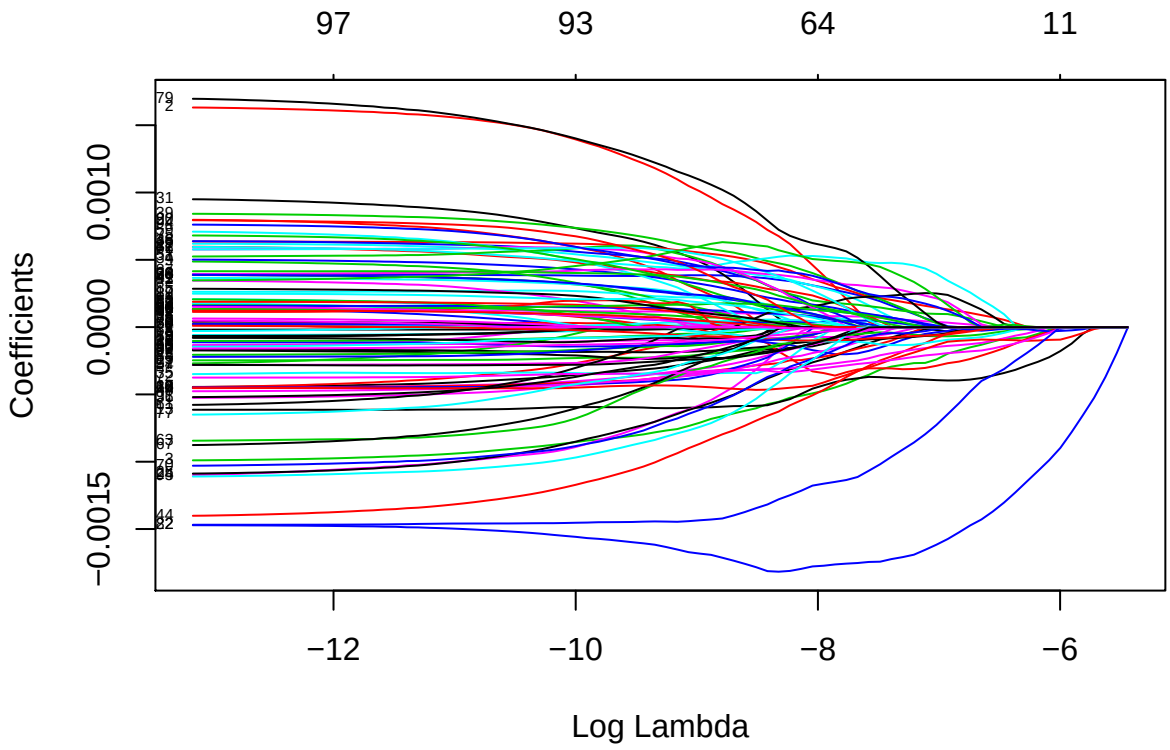
46-926, Statistical Machine Learning 1: Homework 2

Jordan Giebas

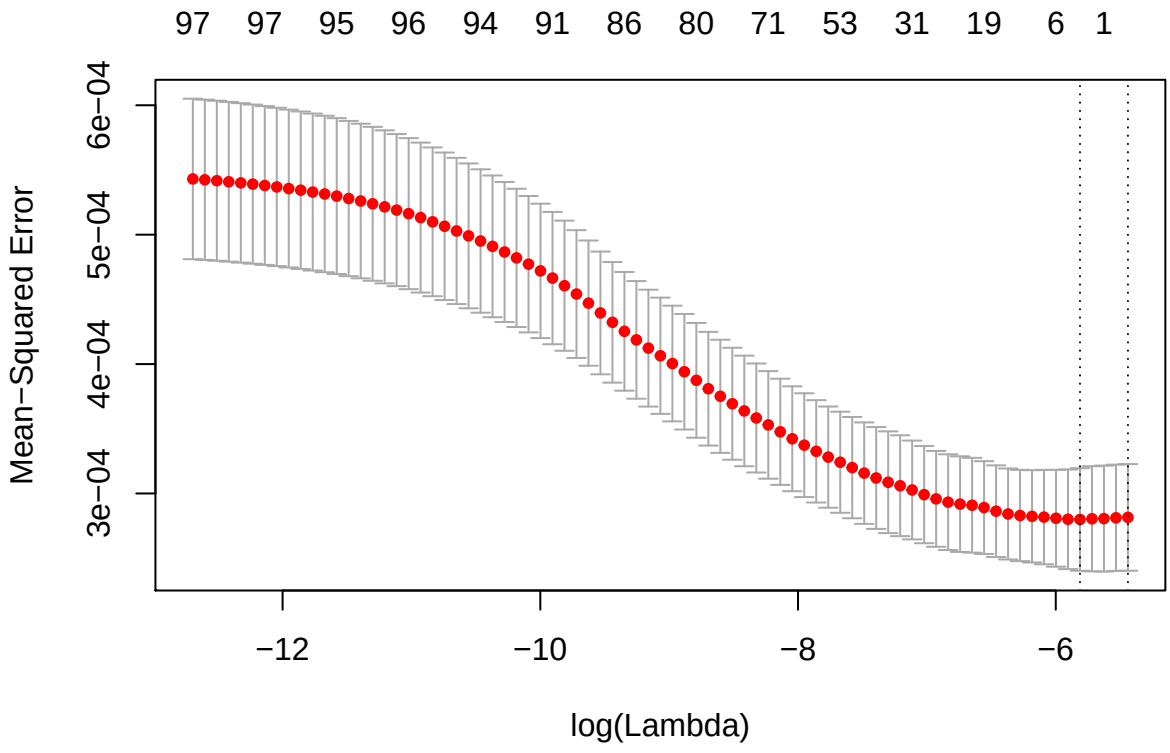
Due January 28th, 2018

```
## Question 3 for Machine Learning

# Load the data
df = read.csv('trends_train.csv')

# Separate predictors and response
x = data.matrix(df[,!(names(df) %in% c("logret"))])
y = df$logret

## Question 3(a)

# Fit the lasso
fit_lasso = glmnet(x, y, alpha=1)

# Plot coefficients as function of log(lambda)
plot(fit_lasso, xvar="lambda", label=TRUE, title("Coefficients vs. Log(Lambda)"))
```




```
## Question 3(b)
```

```r
# Carry out cross-validation
cvfit = cv.glmnet(x, y)

# Plot the cross validation curve w/ SE's
plot(cvfit)
```




```r
# Print out lambda_min, lambda_1se
print(paste("Lambda_min:", as.character(cvfit$lambda.min)))
```

```
## [1] "Lambda_min: 0.00299165448719026"
```

```r
print(paste("Lambda_1se:", as.character(cvfit$lambda.1se)))
```

```
## [1] "Lambda_1se: 0.00434037842533214"
```

```r
## Question 3(c)

min_model = as.vector(coef(cvfit, s="lambda.min"))
ose_model = as.vector(coef(cvfit, s="lambda.1se"))

# Get the number of non zero coefficients (excluding intercept)
min_nzc = sum(min_model != 0) - 1
ose_nzc = sum(ose_model != 0) - 1

print(paste("Number non-zero coefficients in Min Model:", as.character(min_nzc)))
```

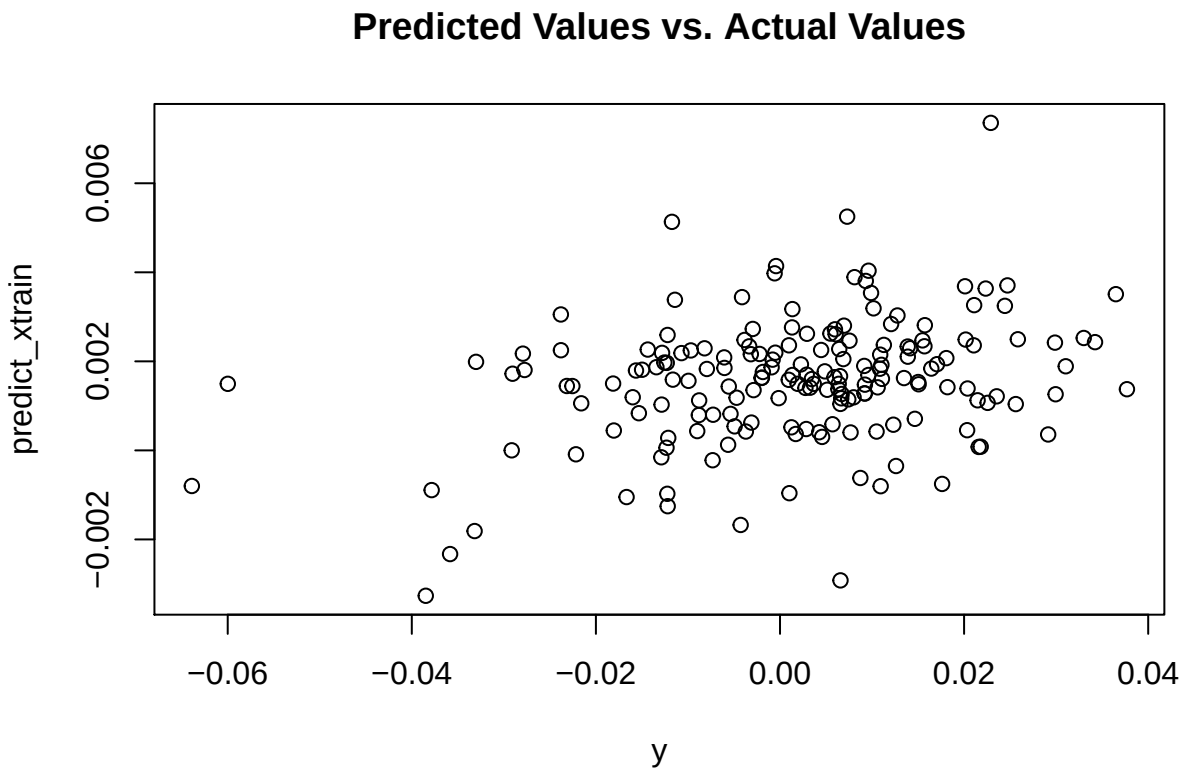
```
## [1] "Number non-zero coefficients in Min Model: 4"
```

```
print(paste("Number non-zero coefficients in 1SE Model:", as.character(ose_nzc)))
```

```
## [1] "Number non-zero coefficients in 1SE Model: 0"
```

In the model, the number of non-zero coefficients (exlcluding the intercept) is four. We see that the lasso penalty selects four of the ninety-eight possible predictors to describe the response variables. In contrast, in the model, zero predictors are chosen and the description of the response is only within the intercept. It seems that this is ample evidence suggesting that the signal in this problem is nearly non-existent. It's interesting that the nubmer of signals to be contained within one standard error of the minimum decreases.

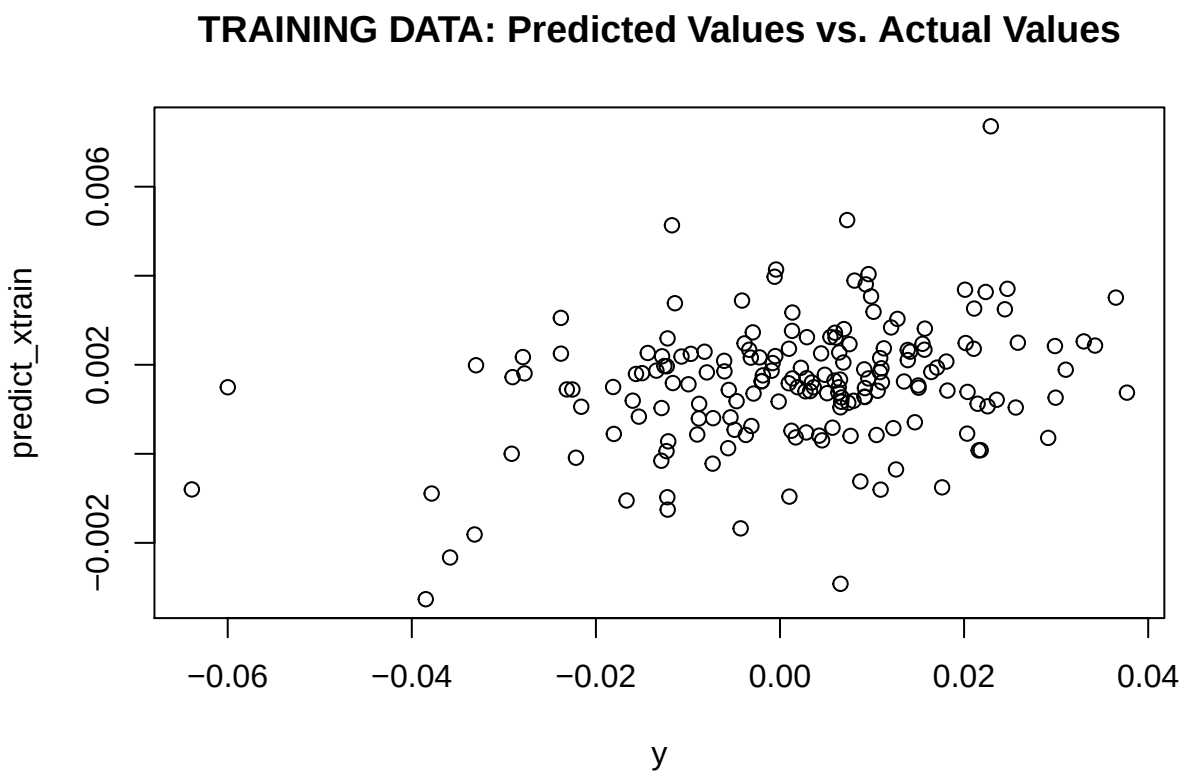
```
## Question 3(d)
predict_xtrain = predict(cvfit, newx=x, s="lambda.min")
plot(y,predict_xtrain, title("Predicted Values vs. Actual Values"))
```



If the predicted values were exactly the response values, then we would essentially see the function $f(x) = x$. There is some correspondence resembling that line, especially because there are clustering effects around $y = 0$ associated with $\hat{y} = 0$, but there are many outliers as well that. The predictions do a moderately good job at explaining the true y-values of the training data. However, we're interested in out they predict *new* data. We analyze this in part (f).

```
## Question 3(e)

# WRITE THE STRATEGY AND RESULTS AS A FUNCTION TO BE RE-USED IN NEXT PART
strat = function( predict_vals, actual_vals, te_or_tr ) {

  # Create the signal vector, buy when positive,
  signal_vec =  sign(predict_vals)
```

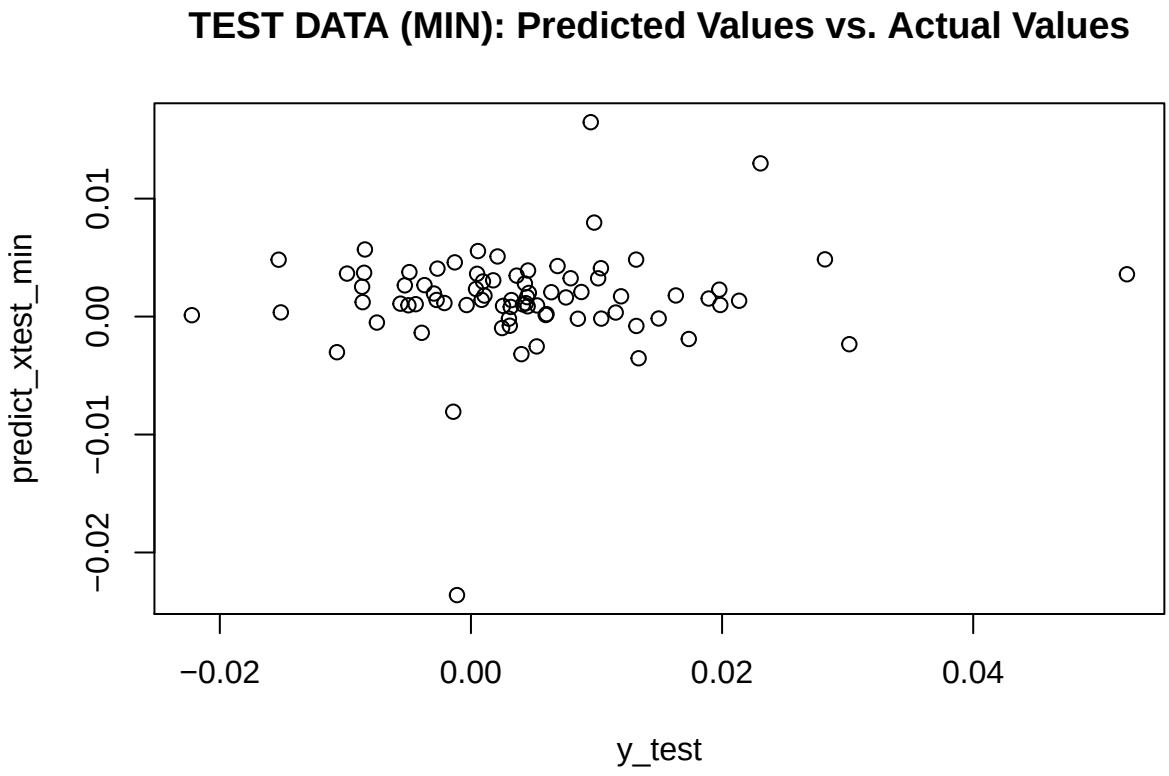
```
  # Multiply entry wise to get the log return
  ret = signal_vec * actual_vals

  # Get return statistics
  our_ret = (exp(sum(ret)) - 1) * 100.0
  bnh_ret = (exp(sum(actual_vals)) - 1) * 100.0
  factor = our_ret/bnh_ret

  # Print results
  print(paste(paste("ON THE", te_or_tr),"DATA"))
  print(paste(paste("Prediction Strategy returns:", as.character(round(our_ret,digits = 2)),"%")))
  print(paste(paste("Buy n Hold Strategy returns:", as.character(round(bnh_ret,digits = 2)),"%")))
  print(paste(paste("Our strategy performs", factor),"times better than buying-holding"))

}

# Plot as desired
plot(y, predict_xtrain, title("TRAINING DATA: Predicted Values vs. Actual Values"))
```




```
# Gather return data
strat(predict_xtrain, y, "TRAIN")
```

```
## [1] "ON THE TRAIN DATA"
## [1] "Prediction Strategy returns: 114.28 %"
## [1] "Buy n Hold Strategy returns: 32.72 %"
## [1] "Our strategy performs 3.49307978266665 times better than buying-holding"
```

```r
## Question 3(f)
library(glmnet)

# Load in test data
df_test = read.csv('trends_test.csv')

# Separate predictors and response of test data
x_test = data.matrix(df_test[,!(names(df_test) %in% c("logret"))])
y_test = df_test$logret

##### FOR MIN #####

# Carry out prediction using lambda min, plot results
predict_xtest_min = predict(cvfit, newx=x_test, s="lambda.min")
plot(y_test, predict_xtest_min, title("TEST DATA (MIN): Predicted Values vs. Actual Values"))
```




```r
# Get returns of strategy
strat(predict_xtest_min, y_test, "TEST_MIN")
```

```
## [1] "ON THE TEST_MIN DATA"
## [1] "Prediction Strategy returns: 15.48 %"
## [1] "Buy n Hold Strategy returns: 41.36 %"
## [1] "Our strategy performs 0.374327394619537 times better than buying-holding"
```

```r
##### FOR 1SE #####

# Carry out prediction using lambda 1se
```

```r
predict_xtest_1se = predict(cvfit, newx=x_test, s="lambda.1se")

# Get returns of strategy
strat(predict_xtest_1se, y_test, "TEST_1SE")
```

```
## [1] "ON THE TEST_1SE DATA"
## [1] "Prediction Strategy returns: 41.36 %"
## [1] "Buy n Hold Strategy returns: 41.36 %"
## [1] "Our strategy performs 1 times better than buying-holding"
```

When using the λ_{min} model, the results seems qutie poor. The strategy using our prediction yields returns of 15.48% compared with the buy and hold strategy of 41.36%. The λ_{1SE} case is odd to me, I'm seeing that the returns are identical for that strategy as they are for the buy and hold strategy which is quite a coincidence, or something is wrong (I'm betting the latter). Well actually, I think the λ_{1SE} case is just an intercept model, because the number of non-zero coeffcients (excluding the intercept) were zero. Hence, it does make some sense that the returns will be identical because we are never making a prediction toward the sign of the upcoming log returns based on any features!