Project

Contents

Note: have not finished data preprocessing

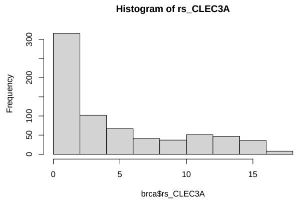
Project Description (est. 1 page, pt. 5)

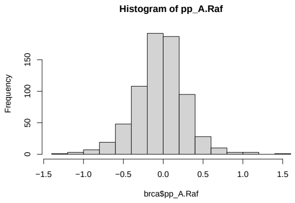
Literature Review(est. 1 page, pt.10)

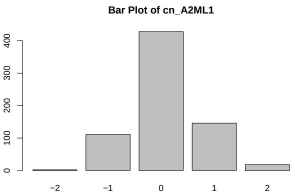
Summary Statistics and Data Preprocessing (est. 1 -2 pages, pt.10)

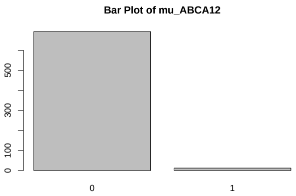
Data Overview

The dataset has 705 observations and 1941 features (1936 predictors and 5 outcomes). There are four different kinds of predictors: `rs` (gene expression), `cn` (copy number variations), `mu` (mutations), and `pp` (protein levels). Among them, `rs` and `pp` are continuous variables, and `cn` and `mu` are categorical variables.









Remove Missing Values

According to the instruction, we dropped `vital.status`, and we only considered each response variable as a binary variable. Therefore, we treated the observations that had other outcomes as missing values and removed them from our dataset.

Then the dataset `sub` had 507 observations and 1940 features.

```
dim(sub)
```

```
## [1]  507 1940
```

Deal with Multicollinearity

One of the noticeable characteristics of the data is its high dimensionality. There are 1936 predictors, almost four times as many as there are observations. Therefore, it is essential to check correlation.

Since there are four kinds of predictors, it is unlikely that two variables that belong to different kinds would be highly correlated. Also, to reduce the computational cost, we split the data into four subsets: `rs`, `cn`, `mu`, and `pp`, each of which contained only ond kind of predictors.

Then, we created the correlation matrix for each subset, and extracted variables that are highly-correlated with at least one other variable. Take `rs` as an example. The dataframe `idx` stores all matrix indices of highly-correlated variables and the corresponding correlation coefficients. If the i-th variable is highly-correlated with the j-th variable, then we only need one of them. Thus, we removed all variables with indices `i`. For `rs`, 94 predictors were removed. We applied the same process to the other three subsets. In total, 882 predictors were removed. There are 1059 predictors remained.

```
names(idx) = c("i", "j", "corr")
idx[1:3,]
```

```
##   i  j corr
## 2 3  4 0.94
## 3 5 56 0.84
## 4 9 10 0.95
```

```
# remove highly-correlated variables
rmv = unique(idx[,1])
length(rmv)
```

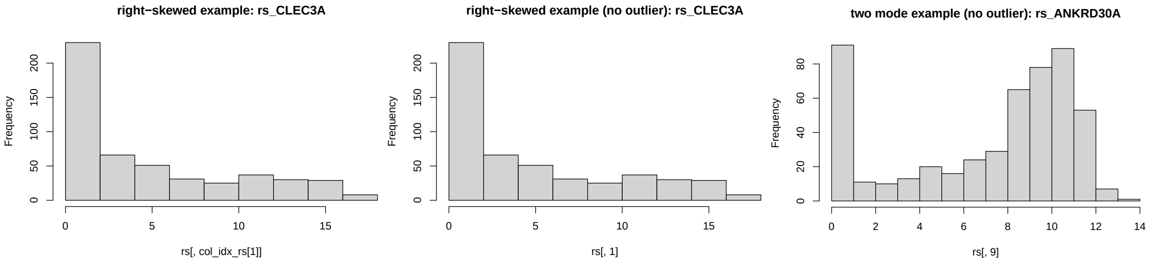
```
## [1] 94
```

```
rs = rs[,-rmv]
```

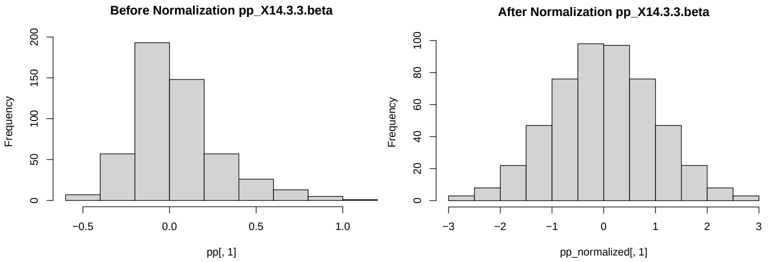
Continuous Variables

As mentioned before, `rs` and `pp` are continuous variables, so we should examine if there are any outliers. We first normalized the variable, and stored row and column indices if the data point was three standard deviations away from the variable mean. For the two subsets, `rs` had 100 outliers, and `pp` had no outlier.

We further looked into `rs` predictors that included outliers, and we found the vast majority of them had a long tail, mostly right and some left. In addition, a number of `rs` predictors that did not contain outliers also had a non-standard distribution. As a result, a log transformation of `rs` predictors would be beneficial.



Unlike `rs`, `pp` variables were distributed quite normally. However, many of the variables would contain outliers without normalization. Therefore, we normalized `pp` variables.



Categorical Variables

All four outcomes were imbalanced, among which `histological.type` was the most imbalanced. Only 10% of the responses were IDC. In some cases, such imbalance would cause models to learn nothing from the minority class. If our data also suffer from problems like this, we should resolve the imbalance by techniques like undersampling. However, if our models perform well enough on current data, no further action needs to be taken.

Fortunately, we trained our models first, and found the models obtained high enough accuracy and AUC scores. Therefore, we decided to not address imbalanced outcomes.

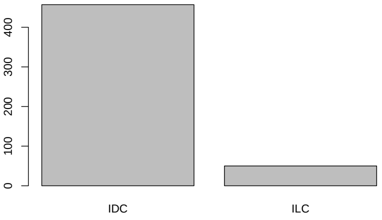
```
##
## Negative Positive
##  0.34714  0.65286
```

```
##
##  Negative  Positive
## 0.2445759 0.7554241
```

```
##
##  infiltrating ductal carcinoma infiltrating lobular carcinoma
##                     0.90138067                     0.09861933
```

```
##
##  Negative  Positive
## 0.8382643 0.1617357
```

```
# par(mfrow=c(2, 1))
# barplot(table(sub$PR.Status))
# barplot(table(sub$ER.Status))
# barplot(table(sub$HER2.Final.Status))
barplot(table(sub$histological.type), names.arg = c("IDC", "ILC"))
```



Modeling PR Status (est. 2-3 pages, pt.20)

Before modeling, we split the train and test datasets. We used 25% of the samples (126) for testing and 75% of the samples for trianing (381).

Support Vector Machine (SVM)

The goal of the project was to make classifications. Plus, we needed to deal with high-dimensional data. Therefore, we should choose classification models that perform well for high-dimensional data. Support vector machines are famous for its capability in high-dimensional spaces, so we first fitted a basic linear SVM, with the default `cost = 1` to see how it worked.

As the confusion matrix showed, the in-sample accuracy was 1.0, which implies that we may prefer the linear kernel to the radial kernel.

```
##            actual
## predicted  Negative Positive
##   Negative      133        0
##   Positive        0      248
```

Then we constructed two grids of tuning parameters for both linear and radial kernels, and we used 5-fold cross-validation to tune the parameters and to determine which kernel was better.

For the linear kernel, the best `C` was 0.001 with an in-sample accuracy of 0.8324.

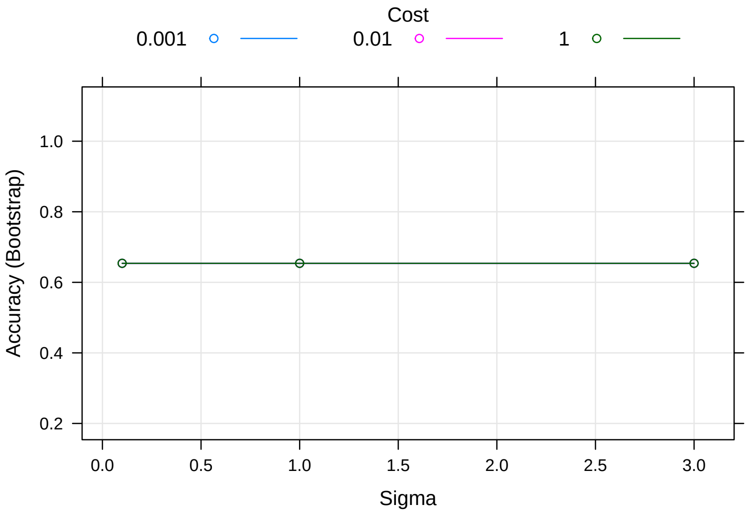
```
##       C
## 1 0.001
```

```
##       C  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.001 0.8336029 0.6058770 0.02730027 0.06541415
## 2 0.010 0.7999388 0.5464954 0.02526173 0.06257166
## 3 0.050 0.8014261 0.5503720 0.02442031 0.06150767
## 4 0.100 0.8014261 0.5503720 0.02442031 0.06150767
## 5 0.500 0.8014261 0.5503720 0.02442031 0.06150767
## 6 1.000 0.8014261 0.5503720 0.02442031 0.06150767
```

For the radial kernel, the best `C` was 0.001 and the best `sigma` was 3. However, the results showed that the radial SVM fitted poorly, since the accuracies remained the same and were merely 0.6677. The fact verified our hypothesis that a linear kernel would work better. Thus, we picked the linear SVM with `C` equals 0.001 to make classifications.

```
##   sigma     C
## 3     3 0.001
```

```
##       C sigma  Accuracy Kappa AccuracySD KappaSD
## 1 0.001   0.1 0.6539067     0 0.02710967       0
## 4 0.010   0.1 0.6539067     0 0.02710967       0
## 7 1.000   0.1 0.6539067     0 0.02710967       0
## 2 0.001   1.0 0.6539067     0 0.02710967       0
## 5 0.010   1.0 0.6539067     0 0.02710967       0
## 8 1.000   1.0 0.6539067     0 0.02710967       0
## 3 0.001   3.0 0.6539067     0 0.02710967       0
## 6 0.010   3.0 0.6539067     0 0.02710967       0
## 9 1.000   3.0 0.6539067     0 0.02710967       0
```



We made predictions for the test data, and printed the confusion table below. The accuracy was 0.9048. Thus, the linear SVM performed quite well.

```
##           actual
## predicted  Negative Positive
##   Negative       33        2
##   Positive       10       81
```

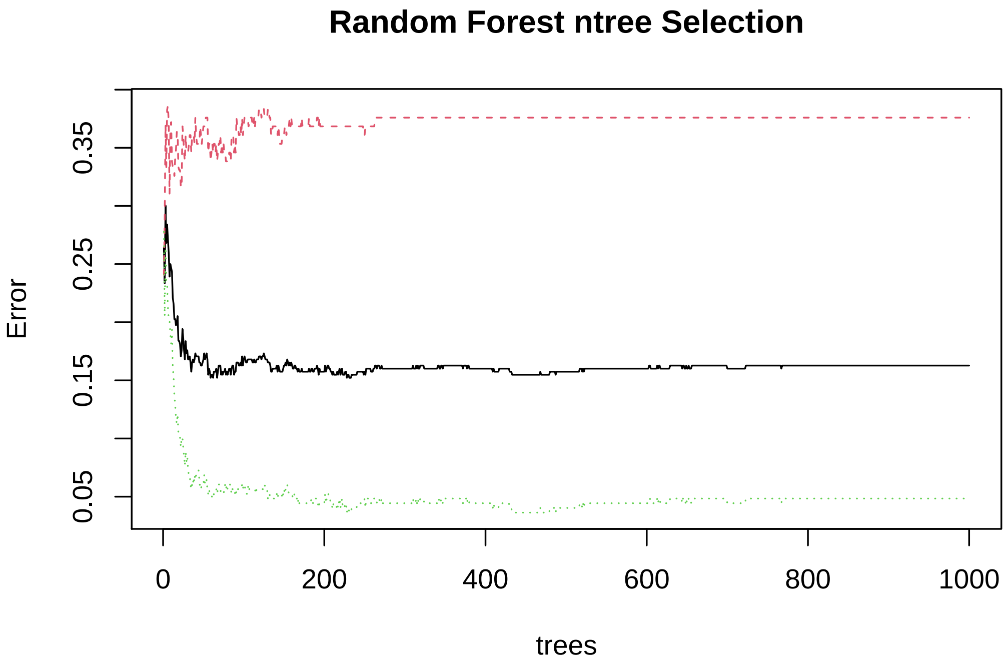
```
## [1] 0.9047619
```

Random Forest

Random forests are another classification model that is less vulnerable to “the curse of dimensionality”. Since there were many parameters that needed to be tuned, we again utilized `caret` package to cross validate the model.

Before applying cross-validation, we should first determine `num.trees`, because it could not be included in the grid of parameters. Therefore, we first fitted a random forest using the `randomForest()` method with `ntree = 1000`, and plotted the error against trees.

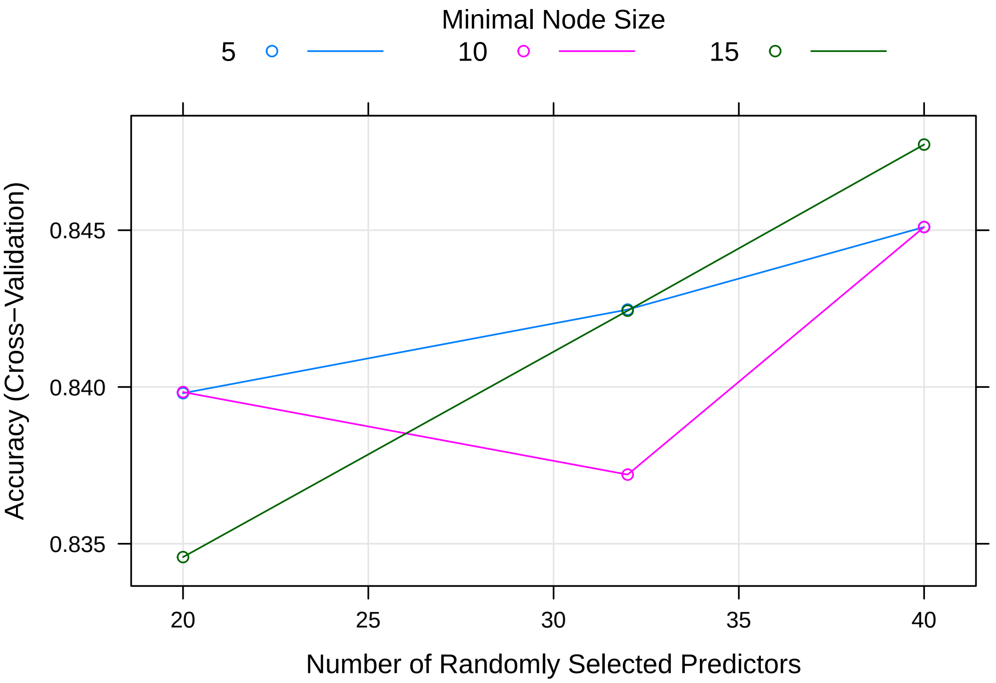
As the plot demonstrated, the error stopped decreasing after the number of trees reached around 500. Thus, `ntree = 500` should be sufficient for training.



In a 5-fold cross-validation, we tuned `mtry` and `min.node.size`. The output showed that the best parameters were `mtry = 40` and `min.node.size = 15` when `num.trees = 500` according to the test accuracy.

```
##   mtry splitrule min.node.size
## 9   40      gini            15
```

```
##   mtry splitrule min.node.size  Accuracy     Kappa AccuracySD    KappaSD
## 1   20      gini             5 0.8398027 0.6199996 0.01816714 0.04744967
## 2   20      gini            10 0.8398378 0.6191721 0.01181501 0.02631224
## 3   20      gini            15 0.8345746 0.6097325 0.01575637 0.03413845
## 4   32      gini             5 0.8424694 0.6272972 0.01375275 0.03444365
## 5   32      gini            10 0.8372062 0.6150392 0.01262152 0.02909492
## 6   32      gini            15 0.8424343 0.6268844 0.01948896 0.05058240
## 7   40      gini             5 0.8451009 0.6341821 0.01487847 0.03698468
## 8   40      gini            10 0.8451009 0.6356210 0.02177987 0.05139481
## 9   40      gini            15 0.8477325 0.6408985 0.01796807 0.04555969
```



The confusion matrix and the highest test accuracy, 0.9048, were shown here.

```
##           actual
## predicted  Negative Positive
##   Negative       33        2
##   Positive       10       81
```

```
## [1] 0.9047619
```

Histological Type (hcluster and knn regression) (est 2-3 pages, pt.20)

To establish the Modeling for histological, We first use logistical regression, However, when we build the model, we found that the logistic model's algothrim is not coverge. The reason that this error occur, because the variable x can divide the reponse variable y into 0 and 1 perfectly. The accurate will be 100%. To solve the problem we decided to use penalized regression. Thus, we choose the modle of logistic regession with ridge penalty

Logistic Regression

To performance

```r
y = as.factor(sub$histological.type)

y = as.factor(ifelse(y == "infiltrating lobular carcinoma", 1, 0))
sub3 = cbind(rs, cn, mu, pp, y) # cleaned dataset with PR.status as response
```

```r
set.seed(651978735)
n = dim(sub)[1]
test_size = as.integer(0.25 * n)
test_idx = sample(1:n, test_size) # 25% of the sample size
```

```
Xtest = sub3[test_idx, -ncol(sub2)]
Xtrain = sub3[-test_idx, -ncol(sub2)]

ytest = sub3[test_idx, ncol(sub2)]
ytrain = sub3[-test_idx, ncol(sub2)]
```

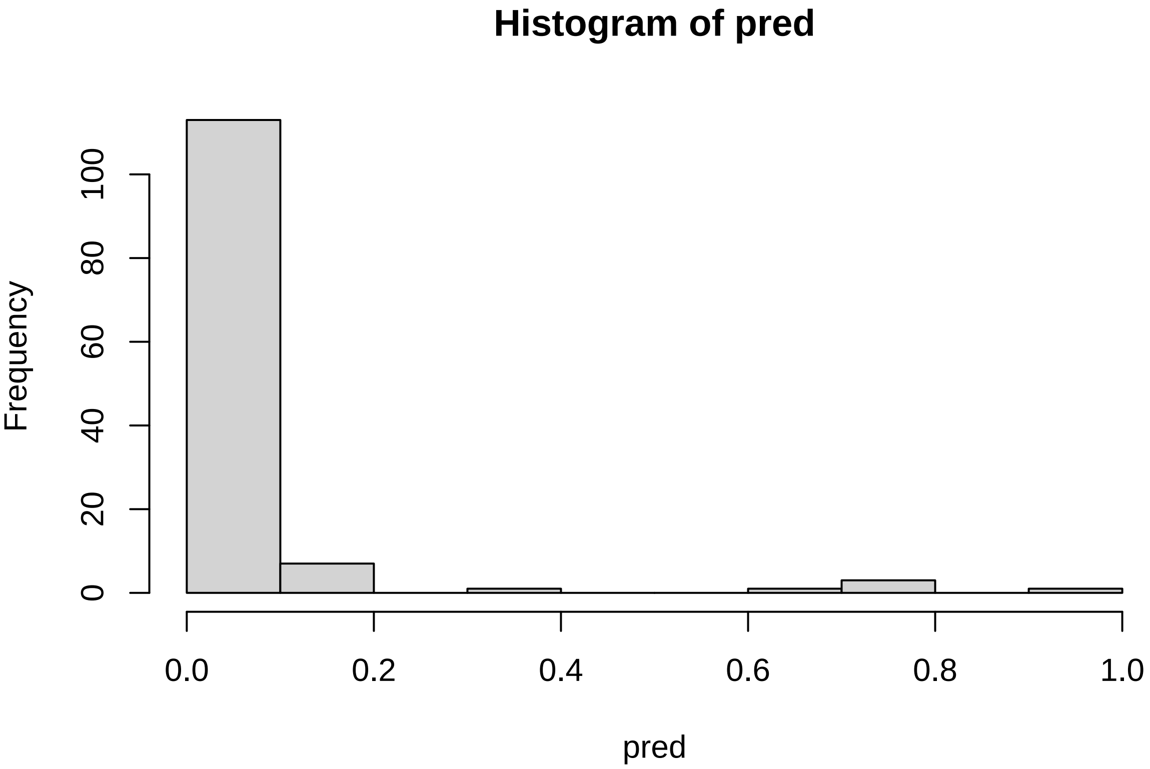
logistic regression with ridge penalty and 10 fold corss validation.

```
library(glmnet)
```

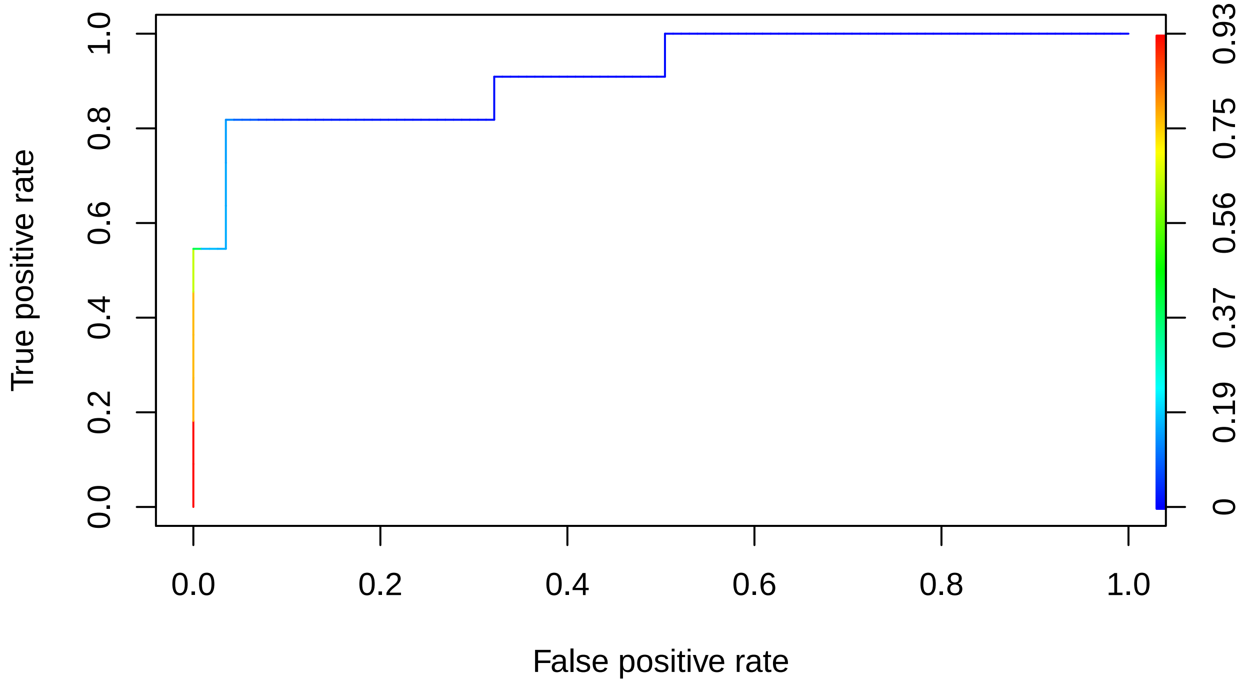
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
fit1 = cv.glmnet(x = data.matrix(Xtrain), y = ytrain, nfolds = 10,
                 type.measure = "auc", family = "binomial")
pred = predict(fit1, newx = data.matrix(Xtest), type = "response", s = fit1$lambda.min)
hist(pred)
```



```
library(ROCR)
roc2 <- prediction(pred, ytest)
# calculates the ROC curve
perf2 <- performance(roc2,"tpr","fpr")
plot(perf2,colorize=TRUE)
```



```
performance(roc2, measure = "auc")@y.values[[1]]
```

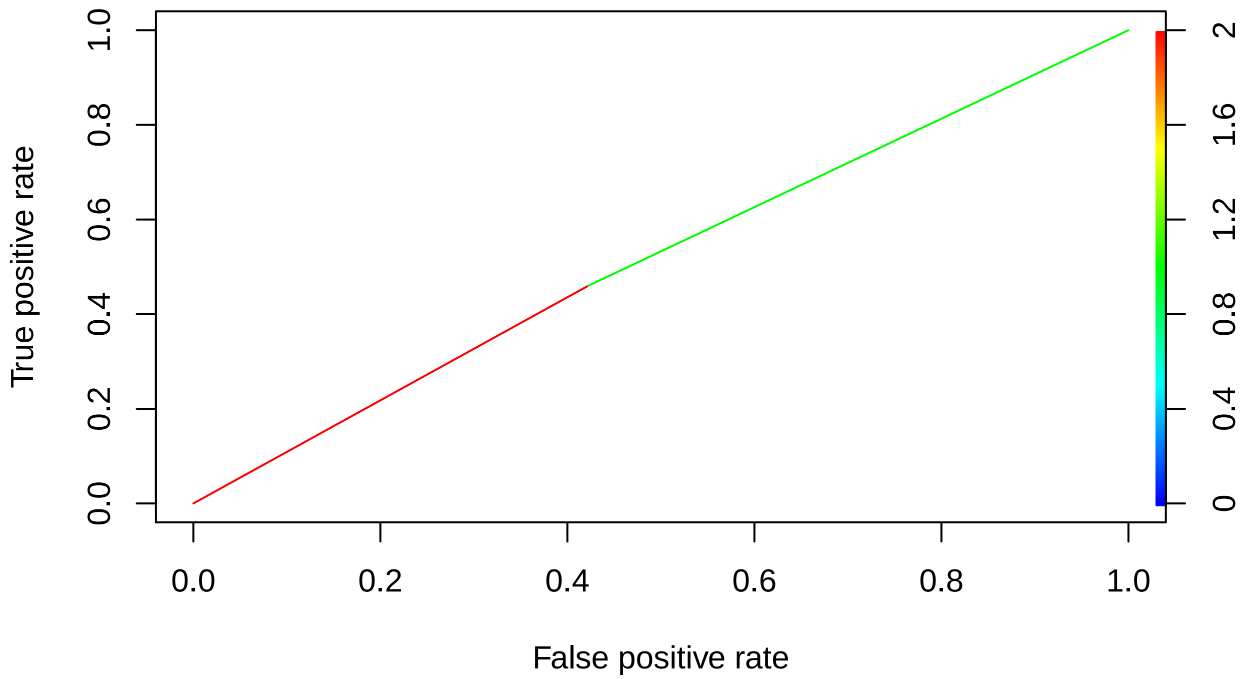
```
## [1] 0.915415
```

hcluster clustering

```
# sub4 = sub3[,-1099]
# kmeanfit <- kmeans(sub4, 2)
# table((kmeanfit$cluster - 1),sub3$y)

data1 = dist(sub3)
complete_hc <- hclust(data1,method = "complete")
# plot(complete_hc)
single_hc <- hclust(data1,method = "single")
# plot(single_hc)
average_hc <- hclust(data1,method = "average")
# plot(average_hc)
clusters = cutree(complete_hc, k = 2)
table((clusters - 1),sub3$y)
```

```
##
##         0   1
##     0 264  27
##     1 193  23
```

```
roc2 <- prediction((clusters - 1), sub3$y)
# calculates the ROC curve
perf2 <- performance(roc2,"tpr","fpr")
plot(perf2,colorize=TRUE)
```



```r
performance(roc2, measure = "auc")@y.values[[1]]
```

```
## [1] 0.5188403
```

Variable Selection for All Outcomes (random forest?) (est. 2-3 pages. pt.20)

Using Random Forest, select the most important 50 variables, and make predictions based on these variables.

```r
impt = rf.fit$importance[order(rf.fit$importance[,3], decreasing=TRUE),][1:50,]
vars = rownames(impt)
```

```r
# sub4 is the cleaned dataset with all four response variables
sub4 = subset(sub, select = vars)
sub4 = cbind(sub4, sub[1937:1940])
sub4$PR.Status = as.factor(sub4$PR.Status)
sub4$histological.type = as.factor(sub4$histological.type)
sub4$ER.Status = as.factor(sub4$ER.Status)
sub4$HER2.Final.Status = as.factor(sub4$HER2.Final.Status)
```

```r
Xtest = sub4[test_idx, 1:50]
Xtrain = sub4[-test_idx, 1:50]
```

```r
ytest = sub4$ER.Status[test_idx]
ytrain = sub4$ER.Status[-test_idx]
svm.fit = svm(ytrain ~., data=Xtrain,
              type="C-classification", kernel="linear", scale=F, cost=1)
table("predicted" = svm.fit$fitted, "actual" = ytrain)
```

```
##           actual
## predicted  Negative Positive
##   Negative       75        9
##   Positive       15      282
```

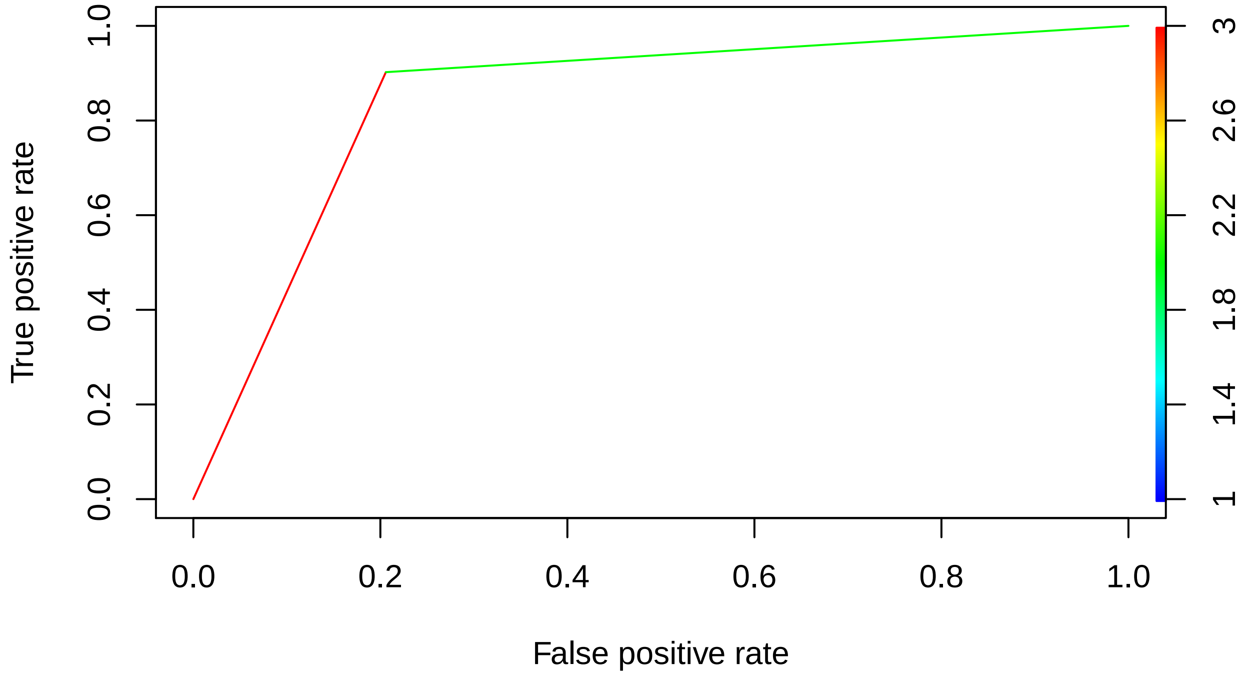
```
pred = predict(svm.fit, newdata = Xtest)
table("predicted" = pred, "actual" = ytest)
```

```
##           actual
## predicted  Negative Positive
##   Negative       27        9
##   Positive        7       83
```

```
library(ROCR)
roc = prediction(as.numeric(pred), ytest)
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8481458
```

```
perf = performance(roc, "tpr", "fpr")
plot(perf, colorize = T)
```



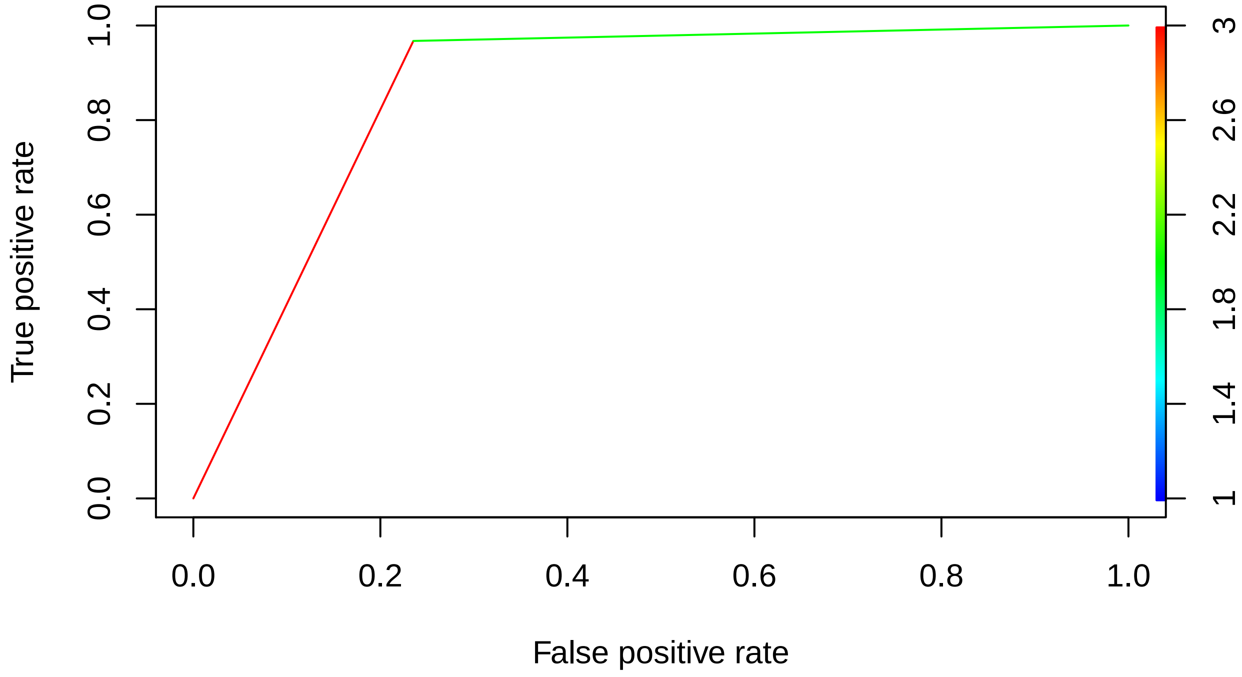
```
rf.fit = randomForest(Xtrain, ytrain,
                      ntree=500,
                      mtry=7,
                      nodesize=10,
                      samplesize=400,
                      importance=TRUE)

pred = predict(rf.fit, Xtest)
table("predicted" = pred, "actual" = ytest)
```

```
##           actual
## predicted  Negative Positive
##   Negative       26        3
##   Positive        8       89
```

```
roc = prediction(as.numeric(pred), ytest)
performance(roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8660486
```

```
perf = performance(roc, "tpr", "fpr")
plot(perf, colorize = T)
```


True positive rate
False positive rate