurs when there are variables that are highly correlated to each other as seen above.

Create a Scree plot for visualization of percentage of variability and PCs:

```
# scree plot
scree.p <- fviz_eig(pc, barfill = "white", barcolor = "darkblue",
                    linecolor = "red", addlabels = T,
                    xlab = "Principal Components") + theme_minimal() +
                    ggtitle('Scree Plot') +
                    theme(plot.title = element_text(hjust = 0.5, face="bold"))
# print
scree.p
```



Looking at the scree plot, we see that keeping 1 or 2 PCs is the best choice as the 1PC explains 89.7% of variance and 2PC explains 4.4%. In addition, variance starts to level off starting at the 3rd PC.

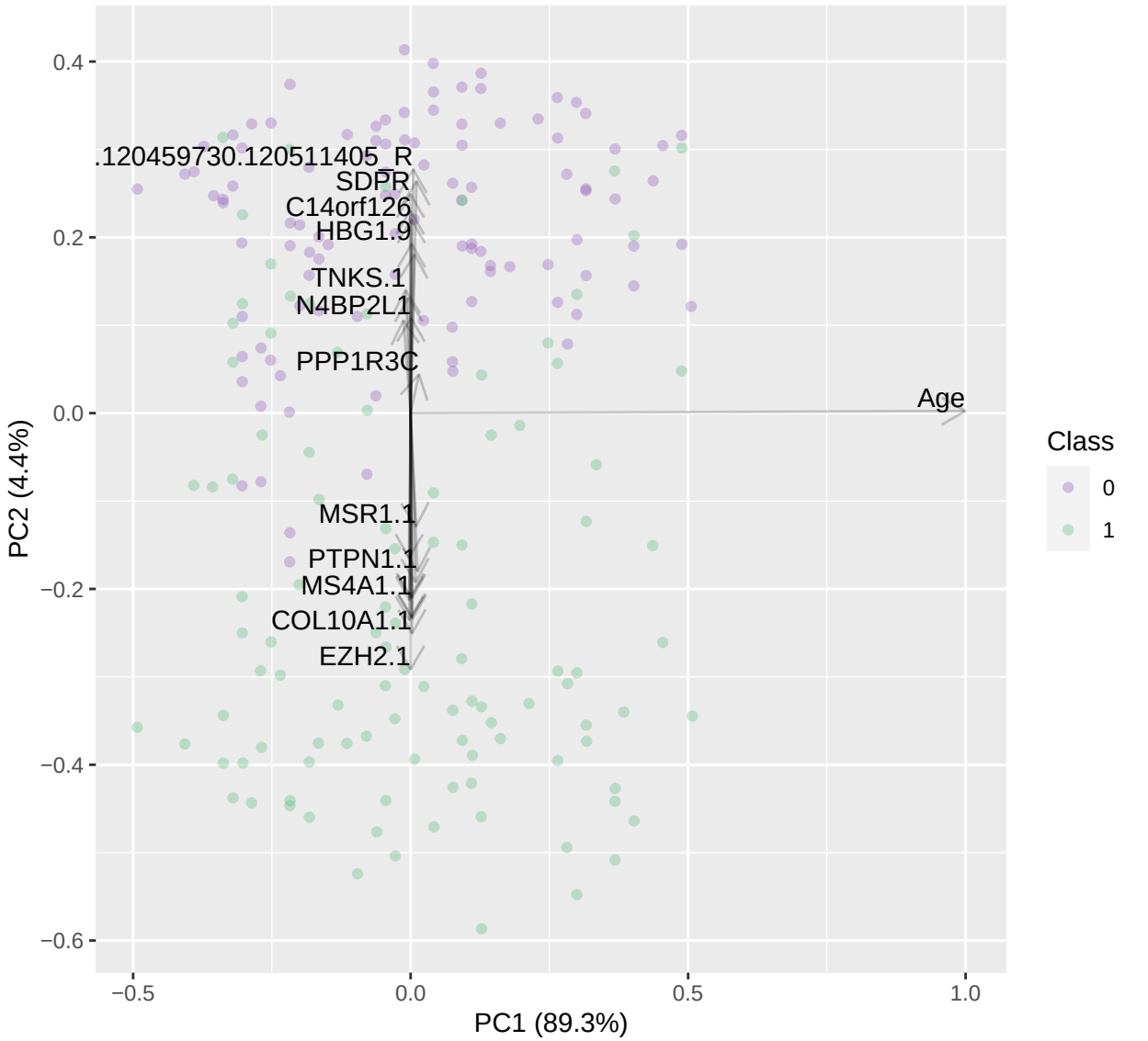
Create a Bi-Plot

```
# pieces from the PCA used in the biplot
pc.loadings <- pc$rotation  # loadings matrix
pc.scores <- predict(pc) # scores matrix
pc.importance <- summary(pc)$importance # explained variance
```

```
# The plot data includes class labels and the first two PC components.
plot.pc.scores <- cbind(Class = genes.fcbf.training$Class,
                        as.data.frame(pc.scores[, c("PC1", "PC2")]))

# We want to use min-max feature scaling on the scores so they are between zero
# and one, the same as the loadings.
normalize <- function(x) return ((x - min(x)) / (max(x) - min(x)))

plot.pc.scores[, "PC1"] <- scale(normalize(plot.pc.scores[, "PC1"]),
                                 center = TRUE, scale = FALSE)
plot.pc.scores[, "PC2"] <- scale(normalize(plot.pc.scores[, "PC2"]),
                                 center = TRUE, scale = FALSE)

# covert to data frame
plot.pc.loadings <- as.data.frame(pc.loadings)
```

```
# plot scores
bip1 <- ggplot() +
```

```
  geom_point(data = plot.pc.scores, mapping = aes(x = PC1, y = PC2,
                                                  colour = Class)) +
  scale_color_manual(values = c("#8F57BA55", "#57BA7D55"))

# plot loadings
bip2 <- bip1 +
  geom_segment(data = plot.pc.loadings, aes(x = 0, y = 0,
                       xend = PC1, yend = PC2),
                       arrow = arrow(length = unit(0.03, "npc")),
                       alpha = 0.2) +
                       geom_text(data = plot.pc.loadings,
                       mapping = aes(x = PC1, y = PC2,
                       label = rownames(plot.pc.loadings)),
                       hjust = 1, vjust = -0.2, colour = "black",
                       size = 4, check_overlap = TRUE)
# label axes
bip3 <- bip2 +
  xlab(paste("PC1 (", round(pc.importance["Proportion of Variance",
                                          "PC1"]*100, 1),"%)", sep = "")) +
  ylab(paste("PC2 (", round(pc.importance["Proportion of Variance",
                                          "PC2"]*100, 1),"%)", sep = ""))

# view plot
bip3
```



Looking at the biplot, on the x-axis we have PC1 which explains 89.7% of variability , and PC2 on the y-axis which explains 4.4% of variability. The arrows represents the features in the data set and arrows that are closer together are highly correlated –we can see that there are two distinct groups of features or arrows that are pointing two opposite directions, one up and one down. Interestingly, age has a very strong positive correlation with PC1 (0.99) and the rest are either slightly positively or negatively correlated to PC1. For PC2, COL10A1.1 and EZH2.1 are the two negatively correlated genes, and lincRNA.chr2.120459730.120511405_R and SDPR are two positively correlated genes.

Now, we are done with the PCA analysis and I've decided to use the 24 features/genes that I've got from FCBF + age = 25 total features as well as PC1 and PC2 from PCA to train and test my three classification models which includes ANN, SVM and Random Forest. This will give me a total of 6 models (3 FCBF models / 3 PC models).

Build and Train ANN Model#1 for FCBF:

```
# define training control --we're doing 10-fold cross-validation
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 10)
```

```
set.seed(1)

# hyperparameter tuning for size and decay
nnetGrid <-  expand.grid(size = seq(from = 1, to = 5, by = 1),
                         decay = seq(from = 0.0, to = 0.4, by = 0.1))

# train the fcbf model
fcbf.ann.model <- train(Class ~., data = genes.fcbf.training,
                        method = "nnet",
                        trControl = fitControl,  # 10-fold cv
                        verbose = F,
                        tuneGrid = nnetGrid,  # tune
                        trace = F)
```

Check ANN Model#1 Summary for FCBF

```
# model summary
#fcbf.ann.model
```

Upon training the model with 10 fold cross-validation, the final values for size = 1 and decay = 0.2 is used, since it gives the highest accuracy %.

Make Predictions

```
# make predictions with test set
fcbf.ann.p1 <- predict(fcbf.ann.model, newdata= genes.fcbf.testing[1:25])
```

ANN Model#1 for FCBF Evaluation:

```
# create a confusion matrix
fcbf.ann.pred <- confusionMatrix(data = fcbf.ann.p1,
                                 reference = genes.fcbf.testing$Class,
                                 positive = "1",
                                 mode = "everything")
fcbf.ann.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 42  4
##          1  2 40
##
##                Accuracy : 0.9318
##                  95% CI : (0.8575, 0.9746)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8636
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9091
##             Specificity : 0.9545
##          Pos Pred Value : 0.9524
##          Neg Pred Value : 0.9130
##               Precision : 0.9524
```

```
##                  Recall : 0.9091
##                      F1 : 0.9302
##              Prevalence : 0.5000
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.4773
##       Balanced Accuracy : 0.9318
##
##        'Positive' Class : 1
##
```

Results:

For ANN Model#1 of FCBF, we have an accuracy of 40 + 42 = 82/88= 93%, and an error rate of 2+4 = 6/88= 7%. In addition, the sensitivity rate is 0.90 and with a specificity of 0.95. The Kappa coefficient is 0.86, which indicates good agreement –this high value is likely to be due to the balanced target classes in the train and test sets. Moreover, ANN incorrectly classified 4 tumor samples as normal, hence 4 false negatives and has an F1 score of 0.93.

Build and Train SVM Model#1 for FCBF:

```
set.seed(1)
# train the fcbf model
fcbf.svm.model <- train(Class ~., data = genes.fcbf.training,
                        method = "svmRadialSigma",
                        prob.model = T,
                        trControl = fitControl,     # 10-fold cv
                        verbose = F,
                        tuneLength = 5,  # try 10 values of the cost function
                        trace = F)
```

Check SVM Model#1 Summary for FCBF

```
# model summary
#fcbf.svm.model
```

Upon training the model with 10 fold cross-validation, the final values for C = 2 and sigma = 0.03 is used after trying 5 values of cost function, since it gives the highest accuracy %.

Make Predictions

```
# make predictions with test set
fcbf.svm.p1 <- predict(fcbf.svm.model, newdata= genes.fcbf.testing[1:25])
```

SVM Model#1 for FCBF Evaluation:

```
# create a confusion matrix
fcbf.svm.pred <- confusionMatrix(data = fcbf.svm.p1,
                                 reference = genes.fcbf.testing$Class,
                                 positive = "1",
                                 mode = "everything")
fcbf.svm.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 41  3
##          1  3 41
##
```

```
##                Accuracy : 0.9318
##                  95% CI : (0.8575, 0.9746)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8636
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9318
##             Specificity : 0.9318
##          Pos Pred Value : 0.9318
##          Neg Pred Value : 0.9318
##               Precision : 0.9318
##                  Recall : 0.9318
##                      F1 : 0.9318
##              Prevalence : 0.5000
##          Detection Rate : 0.4659
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9318
##
##        'Positive' Class : 1
##
```

Results:

For SVM Model#1 of FCBF, we have an accuracy of 41 + 41 = 82/88= 93%, and an error rate of 3+3 = 6/88= 7%. In addition, the sensitivity rate is 0.93 and with a specificity of 0.93 as well. The Kappa coefficient is 0.86, which again indicates good agreement –this high value is likely to be due to the balanced target classes in the train and test sets. The accuracy rate and error rate is the same as ANN, however, SVM incorrectly predicted slightly less false negatives (3) than ANN (4) –meaning, classifying more tumor samples as normal ones. SVM also has an F1 score of 0.93 as well.

Build and Train Random Forest Model#1 for FCBF:

```
set.seed(1)
# train the fcbf model
fcbf.rf.model <- train(Class ~., data = genes.fcbf.training,
                       method = "rf",
                       trControl = fitControl,     # 10-fold cv
                       verbose = F,
                       tuneLength = 5,  # try 5 values of mtry
                       trace = F)
```

Check Random Forest Model#1 Summary for FCBF

```
# model summary
#fcbf.rf.model
```

Upon training the model with 10 fold cross-validation, the final value for mtry = 13 is used after trying 5 values of mtry since it gives the highest accuracy %.

Make Predictions

```
# make predictions with test set
fcbf.rf.p1 <- predict(fcbf.rf.model, newdata= genes.fcbf.testing[1:25])
```

Random Forest Model#1 for FCBF Evaluation:

```
# create a confusion matrix
fcbf.rf.pred <- confusionMatrix(data = fcbf.rf.p1,
                                reference = genes.fcbf.testing$Class,
                                positive = "1",
                                mode = "everything")
fcbf.rf.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40  4
##          1  4 40
##
##                Accuracy : 0.9091
##                  95% CI : (0.8287, 0.9599)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.301e-16
##
##                   Kappa : 0.8182
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9091
##             Specificity : 0.9091
##          Pos Pred Value : 0.9091
##          Neg Pred Value : 0.9091
##               Precision : 0.9091
##                  Recall : 0.9091
##                      F1 : 0.9091
##              Prevalence : 0.5000
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9091
##
##        'Positive' Class : 1
##
```

Results:

For Random Forest Model#1 of FCBF, we have an accuracy of 40 + 40 = 80/88= 91%, and an error rate of 4+4 = 8/88= 9%. In addition, the sensitivity rate is 0.91 and with a specificity of 0.91 as well. The Kappa coefficient is 0.82, which again indicates good agreement –this high value is likely to be due to the balanced target classes in the train and test sets. The accuracy and error rate are lower than both ANN and SVM, and random forest incorrectly predicted slightly more false negatives (4) than SVM (3) and the same FNs as ANN (4). Random forest also has the lowest F1 score of 0.91.

Now, let's formally compare the accuracy, precision and recall between ANN, SVM and Random Forest models for FCBF.

```
# FCBF ANN
fcbf.ann.tab <- table(Predicted = fcbf.ann.p1, Actual = genes.fcbf.testing$Class)
fcbf.ann.tab
```

```
##           Actual
## Predicted  0  1
```

```
##         0 42  4
##         1  2 40
```

```
# FCBF SVM
fcbf.svm.tab <- table(Predicted = fcbf.svm.p1, Actual = genes.fcbf.testing$Class)
fcbf.svm.tab
```

```
##          Actual
## Predicted  0  1
##         0 41  3
##         1  3 41
```

```
# FCBF Random Forest
fcbf.rf.tab <- table(Predicted = fcbf.rf.p1, Actual = genes.fcbf.testing$Class)
fcbf.rf.tab
```

```
##          Actual
## Predicted  0  1
##         0 40  4
##         1  4 40
```

Accuracy: FCBF ANN model achieved an accuracy rate of 93% with SVM achieving an accuracy rate of 93% as well (may consult accuracy calculations above) and Random forest achieved an accuracy rate of 91%, which is the model with lowest accuracy. In addition, three models also differed in the number of FNs –ANN has 4, SVM has 3 and Random forest has 4.

Precision: Equation for precision is = TP/TP+FP. Note that 0 = normal and 1 = tumor samples. ANN achieved a precision of 40/(40+2) = 0.95 while SVM obtained a precision of 41/(41+3)= 0.93, and Random forest 40/(40+4) = 0.91. Thus ANN achieved highest precision, and comes SVM and Random forest the lowest.

Recall: Equation for recall is = TP/TP+FN. ANN achieved a recall of 40/(40+4) = 0.91, while SVM achieved a recall of 41/(41+3)= 0.93, and Random forest achieved a recall of 40/(40+4) = 0.91. Hence, SVM has the highest recall, with ANN and Random forest having the same recall.

Create Train/Test sets for PCs

```
# TRAIN set with PCs as new features
pc.training <- predict(pc, genes.fcbf.training)

# add class label to train set
pc.training <- data.frame(pc.training, genes.fcbf.training[26])

# check structure of train set --208 observations with 25 PCs
#str(pc.training)
```

```
# TEST set with PCs as new features
pc.testing <- predict(pc, genes.fcbf.testing)

# add class label to test set
pc.testing <- data.frame(pc.testing, genes.fcbf.testing[26])

# check structure of test set --88 observations with 25 PCs
#str(pc.testing)
```

Build and Train ANN Model#1 for PC1 and PC2:

```
set.seed(1)
# train pc model
```

```
pc.ann.model <- train(Class ~ PC1 + PC2, data = pc.training,
                      method = "nnet",
                      trControl = fitControl,  # 10-fold cv
                      verbose = F,
                      tuneGrid = nnetGrid,   # tune
                      trace = F)
```

Check ANN Model#1 Summary for PC

```
# model summary
#pc.ann.model
```

Upon training the model with 10 fold cross-validation, the final values for size = 1 and decay = 0 is used, since it gives the highest accuracy %.

Make Predictions

```
# make predictions with test set
pc.ann.p1 <- predict(pc.ann.model, newdata= pc.testing[1:25])
```

ANN Model#1 for FCBF Evaluation:

```
# create a confusion matrix
pc.ann.pred <- confusionMatrix(data = pc.ann.p1,
                               reference = pc.testing$Class,
                               positive = "1",
                               mode = "everything")
pc.ann.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 41  6
##          1  3 38
##
##                Accuracy : 0.8977
##                  95% CI : (0.8147, 0.9522)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.076e-15
##
##                   Kappa : 0.7955
##
##  Mcnemar's Test P-Value : 0.505
##
##             Sensitivity : 0.8636
##             Specificity : 0.9318
##          Pos Pred Value : 0.9268
##          Neg Pred Value : 0.8723
##               Precision : 0.9268
##                  Recall : 0.8636
##                      F1 : 0.8941
##              Prevalence : 0.5000
##          Detection Rate : 0.4318
##    Detection Prevalence : 0.4659
##       Balanced Accuracy : 0.8977
##
```

```
##        'Positive' Class : 1
##
```

Results:

For ANN Model#1 of PC, we have an accuracy of 41 + 38 = 79/88= 90%, and an error rate of 3+6 = 9/88= 10%. In addition, the sensitivity rate is 0.86 and with a specificity of 0.93. The Kappa coefficient is 0.8, which is slightly lower than the FCBF models above but still indicates good agreement and is perhaps due to the balanced target classes in the train and test sets. In addition, the F1 score for ANN model is 0.89. Notice how just using 2 PCs can yield an accuracy rate that is close to the FCBF models which utilized 25 features, that's pretty amazing.

Build and Train SVM Model#1 for PC:

```
set.seed(1)
# train the pc model
pc.svm.model <- train(Class ~ PC1 + PC2, data = pc.training,
                      method = "svmRadialSigma",
                      prob.model = T,
                      trControl = fitControl,     # 10-fold cv
                      verbose = F,
                      tuneLength = 5,  # try 10 values of the cost function
                      trace = F)
```

Check SVM Model#1 Summary for FCBF

```
# model summary
#pc.svm.model
```

Upon training the model with 10 fold cross-validation, the final values for C = 0.25 and sigma = 0.11 is used after trying 5 values of cost function, since it gives the highest accuracy %.

Make Predictions

```
# make predictions with test set
pc.svm.p1 <- predict(pc.svm.model, newdata= pc.testing[1:25])
```

SVM Model#1 for PC Evaluation:

```
# create a confusion matrix
pc.svm.pred <- confusionMatrix(data = pc.svm.p1,
                               reference = pc.testing$Class,
                               positive = "1",
                               mode = "everything")
pc.svm.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40  4
##          1  4 40
##
##                Accuracy : 0.9091
##                  95% CI : (0.8287, 0.9599)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.301e-16
##
##                   Kappa : 0.8182
```

```
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9091
##             Specificity : 0.9091
##          Pos Pred Value : 0.9091
##          Neg Pred Value : 0.9091
##               Precision : 0.9091
##                  Recall : 0.9091
##                      F1 : 0.9091
##              Prevalence : 0.5000
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9091
##
##        'Positive' Class : 1
##
```

Results:

For SVM Model#1 of PC, we have an accuracy of 40 + 40 = 80/88= 91%, and an error rate of 4+4 = 8/88= 9%. In addition, the sensitivity rate is 0.91 and with a specificity of 0.91 as well. The Kappa coefficient is 0.82, which is slightly lower than the average of FCBF models above but still indicates good agreement and is perhaps due to the balanced target classes in the train and test sets. The F1 score is 0.91, which is higher than ANN. Lastly, the accuracy is higher and error rate is lower than ANN, and SVM incorrectly predicted slightly less false negatives (4) than ANN (6) –meaning, classifying less tumor samples as normal ones.

Build and Train Random Forest Model#1 for PC:

```
set.seed(1)
# train the pc model
pc.rf.model <- train(Class ~ PC1 + PC2, data = pc.training,
                         method = "rf",
                         trControl = fitControl,     # 10-fold cv
                         verbose = F,
                         tuneLength = 5,  # tune
                         trace = F)
```

```
## note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .
```

Check Random Forest Model#1 Summary for PC

```
# model summary
#pc.rf.model
```

Upon training the model with 10 fold cross-validation, the final value for mtry = 2 (since there's only two features) is used.

Make Predictions

```
# make predictions with test set
pc.rf.p1 <- predict(pc.rf.model, newdata= pc.testing[1:25])
```

Random Forest Model#1 for PC Evaluation:

```
# create a confusion matrix
pc.rf.pred <- confusionMatrix(data = pc.rf.p1,
                              reference = pc.testing$Class,
                              positive = "1",
```

```
                                mode = "everything")
pc.rf.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 36  4
##          1  8 40
##
##                Accuracy : 0.8636
##                  95% CI : (0.7739, 0.9275)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 7.837e-13
##
##                   Kappa : 0.7273
##
##  Mcnemar's Test P-Value : 0.3865
##
##             Sensitivity : 0.9091
##             Specificity : 0.8182
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 0.9000
##               Precision : 0.8333
##                  Recall : 0.9091
##                      F1 : 0.8696
##              Prevalence : 0.5000
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.5455
##       Balanced Accuracy : 0.8636
##
##        'Positive' Class : 1
##
```

Results:

For Random Forest Model#1 of PC, we have an accuracy of 36 + 40 = 76/88= 86%, and an error rate of 8+4 = 12/88= 14%. In addition, the sensitivity rate is 0.91 and with a specificity of 0.82. The Kappa coefficient is 0.73, which is slightly lower than the FCBF models above but still indicates good agreement and is perhaps due to the balanced target classes in the train and test sets. The F1 score is 0.87, which is the lowest of all models. Lastly, the accuracy rate for random forest is lower than both ANN and SVM and random forest incorrectly predicted same amount of false negatives (4) as SVM (4), but lower false negatives than ANN (6).

Moving on, let's formally compare the accuracy, precision and recall between ANN, SVM and Random Forest models for PC models.

```
# PC ANN
pc.ann.tab <- table(Predicted = pc.ann.p1, Actual = pc.testing$Class)
pc.ann.tab
```

```
##           Actual
## Predicted  0  1
##         0 41  6
##         1  3 38
```

```
# PC SVM
pc.svm.tab <- table(Predicted = pc.svm.p1, Actual = pc.testing$Class)
pc.svm.tab
```

```
##           Actual
## Predicted  0  1
##         0 40  4
##         1  4 40
```

```
# PC Random Forest
pc.rf.tab <- table(Predicted = pc.rf.p1, Actual = pc.testing$Class)
pc.rf.tab
```

```
##           Actual
## Predicted  0  1
##         0 36  4
##         1  8 40
```

Accuracy: PC ANN model achieved an accuracy rate of 90% with SVM achieving an accuracy rate of 91% (may consult accuracy calculations above), and Random forest, on the other hand, achieved an accuracy rate of 86%.

Precision: Equation for precision is = TP/TP+FP. Note that 0 = normal and 1 = tumor samples. ANN achieved a precision of 38/(38+3) = 0.93 while SVM obtained a precision of 40/(40+4)= 0.91, and Random forest 40/(40+8) = 0.82. Thus ANN, achieved highest precision, then comes SVM and Random forest achieving poorest precision.

Recall: Equation for recall is = TP/TP+FN. ANN achieved a recall of 38/(38+6) = 0.86, while SVM achieved a recall of 40/(40+4)= 0.91, and Random forest achieved a recall of 40/(40+4) = 0.91. Hence, SVM and Random forest has the highest recall and ANN the lowest.

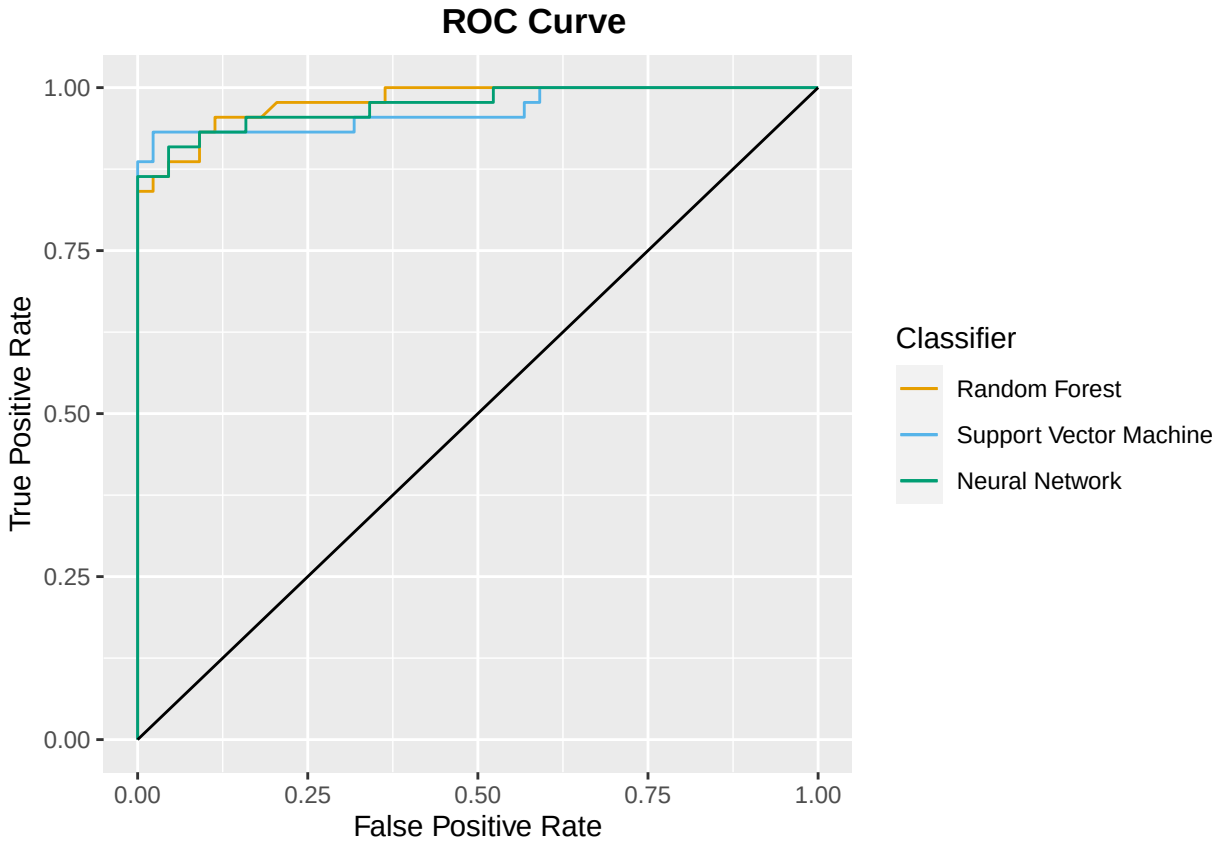
Next, I will be constructing a ROC Curve for the three FCBF models since they achieved better performance overall when compared to the PC models. The PC models are still very impressive, knowing that they achieved fairly good performance that is close to how the FCBF models performed with just 2 PCs!

Construct a ROC curve for FCBF models:

```
# RANDOM FOREST
fcbf.rf.pred.prob <- predict(fcbf.rf.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.rf.prob <- prediction(fcbf.rf.pred.prob[,2], genes.fcbf.testing$Class)
fcbf.rf.perform <- performance(fcbf.rf.prob, measure = "tpr", x.measure = "fpr")

# create a data frame for TP and FP rates
roc.df1 <- data.frame(FP = fcbf.rf.perform@x.values[[1]], TP = fcbf.rf.perform@y.values[[1]] )


#SVM
fcbf.svm.pred.prob <- predict(fcbf.svm.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.svm.prob <- prediction(fcbf.svm.pred.prob[,2], genes.fcbf.testing$Class)
fcbf.svm.perform <- performance(fcbf.svm.prob, measure = "tpr", x.measure = "fpr")

# create a data frame for TP and FP rates
roc.df2 <- data.frame(FP = fcbf.svm.perform@x.values[[1]], TP = fcbf.svm.perform@y.values[[1]] )


#ANN
fcbf.ann.pred.prob <- predict(fcbf.ann.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.ann.prob <- prediction(fcbf.ann.pred.prob[,2], genes.fcbf.testing$Class)
```

```
fcbf.ann.perform <- performance(fcbf.ann.prob, measure = "tpr", x.measure = "fpr")

# create a data frame for TP and FP rates
roc.df3 <- data.frame(FP = fcbf.ann.perform@x.values[[1]], TP = fcbf.ann.perform@y.values[[1]] )
```

```
# plot ROC curve for RANDOM FOREST, SVM, ANN

fcbf.roc <- ggplot() +
  geom_line(data = roc.df1, aes(x = FP, y = TP, color = 'Random Forest')) +
  geom_line(data = roc.df2, aes(x = FP, y = TP, color = 'Support Vector Machine')) +
  geom_line(data = roc.df3, aes(x = FP, y = TP, color = 'Neural Network')) +
  geom_segment(aes(x = 0, xend = 1, y = 0, yend = 1)) +
  ggtitle('ROC Curve') +  theme(plot.title = element_text(hjust = 0.6, face="bold")) +
  labs(x = 'False Positive Rate', y = 'True Positive Rate')

# roc plot labels
fcbf.roc +  scale_colour_manual(name = 'Classifier', values = c('Random Forest'='#E69F00',
                                                'Support Vector Machine'='#56B4E9',
                                                'Neural Network'='#009E73'))
```



The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings. Each point of the ROC curve (the threshold) corresponds to specific values of sensitivity and specificity. Thus, when examining a ROC curve, the model with a curve that hits the left-most upper corner is usually the model with the highest TP and lowest FP rate. According the to ROC curve that I've generated, which examined the TPR and FPR of all three models (random forest, SVM and ANN), although it is hard to distinguish which of the model is better (hits more to the left-most upper corner overall), I would suspect that ANN (green line) and Random forest (yellow line) are two models with the

highest TPR and lowest FPR, then comes SVM.

Determine AUC (Area Under the Curve):

```
# calculate AUC
auc.fcbf <- rbind(performance(fcbf.rf.prob, measure = "auc") @y.values[[1]],
                  performance(fcbf.svm.prob, measure = "auc") @y.values[[1]],
                  performance(fcbf.ann.prob, measure = "auc") @y.values[[1]])

# label row and col names for AUC
rownames(auc.fcbf) <- (c("Random Forest", "Support Vector Machine", "Neural Network"))
colnames(auc.fcbf) <- "Area Under the Curve (AUC)"

# examine AUC
auc.fcbf <- round(auc.fcbf, 3)
auc.fcbf
```

```
##                         Area Under the Curve (AUC)
## Random Forest                                0.979
## Support Vector Machine                       0.965
## Neural Network                               0.973
```

The area under the ROC curve (AUC) is a summary measure of performance that indicates whether on "average" a true positive is ranked higher than a false positive. Hence, when examining AUC, the higher the AUC the better the performance of the model is at determining positive and negative classes (AUC = 1 means the classifier is able to perfectly distinguish between all positive and negative class points correctly). According to the computed AUC shown above, Random forest has the highest area under the curve (AUC) then comes ANN, with SVM having the lowest AUC. Therefore, Random forest is the best performance model among them all and SVM is the poorest classifier for this application –cancer diagnosis.

Here, I will create an ensemble model with bagging to see if ANN can improve in performance. Random forest is already an extension of bagging so only SVM and ANN models were considered. ANN seems to be an interesting choice while it also achieved second highest AUC after Random forest, therefore, I will be creating an ANN ensemble model with bagging to see if the bagged model has any improvements.

Build and train ANN bagged model:

```
set.seed(1)

# train ANN bagged model
ann.bag.model <- train(Class ~., data = genes.fcbf.training,
                  B = 100,
                  size = 5,
                  method = "bag",
                  trControl = fitControl,    # 10-fold cv
                  bagControl = bagControl(fit = nnetBag$fit,     # bagctrl
                                          predict = nnetBag$pred,
                                          aggregate = nnetBag$aggregate))
```

Check Bagged Model

```
# bagged model summary
#ann.bag.model
```

```
set.seed(1)
# make predictions
ann.bag.p1 <- predict(ann.bag.model, newdata= genes.fcbf.testing[1:25])
```

ANN Bagged Model Evaluation:

```
# create a confusion matrix
ann.bag.pred <- confusionMatrix(data = ann.bag.p1,
                                reference = genes.fcbf.testing$Class,
                                positive = "1",
                                mode = "everything")
ann.bag.pred
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 43  4
##          1  1 40
##
##                Accuracy : 0.9432
##                  95% CI : (0.8724, 0.9813)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8864
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.9091
##             Specificity : 0.9773
##          Pos Pred Value : 0.9756
##          Neg Pred Value : 0.9149
##               Precision : 0.9756
##                  Recall : 0.9091
##                      F1 : 0.9412
##              Prevalence : 0.5000
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.4659
##       Balanced Accuracy : 0.9432
##
##        'Positive' Class : 1
##
```

Results:

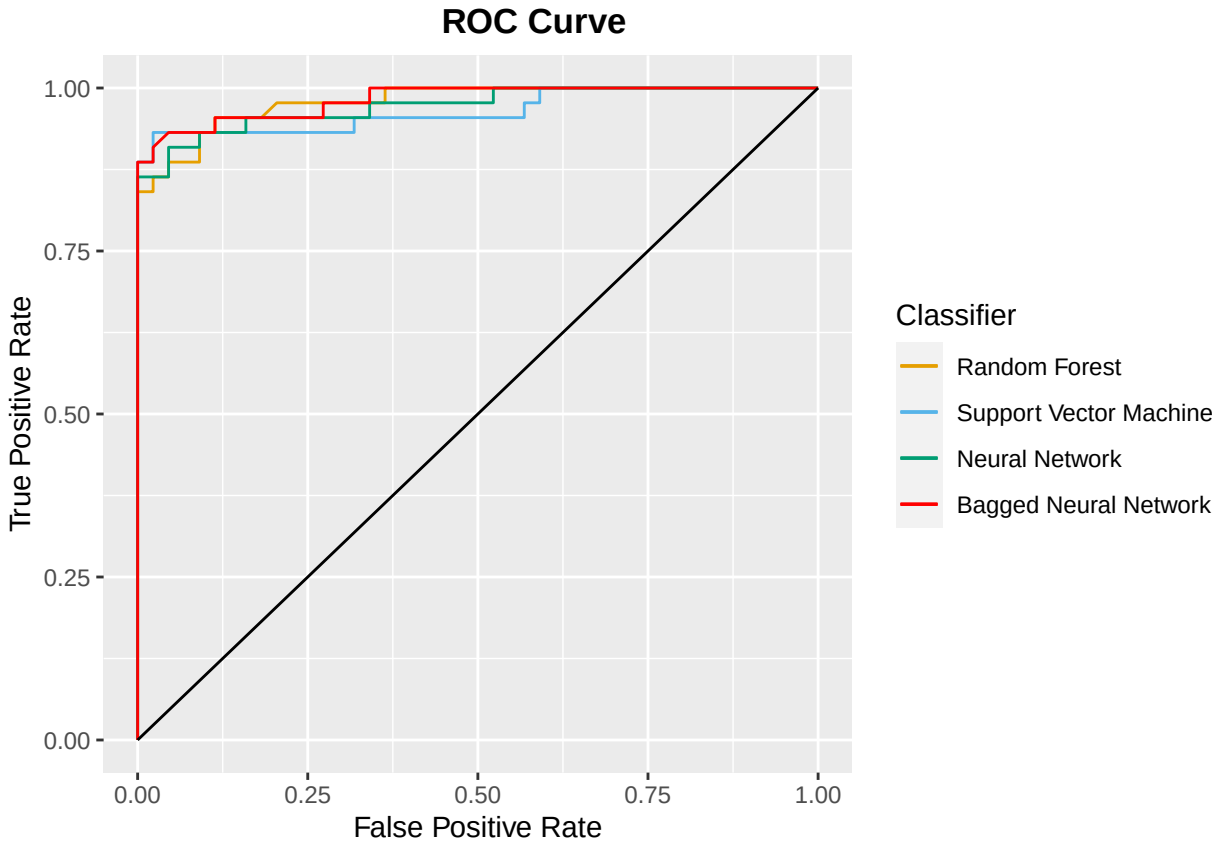
Looking at the confusion matrix created for the bagged model for ANN, the model achieved an accuracy of 41+41/88 = 93% and a sensitivity of 0.93 and specificity of 0.93. The accuracy rate is the same as the original ANN model without bagging, with a higher sensitivity and lower specificity. F1 score is also the same as the original ANN model as well. This is great, since we want sensitivity to have the highest rate –we want to be classifying all cancer samples correctly, and can sacrifice some specificity and accuracy for that.

Construct a ROC curve for all models:

```
set.seed(1)

# RANDOM FOREST
fcbf.rf.pred.prob <- predict(fcbf.rf.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.rf.prob <- prediction(fcbf.rf.pred.prob[,2], genes.fcbf.testing$Class)
fcbf.rf.perform <- performance(fcbf.rf.prob, measure = "tpr", x.measure = "fpr")
# create a data frame for TP and FP rates
roc.df1 <- data.frame(FP = fcbf.rf.perform@x.values[[1]], TP = fcbf.rf.perform@y.values[[1]] )
```

```
#SVM
fcbf.svm.pred.prob <- predict(fcbf.svm.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.svm.prob <- prediction(fcbf.svm.pred.prob[,2], genes.fcbf.testing$Class)
fcbf.svm.perform <- performance(fcbf.svm.prob, measure = "tpr", x.measure = "fpr")
# create a data frame for TP and FP rates
roc.df2 <- data.frame(FP = fcbf.svm.perform@x.values[[1]], TP = fcbf.svm.perform@y.values[[1]] )



#ANN
fcbf.ann.pred.prob <- predict(fcbf.ann.model, newdata= genes.fcbf.testing[1:25], type = "prob")
fcbf.ann.prob <- prediction(fcbf.ann.pred.prob[,2], genes.fcbf.testing$Class)
fcbf.ann.perform <- performance(fcbf.ann.prob, measure = "tpr", x.measure = "fpr")
# create a data frame for TP and FP rates
roc.df3 <- data.frame(FP = fcbf.ann.perform@x.values[[1]], TP = fcbf.ann.perform@y.values[[1]] )



# BAGGED ANN
ann.bag.pred.prob <- predict(ann.bag.model, newdata= genes.fcbf.testing[1:25], type = "prob")
ann.bag.prob <- prediction(ann.bag.pred.prob[,2], genes.fcbf.testing$Class)
ann.bag.perform <- performance(ann.bag.prob, measure = "tpr", x.measure = "fpr")
# create a data frame for TP and FP rates
roc.df4 <- data.frame(FP = ann.bag.perform@x.values[[1]], TP = ann.bag.perform@y.values[[1]] )
```

```
# plot ROC curve for RANDOM FOREST, SVM, ANN, BAGGED ANN

all.roc <- ggplot() +
  geom_line(data = roc.df1, aes(x = FP, y = TP, color = 'Random Forest')) +
  geom_line(data = roc.df2, aes(x = FP, y = TP, color = 'Support Vector Machine')) +
  geom_line(data = roc.df3, aes(x = FP, y = TP, color = 'Neural Network')) +
  geom_line(data = roc.df4, aes(x = FP, y = TP, color = 'Bagged Neural Network')) +
  geom_segment(aes(x = 0, xend = 1, y = 0, yend = 1)) +
  ggtitle('ROC Curve') +  theme(plot.title = element_text(hjust = 0.6, face="bold")) +
  labs(x = 'False Positive Rate', y = 'True Positive Rate')

# roc plot labels
all.roc +  scale_colour_manual(name = 'Classifier', values = c('Random Forest'='#E69F00',
                                                'Support Vector Machine'='#56B4E9',
                                                'Neural Network'='#009E73',
                                                'Bagged Neural Network' = 'red'))
```



Based on the to ROC curve above, which examined the TPR and FPR of all four models (random forest, SVM, ANN and bagged ANN), I would suspect that Random forest (yellow line) and bagged ANN are two models with the highest TPR and lowest FPR, then comes ANN and SVM.

Determine AUC (Area Under the Curve):

```
# calculate AUC
auc.all <- rbind(performance(fcbf.rf.prob, measure = "auc") @y.values[[1]],
                 performance(fcbf.svm.prob, measure = "auc") @y.values[[1]],
                 performance(fcbf.ann.prob, measure = "auc") @y.values[[1]],
                 performance(ann.bag.prob, measure = "auc") @y.values[[1]])


# label row and col names for AUC
rownames(auc.all) <- (c("Random Forest", "Support Vector Machine", "Neural Network",
                        "Bagged Neural Network"))
colnames(auc.all) <- "Area Under the Curve (AUC)"

# examine AUC
auc.all <- round(auc.all, 3)
auc.all
```

```
##                         Area Under the Curve (AUC)
## Random Forest                                0.979
## Support Vector Machine                       0.965
## Neural Network                               0.973
## Bagged Neural Network                        0.982
```

In conclusion, since we are classifying normal versus tumor samples, I would prefer a model with higher

sensitivity (less false negatives) and am willing to sacrifice some accuracy and specificity for that. For this reason, I would also prefer a model with a higher AUC. Therefore, out of all the models that I've constructed the model that I think is best model for this application is the bagged ANN model with a sensitivity of 0.93, specificity of 0.93, accuracy of 93% and lastly, AUC of 0.98, which is the highest among all models.

As a bonus, let's see if we can achieve an even higher performance with a simple ensemble model with stacking, which stacks all three models together to generate a final classification on all samples in the test set.

Ensemble model of all three FCBF models:

```
predictCancerSample <- function(newcase){
  # ANN
  fcbf.ann.p2<- predict(fcbf.ann.model, newcase)

  # SVM
  fcbf.svm.p2 <- predict(fcbf.svm.model, newcase)

  # Random Forest
  fcbf.rf.p2 <- predict(fcbf.rf.model, newcase)


  # create a majority voting system
  return (as.factor(ifelse(fcbf.ann.p2=='0' & fcbf.svm.p2=='0','0',
            ifelse(fcbf.ann.p2=='0' & fcbf.rf.p2=='0','0',
            ifelse(fcbf.svm.p2=='0' & fcbf.rf.p2=='0','0','1')))))
}
```

Make predictions on test set with the preditCancerSample function and save to a variable:

```
# make prediction with ensemble model function
ensemble.pred <- predictCancerSample(genes.fcbf.testing[1:25])
```

Evaluation of Ensemble Model:

```
# construct confusion matrix
ensemble.tab <- confusionMatrix(data = ensemble.pred,
                                reference = genes.fcbf.testing$Class,
                                positive = "1",
                                mode = "everything")
ensemble.tab
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 42  3
##          1  2 41
##
##                Accuracy : 0.9432
##                  95% CI : (0.8724, 0.9813)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8864
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9318
```

```
##             Specificity : 0.9545
##          Pos Pred Value : 0.9535
##          Neg Pred Value : 0.9333
##               Precision : 0.9535
##                  Recall : 0.9318
##                      F1 : 0.9425
##              Prevalence : 0.5000
##          Detection Rate : 0.4659
##    Detection Prevalence : 0.4886
##       Balanced Accuracy : 0.9432
##
##        'Positive' Class : 1
##
```

Results:

Looking at the results, the ensemble model that I've constructed achieved an accuracy rate of 94%. This is higher than all models. In terms of sensitivity, the ensemble model achieved a sensitivity rate of 0.93, which is the same as SVM and bagged ANN, but higher than both Random forest and ANN. Lastly, for specificity, the ensemble model achieved a specificity of 0.95, which is the same as ANN, and higher than both SVM and Random forest and bagged ANN and has an F1 score of 0.94 –highest of all models.