Final Project Document

Duyen Ho, Michael Garbus, Christina Hu

12/6/2021

Contents

Project Description and Summary (NEED GOAL, APPROACH, and CONCLUSION AND THIS NEEDS TO BE ONE PAGE?!!?)

The goal of this project is to find the most accurate model using a variety of statistical model building techniques in order to predict medical outcomes relating to breast cancer. We will use the BRCA Multi-Omics (TCGA) data from Kaggle, and predict outcomes for the `PR.Status`, `ER.Status`, `HER2.Final.Status`, and `histological.type`. For our approach, we will first perform the necessary data cleaning operations, then [method 1] and [method2] to build classification models for `PR.Status` and [method1] and [method2] for `histological.type`, using classification error and AUC respectively as the evalution criteria. Next, we will use [method] to build a model with the goal of accurately predicting **all four** outcomes. Finally, [INSERT CONCLUSION HERE].

- can include any difficulties or describe outcomes for models

Literature Review

We consulted some research involving the subject to guide us in our modelling. The Wisconsin Breast Cancer dataset was used, which involves a digitized image of a fine needle aspirate (FNA) of a breast mass, which describe the characteristics of the cell nuclei present in the image. Unfortunately, this is rather different from our current data, which contains a sample of genetic data instead of a picture of cell data. However, we still feel that their approaches can be useful for our data, as they are predicting binary outcomes on a similar subject matter.

The first study considered was "Breast Cancer Prediction: A Comparative Study Using Machine Learning Techniques", published in 2020 by Islam, M.M. et al. in SN Computer Science. The paper compares 5 different supervised machine learning techniques (support vector machine, K-nearest neighbors, random forests, artificial neural networks, and finally, logistic regression.) They performed little preprocessing on the data, as they only removed 16 NA variables. They divided the data into test and train sets, and used a 10-fold cross validation, with nine-fold used for training and the remaining fold used for testing. The results revealed that ANNs return the highest accuracy (98.57%), KNN and SVM came second (97.1%), and random forests and logistic regression came third, clocking in with an accuracy of 95.71% for the two models. ANNs also achieved the highest specificity (96%), whereas SVM achieved the lowest, at around 92.3%. KNN and random forest had a specificity of 98.53%, whereas logistic regression had a specificity of 95.65%. ANN and SVM achieved the highest sensitivity, clocking in at exactly 100%. This made SVM and ANN very attractive to us when considering our model selection. KNN had a sensitivity of 97.82%, Random Forest had a sensitivity of 95.65%, and logistic regression had a sensitivity of 95.74%.

However, the highest values for AUC were KNN and a random forest model, both achieving an AUC of 99%. The technology that they used for this study was Python's scikit-learn.

In the discussion of related works, the authors mentioned another study which received a 98.83 accuracy for a boosted trees random forest model, which inspired us to test gradient boosted trees for the final part of the project, as they will be able to determine which features are important, and hopefully cut down on features as well.

A similar study, "Using Machine Learning Algorithms for Breast Cancer Risk Prediction and Diagnosis", published in 2016 by Asri, et al in Procedia Computer Science 83 tested SVM, decision tree (C 4.5, a technique avaliable in the open source WEKA data analysis tool), Naive Bayes, as well as KNN on the same UCI Breast Cancer dataset. Asri et al. discovered that although SVM took the longest time to create a model, it resulted in the highest accuracy, which is a different result to the previous model we studied.

While we do not believe that time constraints will be important in regards to our model as we are trying to achieve a high level of classification, it is interesting that these values are different. We imagine that this may be due to a difference in tools used, or perhaps, in the kernel used for the SVM classification. Additionally, the study also found that SVM had the highest AUC value on the ROC curve. For these reason, we decided to use SVM for our model.

Summary Statistics and Data Processing

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brc
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))
# head(brca_data, 10)
```

- 705 observations (breast cancer samples)
- 1936 variables/predictors (4 different omics data types)

#[1:604] are rn - gene expressions 604 <- categorical data

#[605:1464] are cn - copy number variations 860

#[1465:1713] are mu - somatic mutations 249 <- categorical data

#[1714:1936] are pp - protein expression 223

- 4 outcomes [1937:1940]
- There was no missing value in the entire dataset, so there's no missing values in the continuous predictors

Make sure brca_org is the original data but without 'vital.status'

```
#sum(colSums(is.na(brca_org)))
```

- There were some outliers in the continuous predictors, I just did a quick test on the cn variables. But we decided to keep them as they are and filter out the dataset later on with the correlation matrix

```
remove_outliers <- function(x, na.rm = TRUE, ...) {
    qnt <- quantile(x, probs=c(.25, .75), na.rm = na.rm, ...)
    H <- 1.5 * IQR(x, na.rm = na.rm)
    y <- x
    y[x < (qnt[1] - H)] <- NA
    y[x > (qnt[2] + H)] <- NA
    y
}
cont1 <- as.numeric(unlist(brca[605:1464]))
sum(remove_outliers(cont1))
```

- My group wants to focus on filtering out the unsignificant continuous variables first, so we used correlation matrix method. Then we drop the columns that are highly correlated.

#Use correlation matrix

```
cor_mat = cor(non_categorical_brca)
# returns vector of indices to remove
cor_list = findCorrelation(cor_mat, cutoff = .7)
non_categorical_brca =  non_categorical_brca[, -c(cor_list)]
library(vtable)
st(non_categorical_brca)
```

““““““““““““““““““

```
suppressMessages(library(caret))
suppressMessages(library(dplyr))
# store the predictors
predictors = brca_data %>% select(c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))
# remove the categorical var temporarily
brca_data = brca_data %>% select(-c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))


# use correlation matrix
#cor_mat = cor(brca_data)
```

```
# only use variances using the corr mat
non_categorical_brca = cbind(brca_data[1:604], brca_data[1714:1936])

# use correlation matrix
cor_mat = cor(non_categorical_brca)

# returns vector of indices to remove
cor_list = findCorrelation(cor_mat, cutoff = .7)
```

```
non_categorical_brca =  non_categorical_brca[, -c(cor_list)]
```

Modeling PR.Status

Modeling with KNN (predict PR.Status)

First, we will use KNN model to predict `PR.Status`. We will use a test train split of 25/75. We will only be using the non-categorical variables to build the KNN model since we ran into a lot of errors trying to incorporate the categorical variables

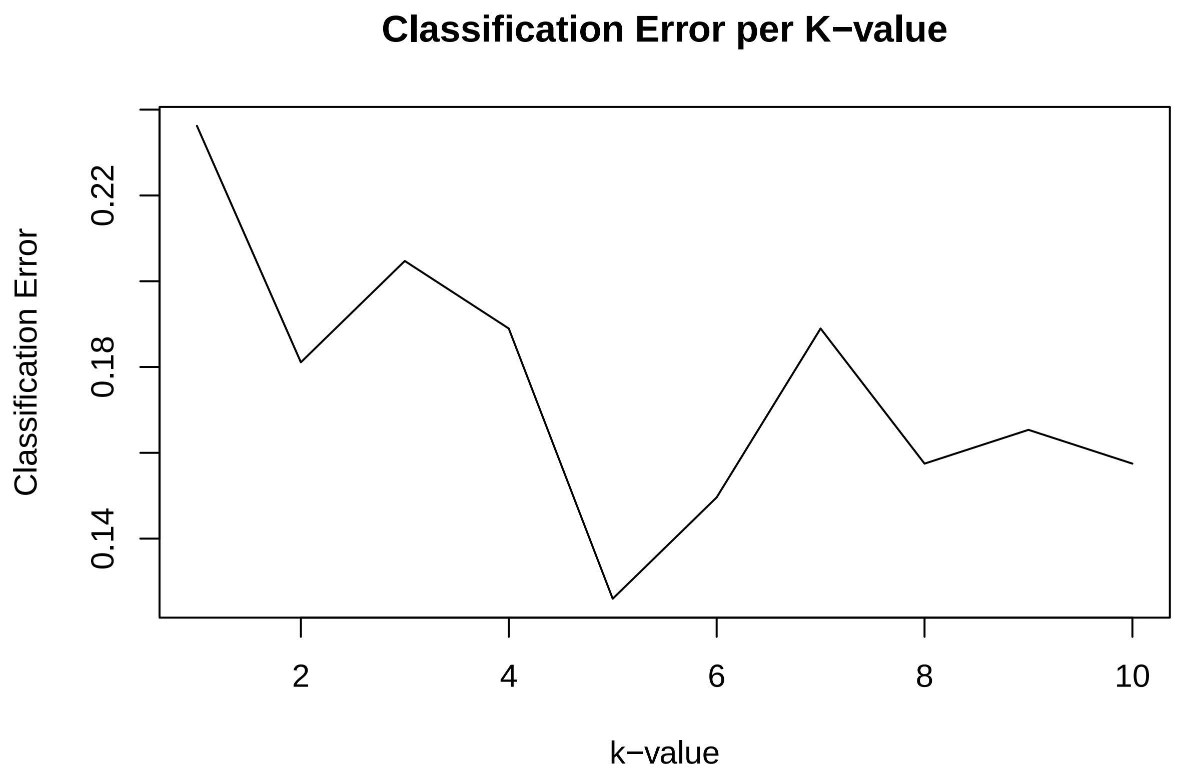
We will find the optimal k-value to use, ranging from k=1-10:

```
library(caret)
library(class)

# set the seed to make partition reproducible
set.seed(432)
```

```r
k_vals = c(1:10)
err_vals = rep(NA, length(k_vals))

## 75% of the sample size
smp_size <- floor(0.75 * nrow(non_categorical_brca))

for(i in 1:length(k_vals)) {

  train_ind <- sample(seq_len(nrow(non_categorical_brca)), size = smp_size)

  brca_train <- as.matrix(non_categorical_brca[train_ind, ])
  y = as.matrix(predictors$PR.Status[train_ind])

  brca_test <- as.matrix(non_categorical_brca[-train_ind, ])

  pred = knn(train = brca_train, test = brca_test, cl = y, k = i)
  cmat = table(pred, as.factor(predictors$PR.Status[-train_ind]))
  # calculate misclassification rate
  err_vals[i] = (cmat[1,2]+cmat[2,1])/sum(cmat)
}
```

Next, we will plot the graph of the classification errors for each k-value and see if we can use the result to determine the best k:

```r
plot(k_vals, err_vals, type = "l", xlab = "k-value", ylab = "Classification Error")
title("Classification Error per K-value")
```



As shown from the plot, the k-value with the lowest classification error is k=5, so we will use that for our model.

Next, use K=5 to create our KNN model and display the confusion matrix:

```
set.seed(432)
train_y = as.matrix(predictors$PR.Status[train_ind])
y = as.matrix(predictors$PR.Status[train_ind])
pred = knn(train = brca_train, test = brca_test, cl = y, k = 5)
cmat = table(pred, as.factor(predictors$PR.Status[-train_ind]))
cmat
```

```
##
## pred       Negative Positive
##   Negative       31        5
##   Positive       17       74
```

Displaying the accuracy:

```
(cmat[1,1]+cmat[2,2])/sum(cmat)
```

```
## [1] 0.8267717
```

The classificaton error for a model with k=5 is 0.1732283.

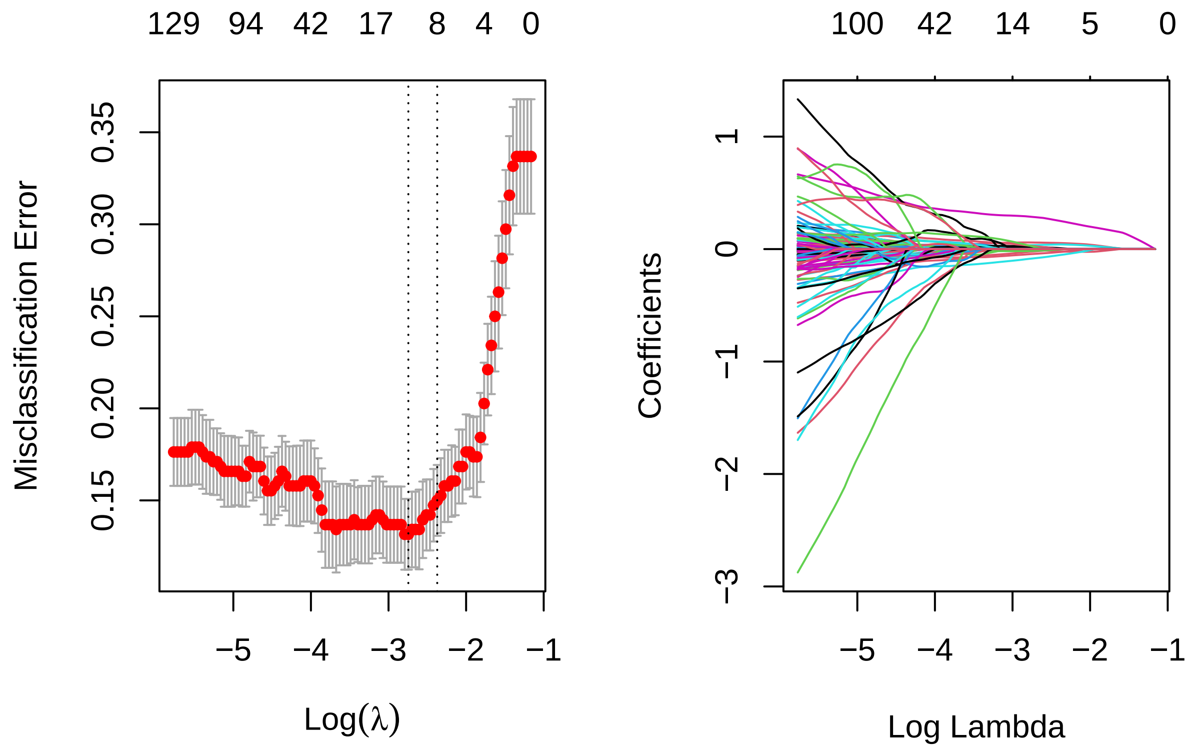
Modeling with Lasso (predict PR.Status)

Next, we will create a Lasso model with 10-fold cross-validation to help create `PR.Status`:

```
suppressMessages(library(glmnet))
set.seed(432)

lasso.fit = cv.glmnet(brca_train, train_y, nfolds = 10, alpha = 1, type.measure = "class", family = "bi
```

Plotting the fit of our Lasso model:

```
par(mfrow = c(1, 2))
    plot(lasso.fit)
    plot(lasso.fit$glmnet.fit, "lambda")
```



We will use the `best_lambda` to use for the prediction on the test data:

```
best_lambda = lasso.fit$lambda.min
test_predict = predict(lasso.fit, brca_test, s = best_lambda, type = "class")
```

Displaying the confusion matrix of the results:

```
cmat_1 = table(predictors$PR.Status[-train_ind], test_predict)
cmat_1
```

```
##           test_predict
##            Negative Positive
##   Negative       32       16
##   Positive        3       76
```

Displaying the accuracy:

```
(cmat_1[1,1]+cmat_1[2,2])/sum(cmat_1)
```

```
## [1] 0.8503937
```

The classification error is 0.1496063

`PR.Status` Summary:

To summarize the model predictions for `PR.Status`, we used KNN and Lasso models to predict the outcome. We only used non-categorical data for both instances since the categorical variables gave many errors while we were in the process of fitting the models. However, even without the categorical predictors, both models yielded relatively low classification errors, with the KNN model (using k=8) having a classification error of 0.1732283 and the Lasso model having a classification error of 0.1496063. Looking at these results, the Lasso model would be better suited to predict `PR.Status`.

Modelling `histological.type`

Next, let us model `histological.type`, using AUC as the evaluation criterion. For both models that we chose to model `histological.type` with, we decided to use ALL of the categorical and non-categorical predictors to see if we could get a more accurate result.

Modelling `histological.type` with SVM

```
# SVM, histological type
library(ROCR)
library(e1071)
library(caret)

brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brca
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))
non_response_brca <- brca_data[,-c(which(colnames(brca_data) %in% c('PR.Status','ER.Status','HER2.Final

set.seed(432)

#SVM WITH MATCORR DATA

# use correlation matrix
cor_mat = cor(non_response_brca)

# returns vector of indices to remove
cor_list = findCorrelation(cor_mat, cutoff = .7)
```

```r
non_response_brca <- non_response_brca[,cor_list]

#New BRCA data

train = brca_data$histological.type
train_data_unsplit = cbind(train,non_response_brca)
#continue to use the combined 10 data? or use the full data

#train_data_unsplit[,'train']  1 and 2
#80 20 split
train_rows <- sample(nrow(train_data_unsplit), size = floor(0.8 *nrow(train_data_unsplit)))
train_data <- train_data_unsplit[train_rows,]
test_data = train_data_unsplit[-train_rows,]

#Let's consider a few different kernels
#Sigma is the distance from the boundary
#Do we want a high cost? Or a low one?

svm.radial <- train(train ~ ., data = train_data, method = "svmRadial",
                preProcess = c("center", "scale"),
                tuneGrid = expand.grid(C = c(0.01, 0.1, 0.5, 1), sigma = c(1, 2, 3)),
                trControl = trainControl(method = "cv", number = 5))


svm.linear <- train(train ~ ., data = train_data, method = 'svmLinear',
                preProcess = c("center", "scale"),
                tuneGrid = expand.grid(C = c(0.01, 0.1, 0.5, 1)),
                trControl = trainControl(method = "cv", number = 5))
```

Let us display the SVM results for radial and linear models respectively:

```r
svm.radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 405 samples
## 979 predictors
##   2 classes: 'infiltrating ductal carcinoma', 'infiltrating lobular carcinoma'
##
## Pre-processing: centered (979), scaled (979)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 324, 325, 324, 323, 324
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##   0.01  1      0.9037285  0
##   0.01  2      0.9037285  0
##   0.01  3      0.9037285  0
##   0.10  1      0.9037285  0
##   0.10  2      0.9037285  0
##   0.10  3      0.9037285  0
##   0.50  1      0.9037285  0
##   0.50  2      0.9037285  0
##   0.50  3      0.9037285  0
##   1.00  1      0.9037285  0
##   1.00  2      0.9037285  0
##   1.00  3      0.9037285  0
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 3 and C = 0.01.
```

```
svm.linear
```

```
## Support Vector Machines with Linear Kernel
##
## 405 samples
## 979 predictors
##   2 classes: 'infiltrating ductal carcinoma', 'infiltrating lobular carcinoma'
##
## Pre-processing: centered (979), scaled (979)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 325, 323, 324, 324, 324
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.01  0.9234831  0.5484948
##   0.10  0.8840334  0.3695863
##   0.50  0.8840334  0.3695863
##   1.00  0.8840334  0.3695863
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

Next, we will use each model to predict on the test data

```
suppressMessages(library(kableExtra))
linear_svm_prediction <- predict(svm.linear, test_data[,-1])

radial_svm_prediction <- predict(svm.radial, test_data[,-1])

#Radial --> unable to classify any infiltrating lobular carcinoma. Linear is the better model
suppressMessages(library(ROCR))

lin.svm <- prediction(as.numeric(as.factor(linear_svm_prediction)), as.factor(test_data[,1]))

rad.svm <- prediction(as.numeric(as.factor(radial_svm_prediction)), as.factor(test_data[,1]))
```

Finally, let us display the results of the Linear and Radial SVM AUC:

```
# Linear has a performance of about 65.98%
performance(lin.svm,measure = "auc")@y.values[[1]]
```

```
## [1] 0.6598402
```

```
# Radial has performance of about 50%
performance(rad.svm,measure = "auc")@y.values[[1]]
```

```
## [1] 0.5
```

Looking at the results above, Linear SVM has AUC = 0.6598402 and Radial SVM has AUC = rperformance(rad.svm,measure = "auc")@y.values [[1]]‘. We can conclude that Linear SVM is probably the better model due to this significant difference.

Modelling `histological.type` with Random Forests

Next, let us use a Random Forest model:

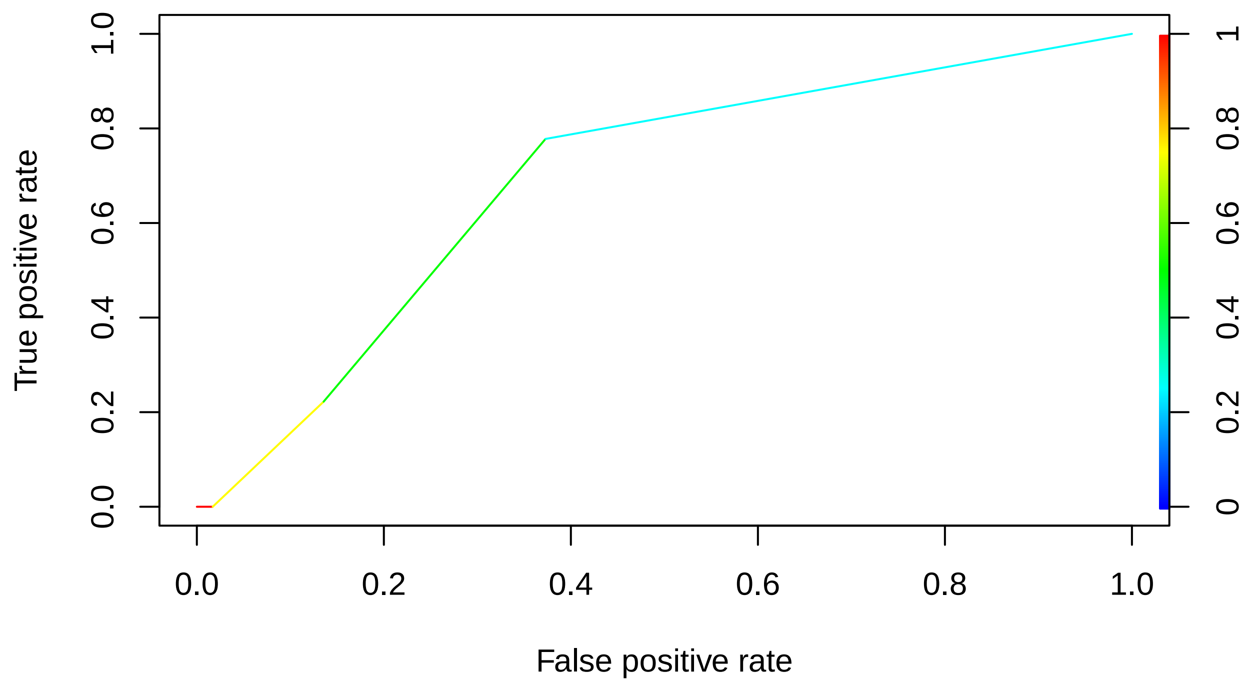
```
train_y_hist= as.matrix(predictors$histological.type[train_ind])
brca_train_2 = as.matrix(cbind(brca_train, train_y_hist))
```

Let us fit the model with `nodesize=8` and `sampsize=50`. Through multiple rounds of experimentation with values, we found those parameters to be most optimal:

```
suppressMessages(library(randomForest))
set.seed(432)
rf.fit = randomForest(data.matrix(brca_train_2[, -c(640)]), y = as.factor(brca_train_2[, 640]), ntree =
```

Next, use this model to predict on the test data:

```
test.predictions <- predict(rf.fit, type="prob", newdata=brca_test)[,2]

correct_hist = as.factor(predictors$histological.type[-train_ind])
rf_pr_test <- prediction(test.predictions, correct_hist)
r_auc_test <- performance(rf_pr_test, "tpr","fpr")
plot(r_auc_test, colorize=TRUE)
```



The AUC:

```r
performance(rf_pr_test, measure = "auc")@y.values[[1]]
```

```
## [1] 0.6892655
```

The 0.6892655.

`histological.type` Summary:

To summarize our models, we used Linear and Radial SVM, with had AUC values of 0.6598402 and 0.5 respectively. For random forest, the AUC value resulted in 0.6892655. Therefore, the random forest model can be determined as the better model, although its AUC score is still a bit low.

50-Variable Selection

PR.Status

```r
library(dplyr)
brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brca
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))
```

```
# store the outcomes
outcomes = brca_data %>% select(c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))
# remove the categorical var temporarily
brca_data = brca_data %>% select(-c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type")

# Create a correlation matrix and select only variables with some correlation to the target
PR.Status = as.numeric(as.factor(outcomes$PR.Status))-1
pr_data = cbind(PR.Status, brca_data)

pr_data[sapply(pr_data, is.character)] <- lapply(pr_data[sapply(pr_data, is.character)], as.numeric)


# head(pr_data)

M <- cor(pr_data)
correlation_threshold <- 0.446
significant_correlation <- abs(M["PR.Status", ]) > correlation_threshold
table(significant_correlation)
```

```
## significant_correlation
## FALSE  TRUE
##  1886    51
```

```
corr_colnames <- colnames(M)[significant_correlation]
```

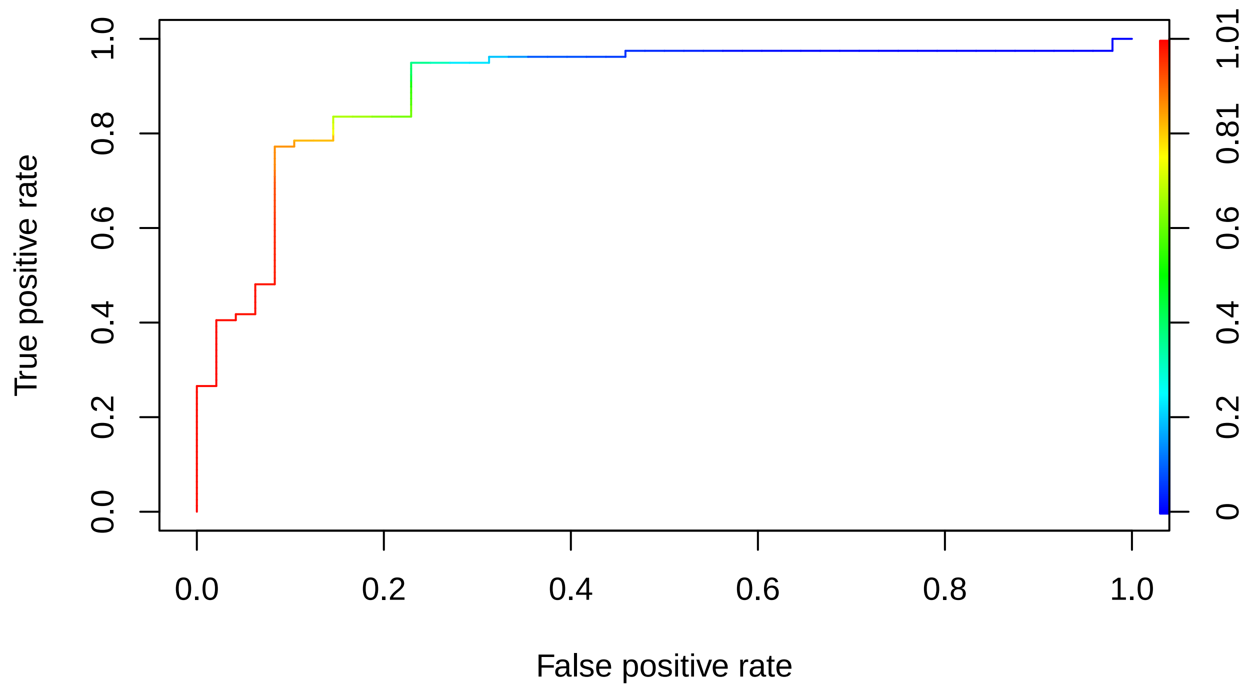
```
suppressMessages(library(caret))
# Linear regression model with selected features, 10 K-fold cross validation using the "caret" package
set.seed(432)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(pr_data))

## set the seed to make your partition reproducible
train_ind <- sample(seq_len(nrow(pr_data)), size = smp_size)

train_y <- outcomes$PR.Status[train_ind]
pr_train <- cbind(train_y, pr_data[train_ind, ])
pr_train <- pr_train[, corr_colnames]

# Training parameters
ctrl <- trainControl(method = "cv",
                     number = 5,
                     savePredictions = TRUE,
                     classProbs = FALSE)

# Fitting model
mod_fit <- train(as.factor(PR.Status) ~ ., data=pr_train,
                 method="glm",
                 family="binomial",
                 trControl = ctrl,
                 tuneLength = 5)
```

```
summary(mod_fit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2824  -0.1375   0.0672   0.2986   2.3869
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.239e+00  3.857e+00   0.581 0.561543
## rs_TFF1      -1.483e-02  1.374e-01  -0.108 0.914037
## rs_CYP2B7P1   2.265e-01  9.449e-02   2.397 0.016525 *
## rs_AGR3       1.365e-01  1.512e-01   0.903 0.366535
## rs_C1orf64   -3.771e-01  1.530e-01  -2.466 0.013673 *
## rs_TFF3       5.694e-02  1.757e-01   0.324 0.745846
## rs_AGR2       1.182e-01  1.681e-01   0.703 0.481818
## rs_GFRA1      1.312e-01  1.096e-01   1.198 0.231024
## rs_PGR        6.742e-01  1.966e-01   3.430 0.000604 ***
## rs_VGLL1     -7.153e-02  1.480e-01  -0.483 0.628819
## rs_SERPINA11  2.004e-02  1.278e-01   0.157 0.875348
## rs_A2ML1     -2.156e-01  1.068e-01  -2.019 0.043456 *
## rs_PTPRT      1.076e-01  1.173e-01   0.917 0.359118
## rs_ESR1      -6.531e-01  2.918e-01  -2.238 0.025204 *
## rs_SYT9      -8.222e-03  1.070e-01  -0.077 0.938773
## rs_GRPR       2.118e-01  1.363e-01   1.554 0.120260
## rs_CHAD      -6.777e-02  1.121e-01  -0.604 0.545638
## rs_ABCC8     -2.369e-01  1.317e-01  -1.799 0.072056 .
## rs_PGLYRP2    9.117e-02  1.115e-01   0.817 0.413644
## rs_SCUBE2    -1.256e-01  1.867e-01  -0.673 0.501028
## rs_NEK10     -6.247e-02  1.636e-01  -0.382 0.702546
## rs_ERBB4     -1.088e-01  1.596e-01  -0.682 0.495434
## rs_FSIP1      4.746e-02  1.781e-01   0.266 0.789873
## rs_FOXA1     -2.325e-01  2.695e-01  -0.863 0.388207
## rs_NAT1       2.792e-01  1.454e-01   1.920 0.054814 .
## rs_SLC7A2    -4.151e-01  1.519e-01  -2.733 0.006271 **
## rs_FLJ45983   2.114e-01  1.541e-01   1.372 0.170134
## rs_ART3       1.355e-01  1.470e-01   0.922 0.356721
## rs_PPP1R14C  -1.188e-01  1.730e-01  -0.687 0.492117
## rs_RGS22     -5.212e-03  1.463e-01  -0.036 0.971592
## rs_C1orf173  -1.565e-01  1.450e-01  -1.079 0.280442
## rs_RBM24     -4.967e-02  1.347e-01  -0.369 0.712331
## rs_GRIK3      1.108e-01  1.449e-01   0.765 0.444500
## rs_TPRG1     -2.989e-01  1.600e-01  -1.868 0.061736 .
## rs_AFF3       5.398e-01  2.279e-01   2.368 0.017871 *
## rs_TUBA3E     2.454e-01  1.668e-01   1.472 0.141087
## rs_SBSN      -3.018e-01  1.493e-01  -2.022 0.043177 *
## rs_SOX11     -8.875e-02  1.319e-01  -0.673 0.501136
## rs_TTC36      6.829e-02  1.464e-01   0.466 0.640947
## rs_GABBR2     4.582e-02  1.453e-01   0.315 0.752504
## rs_DEGS2      1.801e-02  2.165e-01   0.083 0.933714
## rs_ADAMTS15   3.231e-01  1.622e-01   1.992 0.046333 *
## rs_CLSTN2    -2.241e-01  1.943e-01  -1.153 0.248769
## rs_ACE2      -3.671e-02  1.324e-01  -0.277 0.781555
## rs_AMY1A      4.939e-03  1.499e-01   0.033 0.973707
## cn_HAPLN1    -1.302e+01  1.385e+03  -0.009 0.992501
## cn_EDIL3      1.368e+01  1.385e+03   0.010 0.992123
## pp_ASNS      -2.633e-01  5.098e-01  -0.516 0.605516
## pp_ER.alpha   5.648e-01  3.039e-01   1.859 0.063096 .
## pp_GATA3     -1.914e-01  3.922e-01  -0.488 0.625517
## pp_PR         1.195e-01  2.478e-01   0.482 0.629684
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 485.58  on 379  degrees of freedom
## Residual deviance: 169.42  on 329  degrees of freedom
## AIC: 271.42
##
## Number of Fisher Scoring iterations: 15
```

Compute and plot the AUC:

```
test_y <- outcomes$PR.Status[-train_ind]
pr_test <- cbind(test_y, pr_data[-train_ind, ])

# length(test_y)
# dim(pr_data[-train_ind,])

test_predict <- predict(mod_fit, pr_test, type="prob")
roc <- prediction(test_predict[,2], test_y)

# calculates the ROC curve
perf <- performance(roc,"tpr","fpr")
plot(perf,colorize=TRUE)
```



```
# obtaining the AUC
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8989979
```

The AUC for PR.Status is 0.8989979

Histological.Type

```
library(dplyr)
brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brca
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))

# store the outcomes
outcomes = brca_data %>% select(c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))
# remove the categorical var temporarily
brca_data = brca_data %>% select(-c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type")

# Create a correlation matrix and select only variables with some correlation to the target
histological.type = as.numeric(as.factor(outcomes$histological.type))

hist_data = cbind(histological.type, brca_data)
```

```r
# head(hist_data,20)

hist_data[sapply(hist_data, is.character)] <- lapply(hist_data[sapply(hist_data, is.character)], as.num
```

```r
M <- cor(hist_data)
correlation_threshold <- 0.204
significant_correlation <- abs(M["histological.type", ]) > correlation_threshold
table(significant_correlation)
```

```
## significant_correlation
## FALSE  TRUE
##  1886    51
```

```r
corr_colnames <- colnames(M)[significant_correlation]
```

```r
library(caret)

set.seed(432)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(hist_data))

## set the seed to make your partition reproducible
train_ind <- sample(seq_len(nrow(hist_data)), size = smp_size)


train_y <- outcomes$histological.type[train_ind]
hist_train <- cbind(train_y, hist_data[train_ind, ])
hist_train <- hist_train[, corr_colnames]


# Training parameters
ctrl <- trainControl(method = "cv",
                     number = 5,
                     savePredictions = TRUE,
                     classProbs = FALSE)

# Fitting model
mod_fit <- train(as.factor(histological.type) ~ ., data=hist_train,
                 method="glm",
                 family="binomial",
                 trControl = ctrl,
                 tuneLength = 5)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(mod_fit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.07602  -0.06594  -0.01469  -0.00141   2.54704
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -1.476e+01  4.948e+00  -2.984 0.002849 **
## rs_TFAP2B        1.662e-01  1.896e-01   0.877 0.380744
## rs_CYP4Z1       -3.451e-02  1.778e-01  -0.194 0.846115
## rs_C7           -1.996e-01  4.106e-01  -0.486 0.626864
## rs_ABCA8         2.879e-01  4.696e-01   0.613 0.539849
## rs_SLC26A3       2.155e-01  1.744e-01   1.235 0.216705
## rs_ADCY5        -5.141e-02  2.788e-01  -0.184 0.853714
## rs_RERGL         7.007e-01  4.343e-01   1.613 0.106642
## rs_ANKRD43       3.110e-01  2.359e-01   1.318 0.187410
## rs_LOC389033    -1.143e-01  2.593e-01  -0.441 0.659404
## rs_CCL14        -9.174e-02  6.062e-01  -0.151 0.879725
## rs_MMRN1         2.245e-01  3.218e-01   0.698 0.485381
## cn_PART1        -1.521e-01  2.783e+00  -0.055 0.956405
## cn_GTF2H2B      -8.524e-01  1.114e+01  -0.076 0.939026
## cn_CARTPT       -2.972e+00  2.154e+01  -0.138 0.890265
## cn_FOXD1         5.581e+00  2.161e+01   0.258 0.796189
## cn_F2RL2         2.366e+01  8.390e+03   0.003 0.997750
## cn_PDE8B        -2.296e+01  8.390e+03  -0.003 0.997816
## cn_THBS4         1.689e+01  4.711e+03   0.004 0.997139
## cn_CKMT2         7.321e+00  8.693e+03   0.001 0.999328
## cn_HAPLN1       -1.575e+01  1.254e+04  -0.001 0.998998
## cn_EDIL3        -1.154e+01  7.718e+03  -0.001 0.998807
## cn_GPR98        -1.013e+00  1.559e+01  -0.065 0.948220
## cn_FAM81B       -2.754e+00  2.191e+03  -0.001 0.998997
## cn_C5orf27      -3.445e+00  2.313e+03  -0.001 0.998812
## cn_PCSK1         5.015e+00  3.710e+03   0.001 0.998922
## cn_ERAP2         2.211e+00  1.901e+03   0.001 0.999072
## cn_EFNA5         1.214e+00  1.678e+01   0.072 0.942319
## cn_TSLP          1.120e+01  5.599e+03   0.002 0.998403
## cn_TRIM36       -1.126e+01  5.599e+03  -0.002 0.998396
## cn_CDO1          1.438e+01  9.420e+03   0.002 0.998782
## cn_AQPEP        -1.341e+01  9.420e+03  -0.001 0.998864
## cn_CDH3          2.467e+00  1.228e+04   0.000 0.999840
## cn_CDH1         -9.113e+00  9.116e+03  -0.001 0.999202
## cn_HAS3          6.683e+00  8.225e+03   0.001 0.999352
## cn_TAT          -1.883e-01  7.649e+00  -0.025 0.980363
## cn_HP           -9.571e-01  5.139e+00  -0.186 0.852261
## mu_CDH1          4.093e+00  1.179e+00   3.471 0.000518 ***
## pp_Caspase.8     1.723e+00  2.584e+00   0.667 0.504800
## pp_Caveolin.1   -7.661e-02  6.437e-01  -0.119 0.905259
## pp_DJ.1          1.749e+00  1.558e+00   1.123 0.261526
## pp_Dvl3         -3.971e+00  2.517e+00  -1.578 0.114664
## pp_E.Cadherin   -1.129e+00  6.820e-01  -1.655 0.097848 .
## pp_ENY2          1.088e+00  1.313e+00   0.828 0.407406
## pp_Myosin.IIa   -7.800e-01  1.386e+00  -0.563 0.573591
## pp_RBM15         1.714e+00  1.092e+00   1.570 0.116511
## pp_Rab11        -3.036e+00  1.947e+00  -1.559 0.119019
## pp_S6           -1.613e+00  1.307e+00  -1.234 0.217064
## pp_SLC1A5       -7.351e-01  9.635e-01  -0.763 0.445498
## pp_beta.Catenin -4.839e-01  1.378e+00  -0.351 0.725543
## pp_eIF4G        -1.539e+00  1.270e+00  -1.212 0.225454
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 251.428  on 379  degrees of freedom
## Residual deviance:  59.391  on 329  degrees of freedom
## AIC: 161.39
##
## Number of Fisher Scoring iterations: 19
```

Compute and plot the AUC:

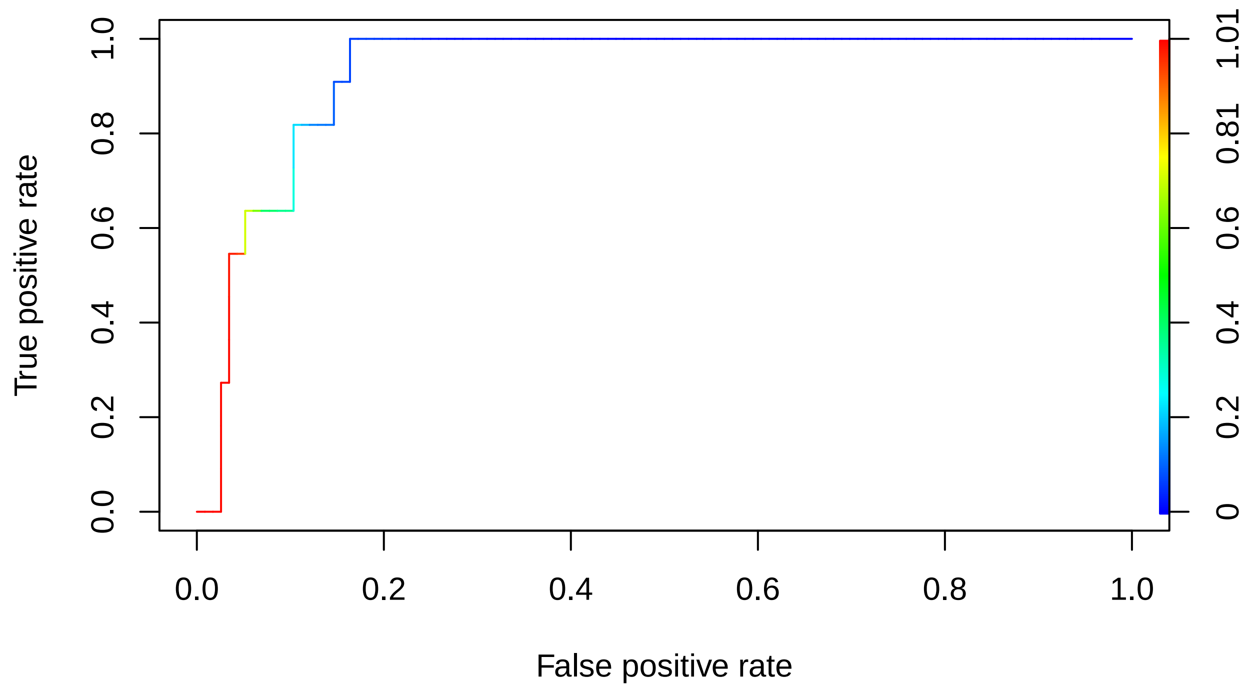
```
test_y <- as.data.frame(outcomes$histological.type[-train_ind])

dim(test_y)
```

```
## [1] 127   1
```

```
dim(hist_data[-train_ind,])
```

```
## [1]  127 1937
```

```
hist_test <- cbind(test_y, hist_data[-train_ind, ])

test_predict <- predict(mod_fit, hist_test, type="prob")
roc <- prediction(test_predict[,2], test_y)

# calculates the ROC curve
perf <- performance(roc,"tpr","fpr")
plot(perf,colorize=TRUE)
```



```
# obtaining the AUC
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.9318182
```

The AUC for histological.type is 0.9318182

ER.Status

```
library(dplyr)
brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brca
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))

# store the outcomes
outcomes = brca_data %>% select(c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))
# remove the categorical var temporarily
brca_data = brca_data %>% select(-c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type")

# Create a correlation matrix and select only variables with some correlation to the target
ER.Status = as.numeric(as.factor(outcomes$ER.Status))

er_data = cbind(ER.Status, brca_data)
```

```
er_data[sapply(er_data, is.character)] <- lapply(er_data[sapply(er_data, is.character)], as.numeric)
```

```
M <- cor(er_data)
correlation_threshold <- 0.501
significant_correlation <- abs(M["ER.Status", ]) > correlation_threshold
table(significant_correlation)
```

```
## significant_correlation
## FALSE  TRUE
##  1886    51
```

```
corr_colnames <- colnames(M)[significant_correlation]
```

```
library(caret)

set.seed(432)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(er_data))

## set the seed to make your partition reproducible
train_ind <- sample(seq_len(nrow(er_data)), size = smp_size)


train_y <- outcomes$ER.Status[train_ind]
er_train <- cbind(train_y, er_data[train_ind, ])
er_train <- er_train[, corr_colnames]


# Training parameters
ctrl <- trainControl(method = "cv",
                     number = 5,
                     savePredictions = TRUE,
                     classProbs = FALSE)

# Fitting model
mod_fit <- train(as.factor(ER.Status) ~ ., data=er_train,
                 method="glm",
                 family="binomial",
                 trControl = ctrl,
                 tuneLength = 5)
```

```
summary(mod_fit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -8.49    0.00    0.00    0.00    8.49
##
## Coefficients:
##                Estimate Std. Error    z value Pr(>|z|)
## (Intercept)   6.956e+15  6.099e+07  114052178   <2e-16 ***
## rs_TFF1       3.932e+14  2.018e+06  194897908   <2e-16 ***
## rs_CYP2B7P1   9.201e+13  1.487e+06   61865404   <2e-16 ***
## rs_AGR3       3.333e+14  2.543e+06  131095335   <2e-16 ***
## rs_GP2        1.626e+13  1.546e+06   10515790   <2e-16 ***
## rs_C1orf64   -1.600e+14  2.082e+06  -76861520   <2e-16 ***
## rs_TFF3      -3.281e+14  2.782e+06 -117932448   <2e-16 ***
## rs_AGR2       2.001e+14  2.718e+06   73633411   <2e-16 ***
## rs_GFRA1      5.871e+13  1.768e+06   33202850   <2e-16 ***
## rs_PGR        6.277e+13  2.000e+06   31392302   <2e-16 ***
## rs_VGLL1     -6.556e+13  2.397e+06  -27352502   <2e-16 ***
## rs_SERPINA11 -2.112e+14  1.889e+06 -111825460   <2e-16 ***
## rs_KRT16     -2.747e+14  2.311e+06 -118860169   <2e-16 ***
## rs_A2ML1     -4.880e+13  2.044e+06  -23879318   <2e-16 ***
## rs_ESR1      -7.516e+13  3.914e+06  -19203128   <2e-16 ***
## rs_SYT9      -4.261e+13  1.871e+06  -22777598   <2e-16 ***
## rs_GRPR       1.063e+14  1.890e+06   56220055   <2e-16 ***
## rs_CHAD      -7.531e+13  1.729e+06  -43558352   <2e-16 ***
## rs_WNK4      -5.822e+13  2.182e+06  -26676185   <2e-16 ***
## rs_ABCC8      1.311e+14  2.053e+06   63864341   <2e-16 ***
## rs_SLC44A4    3.206e+14  2.482e+06  129174389   <2e-16 ***
## rs_SCUBE2    -2.141e+14  2.596e+06  -82500887   <2e-16 ***
## rs_ERBB4      8.612e+13  2.375e+06   36257897   <2e-16 ***
## rs_BCAS1      1.337e+14  2.528e+06   52887597   <2e-16 ***
## rs_HORMAD1   -7.558e+13  2.281e+06  -33132008   <2e-16 ***
## rs_FSIP1     -7.670e+13  2.547e+06  -30114803   <2e-16 ***
## rs_FOXA1     -7.239e+14  4.188e+06 -172844038   <2e-16 ***
## rs_NAT1      -1.273e+14  2.235e+06  -56954494   <2e-16 ***
## rs_SLC7A2    -4.000e+13  2.001e+06  -19986653   <2e-16 ***
## rs_FLJ45983  -8.835e+13  2.440e+06  -36211345   <2e-16 ***
## rs_ART3      -1.188e+14  2.319e+06  -51242674   <2e-16 ***
## rs_PPP1R14C   2.015e+13  2.810e+06    7168824   <2e-16 ***
## rs_RGS22     -1.562e+14  2.261e+06  -69093025   <2e-16 ***
## rs_AR         2.210e+14  3.045e+06   72561576   <2e-16 ***
## rs_AFF3       8.107e+13  2.825e+06   28692783   <2e-16 ***
## rs_BCL11A     8.410e+12  2.428e+06    3464524   <2e-16 ***
## rs_SBSN       1.217e+14  2.359e+06   51569252   <2e-16 ***
## rs_SOX11     -1.590e+14  2.122e+06  -74959804   <2e-16 ***
## rs_TTC36     -1.521e+14  2.511e+06  -60594295   <2e-16 ***
## rs_GABBR2     2.199e+14  2.422e+06   90797108   <2e-16 ***
## rs_SLC15A1    5.099e+13  2.446e+06   20841622   <2e-16 ***
## rs_DEGS2     -9.733e+13  3.343e+06  -29112764   <2e-16 ***
## rs_ADAMTS15   1.447e+14  2.434e+06   59452809   <2e-16 ***
## rs_CLSTN2     5.760e+13  2.860e+06   20139630   <2e-16 ***
## rs_DACH1      4.690e+13  2.448e+06   19160353   <2e-16 ***
## rs_LOC84740  -2.324e+14  2.119e+06 -109662842   <2e-16 ***
## rs_ACE2      -1.016e+14  2.300e+06  -44182529   <2e-16 ***
## rs_AMY1A      1.702e+14  2.500e+06   68058773   <2e-16 ***
## pp_ER.alpha   8.526e+14  4.122e+06  206849284   <2e-16 ***
## pp_GATA3     -2.602e+14  6.658e+06  -39080374   <2e-16 ***
## pp_INPP4B    -1.661e+14  5.823e+06  -28519970   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:  411.29  on 379  degrees of freedom
## Residual deviance: 1009.22  on 329  degrees of freedom
## AIC: 1111.2
##
## Number of Fisher Scoring iterations: 25
```

Compute and plot the AUC:

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

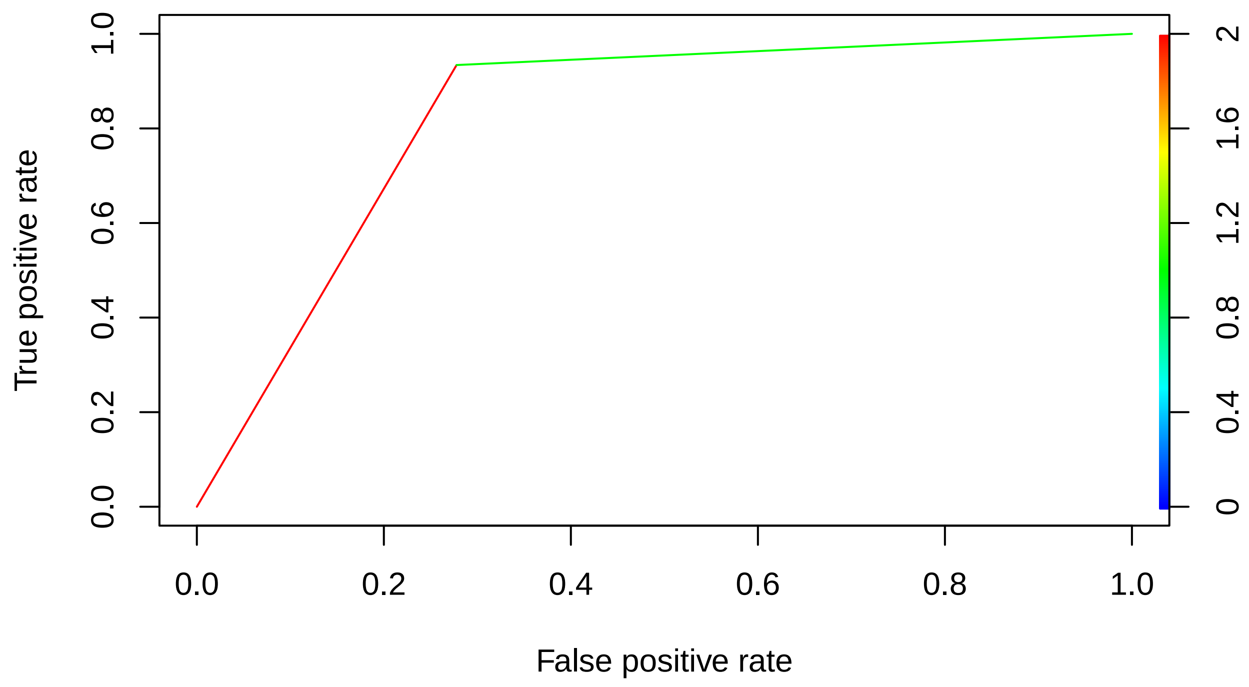
```
library(ROCR)

test_y <- as.data.frame(outcomes$ER.Status[-train_ind])

dim(test_y)
```

```
## [1] 127   1
```

```
dim(er_data[-train_ind,])
```

```
## [1]  127 1937
```

```
er_test <- cbind(test_y, er_data[-train_ind, ])

test_predict <- predict(mod_fit, er_test, type="prob")
roc <- prediction(test_predict[,2], test_y)

# calculates the ROC curve
perf <- performance(roc,"tpr","fpr")
plot(perf,colorize=TRUE)
```



```r
# obtaining the AUC
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8281441
```

The AUC for ER.Status is 0.9318182

HER2.Final.Status

```r
library(dplyr)
brca_data = read.csv("https://raw.githubusercontent.com/mgarbvs/STAT-432-final-project/main/cleaned_brca
# remove X.1 and X columns
brca_data = brca_data %>% select(-c("X.1", "X"))

# store the outcomes
outcomes = brca_data %>% select(c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type"))
# remove the categorical var temporarily
brca_data = brca_data %>% select(-c("PR.Status", "ER.Status", "HER2.Final.Status", "histological.type")

# Create a correlation matrix and select only variables with some correlation to the target
HER2.Final.Status = as.numeric(as.factor(outcomes$HER2.Final.Status))

final_status_data = cbind(HER2.Final.Status, brca_data)
```

```
final_status_data[sapply(final_status_data, is.character)] <- lapply(final_status_data[sapply(final_sta
```

```
M <- cor(final_status_data)
correlation_threshold <- 0.193
significant_correlation <- abs(M["HER2.Final.Status", ]) > correlation_threshold
table(significant_correlation)
```

```
## significant_correlation
## FALSE  TRUE
##  1886    51
```

```
corr_colnames <- colnames(M)[significant_correlation]
```

```
library(caret)

set.seed(432)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(final_status_data))

## set the seed to make your partition reproducible
train_ind <- sample(seq_len(nrow(final_status_data)), size = smp_size)


train_y <- outcomes$HER2.Final.Status[train_ind]
final_status_train <- cbind(train_y, final_status_data[train_ind, ])
final_status_train <- final_status_train[, corr_colnames]


# Training parameters
ctrl <- trainControl(method = "cv",
                     number = 5,
                     savePredictions = TRUE,
                     classProbs = FALSE)

# Fitting model
mod_fit <- train(as.factor(HER2.Final.Status) ~ ., data=final_status_train,
                 method="glm",
                 family="binomial",
                 trControl = ctrl,
                 tuneLength = 5)
```

```
summary(mod_fit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.24054  -0.00491  -0.00009   0.00000   2.80863
##
## Coefficients: (1 not defined because of singularities)
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -27.17755   12.04701  -2.256   0.0241 *
## rs_MUCL1         0.22531    0.20470   1.101   0.2710
## rs_CEACAM5       1.29164    0.61157   2.112   0.0347 *
## rs_HOXB13       -0.11247    0.23362  -0.481   0.6302
## rs_CEACAM6      -0.98037    0.49446  -1.983   0.0474 *
## rs_CRISP3       -0.09467    0.25442  -0.372   0.7098
## rs_ABCC11        1.16431    0.63779   1.826   0.0679 .
## rs_GRIA2        -1.05074    0.58815  -1.787   0.0740 .
## rs_NPY1R         0.66636    0.49945   1.334   0.1821
## rs_LRP2          0.04876    0.30879   0.158   0.8745
## rs_KLK10         0.36831    0.32701   1.126   0.2600
## rs_CYP4Z2P      -0.84002    0.48280  -1.740   0.0819 .
## rs_ABCA12        0.01335    0.37746   0.035   0.9718
## rs_SDR16C5       0.04623    0.21389   0.216   0.8289
## rs_C2orf54      -0.61569    0.38931  -1.581   0.1138
## rs_S100A7A      -0.07818    0.49153  -0.159   0.8736
## rs_SLC9A2       -0.01864    0.30032  -0.062   0.9505
## rs_PPP4R4       -0.57069    0.42206  -1.352   0.1763
## rs_PNMT         -0.26830    0.42068  -0.638   0.5236
## rs_PAX7          1.15414    0.54633   2.113   0.0346 *
## rs_C2CD4A       -1.36189    0.60459  -2.253   0.0243 *
## rs_NPY5R        -0.87043    0.51127  -1.702   0.0887 .
## rs_ONECUT2      -0.61152    0.37080  -1.649   0.0991 .
## rs_RBM24        -0.36237    0.39749  -0.912   0.3620
## rs_AKR1B10       0.75353    0.45295   1.664   0.0962 .
## rs_TDRD1         0.56589    0.45432   1.246   0.2129
## rs_IYD           0.74123    0.50757   1.460   0.1442
## rs_PP14571       0.30492    0.36571   0.834   0.4044
## rs_ABCC12       -0.71509    0.53994  -1.324   0.1854
## rs_SCRG1        -1.92828    0.77897  -2.475   0.0133 *
## rs_OSR1          2.19440    1.06247   2.065   0.0389 *
## rs_S100A1        0.09281    0.33881   0.274   0.7841
## rs_SERHL2       -0.04579    0.36395  -0.126   0.8999
## rs_NTRK3        -0.74699    0.55549  -1.345   0.1787
## rs_TMEM45B       1.16029    0.57894   2.004   0.0451 *
## cn_DEFB1        -4.19934    2.10365  -1.996   0.0459 *
## cn_STAC2        -5.35117    4.16914  -1.284   0.1993
## cn_PPP1R1B       9.97419  799.85370   0.012   0.9901
## cn_PNMT          0.86188  799.87170   0.001   0.9991
## cn_IKZF3         0.79722    5.99596   0.133   0.8942
## cn_RUNDC3A       7.65588    5.27417   1.452   0.1466
## cn_GFAP         -9.62566    6.03064  -1.596   0.1105
## cn_KIF18B             NA         NA      NA       NA
## cn_CA4          -0.31303    1.23958  -0.253   0.8006
## pp_EGFR.pY1068   3.33196    1.46266   2.278   0.0227 *
## pp_FASN          0.02043    0.97955   0.021   0.9834
## pp_G6PD          0.44436    1.43639   0.309   0.7570
## pp_HER2          3.51959    2.24336   1.569   0.1167
## pp_HER2.pY1248  -2.62099    2.95507  -0.887   0.3751
## pp_HER3.pY1289   3.44563    2.87949   1.197   0.2315
## pp_p70S6K        6.00234    2.47261   2.428   0.0152 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 347.739  on 379  degrees of freedom
## Residual deviance:  57.459  on 330  degrees of freedom
## AIC: 157.46
##
## Number of Fisher Scoring iterations: 15
```

Compute and plot the AUC:

```
test_y <- as.data.frame(outcomes$HER2.Final.Status[-train_ind])

dim(test_y)
```

```
## [1] 127   1
```

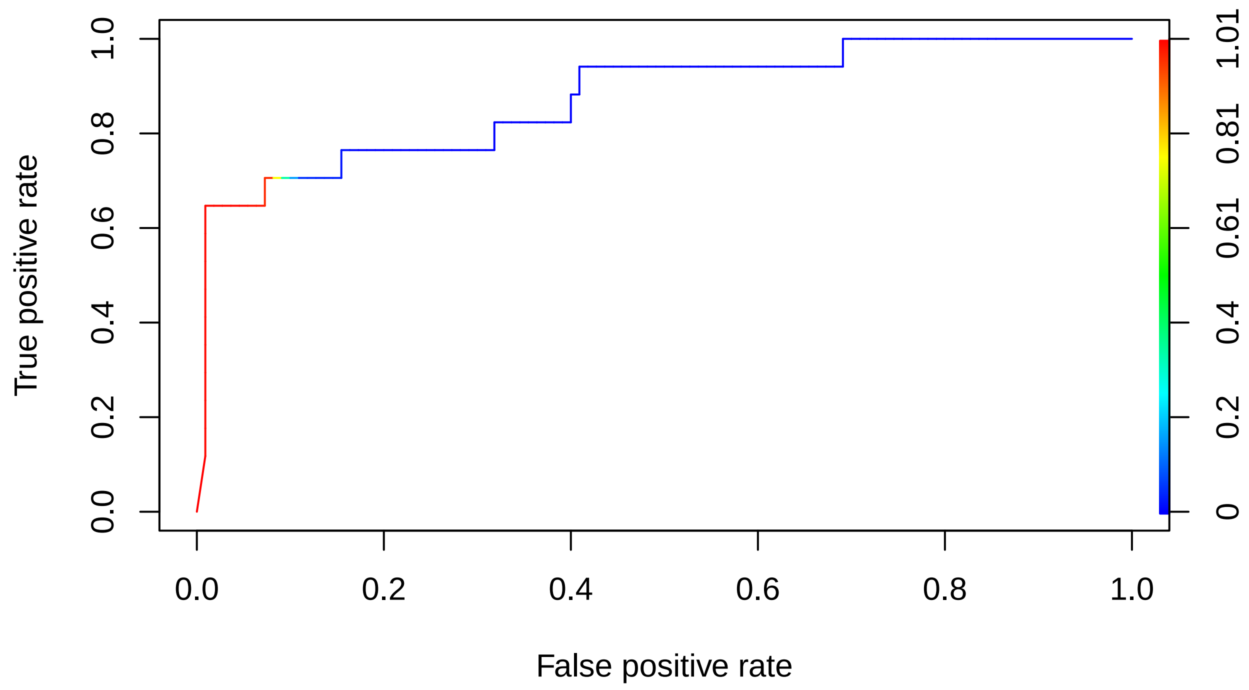
```
dim(final_status_data[-train_ind,])
```

```
## [1]  127 1937
```

```
final_status_test <- cbind(test_y, final_status_data[-train_ind, ])

test_predict <- predict(mod_fit, final_status_test, type="prob")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
roc <- prediction(test_predict[,2], test_y)

# calculates the ROC curve
perf <- performance(roc,"tpr","fpr")
plot(perf,colorize=TRUE)
```



```r
# obtaining the AUC
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8743316
```

The AUC for HER2.Final.Status is 0.8743316

Now, to select the actual 50-variables.

1) Output all of the 50 predictor variables for each outcome
2) Generate a count for top ones
3) pick the top 50 from there