Disease Prediction with Gene Expression Data

Eugine Kang

Sunday, August 30, 2015

Summary  
 Introduction  
 Steps that will follow

# 1. Data Cleaning

dim(data)

## [1] 184 12066

The data is high-dimensional, meaning the number of variables far exceed the number of samples. Classicial approaches such as least squares linear regression are not appropriate in this setting, and other approaches have the danger of overfitting.

We can reduce the dimension by feature selection and dimension reduction techniques, but let's first see if we can discard variables with little information.

var0 <- nearZeroVar(data)  
length(var0)

## [1] 3245

3245 variables have zero or near zero variance. We decide to discard these variables, because they won't be useful when classifying disease.

dataZero <- data[,-var0]  
noNA <- sapply(dataZero, function(x) sum(is.na(x)))  
table(noNA)

## noNA  
## 0 1   
## 8815 6

6 columns have missing values. We only have 184 samples, so removing the samples with missing values will be too valuable of a waste. Imputation fills in the missing values, and will allow us to use every sample to train our model

dataImpute <- knnImputation(dataZero, k=10)

Multicollinearity, the concept that the variables in a regression might be correlated with each other. In the high-dimensional setting, the multicollinearity problem is extreme. Any variable in the model can be written as a linear combination of all of the other variables in the model. Removing variables with a correlation value above 0.7 is an option we can take.

dataScale <- scale(dataImpute, center=TRUE, scale=TRUE)  
corMat <- cor(dataScale)  
highlyCor <- findCorrelation(corMat, 0.70)

length(highlyCor)

## [1] 873

873 variables are removed due to high correlation with other variables. Variables with low information or high correlation are removed. Let's try to select important variables to our models by feature selection.

dataFilter <- as.data.frame(dataScale[,-highlyCor])  
dataFilter$Y <- ifelse(label == 1, 1,0)  
dataFilter$Y <- as.factor(dataFilter$Y)

# 2. Feature Selection

We will be using Random Forest to find out the importance of each variables for our classification model. 10 fold cross validation is the training scheme for our feature selection method.

control <- trainControl(method="repeatedcv", number=10, repeats=1)

Random Forest is a popular method for feature selection for high-dimensional data. We set the seed for every random iteration so that we can reproduce the results.

set.seed(1)  
rfModel <- train(Y~., data=datFilter,   
 method="rf",   
 trControl=control,  
 importance=TRUE,  
 do.trace=FALSE  
)  
# Organize Feature Importance Data  
fullImportance <- varImp(rfModel, scale=FALSE)  
fullVarImp <- fullImportance$importance  
fullVarImp$varName <- row.names(fullVarImp)  
colnames(fullVarImp) <- c("Imp","Imp2","varName")  
fullVarImp <- fullVarImp[order(fullVarImp$Imp, decreasing=T),]

From training a Random Forest model, we get importance levels for each variables that help the classification model. The distribution of importance level is shown by the histogram, and the top variables have an importance higher than 1.

hist(fullVarImp$Imp)

Our goal is to select the fewest amount of variables, so that we can test more type of models and remain the performance. The number of features selected by importance is shown through the graph.

lvl <- seq(1,2,0.05)  
noFeatvsImp <- data.frame(Imp=lvl,   
 noFeat=sapply(lvl, function(x) sum(fullVarImp$Imp>x))  
 )  
ggplot(data=noFeatvsImp, aes(x=Imp,y=noFeat)) +   
 geom\_line(colour="darkmagenta", size=3) +  
 ggtitle("No.Features by Importance Threshold") +  
 labs(x="Importance",y="Number of Features") +   
 theme(plot.title = element\_text(family = "Trebuchet MS", color="#666666", face="bold", size=25, hjust=0)) +  
 theme(axis.title = element\_text(family = "Trebuchet MS", color="#666666", face="bold", size=22))

## Warning in grid.Call(L\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## font family not found in Windows font database

## Warning in grid.Call(L\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## font family not found in Windows font database

## Warning in grid.Call(L\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## font family not found in Windows font database

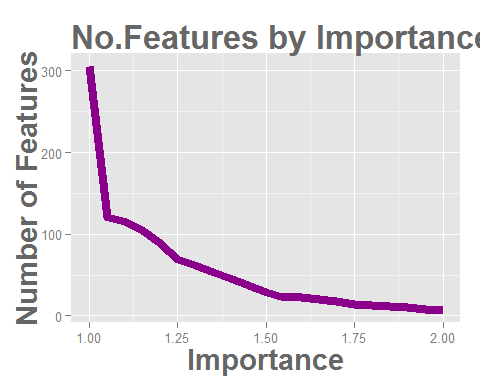
## Warning in grid.Call(L\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## font family not found in Windows font database

## Warning in grid.Call(L\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## font family not found in Windows font database

## Warning in grid.Call.graphics(L\_text, as.graphicsAnnot(x$label), x$x, x$y,  
## : font family not found in Windows font database

## Warning in grid.Call.graphics(L\_text, as.graphicsAnnot(x$label), x$x, x$y,  
## : font family not found in Windows font database

## Warning in grid.Call.graphics(L\_text, as.graphicsAnnot(x$label), x$x, x$y,  
## : font family not found in Windows font database



For a test run, we will select variables with importance higher than 1.1.

# 3. Compare Models

We will be testing the performance of 9 models with the selected features with an importance greater than 1.1.

featSelect <- function(thres){  
 return(fullVarImp[fullVarImp$Imp>thres,"varName"])  
 }  
featSet <- featSelect(1.1)  
# Independent Variables for Model  
X <- dataFilter[,featSet]  
train <- 1:nrow(X)  
# Dependent Variables for Model  
Y <- dataFilter$Y  
levels(Y)[1] <- make.names(levels(Y))[1]  
levels(Y)[2] <- make.names(levels(Y))[2]

The models will go through 10-fold cross-validation, and the performance of the models are evaluated through the ROC value. When considering the real life impact of disease detection, we don't want to miss anyone from detection especially when the disease is sever. Sensitivity measures the proportion of positives that are correctly identified, and specificity measures the proportion of negatives that are correctly identified. Metric that evaluates this performance is ROC, receiver operating characteristic. Our goal is to find a model with a ROC higher than 0.9.

folds=10  
repeats=1  
# ROC will be the metric to evaluate the model "summaryFunction=twoClassSummary"  
myControl <- trainControl(method='cv', number=folds, repeats=repeats,   
 returnResamp='none', classProbs=TRUE,  
 returnData=FALSE, savePredictions=TRUE,   
 verboseIter=TRUE, allowParallel=TRUE,  
 summaryFunction=twoClassSummary,  
 index=createMultiFolds(try$Y, k=folds, times=repeats))  
PP <- c('center', 'scale')

The 9 models that will compared are listed below with the code.

set.seed(1)  
# Stochastic Gradient Boosting  
mdlGBM <- train(X[train,], Y[train],   
 method='gbm',   
 trControl=myControl,   
 tuneLength=10,   
 preProcess=PP)  
# Boosted Tree  
mdlBLACKBOOST <- train(X[train,], Y[train],   
 method='blackboost',   
 trControl=myControl,   
 tuneLength=10,   
 preProcess=PP)  
# Random Forest  
mdlPARRF <- train(X[train,], Y[train],   
 method='parRF',   
 trControl=myControl,   
 tuneLength=10,   
 preProcess=PP)  
# Multi-Layer Perceptron  
mdlMLP <- train(X[train,], Y[train],   
 method='mlpWeightDecay',   
 trControl=myControl,   
 trace=FALSE,  
 tuneLength=3,  
 preProcess=PP)  
# k-Nearest Neighbors  
mdlKNN <- train(X[train,], Y[train],   
 method='knn',   
 trControl=myControl,  
 tuneLength=10,  
 preProcess=PP)  
# Multivariate Adaptive Regression Spline  
mdlEARTH <- train(X[train,], Y[train],   
 method='earth',   
 trControl=myControl,   
 tuneLength=10,  
 preProcess=PP)  
# Generalized Linear Model  
mdlGLM <- train(X[train,], Y[train],   
 method='glm',   
 trControl=myControl,   
 preProcess=PP)  
# Support Vector Machines with Radial Basis Function Kernel  
mdlSVM <- train(X[train,], Y[train],   
 method='svmRadial',   
 trControl=myControl,   
 tuneLength=10,  
 preProcess=PP)  
# Lasso and Elastic-Net Regularized Generalized Linear Models  
mdlGLMNET <- train(X[train,], Y[train],   
 method='glmnet',   
 trControl=myControl,   
 tuneLength=10,  
 preProcess=PP)

Let's organize the results from each model and compare the performance.

models <- c("Stochastic Gradient Boosintg", "Boosted Tree", "Random Forest",   
 "Multi-Layer Perceptron", "k-Nearest Neighbots", "EARTH",   
 "Generalized Linear Model", "SVM Radial", "GLM Net")  
modelList <- list(mdlGBM, mdlBLACKBOOST, mdlPARRF,  
 mdlMLP, mdlKNN, mdlEARTH,  
 mdlGLM, mdlSVM, mdlGLMNET)  
modelResult <- data.frame()  
for(i in 1:length(models)){  
 resultDF <- modelList[[i]]$result[,c("ROC","Sens","Spec")]  
 resultDF$Model <- models[i]  
 modelResult <- rbind(modelResult,resultDF)  
}  
modelResultDT <- data.table(modelResult)  
meanROC <- as.data.frame(modelResultDT[,list(Mean=mean(ROC),Max=max(ROC)),by=Model])  
meanROC <- meanROC[order(meanROC$Mean),]  
modelResult$Model <- factor(modelResult$Model,  
 levels = meanROC$Model,ordered = TRUE)  
  
ggplot(modelResult, aes(x=Model, y=ROC)) + geom\_boxplot(fill="orange") +  
 coord\_flip() +  
 ggtitle("Compare Models with ROC") +  
 labs(x="Models",y="ROC") +   
 theme(plot.title = element\_text(family = "Trebuchet MS", color="#666666", face="bold", size=25, hjust=0)) +  
 theme(axis.title = element\_text(family = "Trebuchet MS", color="#666666", face="bold", size=22)) +  
 theme(axis.text = element\_text(family = "Trebuchet MS", color="#666666", face="bold", size=13))  
  
# We can see that Random Forest shows the best result for our Classification Data