Identifying OGs and TSGs

Yue Wu

November 8th 2020

1 Introduction

Oncogenes and Tumor Suppressor Genes are two of the cancer driver genes, playing an important role in cancer. Oncogenes genes can help cells grow, but it will become a bad gene that causes cancer when it is out of control. Tumor suppressor genes are normal genes which can slow down cell division, repair DNA mistakes, or tell cells when to die. The cancer will happen when these two genes cannot work well to keep the balance between cell growth and apoptosis. In this report, in order to help discover new OGs and TSGs in cancer diagnosis, we are trying to use statistical methods to effectively separate genes into three classes: OG, TSG, and NG(Neutral genes that are not OGs or TSGs.).

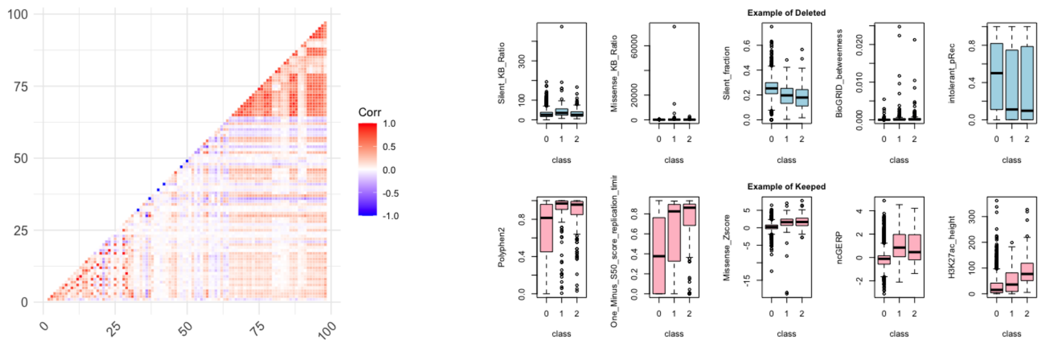
2 Preprocessing

There is a comprehensive gene feature dataset used as our training dataset with a total of 97 predictors which are various features of the genes, so we have to do the feature selection in order to find useful predictors avoiding multicollinearity and overfitting.

2.1 variable classification performance

We firstly created a correctness table that includes each predictor's correctness of prediction for each class using the cross-validation method. The logistic regression method is used during the process due to the majority of variables performed best with it after various testing.

Figure 1: Correlation plot Example Boxplots of Deleted and Keeped Variables



2.2 multicollinearity feature selection

According to the correlation plot, in order to eliminate the multicollinearity, we started to select the predictor combinations which have over 0.95 correlation. Based on the principle of deleting the least number of variables, we preferentially deleted the variables which have high collinearity with the most number of the correlated variables. Then the correctness table is used as a reference for choosing variables between two-variables selections, and the selection processing ends when there is no multicollinearity over 0.85 correlation.

2.3 box plots feature selection

Moreover, in order to keep the informative features, the box plot is used for double-checking, looking for the obvious difference of each class for the remaining variables. Since our purpose is to identify the OGs and TSGs, extreme classified predictors have been left. The Figure 1 shows some examples of what we kept and what we deleted. For now, we kept 64 predictors. According to all the box plots, we also found that there are some very obvious outliers in the data set, such as the points that show up in the top left plot in Figure 1. Since the class of OGs includes most extreme outliers with a disadvantage of quantity, we finally decided to only delete three outliers.

2.4 upsampling method

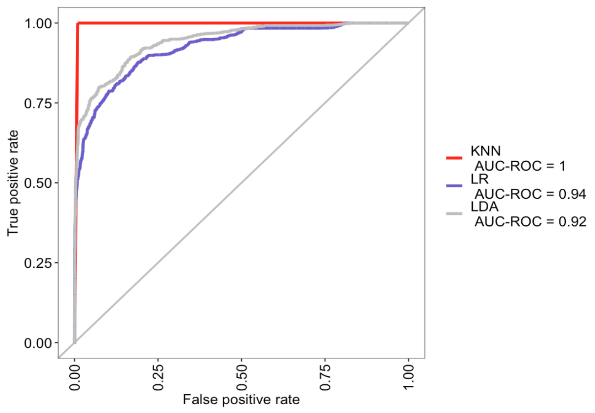
According to the preprocessing, the training data set obviously shows its imbalanced classification that almost 90% of observations are classified as NGs, which may fail to make an accurate prediction on OGs and TSGs. The result of ignoring the imbalanced classification we tested produced a failure prediction with lower accuracy. Since we do not want to lose any information and computation efficiency is not important here, we choose to use an oversampling method to upsample the minority classes OG and TSG after the feature selection.

2.5 significance feature selection

Finally, we did the significance testing of the logistic regression coefficients after upsampling the minority classes, putting all the variables we left into the generalized linear model to check their p-values with 0.05 significance level; and all the significant variables became our final choice.

3 Model

In order to test the effectiveness of predictive ability and generalizability of the model, we used 10-fold cross-validation, that we conserved one subsample of data as the validation data and the rest as the training data. Three classifiers, K-nearest neighbor, logistic regression and linear discriminant analysis, were also used in the model training. Since we need to classify with three unordered categorical levels, the multinomial logistic regression was preferred over logistic regression. Based on the ROC curve, the KNN method has an extremely high ROC score which indicates that the KNN method is overfitted to the training data. Multinomial logistic regression outperformed linear discriminant analysis. Hence we chose the multinomial logistic regression to build our final model, including 42 variables.



4 Result

We created a score function to test the correctness of how our model predicts the training dataset, so we can roughly judge whether our method works well or not. We both consider the validation accuracy and the score to determine how it works. Table 1 shows the best evaluation metric value with this model, including the validation accuracy, score of correctness, the testing result counts, and website correctness score. The correctness score of the training dataset is higher than the website correctness score of the testing dataset, which means the ability of our model to predict new data is not well enough.

Table 1			
Data	NG	OG	TSG
Validation accuracy	0.8779	0.7755	0.72
Testing counts	1062	129	172
Validation score	0.7867		
WCA	0.74615		

5 Conclusion

Most of our test results have more than 70% correctness of both the training dataset and the testing dataset in the kaggle, but we still think our model failed to work well. We face the challenge of the different results from the significance test by setting different seeds. It made us feel confused on what predictors are better to use. As a result, our selected feature may depend too much on the seed we chose, which led to a lack of generalizability by the randomness effect and failed to work well in discovering the OGs and TSGs steadily. Here, our selected predictors are based on the significance test by setting seed 123.

Moreover, our model ignored the interaction effects among variables. Some of them may interact with each other causing a new variable with a significant effect on other independent variables. Lacking a deep understanding of each variable, it would be unwise to ignore the interaction effects, which may cause our model to not match well the true data. Therefore, we believe that our current model failed to work well, and there could be a great improvement in our feature selection and model building.

6 Alternative Model

Since our model includes 42 predictors, we considered this complex model if it can lead to a phenomenon known as overfitting the data. Therefore, we considered building one model with less predictors. In this model, we remove all the predictor combinations that have over 0.8 correlation, and the upsampling method and outlier still to be considered in this model. Finally, our model includes only 21 predictors which are half of the original one. Table 2 shows the results in the same format as Table 1. Compare the two tables, we can see that our alternative model predicts the training dataset better than the original model with a small difference in the website correctness score.

Table 2			
Data	NG	OG	TSG
Validation accuracy	0.8826	0.7959	0.76
Testing counts	1070	147	146
Validation score	0.8093		
WCA	0.74308		

Contribution

Yue Wu created a correctness table as a reference we used to help feature selection. Hanyue Zhang did the feature selection based on the correlation table, and after that Qinyi Chen looked at the box plots for further selection. Then Yue Wu used an oversampling method to upsample the minority classes. Hanyue Zhang determined the final predictors we used using the GLM significance test and created the models. Yue Wu integrated all the codes as an appendix. Qinyi Chen did the final report. Most importantly, we encouraged each other to get all the work done.

Appendix: R code

```r
#  Libraries Used
library(readxl)
library(caret)
library(MLeval)
library(ggcorrplot)



# Import Data
# remove id column from training dataset
training <- read.csv("training.csv")[, -1]
testing <- read.csv("test.csv")



# Functions Used
#######################
### logistic model function
### input: training and validation dataset
### output: logistic model
#######################
fit_LR <- function(training, validation){
  # k-fold cross-validation
  train_control <- trainControl(method="cv",
                                number = 10,
                                classProbs = TRUE,
                                savePredictions = TRUE)
  # fit logistic regression model to training data
  LRfit <- train(class~.,
                 data = training,
                 method = "multinom",
                 preProc = c("center", "scale"),
                 trControl = train_control,
                 trace = FALSE)
  LRfit
}

#######################
### knn model function
### input: training and validation dataset
### output: knn model
#######################
fit_KNN <- function(training, validation){
  # k-fold cross-validation
  train_control <- trainControl(method="cv",
                                number = 10,
                                  classProbs = TRUE,
                                  savePredictions = TRUE)
  # fit knn method to training data
  KNNfit <- train(class~.,
                  data = training,
                  method = 'knn',
                  preProc = c("center", "scale"),
                  trControl = train_control,
                  tuneGrid = expand.grid(k = seq(1, 50, by = 5)))
  KNNfit
}

#######################
### lda analysis function
### input: training and validation dataset
### output: lda model
#######################
fit_LDA <- function(training, validation){
  # k-fold cross-validation
  train_control <- trainControl(method="cv",
                                  number = 10,
                                  classProbs = TRUE,
                                  savePredictions = TRUE)
  # fit linear discriminant analysis
  LDAfit <- train(class~.,
                  data = training,
                  method = "lda",
                 preProc = c("center", "scale"),
                 trControl = train_control)
  LDAfit
}

#######################
### qda model function
### input: training and validation dataset
### output: qda model
#######################
fit_QDA <- function(training, validation){
  # k-fold cross-validation
  train_control <- trainControl(method="cv",
                                  number = 10,
                                  classProbs = TRUE,
                                  savePredictions = TRUE)
  # fit quadratic discriminant analysis
  QDAfit <- train(class~.,
                  data = training,
                  method = "qda",
                 preProc = c("center", "scale"),
                 trControl = train_control)
  QDAfit
}

#######################
### function to built correctness table
### input: training dataset
### output: number of correctly predicted NG, OG, and TSG
#######################
get_proportion <- function(training){
  set.seed(123)
  # split data into training and validation dataset
  trainIndex <- createDataPartition(training$class, p = 0.7, list = FALSE)
  train <- training[trainIndex,]
  validation <- training[-trainIndex,]

  ### we have tested LR, LDA, QDA, and KNN method
  ### majority of variables performed best with LR methods

  LRfit <- fit_LR(train, validation)
  # KNNfit <- fit_KNN(train, validation)
  # LDAit <- fit_LDA(train, validation)
  # QDAfit <- fit_QDA(train, validation)
  fit <- LRfit
  # predict validation class
  pred <- predict(fit, newdata = validation)
  # corrected prediction
  match <- pred[pred == validation$class]
  # number of correctly predicted NG, OG, and TSG
  c(sum(match == "NG"), sum(match == "OG"), sum(match == "TSG"))
}

#######################
### function to return prediction accuracy proportion for each class
### input: validation class prediction, accurate validation class
### output: proportion of correctly predicted NG, OG, and TSG
#######################
each_accuracy <- function(result, answer) {
  # correct predicted value
  match <- result[result == answer]
  # proportion
  acc <- c(sum(match == 0)/sum(answer == 0),
  sum(match == 1)/sum(answer == 1),
  sum(match == 2)/sum(answer == 2))
  names(acc) <- 0:2
  acc
}

#######################
### function to simulate score base on kaggle score evaluation
### input: validation class prediction, accurate validation class
### output: score
#######################
score <- function(result, answer) {
  total <- sum(answer == 0) + sum(answer != 0) * 20
  match <- result[result == answer]
  (sum(match == 0) + sum(match != 0)*20)/total
}



# Data Preprocessing

## Basic Data Exploration and Check for Missing Informations

dim(training)
str(training)
summary(training)
# determine whether sales_train dataframe contains any NA/NULL in every column
all(!is.na(training))
```

```r
all(!is.null(training))
```

```r
## Feature Selection base on Multicollinearity

# deleting potential outliers
training <- training[-which(training$Missense_KB_Ratio >= 10000), ]
training <- training[-which(training$Missense_Damaging_TO_Benign_Ratio >= 30), ]
training <- training[-which(training$LOF_TO_Total_Ratio >= 0.8), ]

valid_training <- training
training_class_factor <- factor(ifelse(valid_training$class == 0, "NG",
                                       ifelse(valid_training$class == 1, "OG", "TSG")))

# evaluate the classification performance of every variable
prop <- t(apply(valid_training[, -ncol(valid_training)], 2, function(x) {
  get_proportion(data.frame(x, class = training_class_factor))
}))
# add column for sumof correctly predicted OG and TSG
prop <- cbind(prop, prop[, 2] + prop[, 3])
colnames(prop) <- c("Correct_NG", "Correct_OG", "Correct_TSG", "Correct_OG_TSG")

# correlation plot
corr <- cor(training)
colnames(corr) <- 1:98; rownames(corr) <- 1:98
ggcorrplot(corr, type = "lower", outline.col = "white")

# check collinearity
c <- cor(valid_training)
h <- list()
n <- colnames(c)[-ncol(c)]
for(i in 1:(ncol(c)-1)){
  if(length(n[abs(c[i, ]) >= 0.85 & c[i, ] != 1]) != 0){
    h[[n[i]]] <- n[abs(c[i, ]) >= 0.85 & c[i, ] != 1]
  }
}

### first delete variables have high collinearity with most number of correlated variables
### then use prop as reference for deleting variables between two-variables selections

# delected variables
delete <- c("Broad_H3K9ac_percentage",
            "Broad_H3K4me2_percentage",
            "LOF_KB_Ratio",
            "LOF_TO_Benign_Ratio",
            "Splice_TO_Benign_Ratio",
            "Missense_TO_Silent_Ratio",
            "Missense_TO_Benign_Ratio",
            "Missense_Damaging_TO_Benign_Ratio",
            "H3K79me2_width",
            "H3K27me3_width",
            "H3K4me1_width",
            "Broad_H3K36me3_percentage",
            "Broad_H3K4me3_percentage",
            "LOF_TO_Total_Ratio",
            "Missense_fraction",
            "CNA_amplification",
            "S50_score_replication_timing",
            "Gene_expression_Minus_Z_score",
            "Minus_Cell_proliferation_rate_CRISPR_KD",
            "Promoter_hypomethylation_in_cancer",
            "Gene_body_hypomethylation_in_cancer",
            "Splice_TO_Total_Ratio",
            "Broad_H3K4me1_percentage",
            "Broad_H4K20me1_percentage")
# update
valid_training <- valid_training[, -which(colnames(valid_training) %in% delete)]



## Feature Selection base on Boxplot

total <- colnames(valid_training)[-ncol(valid_training)]

# delete variable
delete <- c("Silent_KB_Ratio",
            "Missense_KB_Ratio",
            "Silent_fraction",
            "Frameshift_indel_fraction",
            "Lost_start_and_stop_fraction",
            "BioGRID_betweenness",
            "intolerant_pRec",
            "intolerant_pNull",
            "ncRVIS")
keeped <- total[!(total %in% delete)]

par(mfrow = c(2, 5))
# example of deleted variable boxplot
boxplot(valid_training[, delete[1]] ~ valid_training$class,
        ylab = delete[1],
        xlab = "class",
        col = "lightblue",
        cex.labels = 0.8)
boxplot(valid_training[, delete[2]] ~ valid_training$class,
        ylab = delete[2],
        xlab = "class",
        col = "lightblue",
        cex.labels = 0.8)
boxplot(valid_training[, delete[3]] ~ valid_training$class,
        ylab = delete[3],
        xlab = "class",
        col = "lightblue",
        main = "Example␣of␣Deleted",
        cex.main = 1,
        cex.labels = 0.8)
boxplot(valid_training[, delete[6]] ~ valid_training$class,
        ylab = delete[6],
        xlab = "class",
        col = "lightblue",
        cex.labels = 0.8)
boxplot(valid_training[, delete[7]] ~ valid_training$class,
        ylab = delete[7],
        xlab = "class",
        col = "lightblue",
        cex.labels = 0.8)
```

```r
# example of keeped variable boxplot
boxplot(valid_training[, keeped[8]] ~ valid_training$class,
        ylab = keeped[8],
        xlab = "class",
        col = "pink",
        cex.labels = 0.8)
boxplot(valid_training[, keeped[21]] ~ valid_training$class,
        ylab = keeped[21],
        xlab = "class",
        col = "pink",
        cex.labels = 0.8)
boxplot(valid_training[, keeped[33]] ~ valid_training$class,
        ylab = keeped[33],
        xlab = "class",
        col = "pink",
        main = "Example␣of␣Keeped",
        cex.main = 1,
        cex.labels = 0.8)
boxplot(valid_training[, keeped[38]] ~ valid_training$class,
        ylab = keeped[38],
        xlab = "class",
        col = "pink",
        cex.labels = 0.8)
boxplot(valid_training[, keeped[50]] ~ valid_training$class,
        ylab = keeped[50],
        xlab = "class",
        col = "pink",
        cex.labels = 0.8)

# update
valid_training <- valid_training[, c(keeped, "class")]



## Upsampling

valid_training$class <- factor(ifelse(valid_training$class == 0, "NG",
ifelse(valid_training$class == 1, "OG", "TSG")))

# split data into training and validation
set.seed(123)
trainIndex <- createDataPartition(valid_training$class, p = 0.7, list = FALSE)
imtraining <- valid_training[trainIndex, ]
validation <- valid_training[-trainIndex, ]
correct_result <- training$class[-trainIndex]

# upsampling
training_upSample <- upSample(x = imtraining[, -ncol(imtraining)], y = imtraining$class)

upsampling_prop <- c(sum(training_upSample$Class == "NG"),
sum(training_upSample$Class == "OG"),
sum(training_upSample$Class == "TSG")) / nrow(training_upSample)
names(upsampling_prop) <- c("NG", "OG", "TSG")
prop.table(table(imtraining$class))
upsampling_prop



## Feature Selection base on Variable Significance
```

```
t <- training_upSample
t$Class <- as.numeric(t$Class)-1
# delete variables base on the p-value significance
summary(glm(Class~., data = t))

delete <- c("N_LOF",
            "N_Splice",
            "Missense_Damaging_TO_Missense_Benign_Ratio",
            "Polyphen2",
            "Missense_TO_Total_Ratio",
            "Nonsense_fraction",
            "Exon_Cons",
            "BioGRID_clossness",
            "Promoter_hypermethylation_in_cancer",
            "Gene_body_hypermethylation_in_cancer",
            "intolerant_pLI",
            "Missense_Zscore",
            "dN_to_dS_ratio",
            "ncGERP",
            "Length_H3K4me3",
            "H3K4me3_height",
            "H3K4me1_height",
            "H3K36me3_height",
            "H3K9me3_height",
            "H3K9ac_height",
            "H3K79me2_height",
            "H4K20me1_height")

training_upSample <- training_upSample[, -which(colnames(training_upSample) %in% delete)]

# selected variables
colnames((training_upSample))[-ncol(training_upSample)]



# Model

## Validation

# k-fold cross-validation
train_control <- trainControl(method="cv",
                              number = 10,
                              classProbs = TRUE,
                              savePredictions = TRUE)

# evaluate the classification performance of LR, KNN, LAD, QDA on training data
LRfit <- train(Class~.,
               data = training_upSample,
               method = "multinom",
               preProc = c("center", "scale"),
               trControl = train_control,
               trace = FALSE)
KNNfit <- train(Class~.,
                data = training_upSample,
                method = 'knn',
                preProc = c("center", "scale"),
                trControl = train_control,
                tuneGrid = expand.grid(k = seq(1, 50, by = 5)))
```

```
LDAfit <- train(Class~.,
                data = training_upSample,
                method = "lda",
               preProc = c("center", "scale"),
               trControl = train_control)

# evaluate using ROC curves
res <- evalm(list(KNNfit, LRfit, LDAfit), gnames = c('KNN','LR', 'LDA'))
res$roc
### choose logistic regression

# predict on validation
result_train <- as.numeric(predict(LRfit, newdata = validation)) - 1
# evaluation metric value
each_accuracy(result_train, correct_result)
# simulate score base on kaggle score evaluation
score(result_train, correct_result)



## Testing

# predict on testing data
result <- as.numeric(predict(LRfit, newdata = testing[, -1])) - 1
result <- data.frame(id = testing$id, class = result)
# number of predicted NG, OG, TSG
table(result$class)
# proportion of predicted NG, OG, TSG
prop.table(table(result$class))
write.csv(result, "sample.csv", row.names = FALSE)



# Alternative Model

# alternative model with 21 variables
alternative_var_21 <- c("N_Missense",
                "N_LOF",
                "Missense_KB_Ratio",
                "Missense_Entropy",
                "LOF_TO_Silent_Ratio",
                "Missense_Damaging_TO_Benign_Ratio",
                "LOF_TO_Total_Ratio",
                "VEST_score",
                "BioGRID_betweenness",
                "BioGRID_clossness",
                "BioGRID_log_degree",
                "pLOF_Zscore",
                "Length_H3K4me3",
                "H3K4me1_width",
                "H3K36me3_width",
                "H3K27ac_height",
                "Broad_H3K9ac_percentage",
                "H3K9ac_height",
                "Broad_H3K79me2_percentage",
                "H4K20me1_width",
                "H4K20me1_height")
alternative_var_21
```