# Analysis of Liver Hepatocellular Carcinoma Data in R

Bo Yang, Rui Nie, Yueying Hu

## Introduction

Hepatocellular carcinoma (LIHC/HCC) is the one of the leading cause of cancer death, which accounts for 8.2% of global cancer incidence and 4.7% of cancer-related mortality []. Previous study suggested that lifestyles and personal habits, including drinking alcohol, smoking, are positively associated with the onset LIHC [], which has greatly enhanced the efficiency of screening for the disease by health providers. Nevertheless, the difficulty for disease detection remains alarmingly high. Therefore, an indicator with higher sensitivity would advance the diagnosis stage and prolong patients' survival.

RNA Sequencing (RNA-seq) using the capabilities of high-throughput sequencing methods provides insight into the transcriptome of a cell. Through understanding the gene expression patterns reflected by RNA sequencing results, we aim to locate indicator genes for LIHC with the help of statistical learning methods Furthermore, given the massive amount of existing genes in human genome, our problem can be formulated as a high-dimensional problem. One of our side goal is to perform variable selection to reduce the risk of overfitting.

## Methods

### LIHC data from TCGA

We're primarily interested in analyzing RNA-seq data provided through The Cancer Genome Atlas (TCGA) for liver cancer patients. The dataset includes data from 424 LIHC participants and over 20,000 genes for expression level analysis. The inclusion of both normal and tumor samples allows us for comparative analysis between cancerous and non-cancerous liver tissues.

To download and prepare the LIHC data from TCGAbiolinks, we used `GDCquery`, `GDCdownload`, and `GDCprepare` functions for data accessing.

```r
query_TCGA_LIHC = GDCquery(
  project = "TCGA-LIHC",
  data.category = "Transcriptome Profiling",
  experimental.strategy = "RNA-Seq",
  workflow.type = "STAR - Counts",
  data.type = 'Gene Expression Quantification')

GDCdownload(query = query_TCGA_LIHC)

tcga_data_LIHC = GDCprepare(query_TCGA_LIHC)
```

```
## |                                                    |  0%                          |
```

```r
# transform a large summarized experiment into readable matrix
count=assay(tcga_data_LIHC)
```

Normalization for reads count help account for gene length and sequencing depth. Fragments Per Kilobase of transcript per Million mapped reads (FPKM) provides a comparable measure to interpret and utilize

RNA-seq results. We standardized our data into FPKM format for fair representation and comparison gene expressions.

```r
## load gene annotation data
file.annotations <- system.file("extdata",
                                "Biomart.annotations.hg38.txt",
                                package="countToFPKM")
gene.annotations <- read.table(file.annotations, sep="\t", header=TRUE)

## combine gene annotation and expression data
genelength=data.frame(gene=gene.annotations$Gene_ID,
                      length=gene.annotations$length,
                      type=gene.annotations$Gene_type)

rownames(count)<-substring(rownames(count),1,15)
df_temp = merge(genelength, count, by.x="gene",by.y="row.names")

## FPKM calculation by hand
countToFpkm <- function(counts, effLen)
{
  N <- sum(counts)
  exp( log(counts) + log(1e9) - log(effLen) - log(N) )
}

df<-countToFpkm(df_temp[,-c(1:3)], df_temp$length)
rownames(df)<-df_temp$gene; X = as.matrix(t(df));dim(df)
```

```
## [1] 20063   424
```

## Preprocessing

We collected the sample IDs from normal samples and from tumor samples separately and retrieved their index for future train-test dataset split. In summary, out of 424 samples, there are 50 normal samples and 374 tumor samples. This suggests we consider additional metrics such as precision, F1, or AUC-ROC scores to account for potential bias introduced by the imbalanced labels aside from crude accuracy for classification tasks.

```r
## retrieve the label for each sample
sample_ID = colnames(df)
tumor_sample_ID = sample_ID[which(substring(sample_ID,14,15)!=11)]
normal_sample_ID = sample_ID[which(substring(sample_ID,14,15)==11)]

## index for each label group
tumor_index = which(sample_ID %in% tumor_sample_ID)
normal_index = which(sample_ID %in% normal_sample_ID)
label = rep(0, length(sample_ID)); label[tumor_index] = 1
label = factor(label, labels = c("normal", "tumor")); table(label)
```

```
## label
## normal  tumor
##     50    374
```

## Variable Selection

Of all the variables (genes) selected for our preprocessed data, we'd like to filter out the ones with constant values across samples, as they provide no information in differentiating tumor samples from normal samples.

This leads to the removal of 381 genes from the entire sample.

```r
sd_x = apply(X, 2, sd)
ind = which(sd_x == 0)
X = X[, -ind];length(ind)
```

```
## [1] 381
```

To select variables while maintaining interpretation along the process, we will use the variance filtering method to select genes with a more variable expression activity from the samples we included. We kept the variance cutoff to be as high as 0.95, leaving only 5% (1004) of the original genes for future statistical analysis.

```r
Xnew = t(varFilter(t(X), var.func=IQR, var.cutoff = 0.95, filterByQuantile = TRUE));dim(Xnew)
```

```
## [1] 424 985
```

## Statistical Leaning

For building predictive statistical learning methods in LIHC classification, we continue with 985 selected genes using Elastic Net, decision tree, and random forest and assess their performances under the idea of train-test split validation. We will also use cross-validation to tune the hyperparameters for each model and compare their performances in terms of specificity, sensitivity, precision, F1, and AUC-ROC scores.

## Train-Test Split

Considering the imbalanced labels of two classes, it is important to preserve the representation of both classes in the training and testing samples to ensure the model is not overly biasing towards one class over the other. As a consequence, we use `createDataPartition` to create a stratified split for training and testing purposes with a 3:1 ratio.

```r
set.seed(123)
train_index = createDataPartition(label, p = 0.75, list = FALSE)
trainX = Xnew[train_index, ]; testX = Xnew[-train_index, ]
train_label = label[train_index]; test_label = label[-train_index]
```

## Elastic Net

Elastic Net is a penalized regression method that combines the L1 and L2 penalties of Lasso and Ridge regression. The L1 penalty suppress the coefficients of some variables to zero, which is equivalent to variable selection. The L2 penalty shrinks the coefficients of some variables towards a smaller scale. We take the idea of logistic regression and use below formula as the loss function for Elastic Net in classification problems:

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^{N} \log(1+\exp(-y_i(\beta_0 + x_i^T\beta))) + \lambda\left[\frac{1}{2}(1-\alpha)\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|\right]$$

We determine the alpha and lambda coefficient through grid search, where alpha determines the balance between L1 and L2 penalty and lambda determines the strength of the penalty.

```r
## grid search for hyperparameter tuning
set.seed(123)
alpha_grid = seq(0, 1, 0.1)
lambda_grid = seq(0.001, 0.1, 0.001)
cv_control <- trainControl(method = "cv", number = 5,classProbs = TRUE)
elastic <- train(x = trainX, y = train_label, method = "glmnet",
  trControl = cv_control, family = "binomial", tuneGrid = expand.grid(alpha = alpha_grid, lambda = lamb
elastic$bestTune
```

```
##     alpha lambda
## 236   0.2  0.036
```

```
## Evaluate the performance of Elastic Net
set.seed(123)
elastic_model = glmnet(trainX, train_label, family="binomial", alpha = 0.2, lambda = 0.036)
elastic_pred = predict(elastic_model, testX, type = "class")
result.elastic = confusionMatrix(factor(elastic_pred), test_label)$byClass[c(1, 2, 5, 7)]
## Variables selected by Elastic Net
coef.elastic = coef(elastic_model, s = "lambda.min")
idx = which(coef.elastic[,1]!=0)
variables <- row.names(coef.elastic)[idx]
variables <- variables[!variables %in% c('(Intercept)')]
names = rownames(rowData(tcga_data_LIHC))
names = gsub("\\..*","", names)
genes <- rowData(tcga_data_LIHC)[which(variables %in% names), "gene_name"]
genes[1:10]
```

```
##  [1] "TSPAN6"   "TNMD"     "DPM1"     "SCYL3"    "C1orf112" "FGR"
##  [7] "CFH"      "FUCA2"    "GCLC"     "NFYA"
```

## Decision Tree

Decision Tree is a non-parametric supervised learning method that can be used for both classification and regression problems. It works by recursively partitioning the data into smaller subsets based on the most significant variable at each step. The partitioning process is repeated until the data is completely separated or the stopping criteria is met. We displayed the gene names of the most important variables selected by Decision Tree.

```
## Pruning the tree by ten-fold cross validation
rt = rpart(train_label~., data = data.frame(trainX))
cp = rt$cptable[which.min(rt$cptable[,"xerror"]),"CP"]
rt.prune = prune(rt, cp = cp)
variables <- rt.prune$variable.importance[order(rt.prune$variable.importance, decreasing = TRUE)]
genes <- rowData(tcga_data_LIHC)[which(names(variables) %in% names), "gene_name"]
genes
```

```
##  [1] "TSPAN6"   "TNMD"     "DPM1"     "SCYL3"    "C1orf112" "FGR"
##  [7] "CFH"      "FUCA2"    "GCLC"     "NFYA"     "STPG1"    "NIPAL3"
```

```
## Evaluate the performance of Decision Tree
rt_pred = predict(rt.prune, newdata = data.frame(testX), type = "class")
result.decision = confusionMatrix(factor(rt_pred), test_label)$byClass[c(1, 2, 5, 7)]
```

## Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve the performance of the model. It works by building multiple decision trees on different subsets of the data and then averaging the predictions of all the trees. The random forest model is more robust to overfitting and less sensitive to outliers than a single decision tree. Here, we use the `randomForest` function to build the random forest model and tune the hyperparameters by cross validation.

```r
set.seed(123)
rf = randomForest(x = trainX, y = train_label, importance=TRUE)
## Evaluate the performance of Random Forest
rf.pred = predict(rf, newdata = testX)
result.rf = confusionMatrix(factor(rf.pred), test_label)$byClass[c(1, 2, 5, 7)]
```

Further, we explore the the variable importance, we consider two types of measure of variable importance: Mean Decrease Accuracy and Mean Decrease Gini. We see the gene TSPAN6 is the most important predictor in Mean Decrease Accuracy and in Mean Decrease Gini, which is consistent with the results of the decision tree and elastic net regression.

```r
imp <- rf$importance
variables <- names(head(sort(imp[, 3], decreasing = T)))
genes <- rowData(tcga_data_LIHC)[which(variables %in% names), "gene_name"]
genes
```

```
## [1] "TSPAN6"   "TNMD"     "DPM1"     "SCYL3"    "C1orf112" "FGR"
```

```r
variables <- names(head(sort(imp[, 4], decreasing = T)))
genes <- rowData(tcga_data_LIHC)[which(variables %in% names), "gene_name"]
genes
```

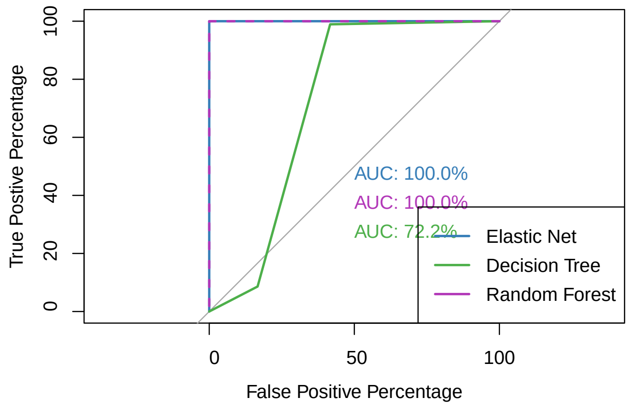
```
## [1] "TSPAN6"   "TNMD"     "DPM1"     "SCYL3"    "C1orf112" "FGR"
```

## Discussion

Overall, the three statistical learning methods have similar performance in terms of specificity and raw accuracy, and they all successfully identify similar genes as the most important predictors ("TSPAN6", "TNMD", "DPM1", "SCYL3", "C1orf112", "FGR"). However, the single decision tree model has the lowest sensitivity, F1 score, and AUC-ROC score, which indicates that the model is overfitting. Comparably, the elastic net regression model and the random forest model are more robust to such unbalanced dataset.

```r
data.frame(elastic = result.elastic, decision = result.decision, rf = result.rf)
```

```
##               elastic  decision        rf
## Sensitivity 0.9166667 0.5833333 0.7500000
## Specificity 1.0000000 0.9892473 1.0000000
## Precision   1.0000000 0.8750000 1.0000000
## F1          0.9565217 0.7000000 0.8571429
```


AUC: 100.0%
AUC: 100.0%
AUC: 72.2%
Elastic Net
Decision Tree
Random Forest
True Postive Percentage
False Positive Percentage