Stat 897 Fall 2017 Data Analysis Assignment 5

Penn State

Due September 24, 2017

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.2.5
```

```
library(leaps)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.2.4
```

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

```
## Warning: package 'Matrix' was built under R version 3.2.5
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

(1) In the first part of this assignment we will use the College data found in the ISLR library. Split the data into a training set of size 100 and test set with the rest. Your goal is to predict the number of applications received using the other variables in the data set.

(a) Fit a "best" model obtained from your previous assignment on the training set and report the test error for this model.

```
set.seed(5948)
train = sample(seq(1:777), 100, replace = FALSE)
College.train = College[ train, ]
College.test = College[-train, ]

ntrain = nrow(College.train)
ntest = nrow(College.test)
fit1 = regsubsets(Apps ~ ., data = College.train, nvmax = 17)

### predict function for regsubsets object
predict.regsubsets = function(object, newdata, id, ...){
  form = as.formula(~ .)
  mat = model.matrix(form, newdata)
  coefi = coef(object, id)
  xvars = names(coefi)
  mat[, xvars] %*% coefi
}

mse.subset.test = rep(NA, 17)

for(i in 1:17){
  yhat.i = predict.regsubsets(fit1, newdata = College.test, id = i)
  mse.subset.test[i] = sum((College.test$Apps - yhat.i) ^ 2) / ntest
}
```

```r
# number of the predictors selected by the best subset
which.min(mse.subset.test)
```

```
## [1] 14
```

The predictors and coefficients are as follows:

```r
# best model given by best subset
coef(fit1, id = which.min(mse.subset.test))
```

```
##   (Intercept)    PrivateYes        Accept        Enroll     Top10perc
## -708.82867474 -945.72916251    1.51473461   -0.64707927    3.73619300
##     Top25perc      Outstate    Room.Board         Books      Personal
##   14.51737761   -0.09640373    0.22054804    0.44645755   -0.07225778
##            PhD      Terminal     S.F.Ratio   perc.alumni        Expend
##    7.30194687   -9.79010112   20.68814241   19.67621350    0.02587655
```

The test error for this model is

```r
mse.subset.test[which.min(mse.subset.test)]
```

```
## [1] 1630652
```

(b) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```r
x.train = data.matrix(College.train[, -2])
y.train = College.train$Apps
x.test  = data.matrix(College.test[, -2])
y.test  = College.test$Apps
fit.ridge = cv.glmnet(x = x.train, y = y.train, alpha = 0)
# chosen lambda
fit.ridge$lambda.min
```

```
## [1] 364.6635
```

```r
# test error
yhat.ridge = predict(fit.ridge, newx = x.test)
ridge.test.error = mean((y.test - yhat.ridge) ^ 2)
ridge.test.error
```

```
## [1] 2474284
```

(c) Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```r
fit.lasso = cv.glmnet(x = x.train, y = y.train, alpha = 1)
# chosen lambda
fit.lasso$lambda.min
```

```
## [1] 73.2694
```

```r
# test error
yhat.lasso = predict(fit.lasso, newx = x.test)
lasso.test.error = mean((y.test - yhat.lasso) ^ 2)
lasso.test.error
```

```
## [1] 2153213
```

(d) Compare the result obtained in (a) – (c). How accurately can we predict the number of college applications? Is there much difference among the test errors resulting from different approaches?

The MSE for the three models above (best subset, ridge, lasso) are sort of large. I would say these three models are not very accurate in term of predicting the college applications. The best subset regression and lasso regression have similar performance and ridge regression is worse.

(e) Now partition the data into a training set of size 600 and a test set with the rest. Compare the test errors from the 'best' linear regression model, ridge and lasso models. Note that, 'best' model here may not be the 'best' model obtained before.

```
train = sample(seq(1:777), 600, replace = FALSE)
College.train = College[ train,]
College.test  = College[-train,]

ntrain = nrow(College.train)
ntest  = nrow(College.test)
fit1   = regsubsets(Apps ~., data = College.train, nvmax = 17)

mse.subset.test = rep(NA, 17)

for(i in 1:17){
  yhat.i = predict.regsubsets(fit1, newdata = College.test, id = i)
  mse.subset.test[i] = sum((College.test$Apps - yhat.i) ^ 2) / ntest
}

# number of the predictors selected by the best subset
which.min(mse.subset.test)
```

```
## [1] 4
```

```
coef(fit1, id = which.min(mse.subset.test))
```

```
##   (Intercept)        Accept     Top10perc      Outstate        Expend
## -726.92651335    1.43690800   21.07283561   -0.07167623    0.10019581
```

The test error for this model is

```
mse.subset.test[which.min(mse.subset.test)]
```

```
## [1] 1839755
```

```
x.train = data.matrix(College.train[, -2])
y.train = College.train$Apps
x.test  = data.matrix(College.test[, -2])
y.test  = College.test$Apps
fit.ridge = cv.glmnet(x = x.train, y = y.train, alpha = 0)
# chosen lambda
fit.ridge$lambda.min
```

```
## [1] 393.0932
```

```
# test error
yhat.ridge = predict(fit.ridge, newx = x.test)
ridge.test.error = mean((y.test - yhat.ridge) ^ 2)
ridge.test.error
```

```
## [1] 2197970
```

```
fit.lasso = cv.glmnet(x = x.train, y = y.train, alpha = 1)
# chosen lambda
fit.lasso$lambda.min
```

```
## [1] 2.773195
```

```
# test error
yhat.lasso = predict(fit.lasso, newx = x.test)
lasso.test.error = mean((y.test - yhat.lasso) ^ 2)
lasso.test.error
```

```
## [1] 2168044
```

(f) Do you see any difference between the two sets of results? Comment.

After we increase the training sample size from 100 to 600, we don't see much improvement in the three models in terms of test error, which is surprising.

(2) The file Sp.Rdv contains daily returns for 501 stocks in 2016. It was created as follows.

(You don't need to run this, i.e. leave eval=FALSE)

```
library("quantmod")
sp = read.csv("SP500HistoricalComponents.csv")
Symbols = sp[which(sp$X12.31.2016 == "X"), "Ticker"]
StartDate = '2016-01-01'
EndDate = '2016-12-31'

Stocks = lapply(Symbols, function(sym) {
  print(sym)
  dailyReturn(na.omit(getSymbols(as.character(sym), from = StartDate, to = EndDate,
                                 auto.assign = FALSE, src = "yahoo")))
})

SPreturns = do.call(merge, Stocks)
colnames(SPreturns) = Symbols

#Clean out cols with NA
spreturns = SPreturns[ , colSums(is.na(SPreturns)) == 0]
save(spreturns, file = "spreturns.Rda")
```

Make sure you downloaded the data from the assignment page, and you can load it using:

```
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 3.2.5
## Loading required package: xts
## Warning: package 'xts' was built under R version 3.2.5
## Loading required package: zoo
## Warning: package 'zoo' was built under R version 3.2.5
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: TTR
## Warning: package 'TTR' was built under R version 3.2.5
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
load("../spreturns.Rda")
```

I have constructed a secret long-only portfolio chosen from these stocks. It contains between five and twenty stocks. The daily return of this portfolio for each trading day of 2016 is in the object `portfolioreturns` which you can load from file with:

```
load("../portfolioreturnsstatic.Rda")
```

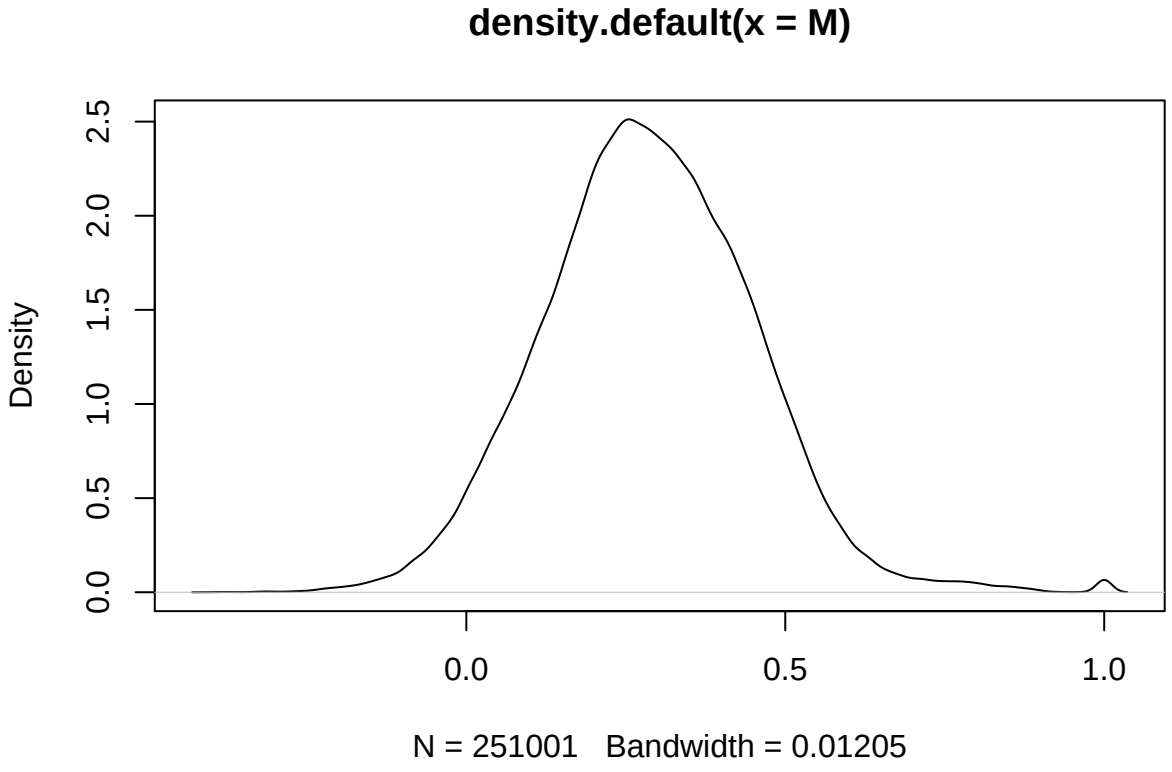
Your goal is to recover the stocks and weights of the secret portfolio.

Note that you can think of a portfolio as a vector of nonnegative weights that sum to one. For simplicity, we are assuming that this portfolio is rebalanced daily at the closing prices. Then if the daily returns vector on date d is r_d and the weight vector is w_d, the daily return for the portfolio is the dot product $r_d \cdot w_d$. If this were a buy-and-hold portfolio, we would have to back into the returns more carefully.

(a) First try to fit an ordinary regression (lm) with `portfolioreturns` as the response and `spreturns` as the predictors. What happens? What problem do you run into?

Of course, the returns of the S&P 500 are highly correlated.

```
M = cor(spreturns)
plot(density(M))
```



The stocks on the right hand part have nearly colinear returns. Moreover since the sample size (number of periods) is smaller than the number of variables, generically we expect that there are many ways to get a perfect or near-perfect fit to the data, but they are not meaningful.

```
mod = lm(portfolioreturns ~ 0 + spreturns)
length(coef(mod)[!is.na(coef(mod))]) == nrow(spreturns)
```

```
## [1] TRUE
```

Note that the number of nonzero coefficients returned in the fit equals the number of rows.

(b) Now use elasticnet (Lasso, Ridge, or a combination, i.e. glmnet) instead of linear regression to model the secret portfolio. Once get a smaller set of variables from the shrinkage, refit a linear model with only those predictors. [Note, you may want to check out the `plotmo` and `pander` packages for nice plots and outputs for your models.]

I'll use the plotmo package which has nicely labelled glmnet plots, and the pander package to format output tables. We use glmnet for the regularized regression. [If you were using `Python` you could use lasso_path or enet_path form sklearn.linear_model for regularized regression (or even implement it yourself; it uses a fairly straightforward coordinate gradient descent algorithm).]

```
library(glmnet)
library(plotmo)
```

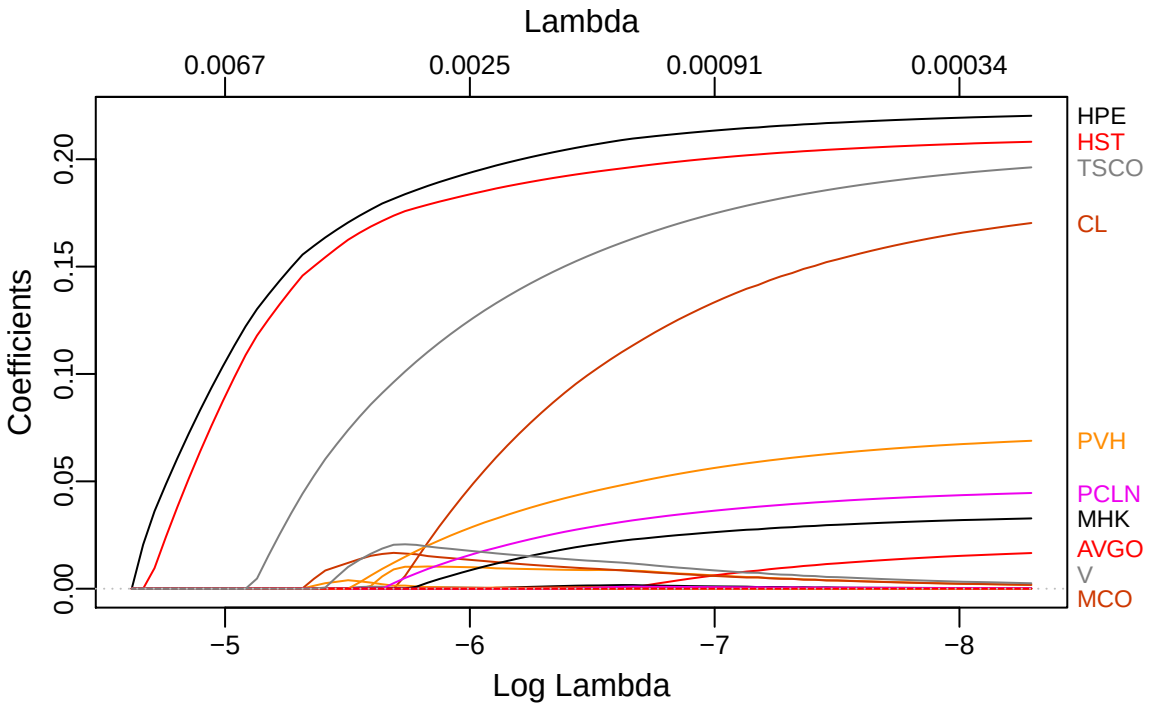
```
## Warning: package 'plotmo' was built under R version 3.2.5
## Loading required package: plotrix
## Warning: package 'plotrix' was built under R version 3.2.5
## Loading required package: TeachingDemos
## Warning: package 'TeachingDemos' was built under R version 3.2.3
```

```
library(pander)
```

```
## Warning: package 'pander' was built under R version 3.2.5
```

First let's look at the default

```
mod = glmnet(spreturns, portfolioreturns)
plot_glmnet(mod)
```



As the lasso penalty is relaxed, lasso (the default for glmnet) picks out eight stocks while all the other coefficients seem to converge to zero. This is in our range of 5-20, so good candidates for our secret portfolio. By the way, sometimes people say (including in the book) that at zero penalty, lasso/glmnet and ordinary regression give the same answer. Do you still believe that? What do you think happened to get such different answers?

```
# or use "broom" package
lassocoeff = predict(mod, s = 0, type = "coefficients")
bigcoef = lassocoeff[which(lassocoeff > 0.01), ]
pander(bigcoef)
```

AVGO	CL	HPE	HST	MHK	PCLN	PVH	TSCO
0.0166	0.1703	0.2202	0.2081	0.03273	0.04457	0.06889	0.1962

These eight coefficients sum to 0.9576714, which is not quite one. There is some bias introduced by shrinking, so let's rerun an OLS regression against these eight. Loading quantmod will make our life easier by coercing time series into data frames.

```
library(quantmod)
redmod = lm(portfolioreturns ~ 0 + ., data = spreturns[, names(bigcoef)])
pander(summary(redmod))
```

```
## Warning in summary.lm(redmod): essentially perfect fit: summary may be
```

```
## unreliable
```

	Estimate	Std. Error	t value	Pr(>|t|)
AVGO	0.02066	3.653e-17	5.655e+14	0
CL	0.1845	5.833e-17	3.164e+15	0
HPE	0.2229	2.964e-17	7.52e+15	0
HST	0.211	3.346e-17	6.305e+15	0
MHK	0.03527	4.515e-17	7.81e+14	0
PCLN	0.04767	3.494e-17	1.365e+15	0
PVH	0.07369	2.802e-17	2.629e+15	0
TSCO	0.2043	3.086e-17	6.62e+15	0

Table 3: Fitting linear model: portfolioreturns ~ 0 + .

Observations	Residual Std. Error	R^2	Adjusted R^2
252	7.863e-18	1	1

In this case the perfect fit is good news! So our preliminary answer is:

```
pander(coef(redmod))
```

AVGO	CL	HPE	HST	MHK	PCLN	PVH	TSCO
0.02066	0.1845	0.2229	0.211	0.03527	0.04767	0.07369	0.2043

And, the sum of these coefficients is 1, which is encouraging. Let's compare to the true portfolio.

(c) Here is how the portfolio was created. We're essentially randomly sampling columns from spreturns to have non-zero coefficients (portfoliowts) and then generating returns from that.

```
t = runif(ncol(spreturns))
thresh = .98
mask = t > thresh
w = runif(ncol(spreturns))
sum(mask) # number of chosen coefficients
portfoliowts = w * mask / sum(w * mask)
myportfolioreturns = spreturns %*% portfoliowts
```

Write a function that takes a threshold as input and produces the total error for estimated weights from lasso to the true weights. To do this you will need (1) a function that generates weights and returns for a given threshold function, (2) a function that takes the the returns and outputs the estimated coefficeints from a lasso, and (3) a function that takes the estimated coefficients and returns the error relative to the true weights. Plot the errors for a variety of different thresholds between 0.5 and 1.

What we'll do here is package up the portfolio creation procedure and the portfolio extraction procedure and run them with decresing threshold. We'll use one-half the l1 distance between the true portfolio and the inferred portfolio as a measure of accuracy; this should vary between zero (total agreement) and one (total disagreement).

Here is a function that creates a portfolio weight vector and return vector.

```
makereturns = function(thresh){
  t = runif(ncol(spreturns))
  mask = t > thresh
  w = runif(ncol(spreturns))
  portfoliowts = w * mask / sum(w * mask)
  names(portfoliowts) = colnames(spreturns)
  myportfolioreturns = spreturns %*% portfoliowts
  retobj = list("returns" = myportfolioreturns, "weights" = portfoliowts)
  return(retobj)
}
```

Then we can access the weights and returns like this:

```
test = makereturns(.97)
summary(test$weights)
summary(test$returns)
```

Now let's wrap up our two-stage portfolio inference procedure. In general we might want to know how good the fit is, or be smart about cross validation (and many of you were), but for now let's just take the zero-penalty result and return the weights from a second regression as we did above. So our function takes a vector of returns from a secret portfolio and returns its best guess as to the weights. We just rehash the above in a function.

```
inferweights = function(portreturns, xreturns){
  mod = glmnet(xreturns, portreturns)
  lassocoeff = predict(mod, s = 0, type = "coefficients")
  bigcoef = lassocoeff[which(lassocoeff > .01), ]
  redmod = lm(portreturns ~ 0 + ., data = xreturns[, names(bigcoef)])
  return(coef(redmod))
}
```

Of course coef(redmod) will only have entries for the nonzero coefficients; we need to put it back in a labelled 501-vector to compare it to the true portfolio weights, which may have different nonzero entries.

```
desparse = function(coefvect){
  dense = rep(0, ncol(spreturns))
  names(dense) = colnames(spreturns)
  dense[names(coefvect)] = coefvect
  return(dense)
}
```

Now let's put it together into a function which draws, infers, and just reports the error.

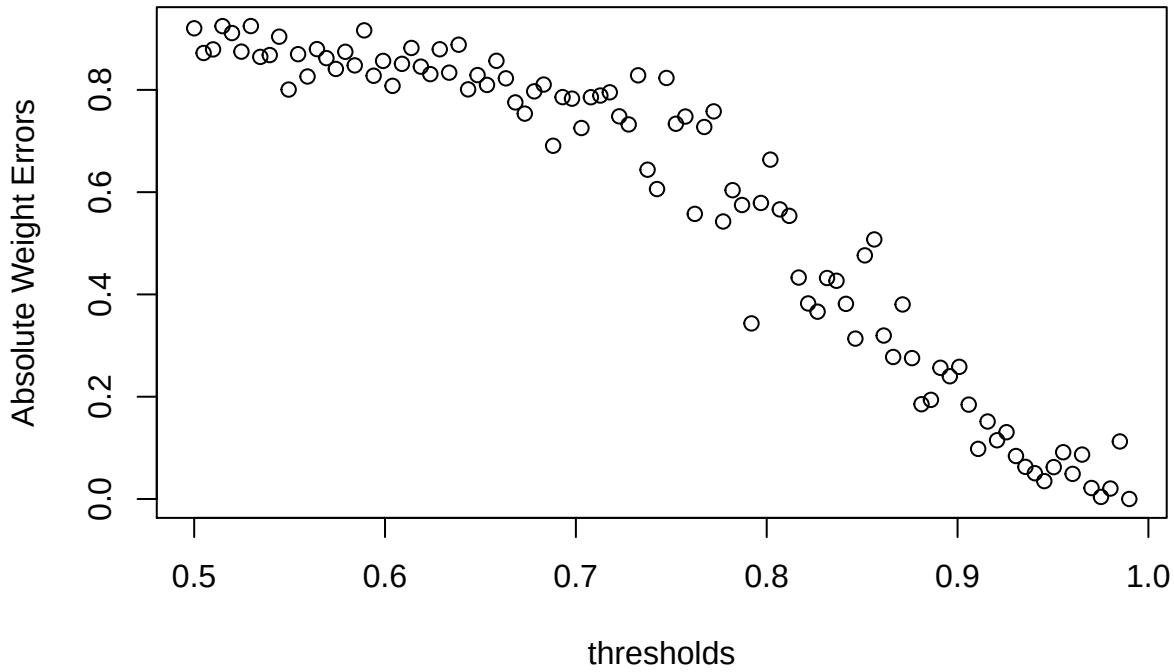
```
errat = function(thresh){
  pr = makereturns(thresh)
  sparsewt = inferweights(pr$returns, spreturns)
  tver = sum(abs(desparse(sparsewt) - pr$weights))
  return(tver / 2)
}

errat(.98) ## test it
```

```
## [1] 5.931825e-16
```

Finally let's do that a lot and plot it.

```r
thresholds = seq(.5, .99, length = 100)
plot(thresholds, mapply(errat, thresholds), ylab = "Absolute Weight Errors")
```



The errors seems to increase for lower thresholds but eventually flatten as it nears 0.5.