Thomas Huelsnitz
ISYE 6501
Week 5 Homework

HW05

Question 11.1

Using the crime data set uscrime.txt from Questions 8.2, 9.1, and 10.1, build a regression model using: 1. Stepwise regression 2. Lasso 3. Elastic net For Parts 2 and 3, remember to scale the data first - otherwise, the regression coefficients will be on different scales and the constraint won't have the desired effect. For Parts 2 and 3, use the glmnet function in R.

For this analysis, we will perform stepwise regression, Lasso regression (glmnet function with alpha = 1), elasticnet regression (glmnet with alpha = 0.75, 0.5, 0.25) and finally Ridge regression (glmnet function with alpha = 0).

Stepwise regression is carried out using the stepAIC function from the MASS library. We calculate the average of the root mean square error (RMSE) across a 10-fold cross validation. This value is show below.

```
set.seed(42)
crime <- read.table("11.1uscrimeSummer2018.txt", header=TRUE)
crime$So <- as.factor(crime$So)

RMSE_stepwise_list <- c()
for (trialnum in 1:10){

  train_indices <- sample(1:nrow(crime), round(0.9*nrow(crime)))
  train <- crime[train_indices,]
  test <- crime[-train_indices,]

  #Tempting to try a grid search, but with 16 factors, there are 16!
different possible combinations.

  fit <- lm(Crime~.,data=train)
  step <- stepAIC(fit, direction="both", trace=0)

  RMSE_stepwise <- sqrt(mean((predict(step,test) - test[,'Crime'])^2))
  RMSE_stepwise_list <- c(RMSE_stepwise_list, RMSE_stepwise)
}
sprintf("Stepwise-regression RMSE: %s", mean(RMSE_stepwise_list))

## [1] "Stepwise-regression RMSE: 255.697355194326"
```

Next we do a similar process for lasso regression. Note that this time, the cross fold validation is done with the built in cv function rather than carried out explicity. For this model and all other glmnet-based models, the input data is scaled prior to training.

```
scalefunc <- preProcess(train[,-ncol(train)])
train_scaled <- cbind(predict(scalefunc, train[,-ncol(train)]),
```

```
Crime=train[,ncol(train)])
lassomodel <- cv.glmnet(x=data.matrix(train_scaled[,-ncol(train)]),
y=data.matrix(train_scaled[,ncol(train)]), alpha=1)
test_scaled <- cbind(predict(scalefunc, test[,-ncol(train)]),
test[,ncol(train)])
lassomodel_preds <-
predict(lassomodel$glmnet.fit,as.matrix(as.matrix(sapply(test_scaled,
as.numeric))[,-ncol(train)]))
lassomodel_pred_col <- paste("s",which(lassomodel$lambda ==
lassomodel$lambda.min)[length(which(lassomodel$lambda ==
lassomodel$lambda.min))] + 1, sep="")
RMSE_lasso <- sqrt(mean((lassomodel_preds[,lassomodel_pred_col] -
test[,'Crime'])^2))
sprintf("Lasso regression RMSE: %s", RMSE_lasso)

## [1] "Lasso regression RMSE: 143.229094909264"
```

In the next block of code we are doing the same process as in the lasso regression section, but this time it is for 3 elastic net models with alpha = 0.75, 0.5 and 0.25 respectively. Their cross-validated RMSE averages are reported below.

```
RMSE_elastic_list <- c()
for (a in c(0.75, 0.5, 0.25)){
  scalefunc <- preProcess(train[,-ncol(train)])
  train_scaled <- cbind(predict(scalefunc, train[,-ncol(train)]),
Crime=train[,ncol(train)])
  elasticmodel <- cv.glmnet(x=data.matrix(train_scaled[,-ncol(train)]),
y=data.matrix(train_scaled[,ncol(train)]), alpha=a)
  test_scaled <- cbind(predict(scalefunc, test[,-ncol(train)]),
test[,ncol(train)])
  elasticmodel_preds <-
predict(elasticmodel$glmnet.fit,as.matrix(as.matrix(sapply(test_scaled,
as.numeric))[,-ncol(train)]))
  elasticmodel_pred_col <- paste("s",which(elasticmodel$lambda ==
elasticmodel$lambda.min)[length(which(elasticmodel$lambda ==
elasticmodel$lambda.min))] + 1, sep="")
  RMSE <- sqrt(mean((elasticmodel_preds[,elasticmodel_pred_col] -
test[,'Crime'])^2))
  RMSE_elastic_list <- c(RMSE_elastic_list, RMSE)
}
sprintf("Elastic Net (alpha=0.75) regression RMSE: %s", RMSE_elastic_list[1])

## [1] "Elastic Net (alpha=0.75) regression RMSE: 112.833003969129"

sprintf("Elastic Net (alpha=0.5) regression RMSE: %s", RMSE_elastic_list[2])

## [1] "Elastic Net (alpha=0.5) regression RMSE: 114.929620108736"

sprintf("Elastic Net (alpha=0.25) regression RMSE: %s", RMSE_elastic_list[3])

## [1] "Elastic Net (alpha=0.25) regression RMSE: 128.474782616003"
```

Finally a ridge regression is calculated with the same methodology.

```
scalefunc <- preProcess(train[,-ncol(train)])
train_scaled <- cbind(predict(scalefunc, train[,-ncol(train)]),
Crime=train[,ncol(train)])
ridgemodel <- cv.glmnet(x=data.matrix(train_scaled[,-ncol(train)]),
y=data.matrix(train_scaled[,ncol(train)]), alpha=0)
test_scaled <- cbind(predict(scalefunc, test[,-ncol(train)]),
test[,ncol(train)])
ridgemodel_preds <-
predict(ridgemodel$glmnet.fit,as.matrix(as.matrix(sapply(test_scaled,
as.numeric))[,-ncol(train)]))
ridgemodel_pred_col <- paste("s",which(ridgemodel$lambda ==
ridgemodel$lambda.min)[length(which(ridgemodel$lambda ==
ridgemodel$lambda.min))] + 1, sep="")
RMSE_ridge <- sqrt(mean((ridgemodel_preds[,ridgemodel_pred_col] -
test[,'Crime'])^2))
sprintf("Ridge regression RMSE: %s", RMSE_ridge)

## [1] "Ridge regression RMSE: 154.326120673133"
```

Working on this problem I learned that stepwise regression is generally discouraged for the following reasons:

1. R 2 values are biased high
2. The F and chi-squared test statistics do not have the claimed distribution.
3. The standard errors of the parameter estimates are too small.
4. Consequently, the confidence intervals around the parameter estimates are too narrow.
5. p-values are too low, due to multiple comparisons, and are difficult to correct.
6. Parameter estimates are biased high in absolute value.
7. Collinearity problems are exacerbated

- Regression Modeling Strategies, Harrell 2001

Comparing the RMSE for these techniques indicates that the glmnet based approaches, specifically the elastic net ones which are constrained by a linear combination of the constraints applied to lasso and ridge regression, outperform stepwise regression. Playing with the seed value can lead to situations where stepwise regression appears to outperform the others, but most of the time this is not the case.

Question 12.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

In a prior job, I was tasked with fitting a neural network to a computationally intensive physics simulation model that was run many thousands of times per day. The statistics

model would be fit to real data points generated by the physics model spread throughout the space of physically possible parameter combinations (for example, air pressure was constrained to be positive, maximum absolute humidity depended on the temperature and pressure of the air, etc.). I used latin-hypercube sampling to minimize the number of simulation datapoints required to get a good fit throughout the entire parameter space. The latin-hypercube makes this possible because it ensures that for sparse sampling, the points are optimally distributed throughout the parameter space.

Question 12.2

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R's FrF2 function (in the FrF2 package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of FrF2 is "1" (include) or "-1" (don't include) for each feature.

The FrF2 function needs 2 arguments in this case. The first argument, nruns, is the number of trials in the design of experiments. In our scenario, this would be the number of houses which are shown to the buyers (16). Note that, according to the FrF2 documentation, this value must be a power of 2.

The second argument, nfactors, is the number of binary features to consider. In this scenario, this would be 10.

```
FrF2(16,10)
```

```
##     A  B  C  D  E  F  G  H  J  K
## 1   1 -1  1 -1 -1  1 -1 -1  1  1
## 2   1  1 -1 -1  1 -1 -1 -1  1  1
## 3  -1  1  1  1 -1 -1  1 -1  1 -1
## 4   1 -1  1  1 -1  1 -1  1 -1 -1
## 5  -1  1  1 -1 -1 -1  1  1 -1  1
## 6   1  1  1 -1  1  1  1 -1 -1 -1
## 7   1 -1 -1 -1 -1 -1  1 -1 -1 -1
## 8  -1 -1  1 -1  1 -1 -1  1  1 -1
## 9   1  1 -1  1  1 -1 -1  1 -1 -1
## 10  1  1  1  1  1  1  1  1  1  1
## 11  1 -1 -1  1 -1 -1  1  1  1  1
## 12 -1 -1 -1 -1  1  1  1  1 -1  1
## 13 -1  1 -1 -1 -1  1 -1  1  1 -1
## 14 -1 -1  1  1  1 -1 -1 -1 -1  1
## 15 -1  1 -1  1 -1  1 -1 -1 -1  1
## 16 -1 -1 -1  1  1  1  1 -1  1 -1
## class=design, type= FrF2
```

Note that for each feature, half of the trials are -1 and half are 1. The DOE is designed such that we can isolate the effects of each factor with a minimal number of trials.

Thomas Huelsnitz
ISYE 6501
Week 5 Homework

Question 13.1

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).

a. Binomial b. Geometric c. Poisson d. Exponential e. Weibull

a. Binomial

If individual with some gene have a probability p of contracting a disease, then the distribution of the number of individuals k who would contract the disease out of a population of N would be binomial.

b. Geometric

Using the same scenario as in a., the geometric distribution would describe the distribution of the number of trials required before encountering the first instance of k (patient who contracts the disease.)

c. Poisson

CCD sensors are subject to a type of noise called shot noise. Assume a CCD is experiencing an average of l photons per timeperiod t. For a given snapshot of time t, the distribution of the number of photons hitting the CCD is Poisson.

d. Exponential

Assuming the CCD scenario from c., the exponential distribution would describe the distribution of the amount of time between subsequent photon impacts on the CCD.

e. Weibull

The Weibull distribution is used in various types of failure analysis. The distribution of the lifetime of a jet engine blade would follow a weibull distribution with $k > 1$. Recall that $k < 1$ indicates a higher probability of failure early in the lifetime, $k = 1$ indicates equal probability throughout the lifetime and $k > 1$ indicates an increasing probability of failure throughout the lifetime of the process/object.