STAT 301-2 Final Report: Biomarkers

Introduction

Cancer classification is a worthwhile but complicated endeavor. In this project, we attempt to build a classifier based solely on gene data by using statistical learning methods. Our examined dataset contains data for over 7,000 genes of 72 patients, each of whom has a certain type of leukemia: either acute lymphoblastic leukemia (ALL), or acute myeloid leukemia (AML). This gene data was taken from a microarray research experiment. Our goal is to build an accurate statistical model that, as accurately as possible, correctly predicts the type of leukemia a patient has, given the patient's gene expression data. In order to create the such a model, we apply and compare a variety of statistical learning methods, including KNN, Lasso, Ridge Regression, PCR, and Random Forest.

What Manipulations we Made to Data, if Any

Throughout the course of our model-building, we used four different subsets of the predictors. The first set consisted of all 7129 genes. The second subset was comprised of the genes with the highest variance. (We saved this predictor subset as a dataset, referenced here; the full code script is included in our project). The third subset of predictors we used were the genes we outlined to be the 37 "best." To find the best 37 genes, we first calculated a measure (P) of how much each genes correlates with one of the two cancer classes. For each gene, P = (ALLmean - AMLmean)/(ALLsd - AMLsd). This calculation uses the means and standard deviations for each gene, across all 38 patients (in the test dataset). We then selected the 37 genes with the highest P correlation measures. (Again, we saved this dataset, as outlined in our codebook, and left the actual R script in the R scripts folder.) The fourth subset of predictors was found not by data manipulation but by a Lasso model. Altogether, these sets of predictors formed the data bases for our models.

Loading Packages and Data

```r
library(tidyverse)
library(class)
library(scales)
library(stringr)
library(glmnet)
library(ridge)
library(plyr)
library(randomForest)
library(pls)
library(knitr)

#make code wrap!
opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE)
```

First, we loaded in and cleaned the training dataset:

```r
# reading in data
cancer_data_v <- read_delim(file = "Data/Unprocessed/data_set_ALL_AML_train.txt",
    delim = "\t")

# reading in the actual types of cancer are
```

```r
cancer_sample <- read_delim("Data/Unprocessed/table_ALL_AML_samples.txt", delim = "\t",
    skip = 4)

# getting index of each participant
index <- cancer_sample[[1]][2:39]

# getting type of cancer
type_cancer <- cancer_sample[[3]][2:39]

# merging index and type of cancer for each participant in the training set
index_type <- tibble(index, type_cancer)

# removing 'calls' variables
cancer_data_v <- cancer_data_v %>% select(-(seq(4, 78, by = 2)))


# transpose the data
cancer_transposed <- as_tibble(t(cancer_data_v))

# cleaning
colnames(cancer_transposed) <- cancer_transposed[2, ]

# removing extra columns
cancer_transposed <- cancer_transposed[-(1:2), ]

cancer_transposed <- cancer_transposed %>% mutate(cancer_type = index_type$type_cancer)

# rename
cancer_cleaning <- cancer_transposed

# Working with: cancer_cleaning

# Further cleaning the data, dimensions are now 40 x 7000:
cancer_cleaning <- cancer_cleaning %>% mutate(cancer_type = as.factor(cancer_type))

# check: cancer type should now be a factor, so check that a typeof() call
# returns type int.
typeof(cancer_cleaning$cancer_type)  # check - is correct.
```

```
## [1] "integer"
```

Next, we did the same for the testing dataset.

```r
# Repeat cleaning process for the testing dataset.

# reading in data
cancer_data_testing <- read_delim(file = "Data/Unprocessed/data_set_ALL_AML_independent.txt",
    delim = "\t")

# getting index of each participant in the test data
index_2 <- cancer_sample[[1]][41:74]

# getting type of cancer
type_cancer_2 <- cancer_sample[[3]][41:74]
```

```
# merging index and type of cancer for each participant in the testing set
index_type_2 <- tibble(index_2, type_cancer_2)

# removing calls
cancer_data_testing <- cancer_data_testing %>% select(-(seq(4, 78, by = 2)))


# transpose
cancer_transposed_2 <- as_tibble(t(cancer_data_testing))

# pick out the column names
colnames(cancer_transposed_2) <- cancer_transposed_2[2, ]

# removing extra columns

cancer_transposed_2 <- cancer_transposed_2[-(1:2), ]

cancer_transposed_2 <- cancer_transposed_2 %>% mutate(cancer_type = index_type_2$type_cancer_2)


cancer_cleaning_2 <- cancer_transposed_2

# Working with: cancer_cleaning

# Cleaning the 40 x 7000 data table:


cancer_cleaning_2 <- cancer_cleaning_2 %>% mutate(cancer_type = as.factor(cancer_type))

# again, make sure the type of cancer_type is correct.
typeof(cancer_cleaning_2$cancer_type)  # check - is correct.
```

```
## [1] "integer"
```

EDA

First, we identified the genes whose variance falls the in 99.5th percentile.

```
# Import cleaned train data
cancer <- read_csv("Data/Processed/cancerdata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   type = col_character(),
##   AB000464_at = col_double(),
##   AC002115_rna2_at = col_double(),
##   AJ000480_at = col_double(),
##   D00761_at = col_double(),
##   D13748_at = col_double(),
##   D38128_at = col_double(),
##   D38548_at = col_double(),
##   D49400_at = col_double(),
##   D61380_at = col_double(),
##   D63881_at = col_double(),
##   D86479_at = col_double(),
##   D86965_at = col_double(),
##   `HG2320-HT2416_at` = col_double(),
##   `HG2662-HT2758_at` = col_double(),
##   `HG3039-HT3200_at` = col_double(),
##   J04173_at = col_double(),
##   L18983_at = col_double(),
##   L20773_at = col_double(),
##   M13792_at = col_double()
##   # ... with 42 more columns
## )

## See spec(...) for full column specifications.
```

```r
colnames(cancer)[1] <- "type"

# # Calculate SD for each gene cancerSD <- cancer %>% gather(-type, key = 'gene',
# value = 'expression') %>% group_by(gene) %>% mutate(sd = sd(expression)) #
# Visualize SD distribution cancerSD %>% ggplot(aes(x = sd)) +
# geom_histogram(bins = 100) + ggtitle('SD distribution') #Percentile for gene
# SD's cancerSD$sd %>% quantile(prob = seq(0, 1, length = 201), type = 5) #99.5th
# percentile topgenes <- cancerSD %>% filter(sd >= 4636.25218) %>% select(-type,
# -expression) %>% unique()
```

The above script was used to select the 36 most variant genes. Since the same script is inexplicably failing to reproduce, we include a table of these "top" 36 genes below.

```r
top36genes <- read_csv("Data/Processed/top36genes.csv")
```

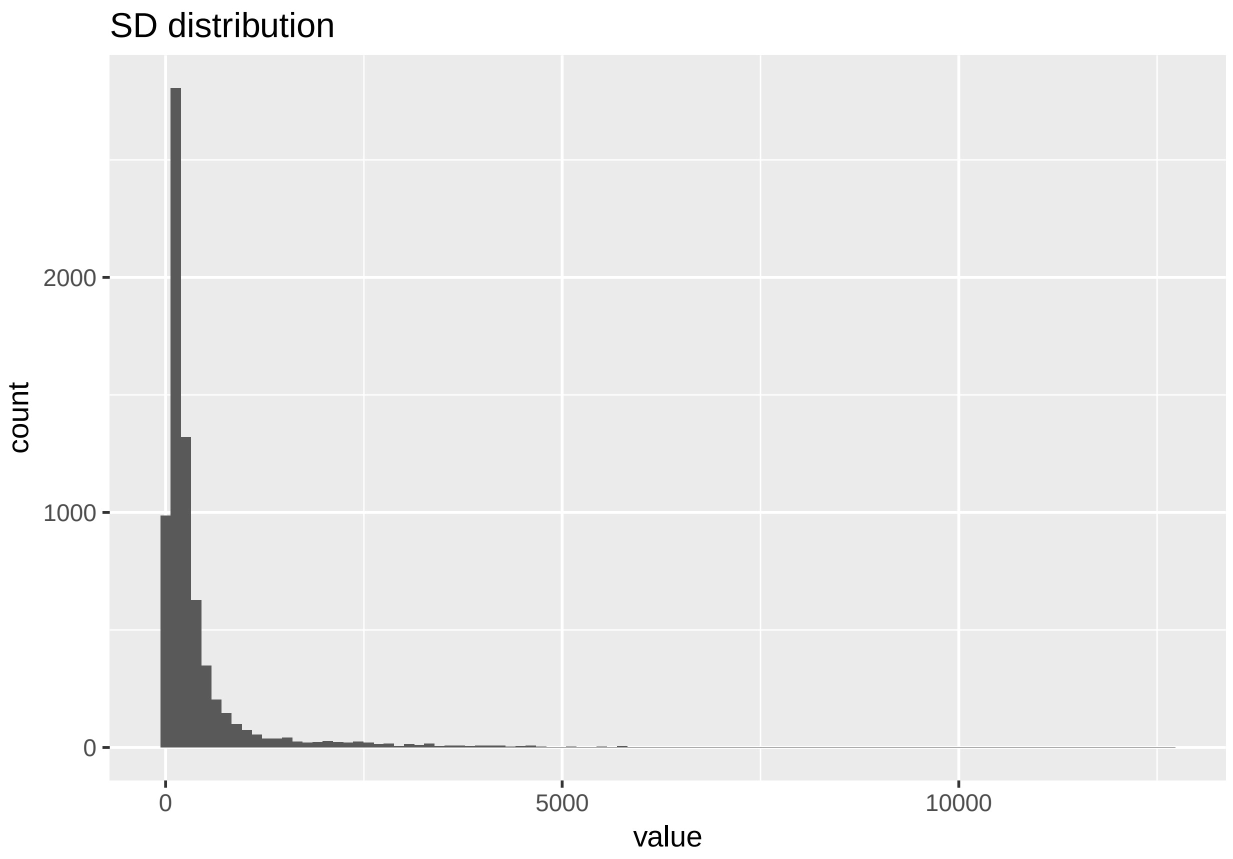
```
## Parsed with column specification:
## cols(
##   gene = col_character(),
##   sd = col_double()
## )
```

```r
top36genes
```

```
## # A tibble: 36 x 2
##    gene                         sd
##    <chr>                     <dbl>
##  1 hum_alu_at                 7546
##  2 AFFX-HUMRGE/M10098_5_at    5691
##  3 AFFX-HUMRGE/M10098_3_at    4993
##  4 AFFX-HUMGAPDH/M33197_5_at  6333
##  5 AFFX-HSAC07/X00351_5_at    5451
##  6 AFFX-HSAC07/X00351_M_at    5873
##  7 D64142_at                  4803
##  8 D79205_at                  4794
##  9 L06499_at                  5528
## 10 L19779_at                  5084
## # ... with 26 more rows
```

We compared the genes' variance distribution with a visualization.

```
# Visualize the distribution of variance for each gene
cancer %>% select(-type) %>% apply(2, sd) %>% as_tibble() %>% ggplot(aes(x = value)) +
    geom_histogram(bins = 100) + ggtitle("SD distribution")
```



We identified the 37 most informative genes– that is, the genes that correlate most strongly with only one of the two cancer classes.

```
# # Find lowest gene expression value genemin <- apply(cancer[,2:7130], 1, min)
# %>% min() # Set minimun expression value to zero by adding constant
# cancer[,2:7130] <- apply(cancer[,2:7130], 1, function (x) (x - genemin + 1)) #
# Log transorm each gene's expression distrivution cancer[,2:7130] <-
# apply(cancer[,2:7130],1, log) # Calulate means and SD's for each gene's
# expression cancer_summary <- cancer %>% gather(-type, key = 'gene', value =
# 'expression')%>% group_by(type, gene) %>% mutate(mean = mean(expression), sd =
# sd(expression)) %>% select(-expression) %>% unique() # cancer_summary %>%
# write_csv('cancer_summary.csv') ALL <- cancer_summary %>% filter(type=='ALL')
# %>% mutate(ALLmean = mean, ALLsd = sd) %>% ungroup() %>% select(-type, -mean,
# -sd) AML <- cancer_summary %>% filter(type=='AML') %>% mutate(AMLmean = mean,
# AMLsd = sd) %>% ungroup() %>% select(-type, -mean, -sd) # Calculate
# correleation of each gene with a class (positive values indicate ALL) Pcorr <-
# inner_join(ALL, AML, by = 'gene') %>% transmute(gene = gene, P = (ALLmean -
# AMLmean)/(ALLsd - AMLsd), Pabs = abs(P)) # 37 most informative genes.
# bestgenes <- Pcorr %>% arrange(desc(Pabs)) %>% .[1:37,]
```

The above script was used to select the 37 most informative genes. Since the same script is inexplicably failing to reproduce, we have included a table of the 37 most informative genes below.

```
best37genes <- read_csv("Data/Processed/best37genes.csv")
```

```
## Parsed with column specification:
## cols(
##   gene = col_character(),
##   description = col_character(),
##   P = col_double(),
##   Pabs = col_double()
## )
```

```
best37genes
```

```
## # A tibble: 37 x 4
##    gene           description                                   P   Pabs
##    <chr>          <chr>                                     <dbl>  <dbl>
##  1 J04621_at      SDC2 Syndecan 2 (heparan sulfate proteog~  2375   2375
##  2 U90904_at      Clone 23773 mRNA sequence                 - 841    841
##  3 U66616_at      SWI/SNF complex 170 KDa subunit (BAF170)~ - 359    359
##  4 U89335_cds2_at NOTCH4 gene (notch4) extracted from Huma~ - 315    315
##  5 L14754_at      IGHMBP2 DNA-binding protein (SMBP2)       - 242    242
##  6 U23942_at      CYP51 Cytochrome P450, 51 (lanosterol 14~   231    231
##  7 U58516_at      Breast epithelial antigen BA46 mRNA         143    143
##  8 L20965_at      Phosphodiesterase mRNA                      132    132
##  9 U06631_at      IEF SSP 9502 mRNA                         - 102    102
## 10 U70735_at      GB DEF = 34 kDa mov34 isologue mRNA       -  95.9   95.9
## # ... with 27 more rows
```

KNN with all predictors

Since the paper that accompanied this dataset suggested we try nonparametric methods, the first method we attempted was K-Nearest Neighbors (KNN) with varying levels of K. We first attempted to do so using *all* the predictors, before correcting with a smaller number of predictors. That first all-predictor KNN model is displayed below; its problem with high dimensionality, i.e. `p > n`, is addressed in subsequent KNN models further below.

```
# training set:
cancer_train <- cancer_cleaning

# testing set:
cancer_test <- cancer_cleaning_2

levels(cancer_train$cancer_type)
```

```
## [1] "ALL"  "ALL " "AML"  "AML "
```

```
levels(cancer_test$cancer_type)
```

```
## [1] "ALL"  "ALL " "AML"  "AML "
```

```
# NOTE: This was fixed below-- factors accidentally had 'AML' and 'AML ', etc.
# See the second declaration of cancer_train and cancer_test.




##### KNN
```

```r
## Set the seed.
set.seed(1)

## testing and training datasets:

# training matrix of predictors: remove the Y variable.
cancer_train_x <- cancer_train %>% select(-cancer_type)

# testing matrix of predictors: remove the Y variable.
cancer_test_x <- cancer_test %>% select(-cancer_type)


## vectors of the Y variables for the train and test sets:

cancer_train_y <- cancer_train$cancer_type

cancer_test_y <- cancer_test$cancer_type


## Make the model: use the knn() function in package `class`.

KNN_model <- knn(train = cancer_train_x, test = cancer_test_x, cl = cancer_train_y,
    k = 5)  # Experimentation with different values of k is continued below.

KNN_confusion <- table(KNN_model, cancer_test_y)
KNN_confusion
```

```
##          cancer_test_y
## KNN_model ALL ALL  AML AML
##      ALL    0    0   0    0
##      ALL    4   10   5    5
##      AML    0    0   0    0
##      AML    5    1   3    1
```

```r
KNN_accuracy_rate <- (KNN_confusion[1] + KNN_confusion[2] + KNN_confusion[5] + KNN_confusion[6] +
    KNN_confusion[11] + KNN_confusion[12] + KNN_confusion[15] + KNN_confusion[16])/nrow(cancer_test_x)
KNN_accuracy_rate
```

```
## [1] 0.5294118
```

```r
# Okay, why are there two versions of ALL and AML..? Is there a space or what?
# Could still find error by quadrant:

KNN_accuracy_rate <- (KNN_confusion[1] + KNN_confusion[2] + KNN_confusion[5] + KNN_confusion[6] +
    KNN_confusion[11] + KNN_confusion[12] + KNN_confusion[15] + KNN_confusion[16])/nrow(cancer_test_x)
KNN_accuracy_rate
```

```
## [1] 0.5294118
```

```r
### Accuracy of k = 5 with this seed is 52.9% - barely better than random guessing.

# How to fix:

# 1. Fix spaces in the 'AML ' etc. categories.
# 2. Try different values of k to find the best one.



## Fix 1: Factors

levels(cancer_train$cancer_type)
```

```
## [1] "ALL"  "ALL " "AML"  "AML "
```

```r
levels(cancer_test$cancer_type)
```

```
## [1] "ALL"  "ALL " "AML"  "AML "
```

```r
# It appears that there was an extra space in the variable names that threw off
# the first KNN model.

# To fix, mutate the variable:
cancer_train <- cancer_cleaning %>% mutate(cancer_type = as.factor(if_else(cancer_type ==
    "AML" | cancer_type == "AML ", "AML", "ALL")))  # again, necessary to call as.factor()

cancer_test <- cancer_cleaning_2 %>% mutate(cancer_type = as.factor(if_else(cancer_type ==
    "AML" | cancer_type == "AML ", "AML", "ALL")))

# Now check the levels again:

levels(cancer_train$cancer_type)
```

```
## [1] "ALL" "AML"
```

```r
levels(cancer_test$cancer_type)
```

```
## [1] "ALL" "AML"
```

```r
# Only two levels, 'AML' and 'ALL', just as the variable should be.



## Fix 2: Different folds of k.


# First, let's try to repeat k = 5 with the fixed factors.

cancer_train_x <- cancer_train %>% select(-cancer_type)
cancer_test_x <- cancer_test %>% select(-cancer_type)

cancer_train_y <- cancer_train$cancer_type
cancer_test_y <- cancer_test$cancer_type

KNN_model_2 <- knn(train = cancer_train_x, test = cancer_test_x, cl = cancer_train_y,
    k = 5)

KNN_confusion_2 <- table(KNN_model_2, cancer_test_y)
KNN_confusion_2
```

```
##            cancer_test_y
## KNN_model_2 ALL AML
##         ALL  14  10
##         AML   6   4
```

```
KNN_accuracy_rate_2 <- (KNN_confusion_2[1] + KNN_confusion_2[4])/nrow(cancer_test_x)
KNN_accuracy_rate_2
```

```
## [1] 0.5294118
```

```
knn_5_success_rate <- KNN_accuracy_rate_2

### Result: 52.9% accuracy. Altering folds:


# For k = 3:


KNN_model_k3 <- knn(train = cancer_train_x, test = cancer_test_x, cl = cancer_train_y,
    k = 3)  # Nearest 3 neighbors.

KNN_confusion_k3 <- table(KNN_model_k3, cancer_test_y)
KNN_confusion_k3
```

```
##             cancer_test_y
## KNN_model_k3 ALL AML
##          ALL  15   9
##          AML   5   5
```

```
KNN_accuracy_rate_k3 <- (KNN_confusion_k3[1] + KNN_confusion_k3[4])/nrow(cancer_test_x)
KNN_accuracy_rate_k3
```

```
## [1] 0.5882353
```

```
knn_3_success_rate <- KNN_accuracy_rate_k3

# Result: 58.8% accuracy if we use all the genes, not even winnowing out
# variables. k = 3.


## k-fold cross-validation

# KNN has its own cross-validation method in the `class` package.  NOTE: package
# details say the CV function is LOOCV, not k-fold.




# For our own exploratory interest: how well would a KNN model perform if given
# the entire dataset?

cancer_full <- rbind(cancer_train, cancer_test)
# Correct size. 72 rows, 2 less than cancer_sample which is dirty data and has
# two nonsense rows (titles).

# Remove Y var:
cancer_full_preds <- cancer_full %>% select(-cancer_type)

KNN_model_k3_cv <- knn.cv(train = cancer_full_preds, cl = cancer_full$cancer_type,
    k = 3)
```

```
# Find confusion: KNN_confusion_k3_cv <- table(KNN_model_k3_cv, cancer_test_y)

KNN_confusion_k3_LOOCV <- table(KNN_model_k3_cv, cancer_full$cancer_type)
KNN_confusion_k3_LOOCV
```

```
##
## KNN_model_k3_cv ALL AML
##             ALL  38   8
##             AML   9  17
```

```
KNN_accuracy_k3_LOOCV <- (KNN_confusion_k3_LOOCV[1] + KNN_confusion_k3_LOOCV[4])/nrow(cancer_full_preds
KNN_accuracy_k3_LOOCV
```

```
## [1] 0.7638889
```

```
# Result: 76% accuracy (whole dataset).

# But, this is for the entire dataset. Our working model needs to be based upon
# the training dataset only.


# Using the training dataset:
KNN_model_k3_cv_TRAIN <- knn.cv(train = cancer_train_x, cl = cancer_train$cancer_type,
    k = 3)
KNN_confusion_k3_cv_TRAIN <- table(KNN_model_k3_cv_TRAIN, cancer_train$cancer_type)
KNN_confusion_k3_cv_TRAIN
```

```
##
## KNN_model_k3_cv_TRAIN ALL AML
##                   ALL  26   4
##                   AML   1   7
```

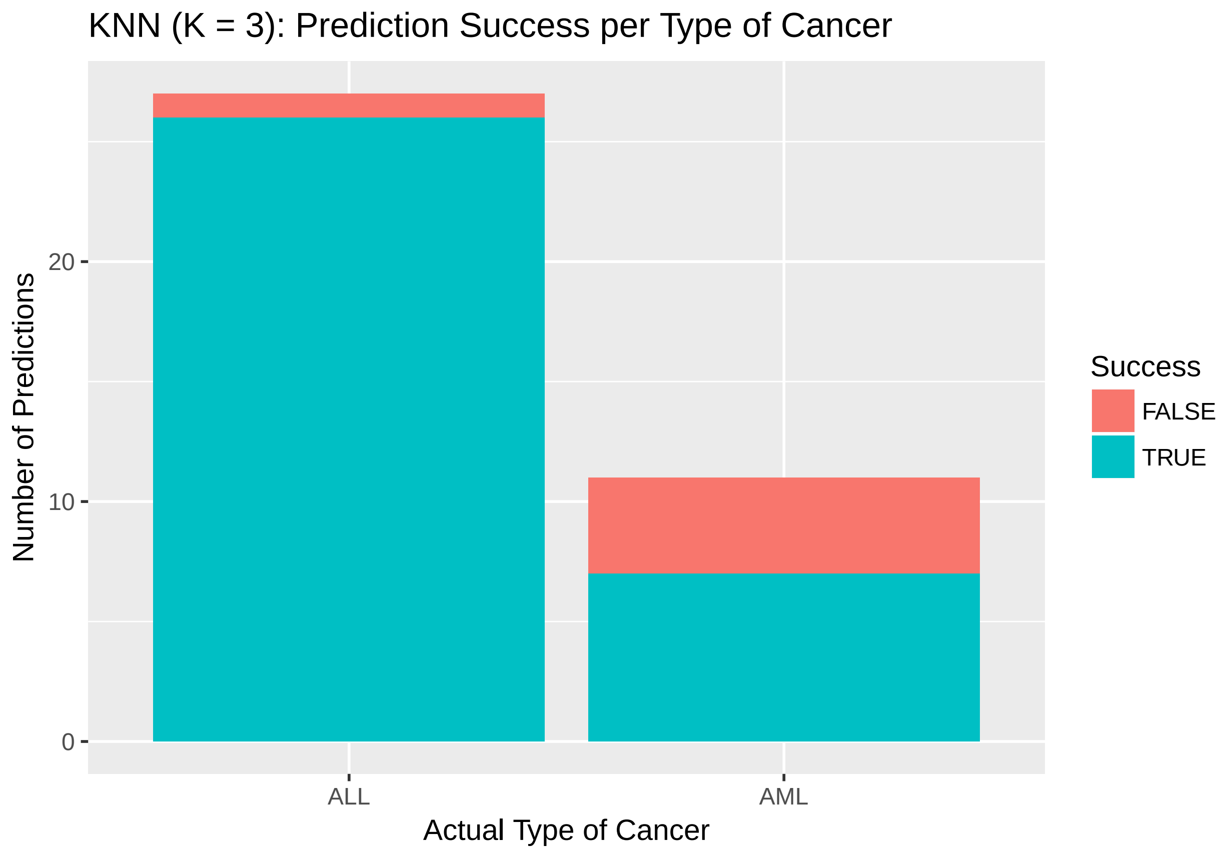
```
KNN_accuracy_k3_cv_TRAIN <- (KNN_confusion_k3_cv_TRAIN[1] + KNN_confusion_k3_cv_TRAIN[4])/nrow(cancer_t
KNN_accuracy_k3_cv_TRAIN
```

```
## [1] 0.8684211
```

```
# Result: 86.8% accuracy (all predictors).

knn_3_success_rate_cv <- KNN_accuracy_k3_cv_TRAIN

#
```

Visualizations for chosen KNN model using all predictors

```
# Metadata tibble
KNN_confusion_tibble <- tibble(Cancer_Type = c("ALL", "AML"), Predictions_Total = c(30,
    8), Predictions_Correct = c(26, 7), Predictions_Incorrect = c(4, 1))

# Predictions
KNN_tibble_predictions <- tibble(Actual_Type = cancer_train$cancer_type, Predicted_Type = KNN_model_k3_
    Success = if_else(Actual_Type == Predicted_Type, TRUE, FALSE))

ggplot(data = KNN_tibble_predictions, aes(x = Actual_Type, fill = Success)) + geom_bar() +
    ggtitle("KNN (K = 3): Prediction Success per Type of Cancer") + ylab("Number of Predictions") +
    xlab("Actual Type of Cancer")
```



After trying various k values for KNN, we ultimately settled upon a model which had k = 3, ultimately resulting in 33 correct predictions out of 38, or around 87% success rate. Even without winnowing down the predictors, this seems to be a very solid success rate. Below, we refined this model by using first the Lasso-chosen predictor set, and then the best variance-chosen predictor set. First, however, we performed the Lasso method, and also attempted to classify cancer type using Ridge Regression.

Ridge and Lasso predictors

```
grid <- 10^seq(10, -2, length = 100)

cancer_train$cancer_type <- as.character(cancer_train$cancer_type)
reboot <- revalue(cancer_train$cancer_type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```
cancer_train$cancer_type <- as.factor(reboot)

cancer_test$cancer_type <- as.character(cancer_test$cancer_type)
reboot2 <- revalue(cancer_test$cancer_type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```r
cancer_test$cancer_type <- as.factor(reboot2)

cancer_train_x <- cancer_train %>% select(-cancer_type)
cancer_test_x <- cancer_test %>% select(-cancer_type)

cancer_train_y <- cancer_train$cancer_type
cancer_test_y <- cancer_test$cancer_type

x.train <- data.matrix(cancer_train_x)
y.train <- cancer_train_y %>% as.numeric(cancer_train_y)


set.seed(1)
cv.out <- cv.glmnet(x.train, y.train, alpha = 0, family = "binomial", lambda = grid)
best.lambda <- cv.out$lambda.min

x.test <- data.matrix(cancer_test_x)
y.test <- cancer_test_y %>% as.numeric(cancer_test_y)

options(max.print = 1e+07)
out <- glmnet(x.test, y.test, alpha = 0, family = "binomial")


predictions_1 <- predict(out, type = "coefficients", s = best.lambda)

ridge_pred_prob <- predict(cv.out, s = best.lambda, newx = x.test, type = "response")

ridge_pred_cancer <- ifelse(ridge_pred_prob >= 0.5, 2, 1)

mean(ridge_pred_cancer != y.test)
```

```
## [1] 0.4117647
```

```r
table(ridge_pred_cancer, y.test)  # 58.8% success rate
```

```
##                  y.test
## ridge_pred_cancer  1  2
##                 1 16 10
##                 2  4  4
```

```r
ridge_success_rate_all <- 1 - mean(ridge_pred_cancer != y.test)

# Lasso

# library(lasso2)
set.seed(12)
cv.out.lasso <- cv.glmnet(x.train, y.train, alpha = 1, family = "binomial", lambda = grid)
best.lamdalasso <- cv.out.lasso$lambda.min

outlasso <- glmnet(x.test, y.test, alpha = 1, lambda = best.lamdalasso, family = "binomial")

lasso.coef <- predict(outlasso, type = "coefficients", s = best.lamdalasso)

temp <- lasso.coef %>% as.matrix() %>% as.data.frame()
```

```r
names(temp) <- "estimate"
temp$var_name <- rownames(temp)

lasso_var <- temp %>% filter(estimate != 0) %>% mutate(estimate = round(estimate,
    4))
```

```
## Warning: package 'bindrcpp' was built under R version 3.3.3
```

```r
lasso_var %>% as.tibble()
```

```
## # A tibble: 24 x 2
##     estimate var_name
##        <dbl> <chr>
##  1 -2.48     (Intercept)
##  2 -0.00760  D26361_at
##  3  0.00310  J03171_at
##  4  0.00300  L17325_at
##  5  0.00300  L40371_at
##  6  0.000800 M35252_at
##  7  0.000300 M64595_at
##  8  0.000100 M96326_rna1_at
##  9 -0.00480  S67156_at
## 10 -0.00130  U09284_at
## # ... with 14 more rows
```

```r
lasso_pred_prob <- predict(cv.out.lasso, s = best.lambda, newx = x.test, type = "response")

lasso_pred_cancer <- ifelse(lasso_pred_prob >= 0.5, 2, 1)

mean(lasso_pred_cancer != y.test)
```

```
## [1] 0.3823529
```

```r
table(lasso_pred_cancer, y.test)
```

```
##                  y.test
## lasso_pred_cancer  1  2
##                 1 16  9
##                 2  4  5
```

```r
lasso_success_rate_all <- 1 - mean(lasso_pred_cancer != y.test)
# This gives 23 variables.
```

This seemed to be by far the most informative method we had tried, as we could end up going from 7130 variables to roughly 23 variables. However, we may want to consider some form of standardization for our x values because these coefficients will end up being low due to the fact that the x values are extremely high and the y values (should be) either 0 or 1. Moreover, this method resulted in only a ~62% success rate among the predictions, which is lower than the current (though fault) KNN model using all predictors.

KNN with Lasso-chosen Predictors

Because `p >> n` for the prior analyses using KNN, we needed to re-run KNN with fewer variables than subjects in order to produce a sound model. Therefore, we tried running KNN with the lasso predictors to test the overall accuracy of the model.

```r
# The following performs a LOOCV (cross-validation) analysis, on K-Nearest
# Neighbors with k = 3, using the training dataset [change to testing dataset to
# see how k = 3 performs on test!], and ONLY on the Lasso variables!

set.seed(1)

# 1. First, use the training dataset with only the Lasso variables, which are
# currently stored in `temp`.

# Original, all-predictor variables for the model:

# cancer_train_x cancer_test_x cancer_train_y cancer_test_y


# Extract the lasso variables:

lasso_variables <- pull(lasso_var, var_name)
lasso_variables <- lasso_variables[2:24]  # Remove the intercept.

# Training dataset with lasso variables.
cancer_train_lassoVars <- cancer_train %>% select(lasso_variables, cancer_type)

# Testing dataset with lasso variables.
cancer_test_lassoVars <- cancer_test %>% select(lasso_variables, cancer_type)

# 2. Prepare the data: 'x' and 'y' versions.
cancer_train_lassoVars_x <- cancer_train_lassoVars %>% select(-cancer_type)
cancer_test_lassoVars_x <- cancer_test_lassoVars %>% select(-cancer_type)


# 3. Make the model - apply it to TRAIN dataset.

KNN_k3_cv_train_lassoVars <- knn.cv(train = cancer_train_lassoVars_x, cl = cancer_train_lassoVars$cance
    k = 4)

KNN_confusion_k3_cv_train_lassoVars <- table(KNN_k3_cv_train_lassoVars, cancer_train_lassoVars$cancer_t

KNN_confusion_k3_cv_train_lassoVars
```

```
##
## KNN_k3_cv_train_lassoVars ALL AML
##                       ALL  26   5
##                       AML   1   6
```

```r
KNN_accuracy_k3_cv_train_lassoVars <- (KNN_confusion_k3_cv_train_lassoVars[1] + KNN_confusion_k3_cv_trai

knn_3_success_rate_lasso <- KNN_accuracy_k3_cv_train_lassoVars

# Tried different values for 'k' by altering the integer in the model call above.
# The results:

# 84.2% success rate on TRAIN set, k = 2 84.2% success rate on TRAIN set, k = 3
# 89.5% success rate on TRAIN set, k = 4 -- BEST. Not k = 3, as we found with the
# all-predictor model.  86.8% success rate on TRAIN set, k = 5
```

Conclusion:

The `k = 4` KNN model performed the best on the *lasso-selected* variable train dataset with an accuracy of 89.5%. This is a *valid model* in terms of `p >> n`– that is, there are fewer variables than subjects: `p = 23` predictors, `n = 38` subjects.

Again, since the paper suggested nonparametric methods, we tried another KNN approach. This time, we calculated the 36 genes which had the highest variance - the logic being that genes with the highest variance would be most likely to be predictive of cancer-type (since the genes with lowest variance wouldn't really matter.) That approach is shown below.

KNN with Variance-chosen Predictors

```
# Get the variance variables:
variance_variables_sd <- read_csv("Data/Processed/top36genes.csv")
```

```
## Parsed with column specification:
## cols(
##   gene = col_character(),
##   sd = col_double()
## )
```

```
variance_variables <- pull(variance_variables_sd, gene)  # No intercept included. Just 36 variables.

# Prepare data:
cancer_train_varianceVars <- cancer_train %>% select(variance_variables, cancer_type)

cancer_test_varianceVars <- cancer_test %>% select(variance_variables, cancer_type)


# Prepare data: 'x' and 'y' versions.
cancer_train_varianceVars_x <- cancer_train_varianceVars %>% select(-cancer_type)
cancer_test_varianceVars_x <- cancer_test_varianceVars %>% select(-cancer_type)

set.seed(17)

# Make the model - apply it to the train set:

KNN_k3_cv_train_varianceVars <- knn.cv(train = cancer_train_varianceVars_x, cl = cancer_train_varianceVa
    k = 3)
KNN_confusion_k3_cv_train_varianceVars <- table(KNN_k3_cv_train_varianceVars, cancer_train_varianceVars$
KNN_confusion_k3_cv_train_varianceVars
```

```
##
## KNN_k3_cv_train_varianceVars ALL AML
##                          ALL  26   2
##                          AML   1   9
```

```
KNN_accuracy_k3_cv_train_varianceVars <- (KNN_confusion_k3_cv_train_varianceVars[1] +
    KNN_confusion_k3_cv_train_varianceVars[4])/nrow(cancer_train_varianceVars_x)

knn_3_sucess_rate_top <- KNN_accuracy_k3_cv_train_varianceVars

# Again, we chose different values of 'k' to see what would give the best
# performance of the model, this time on the 36 best variance-selected variables.
# 89.5% success rate, k = 2, variance variables, TRAIN 92.1% success rate, k = 3,
# variance variables, TRAIN -- BEST.  89.5% success rate, k = 4, variance
# variables, TRAIN 86.8% success rate, k = 5, variance variables, TRAIN
```

Conclusion:

The `k = 3` KNN model performed the best on the *variance-selected* variable train dataset with an accuracy of 92.1%. This model is also a *valid model* in terms of `p >> n`– that is, there are fewer variables than subjects: `p = 36` predictors, `n = 38` subjects.

PCR Approach

Next, we also wanted to try a PCR approach. Using a similar method to the one in the book (except in this scenario, where having AML is equivalent to a prediction of "1" and ALL is equivalent ot "0"), we implemented a PCR approach.

```
###### CLEANING DATA############

# reading in data, removing calls - not using them
cancer_data <- read_delim(file = "Data/Unprocessed/data_set_ALL_AML_train.txt", delim = "\t") %>%
    select(-seq(4, 78, by = 2))
```

```
## Warning: Duplicated column names deduplicated: 'call' => 'call_1' [6],
## 'call' => 'call_2' [8], 'call' => 'call_3' [10], 'call' => 'call_4' [12],
## 'call' => 'call_5' [14], 'call' => 'call_6' [16], 'call' => 'call_7' [18],
## 'call' => 'call_8' [20], 'call' => 'call_9' [22], 'call' => 'call_10' [24],
## 'call' => 'call_11' [26], 'call' => 'call_12' [28], 'call' =>
## 'call_13' [30], 'call' => 'call_14' [32], 'call' => 'call_15' [34], 'call'
## => 'call_16' [36], 'call' => 'call_17' [38], 'call' => 'call_18' [40],
## 'call' => 'call_19' [42], 'call' => 'call_20' [44], 'call' =>
## 'call_21' [46], 'call' => 'call_22' [48], 'call' => 'call_23' [50], 'call'
## => 'call_24' [52], 'call' => 'call_25' [54], 'call' => 'call_26' [56],
## 'call' => 'call_27' [58], 'call' => 'call_28' [60], 'call' =>
## 'call_29' [62], 'call' => 'call_30' [64], 'call' => 'call_31' [66], 'call'
## => 'call_32' [68], 'call' => 'call_33' [70], 'call' => 'call_34' [72],
## 'call' => 'call_35' [74], 'call' => 'call_36' [76], 'call' =>
## 'call_37' [78]
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   `1` = col_integer(),
##   `2` = col_integer(),
##   `3` = col_integer(),
##   `4` = col_integer(),
##   `5` = col_integer(),
##   `6` = col_integer(),
##   `7` = col_integer(),
##   `8` = col_integer(),
##   `9` = col_integer(),
##   `10` = col_integer(),
##   `11` = col_integer(),
##   `12` = col_integer(),
##   `13` = col_integer(),
##   `14` = col_integer(),
##   `15` = col_integer(),
##   `16` = col_integer(),
##   `17` = col_integer(),
##   `18` = col_integer(),
##   `19` = col_integer(),
##   `20` = col_integer()
##   # ... with 18 more columns
## )

## See spec(...) for full column specifications.

## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)

## Warning: 7129 parsing failures.
## row # A tibble: 5 x 5 col     row col   expected   actual     file
## ... ................. ... ..............................................................
## See problems(...) for more details.
```

```r
# reading in what they're actual types of cancer are
cancer_sample <- read_delim("Data/Unprocessed/table_ALL_AML_samples.txt", delim = "\t",
    skip = 4)
```

```
## Warning: Missing column names filled in: 'X1' [1], 'X2' [2], 'X3' [3],
## 'X4' [4], 'X6' [6], 'X7' [7], 'X8' [8], 'X9' [9], 'X10' [10]

## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_character(),
##   X3 = col_character(),
##   X4 = col_character(),
##   `            (if ALL)       (if AML)` = col_character(),
##   X6 = col_character(),
##   X7 = col_character(),
##   X8 = col_character(),
##   X9 = col_character(),
##   X10 = col_character(),
##   ` Response` = col_character()
## )

## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)

## Warning: 74 parsing failures.
## row # A tibble: 5 x 5 col     row col   expected   actual     file
## ... ................. ... ..............................................................
## See problems(...) for more details.
```

```r
# getting index
index <- cancer_sample[[1]][2:39]

# getting type of cancer
type_cancer <- cancer_sample[[3]][2:39] %>% str_trim()

# merging them
index_type <- tibble(index, type_cancer)
```

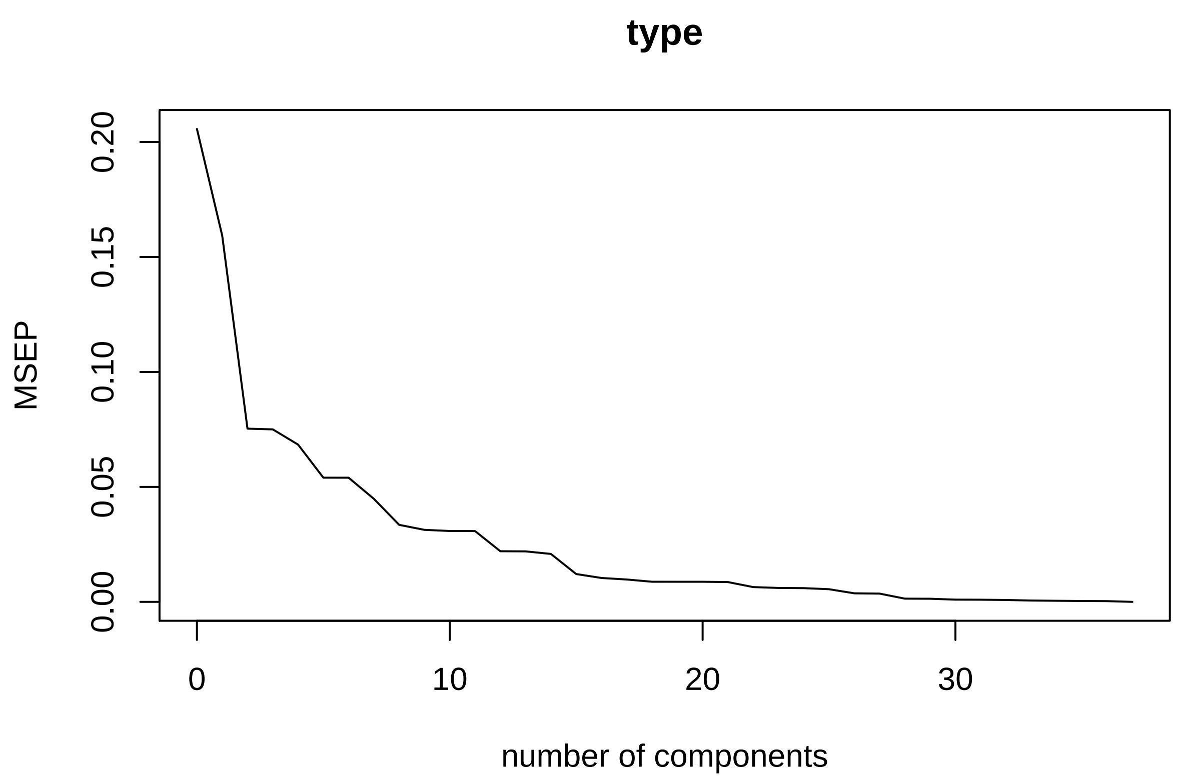
```r
# reordering the data to make it easier to work with
cancer_data_reord <- cancer_data[, c(1:29, 35:40, 30:34)]

transposed <- as.tibble(t(cancer_data_reord))

colnames(transposed) <- transposed[2, ]

# removing extra columns
transposed <- transposed[-(1:2), ]

# merging type of cancer with gene data
transposed <- transposed %>% mutate(type = index_type$type_cancer, type = factor(type),
    type = ifelse(type == "AML", 1, 0))

transposed <- transposed %>% mutate(type = ifelse(type == "AML", 1, 0))


# rearranging data purely for viewing purposes
transposed <- transposed[, c(7130, 1:7129)]

transposed[, c(2:7130)] <- lapply(transposed[, c(2:7130)], function(x) as.numeric(as.character(x)))


################ GETTING TEST DATA##############
cancer_data_testing <- read_delim(file = "Data/Unprocessed/data_set_ALL_AML_independent.txt",
    delim = "\t")
```

```
## Warning: Duplicated column names deduplicated: 'call' => 'call_1' [6],
## 'call' => 'call_2' [8], 'call' => 'call_3' [10], 'call' => 'call_4' [12],
## 'call' => 'call_5' [14], 'call' => 'call_6' [16], 'call' => 'call_7' [18],
## 'call' => 'call_8' [20], 'call' => 'call_9' [22], 'call' => 'call_10' [24],
## 'call' => 'call_11' [26], 'call' => 'call_12' [28], 'call' =>
## 'call_13' [30], 'call' => 'call_14' [32], 'call' => 'call_15' [34], 'call'
## => 'call_16' [36], 'call' => 'call_17' [38], 'call' => 'call_18' [40],
## 'call' => 'call_19' [42], 'call' => 'call_20' [44], 'call' =>
## 'call_21' [46], 'call' => 'call_22' [48], 'call' => 'call_23' [50], 'call'
## => 'call_24' [52], 'call' => 'call_25' [54], 'call' => 'call_26' [56],
## 'call' => 'call_27' [58], 'call' => 'call_28' [60], 'call' =>
## 'call_29' [62], 'call' => 'call_30' [64], 'call' => 'call_31' [66], 'call'
## => 'call_32' [68], 'call' => 'call_33' [70]
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   `39` = col_integer(),
##   `40` = col_integer(),
##   `42` = col_integer(),
##   `47` = col_integer(),
##   `48` = col_integer(),
##   `49` = col_integer(),
##   `41` = col_integer(),
##   `43` = col_integer(),
##   `44` = col_integer(),
##   `45` = col_integer(),
##   `46` = col_integer(),
##   `70` = col_integer(),
##   `71` = col_integer(),
##   `72` = col_integer(),
##   `68` = col_integer(),
##   `69` = col_integer(),
##   `67` = col_integer(),
##   `55` = col_integer(),
##   `56` = col_integer(),
##   `59` = col_integer()
##   # ... with 14 more columns
## )
## See spec(...) for full column specifications.
```

```r
# getting index - of each participant.
index_2 <- cancer_sample[[1]][41:74]

# getting type of cancer
type_cancer_2 <- cancer_sample[[3]][41:74] %>% str_trim()

# merging them
index_type_2 <- tibble(index_2, type_cancer_2)

# removing calls - not using theme
cancer_data_testing <- cancer_data_testing %>% select(-(seq(4, 78, by = 2)))

# Transpose
cancer_transposed_2 <- as_tibble(t(cancer_data_testing))

# Pick out the rows? cols?
colnames(cancer_transposed_2) <- cancer_transposed_2[2, ]

# removing extra columns
cancer_transposed_2 <- cancer_transposed_2[-(1:2), ]

cancer_transposed_2 <- cancer_transposed_2 %>% mutate(type = index_type_2$type_cancer_2)

cancer_cleaning_2 <- cancer_transposed_2

# Cleaning the 40 x 7000 data table:
cancer_cleaning_2 <- cancer_cleaning_2 %>% mutate(type = as.factor(type_cancer_2),
    type = ifelse(type == "AML", 1, 0))

cancer_cleaning_2 <- cancer_cleaning_2 %>% mutate(type = ifelse(type == "AML", 1,
    0))

# reorganizes it
cancer_cleaning_2 <- cancer_cleaning_2[, c(7130, 1:7129)]

# converting all characters to numerics
cancer_cleaning_2[, c(2:7130)] <- lapply(cancer_cleaning_2[, c(2:7130)], function(x) as.numeric(as.char

##################### LITERALLY JUST USING PCR
```

```
# training dataset
pcr_fit_train <- pcr(type ~ ., data = transposed)

validationplot(pcr_fit_train, val.type = "MSEP")
```



```
pcr_pred <- predict(pcr_fit_train, cancer_cleaning_2, ncomp = 30)

pred_tibble <- tibble(predictions = as.vector(pcr_pred)) %>% mutate(type = cancer_cleaning_2$type,
    new_pred = ifelse(predictions > 0.5, 1, 0))

table(pred_tibble$type, pred_tibble$new_pred)
```

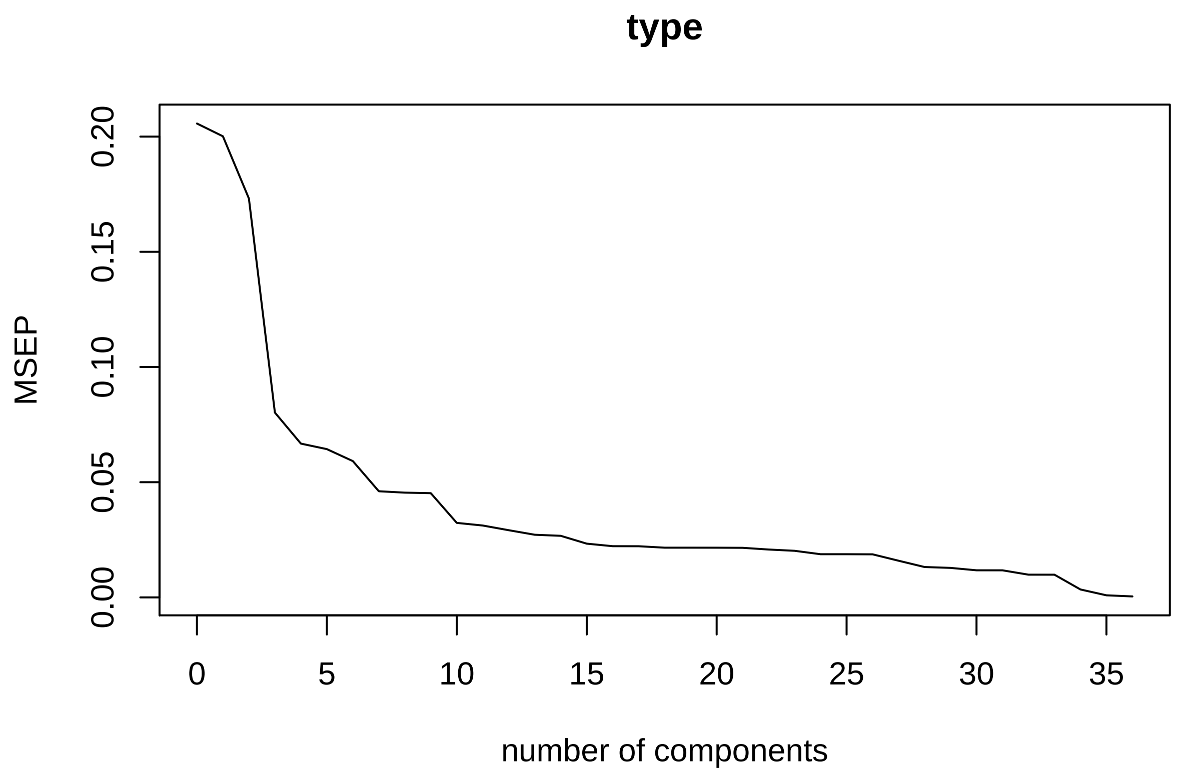
```
##
##       0  1
##   0 12  8
##   1  7  7
```

```
pcr_30_comp_success_rate <- (19)/34  #55.8% correct prediction rate
```

PCR Approach on Only Top (Highest Variance) Data

```
# PCR on only top genes
colnames(cancer_train_varianceVars)[colnames(cancer_train_varianceVars) == "cancer_type"] <- "type"
```

```r
cancer_train_varianceVars <- cancer_train_varianceVars %>% mutate(type = ifelse(type ==
    "AML", 1, 0))

cancer_train_varianceVars[] <- lapply(cancer_train_varianceVars, function(x) as.numeric(x))

pcr_fit_train_top <- pcr(type ~ ., data = cancer_train_varianceVars)

validationplot(pcr_fit_train_top, val.type = "MSEP")
```



```r
pcr_pred_top <- predict(pcr_fit_train_top, cancer_cleaning_2, ncomp = 30)

pred_tibble_top <- tibble(predictions = as.vector(pcr_pred_top)) %>% mutate(type = cancer_cleaning_2$typ
    new_pred = ifelse(predictions > 0.5, 1, 0))

table(pred_tibble_top$type, pred_tibble_top$new_pred)
```

```
##
##      0  1
##   0 10 10
##   1  6  8
```

```r
pcr_30_comp_top_success_rate <- 21/34  #61.76% success rate
```

When we used every gene, PCR had a success rate of around 61%. However, when we only used the top dataset, this success rate fell to around 56%. This makes sense - PCR is good at dealing with high dimensionality, and reduces the variables into components anyway. We might as well let it deal with the entire dataset and produce the best components possible, rather than narrowing it down beforehand.

Random Forest

```r
# These two lines likely don't do this accurately.
cancer_test_lassoVars <- rename(cancer_test_lassoVars, c(`HG3731-HT4001_r_at` = "HG3731_HT4001_r_at"))

cancer_train_lassoVars <- rename(cancer_train_lassoVars, c(`HG3731-HT4001_r_at` = "HG3731_HT4001_r_at")

lasso_rf <- randomForest(cancer_type ~ ., data = cancer_test_lassoVars)
lasso_rf
```

```
##
## Call:
##  randomForest(formula = cancer_type ~ ., data = cancer_test_lassoVars)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##     ALL AML class.error
## ALL  20   0           0
## AML   0  14           0
```

```r
# 0% error rate! This either perfectly matches the report or is wrong.

# Will try doing a different approach.

lasso_rf1 <- randomForest(cancer_type ~ ., data = cancer_train_lassoVars)
lasso_rf1
```

```
##
## Call:
##  randomForest(formula = cancer_type ~ ., data = cancer_train_lassoVars)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 21.05%
## Confusion matrix:
##     ALL AML class.error
## ALL  27   0   0.0000000
## AML   8   3   0.7272727
```

```r
predTrain <- predict(lasso_rf1, cancer_train_lassoVars, type = "class")
table(predTrain, cancer_train_lassoVars$cancer_type)
```

```
##
## predTrain ALL AML
##       ALL  27   0
##       AML   0  11
```

```r
predValid <- predict(lasso_rf1, cancer_test_lassoVars, type = "class")
mean(predValid == cancer_test_lassoVars$cancer_type)
```

```
## [1] 0.7352941
```

```
table(predValid, cancer_test_lassoVars$cancer_type)
```

```
##
## predValid ALL AML
##       ALL  19   8
##       AML   1   6
```

```
random_forest_success_rate <- mean(predValid == cancer_test_lassoVars$cancer_type)
```

Although we did not formally learn about the Random Forest statistical learning method in class, we applied it here with some apparent success, at least compared to several other models. While it isn't perfect, the Random Forest model resulted in a ~73% success rate.

Best 37 Ridge, Lasso, Random Forest

Next, we applied Ridge, Lasso, and Random Forest methods to the 37 genes we determined to be the "best":

```
best37 <- read_csv("Data/Processed/best37data.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   type = col_character(),
##   X54667_s_at = col_double()
## )

## See spec(...) for full column specifications.
```

```
print(colnames(best37))
```

```
##  [1] "type"           "J04621_at"      "U90904_at"      "U66616_at"
##  [5] "U89335_cds2_at" "L14754_at"      "U23942_at"      "U58516_at"
##  [9] "L20965_at"      "U06631_at"      "U70735_at"      "M13903_at"
## [13] "X58964_at"      "L04656_at"      "U27655_at"      "U60061_at"
## [17] "D21255_at"      "M38591_at"      "J00277_at"      "X13810_s_at"
## [21] "L32977_at"      "U90912_at"      "U78095_at"      "M23114_at"
## [25] "X07203_at"      "S67070_at"      "U80669_at"      "M24461_at"
## [29] "U11036_at"      "X89416_at"      "U20860_at"      "M83308_at"
## [33] "M87789_s_at"    "U01833_at"      "L36642_at"      "X74331_at"
## [37] "X02875_s_at"    "X54667_s_at"
```

```
best37t <- cancer_cleaning_2[, c("type", "J04621_at", "U90904_at", "U66616_at", "U89335_cds2_at",
    "L14754_at", "U23942_at", "U58516_at", "L20965_at", "U06631_at", "U70735_at",
    "M13903_at", "X58964_at", "L04656_at", "U27655_at", "U60061_at", "D21255_at",
    "M38591_at", "J00277_at", "X13810_s_at", "L32977_at", "U90912_at", "U78095_at",
    "M23114_at", "X07203_at", "S67070_at", "U80669_at", "M24461_at", "U11036_at",
    "X89416_at", "U20860_at", "M83308_at", "M87789_s_at", "U01833_at", "L36642_at",
    "X74331_at", "X02875_s_at", "X54667_s_at")]

best37$type <- as.character(best37$type)
best37reboot <- revalue(best37$type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```
best37$type <- as.factor(best37reboot)
```

```r
best37t$type <- as.character(best37t$type)
best37treboot <- revalue(best37t$type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```r
best37t$type <- as.factor(best37treboot)

best37x <- best37 %>% select(-type)

best37y <- best37$type

best37tx <- best37t %>% select(-type)

best37ty <- best37t$type

best37x <- data.matrix(best37x)
best37y <- best37y %>% as.numeric(best37y)

best37tx <- data.matrix(best37tx)
best37ty <- best37ty %>% as.numeric(best37ty)

# Ridge

grid <- 10^seq(10, -2, length = 100)

set.seed(1)
cv.37 <- cv.glmnet(best37x, best37y, alpha = 0, family = "binomial", lambda = grid)
best.lambda37 <- cv.37$lambda.min

b37 <- glmnet(best37tx, best37ty, alpha = 0, family = "binomial")

ridge_pred37 <- predict(b37, type = "coefficients", s = best.lambda37)

ridge_pred37_prob <- predict(cv.37, s = best.lambda37, newx = best37tx, type = "response")

ridge_pred37_cancer <- ifelse(ridge_pred37_prob >= 0.5, 2, 1)

mean(ridge_pred37_cancer != best37ty)
```

```
## [1] 0.4117647
```

```r
table(ridge_pred37_cancer, best37ty)
```

```
##                    best37ty
## ridge_pred37_cancer  1  2
##                   1 18 12
##                   2  2  2
```

```r
ridge_best_success_rate <- 1 - mean(ridge_pred37_cancer != best37ty)

# Lasso

set.seed(2)

cv.37lasso <- cv.glmnet(best37x, best37y, alpha = 1, family = "binomial", lambda = grid)
```

```r
best.lambda37lasso <- cv.37lasso$lambda.min

b37lasso <- glmnet(best37tx, best37ty, alpha = 1, family = "binomial")

ridge_pred37lasso <- predict(b37lasso, type = "coefficients", s = best.lambda37lasso)
ridge_pred37lasso
```

```
## 38 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)    -1.430083e+00
## J04621_at       .
## U90904_at       6.588642e-04
## U66616_at       .
## U89335_cds2_at -5.551813e-04
## L14754_at      -1.966761e-03
## U23942_at      -4.218863e-03
## U58516_at       .
## L20965_at       .
## U06631_at       .
## U70735_at      -2.076996e-03
## M13903_at       .
## X58964_at       .
## L04656_at       .
## U27655_at       5.327610e-04
## U60061_at       1.266651e-03
## D21255_at      -5.789193e-03
## M38591_at       .
## J00277_at       .
## X13810_s_at     .
## L32977_at       9.128460e-04
## U90912_at       4.481168e-04
## U78095_at      -2.028597e-04
## M23114_at       .
## X07203_at       .
## S67070_at       .
## U80669_at       .
## M24461_at       .
## U11036_at       3.408963e-03
## X89416_at       .
## U20860_at       .
## M83308_at      -3.104650e-03
## M87789_s_at     7.613653e-05
## U01833_at       .
## L36642_at      -6.046759e-03
## X74331_at      -2.865276e-04
## X02875_s_at     1.066021e-03
## X54667_s_at     .
```

```r
ridge_pred37lasso_prob <- predict(cv.37lasso, s = best.lambda37lasso, newx = best37tx,
    type = "response")

ridge_pred37lasso_cancer <- ifelse(ridge_pred37lasso_prob >= 0.5, 2, 1)

mean(ridge_pred37lasso_cancer != best37ty)
```

```
## [1] 0.3529412
```

```
table(ridge_pred37lasso_cancer, best37ty)
```

```
##                          best37ty
## ridge_pred37lasso_cancer  1  2
##                        1 18 10
##                        2  2  4
```

```
lasso_best_success_rate <- 1 - mean(ridge_pred37lasso_cancer != best37ty)

# Random Forest

rf37 <- randomForest(type ~ ., data = best37)
rf37
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = best37)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 13.16%
## Confusion matrix:
##     ALL AML class.error
## ALL  27   0   0.0000000
## AML   5   6   0.4545455
```

```
predTrain_37 <- predict(rf37, best37, type = "class")
table(predTrain_37, best37$type)
```

```
##
## predTrain_37 ALL AML
##          ALL  27   0
##          AML   0  11
```

```
predValid_37 <- predict(rf37, best37t, type = "class")
table(predValid_37, best37t$type)
```

```
##
## predValid_37  0  1
##          ALL 20 12
##          AML  0  2
```

```
random_forest_best_success_rate <- (22/34)  #64.7% sucess rate
```

Top Genes Ridge, Lasso, Random Forest

Finally, we applied Ridge, Lasso, and Random Forest methods to the genes we determined to have the highest variance:

```
top_datatrain <- read_csv("Data/Processed/topdata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   type = col_character(),
##   X00351_f_at = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
print(colnames(top_datatrain))
```

```
##  [1] "type"                        "hum_alu_at"
##  [3] "AFFX-HUMRGE/M10098_5_at"     "AFFX-HUMRGE/M10098_3_at"
##  [5] "AFFX-HUMGAPDH/M33197_5_at" "AFFX-HSAC07/X00351_5_at"
##  [7] "AFFX-HSAC07/X00351_M_at"     "D64142_at"
##  [9] "D79205_at"                   "L06499_at"
## [11] "L19779_at"                   "M11147_at"
## [13] "M12886_at"                   "M27891_at"
## [15] "M69043_at"                   "M91036_rna1_at"
## [17] "X00274_at"                   "X82240_rna1_at"
## [19] "Y00433_at"                   "Z84721_cds2_at"
## [21] "D86974_at"                   "D49824_s_at"
## [23] "M25079_s_at"                 "HG1428-HT1428_s_at"
## [25] "X57351_s_at"                 "V00594_s_at"
## [27] "V00599_s_at"                 "M13560_s_at"
## [29] "M14483_rna1_s_at"            "Y00787_s_at"
## [31] "Z19554_s_at"                 "M63438_s_at"
## [33] "X00437_s_at"                 "X56681_s_at"
## [35] "HG2887-HT3031_at"            "M33600_f_at"
## [37] "X00351_f_at"
```

```
top_datatest <- cancer_cleaning_2[, c("type", "hum_alu_at", "AFFX-HUMRGE/M10098_5_at",
    "AFFX-HUMRGE/M10098_3_at", "AFFX-HUMGAPDH/M33197_5_at", "AFFX-HSAC07/X00351_5_at",
    "AFFX-HSAC07/X00351_M_at", "D64142_at", "D79205_at", "L06499_at", "L19779_at",
    "M11147_at", "M12886_at", "M27891_at", "M69043_at", "M91036_rna1_at", "X00274_at",
    "X82240_rna1_at", "Y00433_at", "Z84721_cds2_at", "D86974_at", "D49824_s_at",
    "M25079_s_at", "HG1428-HT1428_s_at", "X57351_s_at", "V00594_s_at", "V00599_s_at",
    "M13560_s_at", "M14483_rna1_s_at", "Y00787_s_at", "Z19554_s_at", "M63438_s_at",
    "X00437_s_at", "X56681_s_at", "HG2887-HT3031_at", "M33600_f_at", "X00351_f_at")]

top_datatrain <- rename(top_datatrain, c(`AFFX-HUMRGE/M10098_5_at` = "AFFX_HUMRGE_M10098_5_at",
    `AFFX-HUMRGE/M10098_3_at` = "AFFX_HUMRGE_M10098_3_at", `AFFX-HUMGAPDH/M33197_5_at` = "AFFX_HUMGAPDH
    `AFFX-HSAC07/X00351_5_at` = "AFFX_HSAC07_X00351_5_at", `AFFX-HSAC07/X00351_M_at` = "AFFX_HSAC07_X00
    `HG1428-HT1428_s_at` = "HG1428_HT1428_s_at", `HG2887-HT3031_at` = "HG2887_HT3031_at"))

top_datatest <- rename(top_datatest, c(type = "type", `AFFX-HUMRGE/M10098_5_at` = "AFFX_HUMRGE_M10098_5
    `AFFX-HUMRGE/M10098_3_at` = "AFFX_HUMRGE_M10098_3_at", `AFFX-HUMGAPDH/M33197_5_at` = "AFFX_HUMGAPDH
    `AFFX-HSAC07/X00351_5_at` = "AFFX_HSAC07_X00351_5_at", `AFFX-HSAC07/X00351_M_at` = "AFFX_HSAC07_X00
    `HG1428-HT1428_s_at` = "HG1428_HT1428_s_at", `HG2887-HT3031_at` = "HG2887_HT3031_at"))

top_datatrain$type <- as.character(top_datatrain$type)
topreboot <- revalue(top_datatrain$type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```
top_datatrain$type <- as.factor(topreboot)

top_datatest$type <- as.character(top_datatest$type)
```

```r
topreboot1 <- revalue(top_datatest$type, c(`ALL ` = "ALL", `AML ` = "AML"))
```

```
## The following `from` values were not present in `x`: ALL , AML
```

```r
top_datatest$type <- as.factor(topreboot1)

top_xtrain <- top_datatrain %>% select(-type)

top_ytrain <- top_datatrain$type

top_xtest <- top_datatest %>% select(-type)

top_ytest <- top_datatest$type

top_xtrain <- data.matrix(top_xtrain)
top_ytrain <- top_ytrain %>% as.numeric(top_ytrain)

top_xtest <- data.matrix(top_xtest)
top_ytest <- top_ytest %>% as.numeric(top_ytest)

# Ridge

grid <- 10^seq(10, -2, length = 100)

set.seed(1)
cv.top <- cv.glmnet(top_xtrain, top_ytrain, alpha = 0, family = "binomial", lambda = grid)
best.lambdatop <- cv.top$lambda.min

top <- glmnet(top_xtest, top_ytest, alpha = 0, family = "binomial")

ridge_predtop <- predict(top, type = "coefficients", s = best.lambdatop)

ridge_predtop_prob <- predict(cv.top, s = best.lambdatop, newx = top_xtest, type = "response")

ridge_predtop_cancer <- ifelse(ridge_predtop_prob >= 0.5, 2, 1)

mean(ridge_predtop_cancer != top_ytest)
```

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

```r
table(ridge_predtop_cancer, top_ytest)
```

```
##                     top_ytest
## ridge_predtop_cancer  1  2
##                    1 11  8
##                    2  9  6
```

```r
ridge_top_success_rate <- 1 - mean(ridge_predtop_cancer != top_ytest)

# Lasso

set.seed(2)

cv.toplasso <- cv.glmnet(top_xtrain, top_ytrain, alpha = 1, family = "binomial",
    lambda = grid)
```

```r
best.lambdatoplasso <- cv.top$lambda.min

toplasso <- glmnet(top_xtest, top_ytest, alpha = 1, family = "binomial")

ridge_predtoplasso <- predict(toplasso, type = "coefficients", s = best.lambdatoplasso)
ridge_predtoplasso
```

```
## 37 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                7.115287e+00
## hum_alu_at                 1.321817e-04
## AFFX_HUMRGE_M10098_5_at   -8.370409e-05
## AFFX_HUMRGE_M10098_3_at    .
## AFFX_HUMGAPDH_M33197_5_at  .
## AFFX_HSAC07_X00351_5_at    1.153790e-04
## AFFX_HSAC07_X00351_M_at    .
## D64142_at                  .
## D79205_at                 -2.983951e-04
## L06499_at                 -2.565293e-04
## L19779_at                 -5.063444e-06
## M11147_at                 -1.052127e-04
## M12886_at                  7.139771e-04
## M27891_at                  .
## M69043_at                  .
## M91036_rna1_at            -2.999686e-04
## X00274_at                 -2.328257e-05
## X82240_rna1_at             5.484010e-05
## Y00433_at                  1.911081e-05
## Z84721_cds2_at             1.366102e-04
## D86974_at                  .
## D49824_s_at               -7.558789e-05
## M25079_s_at                .
## HG1428_HT1428_s_at         1.786373e-05
## X57351_s_at               -1.768149e-04
## V00594_s_at                8.010036e-05
## V00599_s_at               -3.162851e-05
## M13560_s_at                .
## M14483_rna1_s_at           .
## Y00787_s_at               -5.507276e-05
## Z19554_s_at                2.235445e-04
## M63438_s_at                1.088892e-04
## X00437_s_at                .
## X56681_s_at               -4.695495e-05
## HG2887_HT3031_at          -1.056423e-04
## M33600_f_at                .
## X00351_f_at                .
```

```r
ridge_predtoplasso_prob <- predict(cv.toplasso, s = best.lambdatoplasso, newx = top_xtest,
    type = "response")

ridge_predtoplasso_cancer <- ifelse(ridge_predtoplasso_prob >= 0.5, 2, 1)

mean(ridge_predtoplasso_cancer != top_ytest)
```

```
## [1] 0.4705882
```

```r
table(ridge_predtoplasso_cancer, top_ytest)
```

```
##                            top_ytest
## ridge_predtoplasso_cancer  1  2
##                          1 12  8
##                          2  8  6
```

```r
lasso_top_success_rate <- 1 - mean(ridge_predtoplasso_cancer != top_ytest)

# Random Forest

lasso_rfv <- randomForest(type ~ ., data = top_datatest)
lasso_rfv
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = top_datatest)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 52.94%
## Confusion matrix:
##    0 1 class.error
## 0 14 6   0.3000000
## 1 12 2   0.8571429
```

```r
# 0% error rate! This either perfectly matches the report or is wrong.

# Will try doing a different approach.

lasso_rfv1 <- randomForest(type ~ ., data = top_datatrain)
lasso_rfv1
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = top_datatrain)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 10.53%
## Confusion matrix:
##     ALL AML class.error
## ALL  26   1  0.03703704
## AML   3   8  0.27272727
```

```r
predTrain_1 <- predict(lasso_rfv1, top_datatrain, type = "class")
table(predTrain_1, top_datatrain$type)
```

```
##
## predTrain_1 ALL AML
##         ALL  27   0
##         AML   0  11
```

```r
predValid_1 <- predict(lasso_rfv1, top_datatest, type = "class")
table(predValid_1, top_datatest$type)
```

```
##
## predValid_1  0  1
##         ALL 15 10
##         AML  5  4
```

```r
random_forest_top_success_rate <- (19/34)
```

Success Rates

Needless to say, we tried a *lot* of models. Below is a basic table that provides the model accuracy of each model.

```r
success_rates <- (tibble(`KNN w/ k = 5 (All)` = knn_5_success_rate, `KNN w/ k = 3 (All)` = knn_3_success
    `KNN w/ k = 3, LOOCV (All)` = knn_3_success_rate_cv, `Ridge Regression (All)` = ridge_success_rate_a
    `Lasso (All)` = lasso_success_rate_all, `KNN w/ k = 4, (Lasso Variables)` = knn_3_success_rate_lasso
    `KNN w/ k = 3, (Top Variables)` = knn_3_sucess_rate_top, `PCR w/ 30 components (All)` = pcr_30_comp_
    `PCR w/ 30 components (Top Variables)` = pcr_30_comp_top_success_rate, `Random Forest (All)` = rando
    `Random Forest (Top Variables)` = random_forest_top_success_rate, `Random Forest (Best Variables)` =
    `Lasso (Best Variables)` = lasso_best_success_rate, `Lasso (Top Variables)` = lasso_top_success_rate
    `Ridge (Best Variables)` = ridge_best_success_rate, `Ridge (Top Variables)` = ridge_top_success_rate

success_rates[2, ] <- colnames(success_rates)

success_rates <- as.tibble(t(success_rates))

success_rates <- success_rates[, c(2, 1)]

colnames(success_rates) <- c("Model", "Success Rate")

success_rates[, 2] <- round(as.numeric(unlist(success_rates[, 2])), 4)

kable(success_rates)
```

Model	Success Rate
KNN w/ k = 5 (All)	0.5294
KNN w/ k = 3 (All)	0.5882
KNN w/ k = 3, LOOCV (All)	0.8684
Ridge Regression (All)	0.5882
Lasso (All)	0.6176
KNN w/ k = 4, (Lasso Variables)	0.8421
KNN w/ k = 3, (Top Variables)	0.9211
PCR w/ 30 components (All)	0.5588
PCR w/ 30 components (Top Variables)	0.6176
Random Forest (All)	0.7353
Random Forest (Top Variables)	0.5588
Random Forest (Best Variables)	0.6471
Lasso (Best Variables)	0.6471
Lasso (Top Variables)	0.5294
Ridge (Best Variables)	0.5882

Model	Success Rate
Ridge (Top Variables)	0.5000

Conclusions

Conclusion: In the end, after looking at all the models, it looks like KNN with `k = 3` using the top variables (the predictors with the highest variance) performed the best, with a 92% success rate. In general, KNN performed better than other models, although Random Forest ended up having an 73% success rate. Fundamentally, this makes sense - the paper suggested using nonparametric methods, and their 100% success rate came solely from using a K-nearest neighbors approach.

Summary

Our goal was to see if we could build an effective cancer classifier. We tried a number of different models, including KNN, Lasso, Ridge Regression, PCR, and Random Forest. We tried these methods on both the entire dataset, as well as genes we determined to have the highest variance. After evaluating all these methods, we determined that the KNN models have the best success rates, although we never received perfect accuracy. In the future, we would refine our code further, making it neater and easier to understand, and would attempt to achieve even closer accuracy.

Ultimately, our biggest troubles came with the sheer size of the dataset. Given that we had 7000+ predictors, it was often difficult to understand what we were doing as we applied the methods we learned in class. Some of us didn't have much content knowledge, which made it even harder to understand the basis of our results. Nevertheless, while our models were by no means perfect, they all at least achieved a success rate of greater than 50%. In all, although this project could be improved even further, the statistical methods we learned in class can definitely help scientists and researchers predict leukemia in cancer patients given gene expression data.