Discovery of Acute Myeloid Leukemia Biomarkers using Ensemble Machine Learning

Dependencies

```
library(knitr)
opts_chunk$set(eval=FALSE, tidy.opts=list(width.cutoff=10, height.cutoff=10),tidy=TRUE)
```

```
# ==============
# Dependencies
# ==============
library(plyr)
library(mlr)
library(magrittr)
library(ggplot2)
library(EnsDb.Hsapiens.v75)
library(glmnet)
library(ROSE)
library(knitr)
library(stringr)
library(dplyr)
library(tibble)
library(tidyr)
library(limma)
library(edgeR)
library(MLSeq)
library(DESeq2)
library(xlsx)
library(VennDiagram)
```

```
#================================
# functions for machine learning
#================================

# Lasso
glm.binom <- function(x,y,df,ref="No", train.names=NULL, test.names=NULL,
                      standardize=FALSE, splitIntoTrain=FALSE){
  # credit: Jenny Smith
  library(glmnet)
  #df is the matrix with the response and  gene expression. Patients as rownames.
  #x is the character vector of column names for genes
  #y is the character vector of column names for the classifier
  #train is a chacter vector of patient IDs
  #test is a chacter vector of Patient IDs.

  response <- y
  predictors <- x

  #check this that referece should be the first level in glmnet package
  #Set-up the x and y matrices
```

```r
y <- factor(df[,y])
y <- relevel(y,ref = ref)  %>% set_names(rownames(df))
x <- as.matrix(df[,x]) #NOTE: for categorical predictors data, should use model.matrix


if (any(c(is.na(y), is.na(x)))) {
  print("There Are Missing Values.")
  return(list(x=x,y=y))
}

#Check the reference level of the response.
contrast <- contrasts(y)

if(splitIntoTrain){
  #Use validation set approach. split observations into approx. equal groups.
  set.seed(1)
  train <- sample(c(TRUE,FALSE), nrow(x), replace = TRUE)
  test <- (!train)

  train.names <- rownames(df)[train]
  test.names <- rownames(df)[test]
}


#grid of lambda values to test.
grid <- 10^ seq(10,-2, length=100)

#training model.
fit <- glmnet(x[train.names,], y[train.names],
              family = "binomial",
              alpha=1,
              standardize = standardize,
              lambda = grid,
              intercept = FALSE)

#use cross-validation on the training model.CV only for lambda
set.seed(2019)
cv.fit <- cv.glmnet(x[train.names,], y[train.names],
                family = "binomial",
                type.logistic="modified.Newton",
                standardize = standardize,
                lambda = grid,
                alpha=1,
                nfolds = length(train.names), #LOOCV
                type.measure = "class",
                intercept = FALSE)

#Select lambda min.
lambda.min <- cv.fit$lambda.min

#predict the classes
pred.class <- predict(fit, newx = x[test.names,], type="class", s=lambda.min)
```

```
  #find the test error
  tab <- table(pred.class,y[test.names])
  testError <- mean(pred.class != y[test.names]) #how many predicted classes were incorrect

  #Fit the full dataset.
  final <- glmnet(x, y,family = "binomial",
                  standardize = standardize,
                  lambda = grid,
                  alpha = 1,
                  intercept = FALSE)

  #Extract the coefficients
  coef <- predict(final, type="coefficients", s=lambda.min)
  idx <- which(coef != 0)
  nonZero <- coef[idx,]

  #Results
  list <- list(train.names, test.names, contrast, fit, cv.fit,tab,testError, final, nonZero)
  names(list) <- c("training.set", "testing.set","contrast", "train.fit",
                   "cv.fit", "confusionMatrix","test.error", "final.model", "nonzero.coef")
  return(list)

}

# SVM
runSVM <- function(seed,kerneltype="linear",trainset,trainclasses,
                   testset,testclasses, weightfilt=FALSE){
  # credit : Sean Maden
  # run SVM optimization
  # Arguments
  #   * seed : set seed (int) for randomization
  #   * kerneltype : (str) type of kernel for SVM, either 'linear' or 'gaussian'
  #   * trainset : training dataset (excluding sample classes)
  #   * trainclasses : classes for training sampels (vector) with 1:1 correspondence
  #       with trainset rows
  #   * testset : test data (data frame or matrix), excluding classes
  #   * testclasses : classes for test samples (vector), with 1:1 row:pos correspondence
  #   * weightfilt : (FALSE or numeric) top percentage weights to use in model
  #       (if FALSE, then all weights used)
  # Returns
  #   * rl (list) : list containing model fitted, predictions, and performacne metrics
  require(e1071); require(ROCR)
  rl <- list(); str.options <- ""
  set.seed(seed)
  ndtr <- trainset
  ndte <- testset
  ndtr.classes <- trainclasses
  ndte.classes <- testclasses

  # train svm model
  svm_model <- svm(as.factor(ndtr.classes)~.,
                   data=ndtr,
                   method="C-classification",
```

```
                  kernel=kerneltype)
weightsvect <- ndtr.weights <- t(svm_model$coefs) %*% svm_model$SV
if(weightfilt){
  str.options <- c(str.options,paste0("weight filt = ",weightfilt))
  # order training data on relative weights
  ndtr.weightsort <- ndtr[,rev(order(abs(ndtr.weights)))]
  # select only top proportion weights
  nweight.col = round(ncol(ndtr.weightsort)*weightfilt,0)
  ndtr.weightfilt <- ndtr.weightsort[,c(1:nweight.col)]
  str.options <- c(str.options,paste("cols_retained:",colnames(ndtr.weightfilt),collapse=";"))
  # redefine training set, rerun SVM optimization
  ndtr <- ndtr.weightfilt
  svm_model <- svm(as.factor(ndtr.classes)~.,
                   data=ndtr,
                   method="C-classification",
                   kernel=kerneltype)
} else{
  str.options <- c(str.options,"no weight filt")
}
pred_train <- predict(svm_model, ndtr, decision.values = TRUE)
pred_test <- predict(svm_model, ndte, decision.values = TRUE)
# get performance metrics
pred <- prediction(as.numeric(attr(pred_test,"decision.values")),ndte.classes)
perf <- performance(pred,"tpr","fpr")
ppred <- pred_test[pred_test==1];
tppred <- ndte.classes[pred_test==1]
ppred <- as.numeric(as.character(ppred))
testprec <- length(ppred[ppred==tppred])/length(ppred) # test precision
rposi <- ndte.classes==1
rtpred <- ndte.classes[rposi];
rppred <- pred_test[rposi]
rppred <- as.numeric(as.character(rppred))
testrec <- length(rppred[rppred==1])/length(rppred) # test recall

# return model, pred's, and performance metrics
rl <- list(str.options,
           svm_model,
           weightsvect,
           pred_train,
           pred_test,
           perf,
           tppred,
           testprec,
           testrec)
names(rl) <- c("options_string",
               "svm_model",
               "weightsvect",
               "predictions_train",
               "predictions_test",
               "performance_test",
               "TPR_test",
               "precision_test",
               "recall_test"
```

```
  )
  return(rl)

}
```

```
#=================================================
# utilities for data summaries and visualization
#=================================================

# Survival by sample groups, plot summaries
{
  # credit: Sean Maden
  ggdat <- as.data.frame(matrix(ncol=2,nrow=0))
ggdat <- rbind(ggdat,data.frame(group='young.overallsurv',survival.time=aml.cd[class.age=='young',]$Ove
ggdat <- rbind(ggdat,data.frame(group='young.efsurv',survival.time=aml.cd[class.age=='young',]$Event.Fr
ggdat <- rbind(ggdat,data.frame(group='old.overallsurv',survival.time=aml.cd[class.age=='old',]$Overall
ggdat <- rbind(ggdat,data.frame(group='old.efsurv',survival.time=aml.cd[class.age=='old',]$Event.Free.Su

ggplot(ggdat, aes(x=ggdat$survival.time, col=ggdat$group))+geom_density()+
  theme(panel.background = element_rect(fill = 'white',colour = 'black'),
        rect = element_rect(fill = 'white',colour = "white"),
        panel.grid.major = element_line(colour = 'grey75', size=0.2),
        panel.grid.minor = element_line(colour = 'white'),
        legend.position = 'right',
        legend.background = element_rect(fill = "white",
                                         colour ="white"),
        legend.key = element_rect(fill = "white"),
        plot.title = element_text(hjust = 0.5))+
  labs(color="Group Survival") +
  ggtitle("Survival Time by Age Classifier")
}

# Categorize DEGs
catExpnData <- function(filenames,regex, cols, header=FALSE,removeFirstLine=FALSE, sep="\t"){
  #credit: Jenny Smith
  # Purpose: Concatenate the expression data-sets downloaded from TCGA/TARGET from GDC or any patient l
  #eg. each individual patient has a single expression-file

  library(magrittr)
  options(stringsAsFactors = FALSE)
  #filenames is a character vector of all filenames.
  #regex is a string with the pattern to extract the patient ID , eg "^.+(Kasumi|MV4)", from filenames
  #cols is the character vector or numeric vector of the columns to select and concatenate.

  extract_cols <-function(filename,cols,rmFirstLine=FALSE){

    if(all(rmFirstLine & header)){
      aFile <- readLines(filename)[-1] #remove first line with extra info.
      aFile <- str_split_fixed(aFile, pattern = "\t",n = length(cols)) %>% #split into a matrix
        set_colnames(.[1,] ) %>%  #set colnames from the first line
        .[-1, ] #remove the header row from matrix
    }else{
      aFile <- read.delim(filename, sep=sep, header=header, as.is=TRUE)
    }
```

```
    output <- list()
    for ( k in 1:length(cols)){
      colname <- cols[k]
      col <- aFile[,colname]
      output[[colname]] <- col
    }
    return(output)
  }

  combineColumns <- function(extract_cols.res,colname){
    sapply(extract_cols.res, '[[', colname)
  }


  IDs <- gsub(regex, "\\1", filenames)

  columns <- lapply(filenames,extract_cols,cols=cols, rmFirstLine=removeFirstLine) %>%
    set_names(IDs)

  catedMatrices <- lapply(cols, combineColumns, extract_cols.res=columns)  %>%
    set_names(cols)


  return(catedMatrices)
}

# Gene summary scatter plots
{
  # credit: Sean Maden
  jpeg("target-aml_gene-meanvar-diff_test-train.jpg",10,15,units="in",res=400)
par(mfrow=c(2,1))
col.deg <- rgb(0.2,0.5,0.2,0.3)
col.all <- rgb(0.7,0.1,0.2,0.3)
test.na <- is.na(test.degdiff) | is.na(test.degvar)
plot(test.degdiff[!test.na], test.degvar[!test.na], pch=16, col=col.deg,
     main = "TARGET AML Test Subset",xlab="Gene mean diff (Low - Not-low)", ylab="Gene var diff (Low - N
test.na <- is.na(test.alldiff) | is.na(test.allvar)
points(test.alldiff[!test.na], test.allvar[!test.na], pch=1, col=col.all)
abline(h=0,col="blue");abline(v=0,col="blue")
legend("topright",legend=c("All Genes","DEGs"),pch=c(1,16),col=c(col.all, col.deg))
train.na <- is.na(train.degdiff) | is.na(train.degvar)
plot(train.degdiff[!train.na], train.degvar[!train.na], pch=16, col=col.deg,
     main = "TARGET AML Train Subset",xlab="Gene mean diff (Low - Not-low)", ylab="Gene var diff (Low -
train.na <- is.na(train.alldiff) | is.na(train.allvar)
points(train.alldiff[!train.na], train.allvar[!train.na], pch=1, col=col.all)
abline(h=0,col="blue");abline(v=0,col="blue")
dev.off()
}

# Volcano plot
volcano_plot <- function(fit, cut.off=4, label.offset=0.5){
  # credit : Jenny Smith
  df <- data.frame(logFC=fit$coefficients[,1],
```

```
                    pValue=fit$p.value[,1],
                    FDR=p.adjust(fit$p.value[,1], method="BH"),
                    MeanExpression=fit$Amean) %>%
      rownames_to_column("Gene") %>%
      mutate(Neg.Log10.P= -log10(pValue),
             DEGs.Groups=case_when(
                  logFC > 1.0 & pValue < 0.05 ~ "FC Greater than 2",
                  logFC < -1.0 & pValue < 0.05 ~ "FC Less than 2",
                  TRUE ~ "Not Significant FC"))


  #Select differentially expressed genes to highlight in the plot.
  ToHighlight <- df[abs(df$logFC) > cut.off & df$FDR < 0.05, "Gene"]
  idx <- which(abs(df$logFC) > cut.off & df$FDR < 0.05)

  vplot <- ggplot(df, aes(x=logFC, y=Neg.Log10.P)) +
    geom_point(data = filter(df, DEGs.Groups == "Not Significant FC"),
               mapping = aes(x=logFC, y=Neg.Log10.P, color=DEGs.Groups), alpha=0.65)  +

    geom_point(data= filter(df, grepl("2", DEGs.Groups)),
               mapping = aes(x=logFC, y=Neg.Log10.P, color=DEGs.Groups)) +

    geom_vline(xintercept=c(-1,1)) +
    geom_hline(yintercept = -log10(0.05)) +

    scale_color_manual(values=c("FC Greater than 2"="red",
                                "FC Less than 2"="blue",
                                "Not Significant FC"="lightgrey")) +

    theme(plot.title = element_text(hjust = 0.5, size = 20),
          panel.background = element_rect(fill="white"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_rect(color = "black", fill=NA),
          axis.text = element_text(color = "black"),
          axis.text.x = element_text(angle = 0,hjust=0.5,vjust = 0.5, size = 26),
          axis.text.y = element_text(size = 25),
          axis.title = element_text(size = 30),
          plot.margin = margin(2,2,2,2, unit = "mm")) +

    geom_text(aes(x=logFC+label.offset, y=Neg.Log10.P, label=ToHighlight),size=3.5,
              data=df[idx, ])


  return(vplot)

}
```

Data Set

This investigation uses pediatric AML cases from the TARGET cohort.

Methods

Dataset

We initially focused on risk group as our primary classifier of interest. Considering sample size, demographics, and other clinical variables, we combined non-low risk groups into a single category and compared these with the low risk group. This resulted in relative balance between the categories across important clinical factors.

Gene Expression Data

We focused on RNA-seq data from Illumina HiSeq. Raw gene counts were converted to TMM log expression. ## Preprocessing Expression Data We then pre-filtered genes by identifying those showing greatest contrasts between our classifier groups of interest (t-test, p-adj < 0.05).

Ensemble Machine learning.

We applied the following methods from R and Python libraries as indicated:.

Results

Summary Statistics

Model Fitting

Feature Selection

Consensus Feature Validation

Conclusions