Homework 6

STAT 430, Spring 2017

Due: Friday, March 20 by 11:59 PM

Exercise 1

```
# data creation

# data generating process
make_hw06_data = function(n = 1000) {

  x01 = rnorm(n = n)
  x02 = rnorm(n = n)
  x03 = rnorm(n = n)
  x04 = rnorm(n = n, mean = 1, sd = 2)
  x05 = rnorm(n = n, mean = 1, sd = 2)
  x06 = runif(n = n, min = -2, max = 2)
  x07 = runif(n = n, min = -2, max = 2)
  x08 = runif(n = n, min = -2, max = 2)
  x09 = x01 + rnorm(n = n)
  x10 = x06 + rnorm(n = n)

  z = 0.5 + 3 * x01 - 2 * x02 + x06 + x07
  prob = exp(z) / (1 + exp(z))
  y = rbinom(length(z), size = 1, prob = prob)

  data.frame(y, x01, x02, x03, x04, x05, x06, x07, x08, x09, x10)

}

# generate data
set.seed(1337)
hw06_data = make_hw06_data(n = 2500)

# write to file
library(readr)
write_csv(hw06_data, "hw06-data.csv")
```

[10 points] For this exercise use the data found in `hw06-data.csv`. We will attempt to classify the y variable.

Do the following:

- Set a seed value equal to your UIN
- Test-train split the data using approximately 50% of the data for training.
- Fit three models:
  - Additive logistic regression
  - Logistic regression with predictors chosen using a variable selection technique of your choice
  - k-nearest neighbors using a well tuned value of k
- Report 5 fold cross-validated, train, and test accuracy for each of the three models
  - You should arrange this in a table

- **You do not need to cross-validate the selection method, but rather only cross-validate the resulting model**

Also answer the following:

- What is the model you chose via selection?
- Plot the cross-validated k-nearest neighbor results. Argue that this plot verifies that you have considered enough values of k.
- What value of k did you select?
- Based on these results, which model do you prefer?

```
uin = 123456789
set.seed(uin)
```

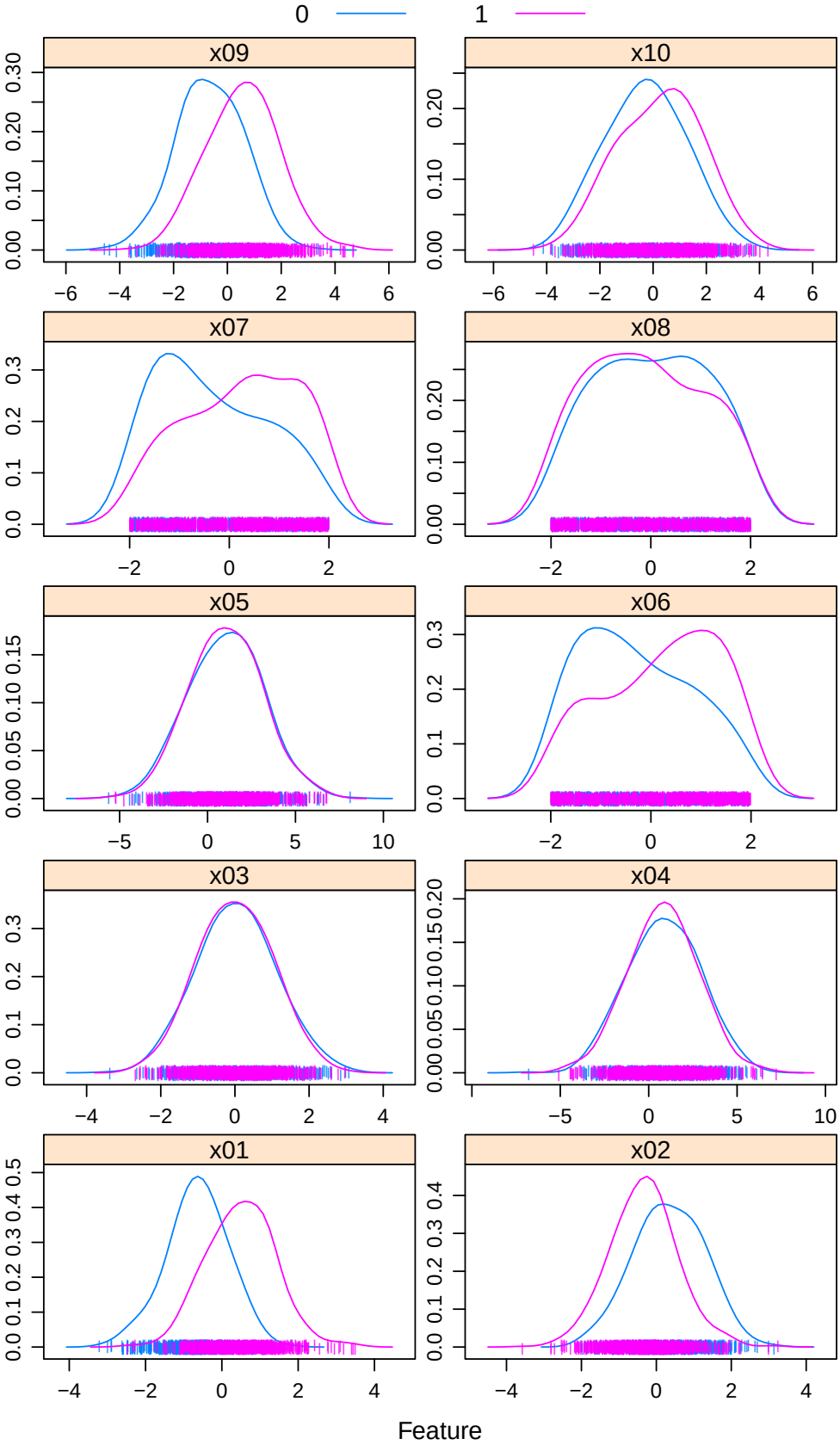
Solution:

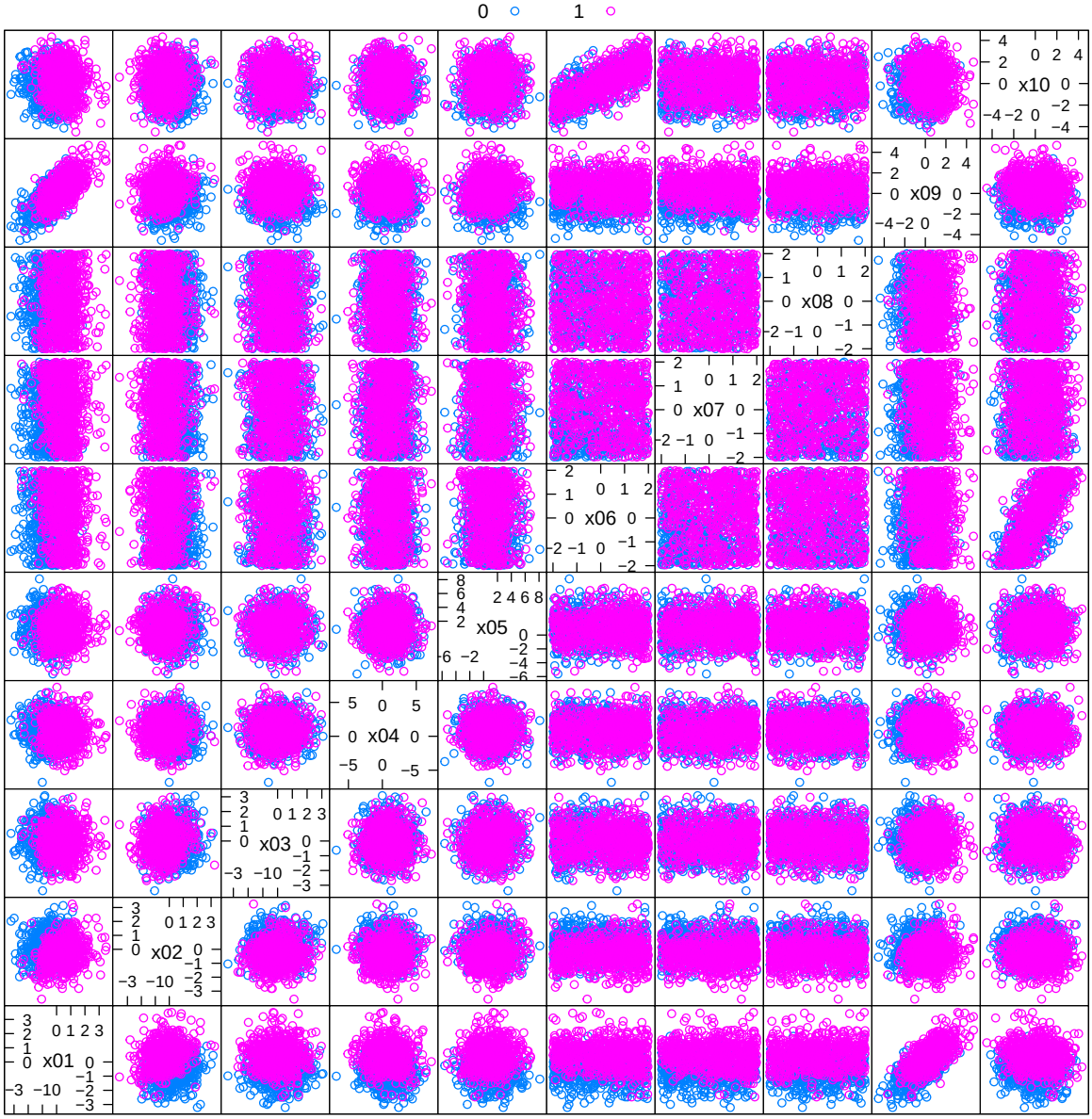
Note that some code, for plotting and summarizing, is hidden. See the .Rmd file for code.

```
# import data
library(readr)
hw06_data = read_csv("hw06-data.csv")
hw06_data$y = as.factor(hw06_data$y)
```

```
# load libraries
library(caret)
```

```
# test-train split the data
train_idx = createDataPartition(hw06_data$y, p = 0.50, list = FALSE)
hw06_trn = hw06_data[ train_idx, ]
hw06_tst = hw06_data[-train_idx, ]
```


0
1
x09
x10
x07
x08
x05
x06
x03
x04
x01
x02
Feature



Scatter Plot Matrix

```
# setup cross validation
cv_5 = trainControl(method = "cv", number = 5)
```

```
# accuracy function
accuracy = function(actual, predicted) {
  mean(actual == predicted)
}
```

```
# additive logistic
hw06_alr = train(y ~ ., data = hw06_trn,
                 method = "glm", family = "binomial",
                 trControl = cv_5)


hw06_alr_cv  = hw06_alr$results$Accuracy
```

```
hw06_alr_trn = accuracy(actual = hw06_trn$y,
                        predicted = predict(hw06_alr, hw06_trn))
hw06_alr_tst = accuracy(actual = hw06_tst$y,
                        predicted = predict(hw06_alr, hw06_tst))
```

```
# logistic after selection
hw06_sel = step(glm(y ~ ., data = hw06_trn, family = "binomial"), trace = FALSE)
(selected_formula = hw06_sel$formula)
```

```
## y ~ x01 + x02 + x06 + x07 + x08 + x09
## <environment: 0x00000000226133b8>
```

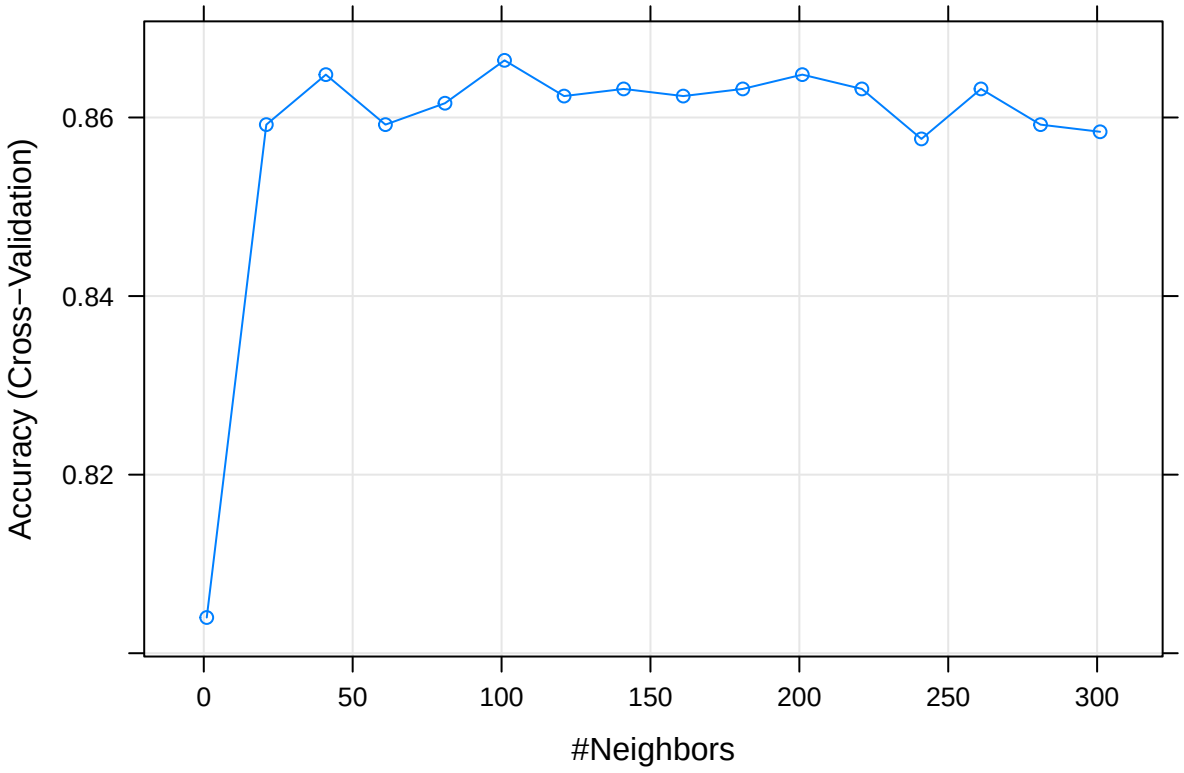
```
hw06_sel = train(selected_formula, data = hw06_trn,
                 method = "glm", family = "binomial",
                 trControl = cv_5)


hw06_sel_cv  = hw06_sel$results$Accuracy
hw06_sel_trn = accuracy(actual = hw06_trn$y,
                        predicted = predict(hw06_sel, hw06_trn))
hw06_sel_tst = accuracy(actual = hw06_tst$y,
                        predicted = predict(hw06_sel, hw06_tst))
```

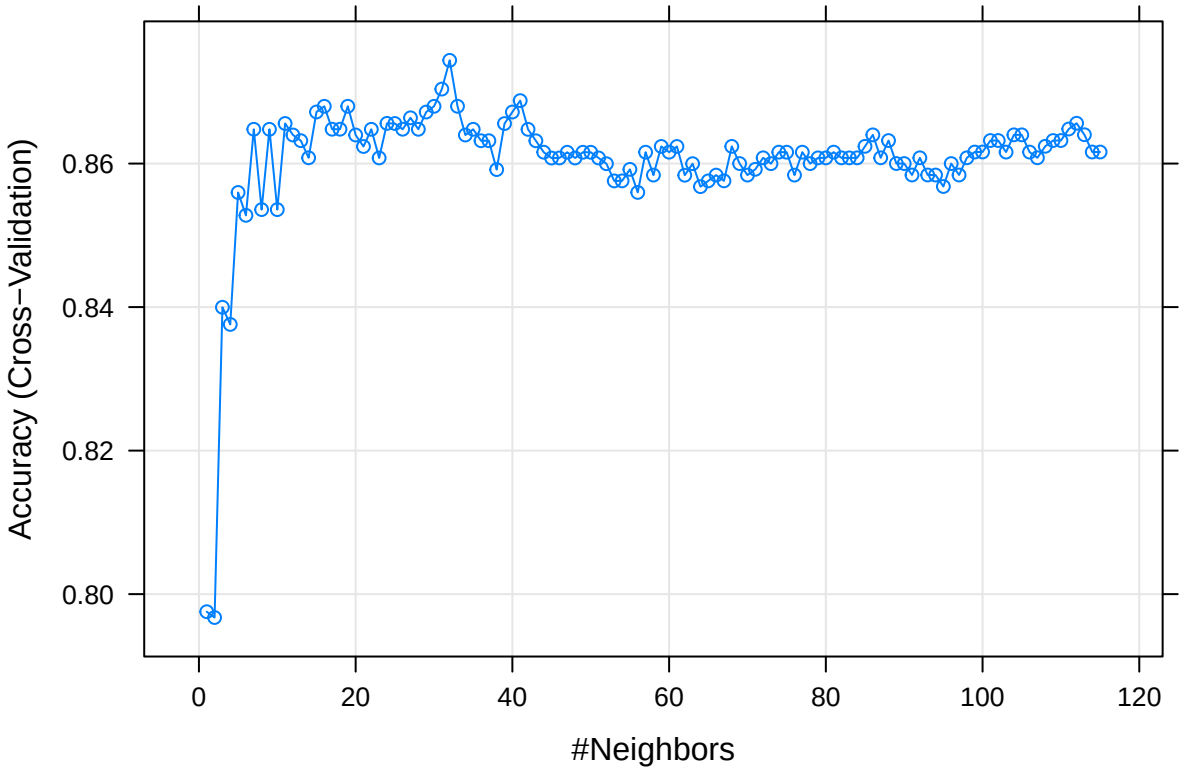
```
# caret helper function
get_best_result = function(caret_fit) {
  best_result = caret_fit$results[as.numeric(rownames(caret_fit$bestTune)), ]
  rownames(best_result) = NULL
  best_result
}
```

```
# knn

# search many k
hw06_knn_search = train(selected_formula, data = hw06_trn,
                        method = "knn",
                        tuneGrid = expand.grid(k = seq(1, 301, by = 20)),
                        trControl = cv_5,
                        preProcess = c("center", "scale"))
plot(hw06_knn_search)
```




```
# refine k search
hw06_knn = train(selected_formula, data = hw06_trn,
                 method = "knn",
                 tuneGrid = expand.grid(k = seq(1, 115, by = 1)),
                 trControl = cv_5,
                 preProcess = c("center", "scale"))
plot(hw06_knn)
```




```
get_best_result(hw06_knn)$k
```

```
## [1] 32
```

```
hw06_knn_cv  = get_best_result(hw06_knn)$Accuracy
hw06_knn_trn = accuracy(actual = hw06_trn$y,
                        predicted = predict(hw06_knn, hw06_trn))
hw06_knn_tst = accuracy(actual = hw06_tst$y,
                        predicted = predict(hw06_knn, hw06_tst))
```

Method	Train Acc	CV-5 ACC	Test Acc
Additive Logistic	0.8992	0.8951931	0.8704
Selected Logistic	0.8968	0.8943930	0.8768
k-Nearest Neighbors	0.8744	0.8743831	0.8520

```
hw06_alr$results$AccuracySD
```

```
## [1] 0.0319412
```

Answering the questions:

-

What is the model you chose via selection?

- Plot the cross-validated k-nearest neighbor results. Argue that this plot verifies that you have considered enough values of k.
  - We actually use two plots. The first is for a wide range of k values. We see the accuracy rise then fall, which indicates that we have check a sufficient number of tuning parameters. We see that a maximum is somewhere in the first 50 k, so we do a more refined search in that space.
- What value of k did you select?
  - We chose 32. Also note that we used preprocessing and the same subset of predictors from the logistic selection. (Both these provided better results.)
- Based on these results, which model do you prefer?
  - Based on these results, it would be wise to select the logistic which was selected via backwards AIC. While the additive does have a better CV accuracy, the standard deviation of that accuracy is 0.032, so the CV accuracy for the smaller model is within one SD. The smaller model also provides a better test accuracy.

Exercise 2

[20 points] For this question we will use the data in `leukemia.csv` which originates from Golub et al. 1999.

The response variable `class` is a categorical variable. There are two possible responses: `ALL` (acute myeloid leukemia) and `AML` (acute lymphoblastic leukemia), both types of leukemia. We will use the many feature variables, which are expression levels of genes, to predict these classes.

Note that, this dataset is rather large and you may have difficultly loading it using the "Import Dataset" feature in RStudio. Instead place the file in the same folder as your `.Rmd` file and run the following command. (Which you should be doing anyway.) Again, since this dataset is large, use 5-fold cross-validation when needed.

```
library(readr)
leukemia = read_csv("leukemia.csv", progress = FALSE)
```

For use with the `glmnet` package, it will be useful to create a factor response variable `y` and a feature matrix `X` as seen below. We won't test-train split the data since there are so few observations.

```
y = as.factor(leukemia$class)
X = as.matrix(leukemia[, -1])
```

Do the following:

- Fit an logistic regression with a lasso penalty. (Don't use cross-validation. Also let `glmnet` choose the λ values.) Create a plot that shows the features entering the model.
- Use cross-validation to tune an logistic regression with a lasso penalty. Again, let `glmnet` choose the λ values. Store both the λ that minimizes the deviance, as well as the λ that has a deviance within one standard error. Create a plot of the deviances for each value of λ considered. Use these two λ values to create a grid for use with `train()` in `caret`. Use `train()` to get cross-validated classification accuracy for these two values of λ. Store these values.
- Fit an logistic regression with ridge penalty. (Don't use cross-validation. Also let `glmnet` choose the λ values.) Create a plot that shows the features entering the model.
- Use cross-validation to tune an logistic regression with a ridge penalty. Again, let `glmnet` choose the λ values. Store both the λ that minimizes the deviance, as well as the λ that has a deviance within one standard error. Create a plot of the deviances for each value of λ considered. Use these two λ values to

create a grid for use with `train()` in `caret`. Use `train()` to get cross-validated classification accuracy for these two values of λ. Store these values.

- Use cross-validation to tune k-nearest neighbors using `train()` in `caret`. Do not specify a grid of k values to try, let `caret` do so automatically. (It will use 5, 7, 9.) Store the cross-validated accuracy for each.

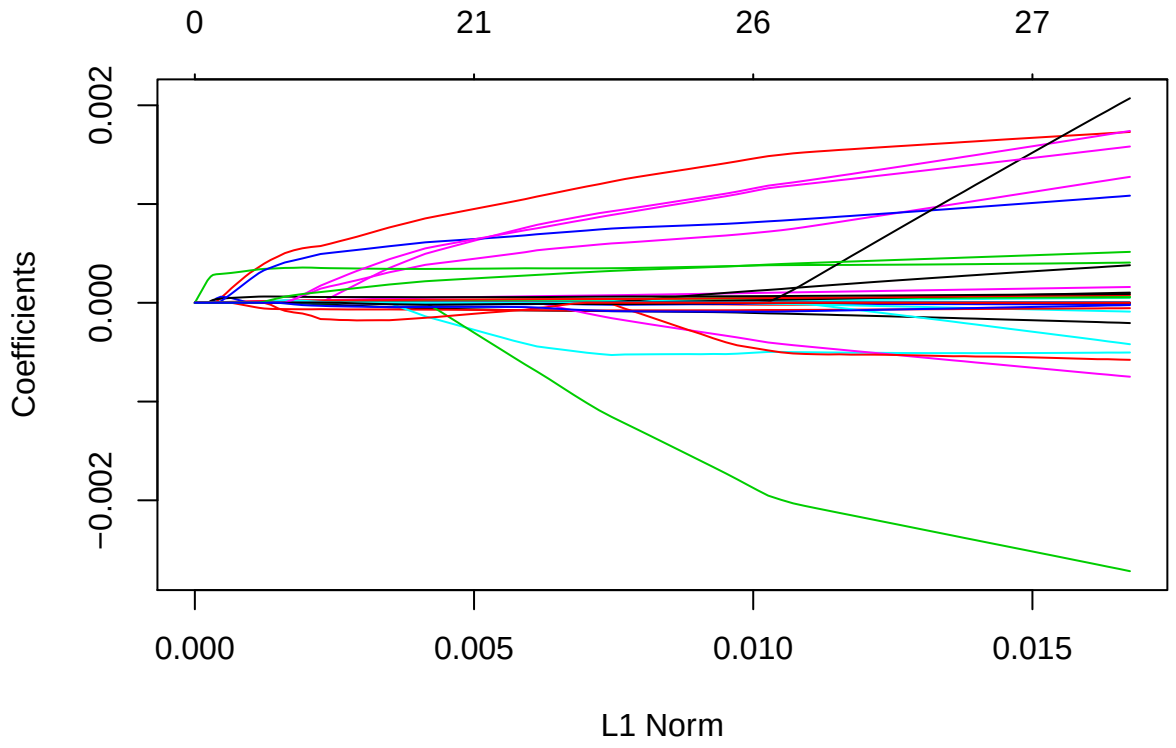
Also answer the following:

- How many observations are in the dataset? How many predictors are in the dataset?
- Based on the deviance plot, do you feel that `glmnet` considered enough λ values for lasso? For ridge?
- How does k-nearest neighbor compare to the penalized methods? Can you explain any difference?
- Summarize these **seven** models in a table. (Two lasso, two ridge, three knn.) For each report the cross-validated accuracy and the standard deviation of the accuracy.
- Based on your results, which model would you choose?

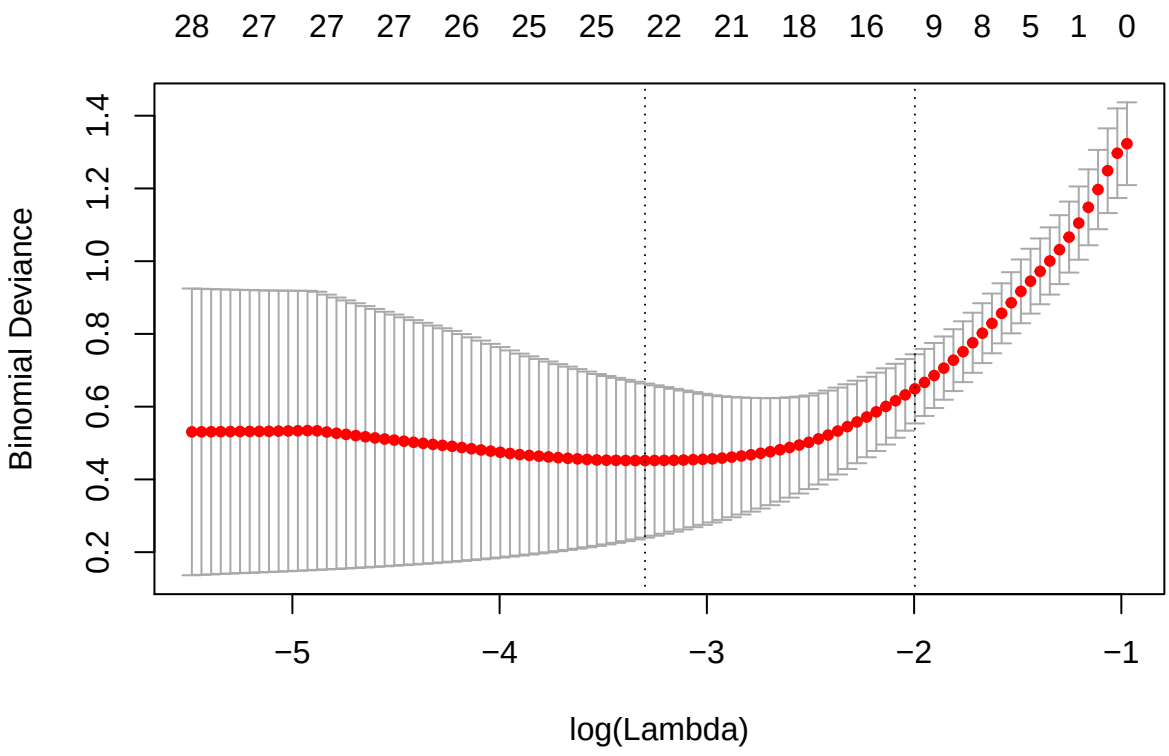
Solution:

```
library(glmnet)
set.seed(uin)
```

```
fit_lasso = glmnet(X, y, family = "binomial", alpha = 1)
plot(fit_lasso)
```




```
fit_lasso_cv = cv.glmnet(X, y, family = "binomial", alpha = 1, nfolds = 5)
plot(fit_lasso_cv)
```

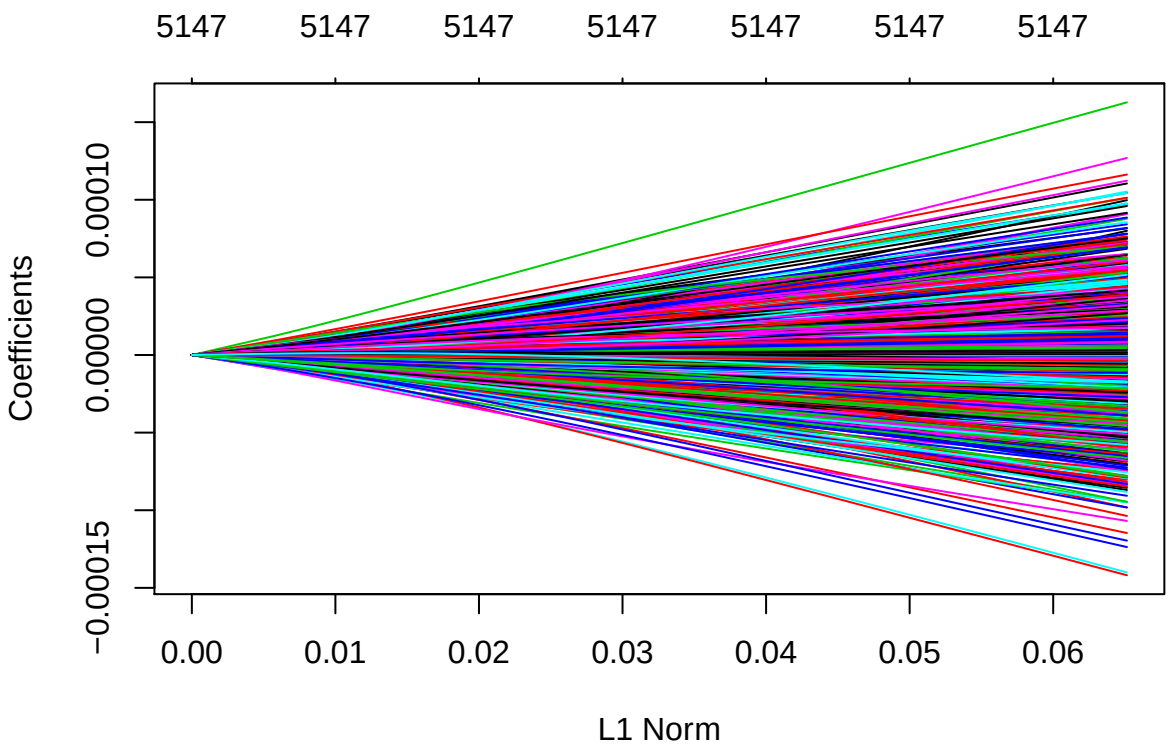



```
lambda_lasso = expand.grid(alpha = 1, lambda = c(fit_lasso_cv$lambda.min, fit_lasso_cv$lambda.1se))
```

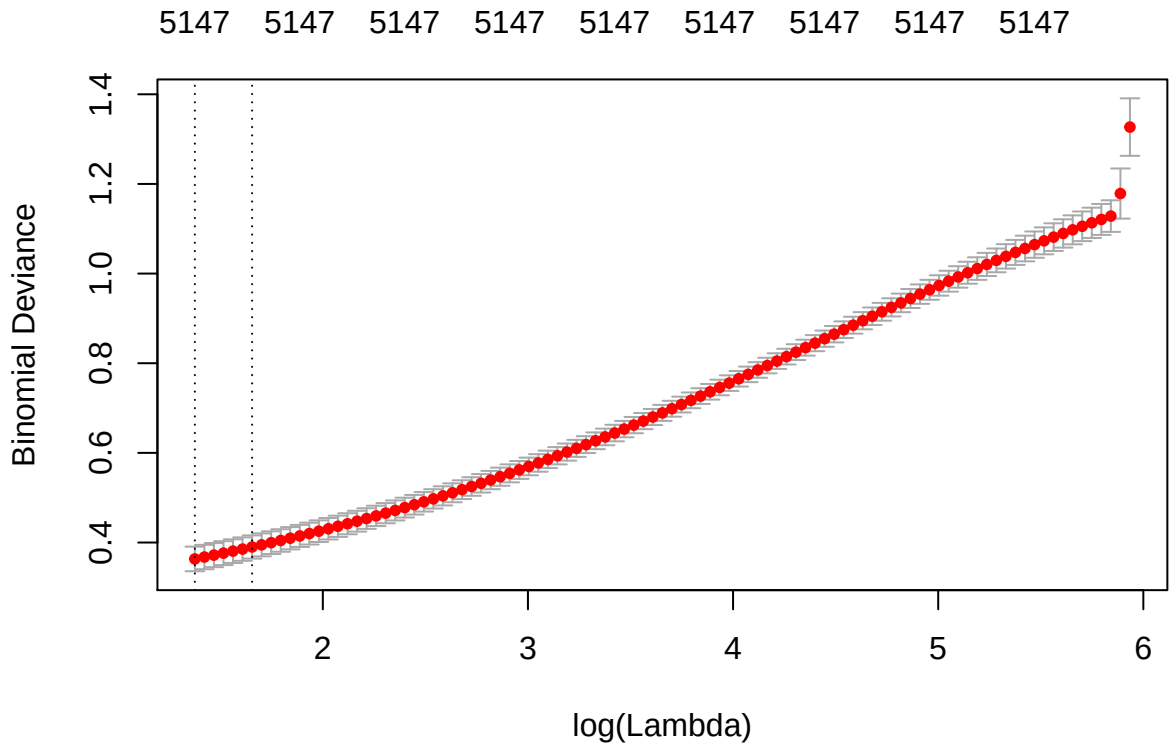
```
train_lasso = train(X, y,
                    method = "glmnet",
                    trControl = cv_5,
                    tuneGrid = lambda_lasso)
train_lasso$results[c("lambda", "Accuracy", "AccuracySD")]
```

```
##       lambda  Accuracy AccuracySD
## 1 0.03692667 0.9438095 0.05951428
## 2 0.13583032 0.8885714 0.03891965
```

```
fit_ridge = glmnet(X, y, family = "binomial", alpha = 0)
plot(fit_ridge)
```




```
fit_ridge_cv = cv.glmnet(X, y, family = "binomial", alpha = 0, nfolds = 5)
plot(fit_ridge_cv)
```




```
lambda_lasso = expand.grid(alpha = 0,
                           lambda = c(fit_ridge_cv$lambda.min,
                                      fit_ridge_cv$lambda.1se))
```

```
train_ridge = train(X, y,
                    method = "glmnet",
                    trControl = cv_5,
                    tuneGrid = lambda_lasso)
train_ridge$results[c("lambda", "Accuracy", "AccuracySD")]
```

```
##     lambda  Accuracy AccuracySD
## 1 3.959526 0.9457143 0.08713115
## 2 5.234260 0.9457143 0.08713115
```

```
train_knn = train(X, y,
                  method = "knn",
                  #preProc = c("center", "scale"),
                  trControl = cv_5
                  )
train_knn$results[c("k", "Accuracy", "AccuracySD")]
```

```
##   k  Accuracy AccuracySD
## 1 5 0.9447619 0.03097069
## 2 7 0.8742857 0.09359293
## 3 9 0.9019048 0.10898418
```

Method	Parameter	Parameter Value	CV-5 Accuracy	Standard Deviation

Method	Parameter	Parameter Value	CV-5 Accuracy	Standard Deviation
Lasso	`lambda`	0.0369267	0.9438095	0.0595143
Lasso	`lambda`	0.1358303	0.8885714	0.0389197
Ridge	`lambda`	3.9595259	0.9457143	0.0871311
Ridge	`lambda`	5.2342602	0.9457143	0.0871311
KNN	`k`	5.0000000	0.9447619	0.0309707
KNN	`k`	7.0000000	0.8742857	0.0935929
KNN	`k`	9.0000000	0.9019048	0.1089842

Answering the questions:

- How many observations are in the dataset? How many predictors are in the dataset?
  - There are 72 observations and 5147 variables, placing us very much in the $p > n$ setting.
- Based on the deviance plot, do you feel that `glmnet` considered enough λ values for lasso? For ridge?
  - For lasso: Yes, we see a nice U-shaped curve for the deviances.
  - For ridge: No, we see that have we tried more λ values, the deviance may have continued to go down.
- How does k-nearest neighbor compare to the penalized methods? Can you explain any difference?
  - k-nearest neighbors is worse on average due to the curse of dimensionality. Interestingly though, KNN works better without scaling and in particular `k = 5` performs as well as some of the shrinkage methods. (At least with this seed value.)
- Summarize these **seven** models in a table. (Two lasso, two ridge, three knn.) For each report the cross-validated accuracy and the standard deviation of the accuracy.
  - See above.
- Based on your results, which model would you choose?
  - There isn't a clear winner, except to say that the penalized methods are much better than the nearest neighbor method. Both ridge give the same result which obtains the highest accuracy. The lasso result with the larger λ is within one SD of the ridge result, and it is the most parsimonious model.