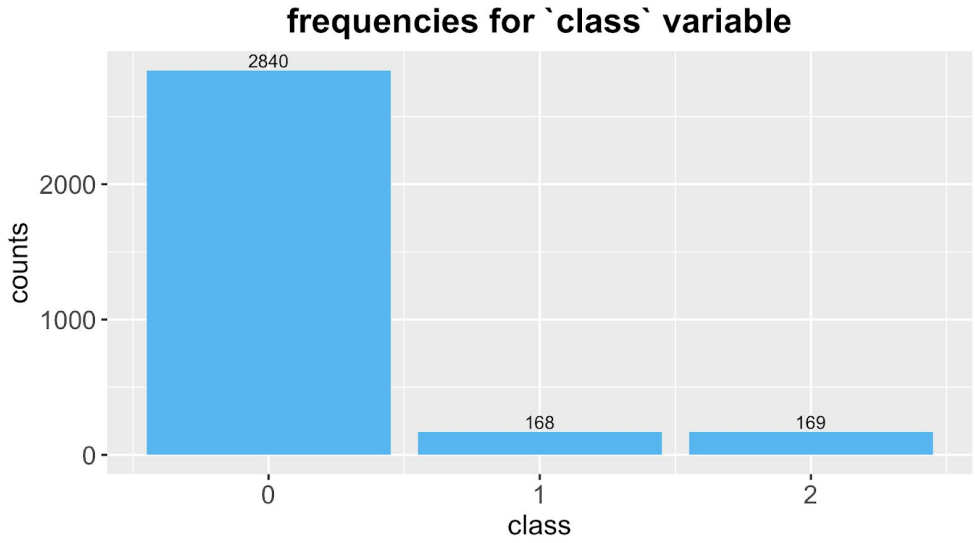
Team Chi Beta Gamma Stats 101C Midterm Project

Eustina Kim, Tiffany Feng, Jonathan Martinez, Kevin Chen

Introduction

In this project, we aimed to create a statistical learning model using a dataset of 3177 gene observations and 97 gene related predictors to predict cancer driver genes—Oncogenes (OGs) and Tumor Suppressor Genes (TSGs)—in a test data set of 1363 gene observations. The genes are encoded as the following in the dataset: neutral gene (NG) as 0, OG as 1, and TSG as 2. We used statistical learning methods discussed in chapters 1-5 in "*An Introduction to Statistical Learning with Applications in R"* to create our models.

For our exploratory data analysis, we graphed the frequencies of the categories within the `class` variable. It is apparent that there is class imbalance in that there are very few 1s and 2s compared to 0s. We discuss how we dealt with this issue in the methodology section.



Methodology

a. Preprocessing our data

First, we made sure to remove the ID column from our predictors, since ID is an identifier and is not related to `class`. We then combined variables that may be highly correlated into one variable by standardizing them and summing them up. For instance, we standardized variables `Broad_H3K4me2_percentage`, `H3K4me2_height`,

`H3K4me2_width` and then added the standardized values across rows into one column with the name “H3K4me2”. We repeated this procedure for other predictors that had a similar name pattern and were likely to be very correlated with one another. Through this process we created combined variables like `H3K4me3`, `H3K4me1`, etc. This technique ensures we are addressing the issue of multicollinearity by combining variables that may be highly correlated with one another.

To further narrow down the number of predictors, we constructed an initial linear model with `class` as our response and all of our remaining variables as our predictors. Using the summary output from this model, we then chose the predictors with a p-value less than 0.1. We opted for a more generous p-value cutoff that is greater than the 0.05 standard in order to include more predictors in our model and to prevent underfitting.

b. Statistical Model

For our model, we opted to use the multinomial logistic model, which applies logistic regression when there are more than 2 classes to predict. This was the case for our project, as our response variable `class` could have 3 different values: 0, 1, or 2. We used k-fold cross validation with k = 10 to compare model performance, choosing the model with the highest cross validation accuracy score as our final model to predict on test data. Next, we changed the probability threshold for predicting class ‘0’ to the mean of the vector of probabilities of predicting ‘0’ when we applied the final logistic model on the test data,which was 0.897209. It was necessary to make the threshold for predicting ‘0’ higher because around 90% of the training data is class of ‘0’. Therefore, we knew that the predicted probabilities for class ‘0’ would be a lot higher than 0.5 and increasing the threshold would be one way to combat the imbalance of categories in our data.

Results

Using our model, we obtained a Kaggle score of 0.82606 We trained the given data using KNN, Multinomial logistic, LDA but chose to use multinomial logistic regression for our final model due to the method having the highest cross validation accuracy of 0.9527816. We did not use QDA as a model because we encountered a rank deficiency error. After researching online, it seems like having too little data--especially on classes ‘1’ and ‘2’--may be the reason for the error.

General Conclusions

For this classification project, we combined our knowledge of data processing and statistical models to create cancer-gene predictions. There were two key steps for our data processing part. We found out that the model performance increased when we combined correlated variables into a single predictor to eliminate multicollinearity. We also chose predictors that are statistically significant but not underfit by setting the p value to be 0.1. Multinomial logistics regression worked the best among our models. We further increased its performance by tuning the probability thresholds. It's also important to note that we compared model performances using 10-fold cross validation instead of the ROC-AUC curves. The reason is because we had little data to work with, especially of classes 1s or 2s in comparison to the 0s. If we could obtain more training data, we could also split it into training and validation sets and use ROC-AUC curves to check model performances.

Our model does have space for improvements. We could change the probability threshold for each class and possibly get better predictions. We could also remove the outliers for each predictor. To deal with the unequal numbers of 0s, 1s, and 2s in the class variable, we could try methods like SMOTE algorithm, penalized model, and decision trees. However, as our model achieved a score of 0.82606, we believe it is a useful model for predicting cancer genes.

Statement of Contributions

- Eustina Kim: Preprocessed data by adding standardized variables and subsetting based on p-value, figured out how to change the probability threshold for the multinomial logistic model, worked on writing the report.
- Tiffany Feng: Worked on creating potential models by subsetting predictors by significance, tried outlier removal from Eustina's model through building and analyzing interactive boxplots of all predictors, worked on writing the report, tried removing multicollinearity through analysis of correlation between variables.
- Jonathan Martinez: Worked on creating models using predictors thought to be good based on correlation and box plots. Tried to change prediction threshold values, but I could not figure it out, but eventually Eustina did. Worked on Midterm report.
- Kevin Chen: worked out midterm report, tried transforming data using log and square root, tried eliminating outliers using boxplots, researched on ROC-AUC curve for three classes of classification problems

Midterm Report

Team Chi Beta Gamma: Eustina Kim, Tiffany Feng, Jonathan Martinez, Kevin Chen

11/09/2020

```
library(dplyr)
library(readr)
library(caret)
library(nnet)
library(ggplot2)
```

```
train <- read.csv("training.csv")
test <- read.csv("test.csv")
```

```
#shows how many class 0,1,2 exist in the training data
num_class<-train %>% group_by(class) %>% summarise(counts=n())
```

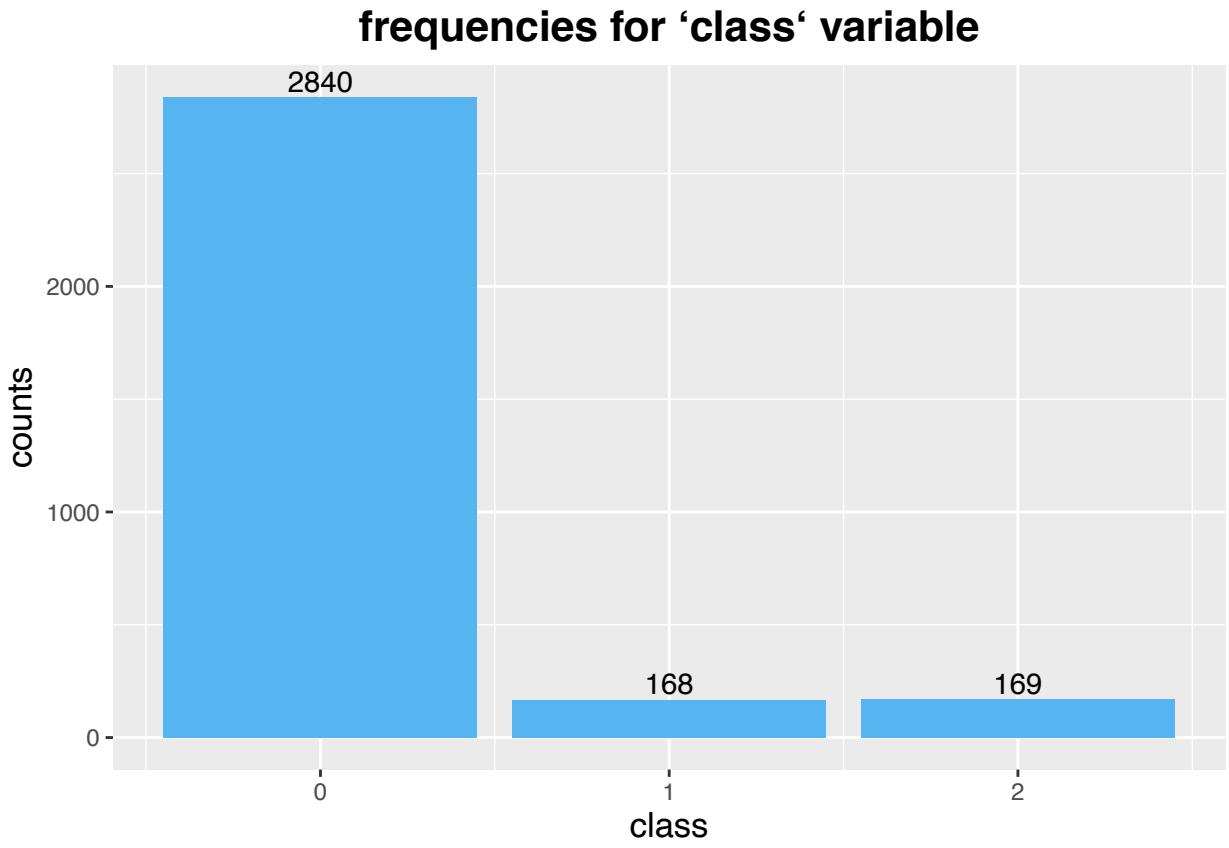
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
print(num_class)
```

```
## # A tibble: 3 x 2
##   class counts
##   <int>  <int>
## 1     0   2840
## 2     1    168
## 3     2    169
```

```
#Graph frequencies of each class
# png("class counts.png", units="px", width= 1920, height= 1080, res=300)

ggplot(num_class, aes(x = class, y = counts)) +
  geom_bar(stat="identity", fill="#56B5F0") +
  labs(title = "frequencies for `class` variable") + theme(axis.title = element_text(size = 13),
    plot.title = element_text(size = 16,
        face = "bold", hjust = 0.5)) +
  geom_text(aes(label=counts), position=position_dodge(width=0.9), vjust=-0.25, size=4)
```




```
# dev.off()
```

```
#Add standardized variables in threes
h1 <- train[,66:98]
h <-scale(h1, center = TRUE, scale = TRUE) #standardize
h <- as.data.frame(h)

sums <- data.frame(matrix(nrow=nrow(h),ncol =11))
j <-1
i <- 1
#combine variables
while(i < 33){
  sums[,j] <- rowSums(h[,i:(i+2)])
  j <- j+1
  i <- i+3
}
```

```
#fix column names
columns <- c("H3K4me3","H3K4me2","H3K4me1","H3K36me3","H3K27ac",
             "H3K27me3","H3K9me3","H3K9ac","H3K9me2","H3K79me2","H4K20me1")
names(sums) <- columns
```

```
#combine standardized columns to make new training data
train1 <- train[,2:65]
train1 <- train1 %>% mutate(sums) %>% mutate(class=train$class)
```

```
#do the same procedure for test data
h2 <- test[,66:98]
h_test <-scale(h2, center = TRUE, scale = TRUE) #standardize
```

```r
h_test <- as.data.frame(h_test)

sums_test <- data.frame(matrix(nrow=nrow(h_test),ncol =11))
j <-1
i <- 1
#combine variables
while(i < 33){
  sums_test[,j] <- rowSums(h_test[,i:(i+2)])
  j <- j+1
  i <- i+3
}

#fix column names
columns <- c("H3K4me3","H3K4me2","H3K4me1","H3K36me3","H3K27ac",
             "H3K27me3","H3K9me3","H3K9ac","H3K9me2","H3K79me2","H4K20me1")
names(sums_test) <- columns

#combine new data
test1 <- test[,2:65]
test1 <- test1 %>% mutate(sums_test)
```

```r
#run lm
mod1 <- lm(class~.,data=train1)
summary(mod1)
```

```
##
## Call:
## lm(formula = class ~ ., data = train1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04003 -0.15293 -0.02047  0.08426  1.99394
##
## Coefficients: (5 not defined because of singularities)
##                                            Estimate Std. Error t value
## (Intercept)                                7.383e+01  1.694e+02   0.436
## Silent_KB_Ratio                            1.977e-03  5.601e-04   3.529
## N_Missense                                 2.684e-02  1.583e-02   1.696
## N_LOF                                     -1.973e-02  1.239e-02  -1.592
## N_Splice                                   6.978e-02  1.217e-02   5.736
## Missense_KB_Ratio                         -2.906e-05  1.149e-05  -2.528
## LOF_KB_Ratio                               6.322e-04  5.189e-04   1.218
## Missense_Entropy                           9.405e-02  3.411e-02   2.757
## LOF_TO_Silent_Ratio                        5.531e-02  3.188e-02   1.735
## Splice_TO_Silent_Ratio                    -2.923e-02  4.713e-01  -0.062
## Missense_TO_Silent_Ratio                   2.021e-02  8.506e-03   2.376
## Missense_Damaging_TO_Missense_Benign_Ratio 5.191e-03  4.299e-03   1.207
## LOF_TO_Benign_Ratio                       -6.829e-04  6.748e-02  -0.010
## Splice_TO_Benign_Ratio                    -6.275e-01  9.803e-01  -0.640
## Missense_TO_Benign_Ratio                  -1.782e-02  1.394e-02  -1.278
## Missense_Damaging_TO_Benign_Ratio         -8.490e-03  1.679e-02  -0.506
## Polyphen2                                 -3.749e-02  2.314e-02  -1.620
## LOF_TO_Total_Ratio                         4.901e+00  8.059e-01   6.082
## Missense_TO_Total_Ratio                    1.018e+00  4.299e-01   2.369
```

```
## Splice_TO_Total_Ratio                      -3.848e+00  2.973e+00  -1.294
## LOF_TO_Missense_Ratio                      -2.685e-01  1.088e-01  -2.468
## Silent_fraction                            -7.501e-01  2.149e-01  -3.490
## Nonsense_fraction                          -4.467e+00  1.732e+00  -2.579
## Missense_fraction                          -1.003e+00  4.367e-01  -2.296
## Recurrent_missense_fraction                -3.581e-01  1.571e-01  -2.280
## Frameshift_indel_fraction                  -4.117e+00  1.758e+00  -2.341
## Inframe_indel_fraction                      2.241e-01  3.332e-01   0.672
## Lost_start_and_stop_fraction               -1.580e+00  1.791e+00  -0.882
## Inactivating_mutations_fraction             1.511e+00  1.732e+00   0.873
## NonSilent_TO_Silent_Ratio                  -1.373e-02  7.533e-03  -1.823
## log_gene_length                            -8.550e-03  4.444e-03  -1.924
## CDS_length                                 -8.162e-03  1.529e-02  -0.534
## CNA_deletion                               -7.372e+01  1.694e+02  -0.435
## CNA_amplification                          -7.378e+01  1.694e+02  -0.436
## Exon_Cons                                  -2.038e-02  3.610e-02  -0.564
## MGAentropy                                  3.716e-02  1.787e-02   2.080
## S50_score_replication_timing               -7.554e-03  3.234e-02  -0.234
## One_Minus_S50_score_replication_timing             NA         NA      NA
## VEST_score                                  3.692e-01  5.448e-02   6.776
## Gene_expression_Z_score                     2.856e-03  1.174e-02   0.243
## Gene_expression_Minus_Z_score                      NA         NA      NA
## Cell_proliferation_rate_CRISPR_KD          -1.476e-01  3.539e-02  -4.172
## Minus_Cell_proliferation_rate_CRISPR_KD            NA         NA      NA
## Super_Enhancer_percentage                   3.440e-01  8.317e-02   4.136
## BioGRID_betweenness                         9.031e+00  7.966e+00   1.134
## BioGRID_clossness                          -3.068e-01  5.924e-02  -5.179
## BioGRID_log_degree                          4.106e-02  4.370e-03   9.396
## Promoter_hypermethylation_in_cancer        -1.128e-02  1.867e-02  -0.604
## Promoter_hypomethylation_in_cancer                 NA         NA      NA
## Gene_body_hypermethylation_in_cancer        6.512e-02  2.492e-02   2.613
## Gene_body_hypomethylation_in_cancer                NA         NA      NA
## Canyon_genebody_hypermethylation            6.954e-03  3.168e-03   2.195
## intolerant_pLI                              9.451e-02  4.254e-02   2.222
## intolerant_pRec                            -1.453e-02  3.111e-02  -0.467
## intolerant_pNull                            2.929e-02  3.218e-02   0.910
## Synonymous_Zscore                          -1.580e-02  5.622e-03  -2.810
## Missense_Zscore                             6.681e-03  7.798e-03   0.857
## pLOF_Zscore                                 1.813e-02  8.666e-03   2.092
## dN_to_dS_ratio                              1.921e-02  1.409e-02   1.364
## GDI                                         4.080e-06  2.785e-06   1.465
## RVIS_percentile                            -1.371e-04  2.796e-04  -0.490
## ncRVIS                                     -6.371e-03  6.219e-03  -1.024
## ncGERP                                      2.244e-02  8.007e-03   2.803
## Gene_age                                    1.053e-03  2.690e-03   0.392
## FamilyMemberCount                           1.058e-05  1.132e-04   0.093
## H3K4me3                                     7.171e-03  6.008e-03   1.194
## H3K4me2                                     2.446e-03  7.013e-03   0.349
## H3K4me1                                    -2.013e-03  6.004e-03  -0.335
## H3K36me3                                    1.990e-02  3.478e-03   5.723
## H3K27ac                                     1.152e-02  5.412e-03   2.129
## H3K27me3                                   -2.074e-04  2.857e-03  -0.073
## H3K9me3                                    -1.263e-02  2.801e-03  -4.510
## H3K9ac                                      1.976e-03  7.861e-03   0.251
```

```
## H3K9me2                                      6.298e-03  3.031e-03   2.078
## H3K79me2                                     7.036e-04  5.007e-03   0.141
## H4K20me1                                     2.534e-02  4.616e-03   5.490
##                                             Pr(>|t|)
## (Intercept)                                 0.663003
## Silent_KB_Ratio                             0.000424 ***
## N_Missense                                  0.089937 .
## N_LOF                                       0.111531
## N_Splice                                    1.06e-08 ***
## Missense_KB_Ratio                           0.011518 *
## LOF_KB_Ratio                                0.223198
## Missense_Entropy                            0.005866 **
## LOF_TO_Silent_Ratio                         0.082882 .
## Splice_TO_Silent_Ratio                      0.950551
## Missense_TO_Silent_Ratio                    0.017547 *
## Missense_Damaging_TO_Missense_Benign_Ratio 0.227334
## LOF_TO_Benign_Ratio                         0.991926
## Splice_TO_Benign_Ratio                      0.522105
## Missense_TO_Benign_Ratio                    0.201300
## Missense_Damaging_TO_Benign_Ratio           0.613131
## Polyphen2                                   0.105313
## LOF_TO_Total_Ratio                          1.33e-09 ***
## Missense_TO_Total_Ratio                     0.017894 *
## Splice_TO_Total_Ratio                       0.195665
## LOF_TO_Missense_Ratio                       0.013647 *
## Silent_fraction                             0.000490 ***
## Nonsense_fraction                           0.009947 **
## Missense_fraction                           0.021751 *
## Recurrent_missense_fraction                 0.022674 *
## Frameshift_indel_fraction                   0.019280 *
## Inframe_indel_fraction                      0.501364
## Lost_start_and_stop_fraction                0.377736
## Inactivating_mutations_fraction             0.382893
## NonSilent_TO_Silent_Ratio                   0.068446 .
## log_gene_length                             0.054438 .
## CDS_length                                  0.593417
## CNA_deletion                                0.663455
## CNA_amplification                           0.663212
## Exon_Cons                                   0.572514
## MGAentropy                                  0.037602 *
## S50_score_replication_timing                0.815346
## One_Minus_S50_score_replication_timing            NA
## VEST_score                                  1.48e-11 ***
## Gene_expression_Z_score                     0.807810
## Gene_expression_Minus_Z_score                     NA
## Cell_proliferation_rate_CRISPR_KD           3.10e-05 ***
## Minus_Cell_proliferation_rate_CRISPR_KD           NA
## Super_Enhancer_percentage                   3.63e-05 ***
## BioGRID_betweenness                         0.257013
## BioGRID_clossness                           2.37e-07 ***
## BioGRID_log_degree                           < 2e-16 ***
## Promoter_hypermethylation_in_cancer         0.545861
## Promoter_hypomethylation_in_cancer                NA
## Gene_body_hypermethylation_in_cancer        0.009017 **
```

```
## Gene_body_hypomethylation_in_cancer             NA
## Canyon_genebody_hypermethylation          0.028241 *
## intolerant_pLI                            0.026385 *
## intolerant_pRec                           0.640469
## intolerant_pNull                          0.362763
## Synonymous_Zscore                         0.004981 **
## Missense_Zscore                           0.391673
## pLOF_Zscore                               0.036545 *
## dN_to_dS_ratio                            0.172737
## GDI                                       0.143060
## RVIS_percentile                           0.623878
## ncRVIS                                    0.305706
## ncGERP                                    0.005093 **
## Gene_age                                  0.695394
## FamilyMemberCount                         0.925535
## H3K4me3                                   0.232740
## H3K4me2                                   0.727220
## H3K4me1                                   0.737406
## H3K36me3                                  1.14e-08 ***
## H3K27ac                                   0.033307 *
## H3K27me3                                  0.942144
## H3K9me3                                   6.72e-06 ***
## H3K9ac                                    0.801583
## H3K9me2                                   0.037817 *
## H3K79me2                                  0.888254
## H4K20me1                                  4.35e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3355 on 3106 degrees of freedom
## Multiple R-squared:  0.542,  Adjusted R-squared:  0.5316
## F-statistic:  52.5 on 70 and 3106 DF,  p-value: < 2.2e-16
```

```
#get predictors with p-value less than 0.1
predictors<-summary(mod1)$coefficients[,4] <0.1
predictors <- names(predictors[predictors==TRUE])
```

```
#subset training and test data with chosen predictors
p_train <- train1 %>% select(class,all_of(predictors))
p_test <- test1 %>% select(all_of(predictors))
```

```
set.seed(1)
p_train$class <- as.factor(p_train$class)
#Log
plog_fit <- train(
  form = class ~.,
  data = p_train,
  trControl = trainControl(method = "cv", number = 10),
  method = "multinom"
)
```

```
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 867.346378
## iter  20 value 725.477853
```

```
## iter  30 value 600.515100
## iter  40 value 482.368918
## iter  50 value 388.081248
## iter  60 value 381.934738
## iter  70 value 380.943859
## iter  80 value 379.419342
## iter  90 value 378.301151
## iter 100 value 377.880555
## final  value 377.880555
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 867.399902
## iter  20 value 728.448300
## iter  30 value 626.395557
## iter  40 value 512.814976
## iter  50 value 428.700693
## iter  60 value 420.073849
## iter  70 value 417.892350
## iter  80 value 417.319771
## iter  90 value 416.039225
## iter 100 value 415.827002
## final  value 415.827002
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 867.346432
## iter  20 value 725.480900
## iter  30 value 600.521933
## iter  40 value 483.087034
## iter  50 value 388.170812
## iter  60 value 382.212224
## iter  70 value 381.152919
## iter  80 value 379.965330
## iter  90 value 378.991562
## iter 100 value 378.696392
## final  value 378.696392
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 999.415370
## iter  20 value 810.873158
## iter  30 value 663.042248
## iter  40 value 531.882376
## iter  50 value 388.196867
## iter  60 value 368.159913
## iter  70 value 366.840525
## iter  80 value 365.745112
## iter  90 value 364.559857
## iter 100 value 364.366053
## final  value 364.366053
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
```

```
## iter  10 value 999.494954
## iter  20 value 812.258427
## iter  30 value 667.404557
## iter  40 value 538.125635
## iter  50 value 424.727967
## iter  60 value 406.801567
## iter  70 value 406.498220
## iter  80 value 406.396879
## iter  90 value 406.333921
## iter 100 value 405.795112
## final  value 405.795112
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 999.415450
## iter  20 value 810.874558
## iter  30 value 663.056282
## iter  40 value 531.909576
## iter  50 value 388.265478
## iter  60 value 368.248557
## iter  70 value 366.967121
## iter  80 value 366.013436
## iter  90 value 365.106666
## iter 100 value 364.964149
## final  value 364.964149
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1256.922551
## iter  20 value 897.687295
## iter  30 value 682.858224
## iter  40 value 508.712352
## iter  50 value 393.095981
## iter  60 value 380.666648
## iter  70 value 379.730238
## iter  80 value 378.397449
## iter  90 value 377.424061
## iter 100 value 377.055138
## final  value 377.055138
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1256.951485
## iter  20 value 901.119961
## iter  30 value 671.277686
## iter  40 value 508.299444
## iter  50 value 431.734754
## iter  60 value 420.262008
## iter  70 value 418.943719
## iter  80 value 418.478158
## iter  90 value 417.696054
## iter 100 value 416.983806
## final  value 416.983806
## stopped after 100 iterations
```

```
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1256.922580
## iter  20 value 897.684477
## iter  30 value 682.946400
## iter  40 value 510.664935
## iter  50 value 392.385751
## iter  60 value 380.632386
## iter  70 value 379.540182
## iter  80 value 378.304505
## iter  90 value 377.411622
## iter 100 value 377.039472
## final  value 377.039472
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 982.189303
## iter  20 value 793.667373
## iter  30 value 621.138349
## iter  40 value 502.103873
## iter  50 value 400.308132
## iter  60 value 377.740920
## iter  70 value 376.772509
## iter  80 value 375.872510
## iter  90 value 375.030474
## iter 100 value 374.693555
## final  value 374.693555
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 982.291517
## iter  20 value 795.586620
## iter  30 value 636.387219
## iter  40 value 517.167817
## iter  50 value 436.391485
## iter  60 value 416.933706
## iter  70 value 415.644602
## iter  80 value 415.537387
## iter  90 value 415.473325
## iter 100 value 414.761843
## final  value 414.761843
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 982.189406
## iter  20 value 793.669465
## iter  30 value 621.147523
## iter  40 value 502.043333
## iter  50 value 400.359496
## iter  60 value 377.816518
## iter  70 value 376.874648
## iter  80 value 376.079171
## iter  90 value 375.369286
## iter 100 value 375.137322
```

```
## final  value 375.137322
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 945.821677
## iter  20 value 786.878662
## iter  30 value 643.651751
## iter  40 value 513.581589
## iter  50 value 398.786097
## iter  60 value 380.626753
## iter  70 value 379.528095
## iter  80 value 378.467462
## iter  90 value 376.977684
## iter 100 value 376.331064
## final  value 376.331064
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 945.872159
## iter  20 value 788.131170
## iter  30 value 650.310677
## iter  40 value 524.892631
## iter  50 value 429.967053
## iter  60 value 421.184407
## iter  70 value 418.059460
## iter  80 value 416.837566
## iter  90 value 416.507949
## iter 100 value 415.708479
## final  value 415.708479
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 945.821728
## iter  20 value 786.879926
## iter  30 value 643.658704
## iter  40 value 513.652723
## iter  50 value 398.990108
## iter  60 value 380.839775
## iter  70 value 379.629305
## iter  80 value 378.814772
## iter  90 value 377.614453
## iter 100 value 377.106381
## final  value 377.106381
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 987.492496
## iter  20 value 829.082312
## iter  30 value 650.009116
## iter  40 value 528.684054
## iter  50 value 407.376192
## iter  60 value 377.377467
## iter  70 value 375.309555
## iter  80 value 374.387583
```

```
## iter  90 value 373.038479
## iter 100 value 372.480797
## final  value 372.480797
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 987.557919
## iter  20 value 831.845443
## iter  30 value 661.365643
## iter  40 value 560.269889
## iter  50 value 436.332528
## iter  60 value 415.710738
## iter  70 value 413.223797
## iter  80 value 411.877404
## iter  90 value 411.575917
## iter 100 value 410.712652
## final  value 410.712652
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 987.492562
## iter  20 value 829.085127
## iter  30 value 650.020778
## iter  40 value 528.323878
## iter  50 value 399.377203
## iter  60 value 377.164364
## iter  70 value 375.250867
## iter  80 value 374.448261
## iter  90 value 373.284974
## iter 100 value 372.807094
## final  value 372.807094
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1404.138490
## iter  20 value 1236.862488
## iter  30 value 867.513733
## iter  40 value 531.053650
## iter  50 value 387.081677
## iter  60 value 360.778351
## iter  70 value 358.983139
## iter  80 value 358.494137
## iter  90 value 357.728258
## iter 100 value 356.520927
## final  value 356.520927
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1404.209288
## iter  20 value 1237.945633
## iter  30 value 896.655588
## iter  40 value 625.128868
## iter  50 value 425.943368
## iter  60 value 398.000737
```

```
## iter  70 value 395.945278
## iter  80 value 395.839374
## iter  90 value 395.812454
## final  value 395.812213
## converged
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 1404.138560
## iter  20 value 1236.863577
## iter  30 value 867.543988
## iter  40 value 531.017043
## iter  50 value 391.928984
## iter  60 value 361.705728
## iter  70 value 359.214540
## iter  80 value 358.707605
## iter  90 value 358.113749
## iter 100 value 357.181291
## final  value 357.181291
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 1283.548895
## iter  20 value 1018.621326
## iter  30 value 725.318841
## iter  40 value 532.228937
## iter  50 value 400.221734
## iter  60 value 380.979961
## iter  70 value 379.565542
## iter  80 value 378.387748
## iter  90 value 376.899837
## iter 100 value 376.236976
## final  value 376.236976
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 1283.586685
## iter  20 value 1027.870672
## iter  30 value 727.966165
## iter  40 value 540.072032
## iter  50 value 434.846492
## iter  60 value 420.843086
## iter  70 value 418.240017
## iter  80 value 417.790723
## iter  90 value 417.783973
## iter 100 value 417.783175
## final  value 417.783175
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 1283.548933
## iter  20 value 1018.631037
## iter  30 value 725.316477
## iter  40 value 532.395955
## iter  50 value 400.526537
```

```
## iter  60 value 381.010856
## iter  70 value 379.549694
## iter  80 value 378.326537
## iter  90 value 377.240928
## iter 100 value 376.834722
## final  value 376.834722
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 944.540654
## iter  20 value 779.396503
## iter  30 value 663.783933
## iter  40 value 503.563817
## iter  50 value 396.048163
## iter  60 value 378.716903
## iter  70 value 377.526365
## iter  80 value 376.301679
## iter  90 value 375.262765
## iter 100 value 374.807055
## final  value 374.807055
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 944.588249
## iter  20 value 781.423143
## iter  30 value 663.287790
## iter  40 value 564.283886
## iter  50 value 431.842382
## iter  60 value 419.903945
## iter  70 value 416.713611
## iter  80 value 415.724959
## iter  90 value 414.131286
## iter 100 value 413.443444
## final  value 413.443444
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3142.031146
## iter  10 value 944.540702
## iter  20 value 779.398558
## iter  30 value 663.790595
## iter  40 value 503.573466
## iter  50 value 396.069364
## iter  60 value 378.808984
## iter  70 value 377.639859
## iter  80 value 376.542805
## iter  90 value 375.696117
## iter 100 value 375.347420
## final  value 375.347420
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 976.018386
## iter  20 value 806.811600
## iter  30 value 662.444418
```

```
## iter  40 value 536.300248
## iter  50 value 405.843614
## iter  60 value 382.461519
## iter  70 value 380.632535
## iter  80 value 379.880768
## iter  90 value 378.581631
## iter 100 value 378.057024
## final  value 378.057024
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 976.080087
## iter  20 value 809.066635
## iter  30 value 672.002172
## iter  40 value 544.489512
## iter  50 value 442.989549
## iter  60 value 419.390437
## iter  70 value 415.169667
## iter  80 value 414.293142
## iter  90 value 413.380034
## iter 100 value 413.043182
## final  value 413.043182
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3140.932533
## iter  10 value 976.018448
## iter  20 value 806.813886
## iter  30 value 662.464491
## iter  40 value 535.789465
## iter  50 value 405.901469
## iter  60 value 382.535095
## iter  70 value 380.735796
## iter  80 value 379.859427
## iter  90 value 378.738693
## iter 100 value 378.305812
## final  value 378.305812
## stopped after 100 iterations
## # weights:  108 (70 variable)
## initial  value 3490.291241
## iter  10 value 900.801847
## iter  20 value 769.714616
## iter  30 value 609.104460
## iter  40 value 521.144843
## iter  50 value 474.258761
## iter  60 value 459.094795
## iter  70 value 457.558443
## iter  80 value 457.557129
## iter  80 value 457.557125
## iter  80 value 457.557125
## final  value 457.557125
## converged
```

```
print(plog_fit)
```

```
## Penalized Multinomial Regression
```

```
##
## 3177 samples
##   34 predictor
##    3 classes: '0', '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2859, 2859, 2859, 2859, 2860, 2859, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.9509047  0.7260421
##   1e-04  0.9509047  0.7262329
##   1e-01  0.9524760  0.7302902
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

```
set.seed(1)
#LDA
lda_fit <- train(
  form = class ~.,
  data = p_train,
  trControl = trainControl(method = "cv", number = 10),
  method = "lda"
)
print(lda_fit)
```

```
## Linear Discriminant Analysis
##
## 3177 samples
##   34 predictor
##    3 classes: '0', '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2859, 2859, 2859, 2859, 2860, 2859, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9499554  0.7096982
```

```
set.seed(1)
#KNN
knn_fit <- train(
  form = class ~.,
  data = p_train,
  trControl = trainControl(method = "cv", number = 5),
  method = "knn",
  preProcess = c("center","scale"),
  tuneGrid = expand.grid(k = seq(1, 20, by = 1))
)
print(knn_fit)
```

```
## k-Nearest Neighbors
```

```
##
## 3177 samples
##   34 predictor
##    3 classes: '0', '1', '2'
##
## Pre-processing: centered (34), scaled (34)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2541, 2542, 2542, 2541, 2542
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9172213  0.5330825
##    2  0.9222557  0.5519454
##    3  0.9339018  0.5820757
##    4  0.9332724  0.5734763
##    5  0.9361061  0.5859969
##    6  0.9335884  0.5646671
##    7  0.9339028  0.5623733
##    8  0.9332739  0.5555526
##    9  0.9310697  0.5297371
##   10  0.9310697  0.5300189
##   11  0.9317006  0.5253331
##   12  0.9310717  0.5219025
##   13  0.9301263  0.5154947
##   14  0.9310712  0.5246848
##   15  0.9313861  0.5237920
##   16  0.9298128  0.5091621
##   17  0.9294969  0.5023939
##   18  0.9279235  0.4895978
##   19  0.9285535  0.4929467
##   20  0.9266642  0.4748575
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
#predict probabilities on test data
log_pred<-predict(plog_fit, p_test, "prob")
```

```
#make a function to predict class based on probabilities
pred_class <- function(prob,threshold){
  preds = rep(NA, nrow(prob)) # vector to hold predictions

  for(i in 1:nrow(prob)){
    if(prob[i,1] >= threshold){ #classify as 0
      preds[i] = 0
    }
    else if(prob[i,2] > prob[i,3]){ #classify as 1 if prob bigger than prob of 2
      preds[i] = 1
    }
    else{
      preds[i] = 2
    }
  }
  return(preds)
}
```

```r
#set threshold as the mean of the probabilities of predicting 0.
prediction <- pred_class(log_pred,mean(log_pred[,1]))
```

```r
final <- data.frame("id" = test$id, "class" = prediction)
```

```r
# write our .csv file
write.csv(final, "threshold2.csv", row.names = FALSE)
```