STATS 101C Midterm Report

Jason Chhay, Kevin Hahn, Newton Peng

November 2020

1 Introduction

All teams were given a data set on 3,177 genes, with 99 variables. The essence of the problem was to classify genes that were cancer-related from those that were not cancer-related; the predictors measured mutation-related features, genomic features, phenotype features and epigenetic features. Since 2 of the variables are the ID and the type of gene, there were effectively 97 predictor variables total. The goal given to the Kaggle teams was to predict if a gene was a neutral gene (NG), oncogene (OG), or a tumor suppressor gene (TSG) using these 97 variables.

2 Methodology

2.1 Preprocessing

We immediately began with exploratory analysis. Observing the entire training set, we saw that there were 2840 NGs, 168 OGs, and 169 TSGs. We compared every predictor variable to the gene type via boxplots [1], and decided which variables were important to keep in our model, since some may be unnecessary. We kept 72 variables out of the 97 potential predictor variables. We also then factorized and relabeled the column "class" since the gene types are discrete values.

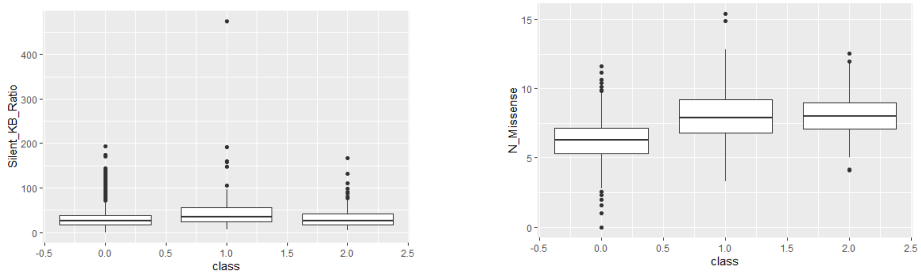

Figure 1: Example boxplots we observed

Afterwards, we removed any variables that were highly correlated with each other using `findCorrelation()` within the `caret` package. This function finds the average pair-wise correlations for all variables with respect to a cutoff value and keeps those that are under the cutoff. We chose our cutoff to be 0.75. We removed an additional 25 variables to bring our list of variables to 47 variables.

Noticing issues when building our model, we decided to further remove variables. The function `nearZeroVar()`, with parameters `freqCut = 19` and `uniqueCut = 10`, finds any variable that has many values near one value and very few towards another. This removes some variables that do not contribute significantly to the classification because across each class, the value of the predictor may not change that much.

The parameters mentioned earlier represent a conservative approach to identifying near-zero-variance predictors by returning `TRUE` if the predictor's values are less than 10% unique and the ratio of frequencies for the most common value over the second most common value is above 19. As described in the `nearZeroVar` documentation, "an example of near zero variance predictor is one that, for 1000 samples, has two distinct values and 999 of them are a single value." This function allowed us to remove 3 more variables: `recurrent_missense_fraction`, `lost_start_and_stop_fraction`, and `canyon_genebody_hypermethylation`. More aggressive approaches, i.e. lower `freqCut` values and higher `uniqueCut` values, were tested, but they removed too many predictors and substantially hurt the accuracy of our model.

Finally, we noticed that our models thus far were able to detect NGs accurately, and OGs and TSGs inaccurately. We believed this to be due to the sheer number of NGs in the data. To deal with this, we randomly selected 500 NG observations and placed them into a different data frame along with the 168 OGs and the 169 TSGs. We then used this different data frame as our training data set to build our model.

2.2 Statistical Model

Our model uses multinomial logistic regression. We found this to be the best fitting model based on our WCA values; LDA and KNN did not perform as well. In particular, KNN classifies TSGs and OGs as NGs most likely due to the abundance of NGs and thus higher probability that edge values were likely to be classified with these NGs.

We applied resampling on our training data to improve the accuracy of our model. We tested both cross-validation (LOOCV, k-folds for various k values) and bootstrap; bootstrap with 1000 resamples was chosen as our resampling method. We chose 1000 based on various values tested for bootstrapping and our corresponding WCA scores.

From our new training data set, we applied a 70/30 split for training and validation data and fed the training data to our model, using multinomial logistic regression and bootstrap with 1000 resamples.

We mentioned earlier that our previous models had trouble identifying OGs and TSGs properly. As a result, we made sure that our model used mean

sensitivity as its evaluation metric. By maximizing this metric, this means that the model will prioritize classifying the OGs and TSGs correctly, since the abundance of NGs throws off the model's ability to correctly classify OGs and TSGs. This also coincides with the weighted scoring, since we are penalized heavily for incorrectly classifying OGs and TSGs. Our mean sensitivity was roughly 73% and more information can be found in Figure 3 in the appendix section.

Applying all these steps yielded an accuracy of 84.4% on our partitioned validation data set. We then used our model to predict the classes of the test data set, and uploaded our predictions to Kaggle.

3 Results

Our final model consisted of multinomial logistic regression with 47 predictors trained to mean sensitivity using bootstrap. As mentioned before, this model was trained on a dataset of 500 NGs, 168 OGs, and 169 TSGs. On the Kaggle leaderboard, we have achieved approximately an 82.3% weighted accuracy rate. On our testing set, which consisted of 250 observations, our accuracy was 84.4% [2] (see appendix for confusion matrix).

4 General Conclusion

Our model could have been improved with additional models of Logistic Regression or a combination of other models that would later perform majority voting. For instance, LDA and KNN might have been better at classifying OGs from TSGs with both of which we can make vote with our final Logistic Regression model.

In general, it seemed like the most significant obstacle that was worth addressing was the large class imbalance in the training set.

5 Appendix

5.1 Sources

We used the following sources in addition to Ch 1-5 of the textbook and any relating lecture material:

http://topepo.github.io/caret/index.html

https://www.r-bloggers.com/2014/03/near-zero-variance-predictors-should-we-remove-them/

https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

5.2 Additional Images

```
Confusion Matrix and Statistics

          Reference
Prediction  NG  OG TSG
       NG  137  10   7
       OG    6  35   4
       TSG   7   5  39

Overall Statistics

               Accuracy : 0.844
                 95% CI : (0.793, 0.8867)
    No Information Rate : 0.6
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7182

 Mcnemar's Test P-Value : 0.7744

Statistics by Class:

                     Class: NG Class: OG Class: TSG
Sensitivity             0.9133    0.7000     0.7800
Specificity             0.8300    0.9500     0.9400
Pos Pred Value          0.8896    0.7778     0.7647
Neg Pred Value          0.8646    0.9268     0.9447
Prevalence              0.6000    0.2000     0.2000
Detection Rate          0.5480    0.1400     0.1560
Detection Prevalence    0.6160    0.1800     0.2040
Balanced Accuracy       0.8717    0.8250     0.8600
```

Figure 2: Confusion matrix of our model on the partitioned validation data set

```
Penalized Multinomial Regression

587 samples
 43 predictor
  3 classes: 'NG', 'OG', 'TSG'

Pre-processing: centered (43), scaled (43)
Resampling: Bootstrapped (1000 reps)
Summary of sample sizes: 587, 587, 587, 587, 587, 587, ...
Resampling results across tuning parameters:

  decay  logLoss    AUC        prAUC      Accuracy   Kappa      Mean_F1    Mean_Sensitivity  Mean_Specificity
  0e+00  1.1629984  0.8905670  0.7453788  0.7805859  0.6105078  0.7166730  0.7189005         0.8790865
  1e-04  1.1513428  0.8907782  0.7462262  0.7808098  0.6108408  0.7169215  0.7190950         0.8791636
  1e-01  0.7441553  0.9003952  0.7710212  0.7938764  0.6315927  0.7310396  0.7306394         0.8847082
  Mean_Pos_Pred_Value  Mean_Neg_Pred_Value  Mean_Precision  Mean_Recall  Mean_Detection_Rate  Mean_Balanced_Accuracy
  0.7200851            0.8793132            0.7200851       0.7189005    0.2601953            0.7989935
  0.7204074            0.8794556            0.7204074       0.7190950    0.2602699            0.7991293
  0.7372084            0.8876797            0.7372084       0.7306394    0.2646255            0.8076738

Mean_Sensitivity was used to select the optimal model using the largest value.
The final value used for the model was decay = 0.1.
```

Figure 3: Our model output, mean sensitivity ≈ 0.73

5.3 Source Code

See Github repository for plain text version

```
# Code for model with 82.298% accuracy
# Removed 3 near-zero variance predictors below:
# "Recurrent_missense_fraction"      "Lost_start_and_stop_fraction"
"Canyon_genebody_hypermethylation"
# SEED 12, SEED 5
# Bootstrap - 1000 resamples, 500 NG resamp, correlation cutoff = 0.75,
partitioned data set 70/30
library(readr)
library(mclust)
library(MASS)
library(class)
library(caret)
library(MLeval)
library(MLmetrics)
library(tidyverse)

# Load data into RStudio
training <- read_csv("C:/Users/kevin/Documents/Classes/STATS
101C/ucla-stats101c-lec4/training.csv")
test <- read_csv("C:/Users/kevin/Documents/Classes/STATS
101C/ucla-stats101c-lec4/test.csv")

# Generating boxplots of class against each predictor, commented out for brevity:
# for ( i in temp[1:length(temp)]) {
#   bp = ggplot(data = training,
#               aes_string(x = names(training)[99],
#                          y = names(training)[i],
#                          group = names(training)[99])) + geom_boxplot()
#   print(bp)
# }

# Variables of interest that should be kept (after examining the box plots)
temp <- c(3, 4, 5, 7, 8, 9, 13, 15, 16, 17, 18, 20, 22, 23, 25, 26, 28, 29, 31,
          35, 36, 37, 38, 39, 40, 41, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54,
          55, 57, 58, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
          77,
          78, 79, 80, 83, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99)
```

```r
# Finding variables that are correlated with each other and removing them
y <- training[,temp]
# Temporarily exclude the response variable column (72 in this new set),
# and look at strictly predictor-to-predictor correlations
y2 <- y[,-72]
corr_mat_y2 <- cor(y2)
corr_y2 <- findCorrelation(corr_mat_y2, cutoff = 0.75)

set.seed(12)
# Attempt to equalize the class imbalance, randomly sample 500 values from NGs
NGs <- y %>% filter(class == 0)
OGs <- y %>% filter(class == 1)
TSGs <- y %>% filter(class == 2)
NG.samp <- sample(1:dim(NGs)[1], 500)

balanced.data <- rbind(NGs[NG.samp, ], OGs, TSGs)
balanced.data$class <- factor(balanced.data$class)
levels(balanced.data$class) <- c("NG", "OG", "TSG")

# Remove highly-correlated variables from each other first from finished model
balanced.data.new <- balanced.data[, -corr_y2]

##############################################################################
# Remove predictors with extremely low, near-zero variance
# recurrent_missense_fraction was causing warning messages to surface when running
train()
# So this motivated further examination of the other predictors' variances.

# Examine whether each predictor is deemed to have a near-zero variance
all_ps = nearZeroVar(balanced.data.new[, -ncol(balanced.data.new)], names = TRUE,
                     freqCut = 19, uniqueCut = 10, saveMetrics = TRUE) # for
                     viewing purposes
print(all_ps)

# Identify and possibly exclude other nonzero predictors:
# NOTE: 2 and 20 for freqCut and uniqueCut are considered an aggressive approach.

#       The default parameter setting is 19 and 10, which is fairly conservative.
other_nzp = nearZeroVar(balanced.data.new[, -ncol(balanced.data.new)], names =
TRUE,
            freqCut = 19, uniqueCut = 10, saveMetrics = FALSE)
print(other_nzp)

# Unselect the near-zero variance predictors:
balanced.data.new = balanced.data.new %>% select(-other_nzp)

##############################################################################
# Test and training set for balanced.data.new
set.seed(5)
trainIndex <- createDataPartition(balanced.data.new$class, p = 0.7, list = FALSE)
gene.train <- balanced.data.new[trainIndex,]
gene.test <- balanced.data.new[-trainIndex,]
# Bootstrap with resampling set to 1000
train_control <- trainControl(method = "boot", number = 1000, summaryFunction =
multiClassSummary,
                              classProbs = TRUE, savePrediction = TRUE)
```

```r
# Multinomial logistic regression model. The LRfit line may take a while to load
due to 1000 resamples
LRfit <- train(class ~ ., data = gene.train, method = "multinom", preProc =
c("center", "scale"),
               trControl = train_control, trace = FALSE, metric =
               "Mean_Sensitivity")
print(LRfit)
# Predict and test out the multinomial logistic regression model
predLR <- predict(LRfit, newdata = gene.test)
confusionMatrix(data = predLR, reference = gene.test$class)

##############################################################################
## PREDICTION ON TEST DATA AND PROCESSING FOR SUBMISSION
saved_prediction_copy =
read_csv("C:/Users/kevin/Documents/boot_predict_1000_with_3_nzvpredictors_removed_nov8_v2.csv")
# NOTE: saved_prediction_copy is a copy of the csv that obtained the 82.298%
accuracy score.
# Comparison will be run against the csv we generate here to make sure the results
are reproducible
test2 <- test[,temp]
test2 <- test2[,-corr_y2]

# Remove the 3 near-zero variance predictors
test2 <- test2 %>% select(-other_nzp) %>% as.data.frame()

# Apply multinomial regression model to the test data
predLR.test <- predict(LRfit, newdata = test2)

# Preparing and assembling the prediction data frame
prediction <- data.frame(test$id, predLR.test)
colnames(prediction) <- c("id", "class")
levels(prediction$class) <- c(0, 1, 2)

# CHECK: which(...) should return integer(0), and length(which(...)) should return
0
# meaning that the predictions are exactly the same for each observation in the
test data
which(saved_prediction_copy$class != prediction$class)
length(which(saved_prediction_copy$class != prediction$class))

# Store file
write_csv(prediction,
"boot_predict_1000_with_3_nzvpredictors_removed_nov8_final.csv")

# Print out the predictors used in the model
predictors_used = names(test2)
predictors_used = subset(predictors_used, predictors_used != "class")
print(predictors_used)
```

6 Statement of Contributions

Our team members worked on the following to create our model and report:

Jason Chhay: Implemented bootstrapping and tested correlation cutoff values on various models (LDA, KNN, multinomial LR), created and structured LaTeX document on Overleaf, and contributed to introduction, methodology, statistical model, and appendix.

Kevin Hahn: Introduced the removal of near-zero-variance predictors, led the investigation of boxplots for removing predictors, and researched possible undersampling techniques. Contributed to the following sections of the report: introduction, preprocessing, statistical model, results, general conclusion, and appendix code.

Newton Peng: Introduced other metrics for tuning our models (Kappa, Mean Sens., ROC) and undersampling NG observations in the test set. Worked with other members in removing variables through boxplots. Contributed to introduction, statistical model, results, general conclusion and appendix.