DSA 5013-001 Homework 6

Group 7: Zachary White, Azadeh Gilanpour, Precious Jatau

October 28, 2018

These packages were used to find the results in Homework 6

```
library(caret) #train conrtol, train(with pls, enet, svmLinear, and lm
library(ipflasso) #lasso and RR
```

Problem 1: Predicting Violent Crime Rate

Given real-world data relating to various communities and their socio-demographics, law enforcement details, and crime statistics, your goal is to predict the community-level per capita violent crimes. The target variable is continuous and you may use any techniques at your disposal to produce a highly predictive model.

Problem 1a

Explore the provided data, conduct any desired data preparation. Summarize your approach.

The Training set and test set are imported as crimeTrain and crimeTest.

```
crimeTrain = read.csv("C:\\Users\\zackw\\Documents\\Classes\\Intelligent Data Analytics\\HW\\HW6\\crimeTrain.csv")
crimeTest = read.csv("C:\\Users\\zackw\\Documents\\Classes\\Intelligent Data Analytics\\HW\\HW6\\crimeTest.csv")
```

There were three criteria for data that was removed: **Significantly High Missingness, Too Unique Values,** and **High Corresposdence**

Columns with an average of missing values greater that 80% were removed. The columns that were removed were: *FTPolicePerPop, FTPoliceFieldPerPop, RacialMatchCommPol, PctPolicWhite, PctPolicBlack, PctPolicHisp, PctPolicAsian, PctPolicMinor, OfficAssgnDrugUnits, PolicAveOTWorked, PolicCars, PolicOperBudg, PctPolicOnPatr, GangUnitDeploy, and PolicBudgPerPop*.

```
myfun = function(x) mean(is.na(x))

missing = apply(crimeTrain,2,myfun)
missInd =  which(missing > 0.8);

# delete all variables with missingnes greater than .8
crimeTrain = crimeTrain[,-missInd]
```

Columns that were too unique were removed since the values they had could be used as categorical and did not bring much new information due to how unique the values were. When One-Hot Encoding occured with the factor parameter, the new parameters would all have near zero variance. The values that were removed were *X, county, state, and communityname.*

```
# remove column id
crimeTrain  = crimeTrain[, - which(colnames(crimeTrain) == "X")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "county")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "communityname")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "state")]
```

Columns that gave very similar information or had high correspondence were removed. Columns that gave very similar information were parameters that showed information that were already given. For example, *whitePerCap, blackPerCap, indianPerCap, AsianPerCap, and otherPerCap* is repetitive since we know the ratio of races and we know the per Capitol income of each community.

```
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "whitePerCap")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "blackPerCap")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "indianPerCap")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "AsianPerCap")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "OtherPerCap")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "HispPerCap")]
```

Another example is *NumImmig, PctImmigRecent, PctImmigRec5, PctImmigRec8, PctImmigRec10, PctRecentImmig, PctRecImmig5,* and *PctRecImmig8* show number of immigrants in total, within 5,8, and 10 years. There is a lot of intersection of these parameters and PctRecImmig10 would suffice for all of these.

```
# keep PctRecImmig10
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "NumImmig")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctImmigRecent")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctImmigRec5")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctImmigRec8")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctImmigRec10")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctRecentImmig")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctRecImmig5")]
crimeTrain = crimeTrain[, - which(colnames(crimeTrain) == "PctRecImmig8")]
```

The changes are then applied to crimeTest

```
usedVar = colnames(crimeTrain)
usedVar = usedVar[-which((usedVar) == "ViolentCrimesPerPop")]
ind = numeric(length(usedVar));
for (i in 1:length(usedVar))
{
  ind[i] =  which(colnames(crimeTest) == usedVar[i])
}
crimeTest = crimeTest[,ind]
```

At the end we have 42 parameters in crimeTrain. Those parameters are the following.

```
colnames(crimeTrain)
```

```
##  [1] "population"          "householdsize"       "racepctblack"
##  [4] "racePctWhite"        "racePctAsian"        "racePctHisp"
##  [7] "agePct12t29"         "agePct65up"          "pctUrban"
## [10] "medIncome"           "pctWWage"            "pctWFarmSelf"
## [13] "pctWInvInc"          "pctWSocSec"          "pctWPubAsst"
## [16] "pctWRetire"          "medFamInc"           "perCapInc"
## [19] "PctPopUnderPov"      "PctNotHSGrad"        "PctBSorMore"
## [22] "PctUnemployed"       "PctEmplManu"         "PctOccupMgmtProf"
## [25] "MalePctDivorce"      "MalePctNevMarr"      "FemalePctDiv"
## [28] "PctFam2Par"          "PctWorkMom"          "PctIlleg"
## [31] "PctRecImmig10"       "PctNotSpeakEnglWell" "PctLargHouseOccup"
## [34] "PctPersDenseHous"    "PctHousNoPhone"      "MedRentPctHousInc"
## [37] "NumInShelters"       "NumStreet"           "PctSameState85"
## [40] "PopDens"             "PctUsePubTrans"      "ViolentCrimesPerPop"
```

Problem 1b

Using cross-validation of your choice, perform PLS, ridge regression, lasso, elastic net, and SVM-regression to tune the associated hyper-parameters and estimate the generalizable error for a model to predict the community-level per capita violent crimes. Provide a chart summarizing the hyper-parameter search for each approach.

The results are shown in the table.

Model Description	R Package	Tuned Hyper-Parameters	CV RMSE	CV R2
Partial Least Squares	caret with pls	ncomp = 8	0.1377	0.6602
SVM (Linear)	caret with svmLinear	Cost = .1	0.1397	0.6560
Ridge Regression	ipflasso	lambda = 0.0192	0.1342	0.6742
Lasso	ipflasso	lambda = 0.000875	0.1336	0.6770
Elastic Net	caret with enet	alpha = 0.525, lambda = 0.0001	0.1337	0.6766

Cross Validation

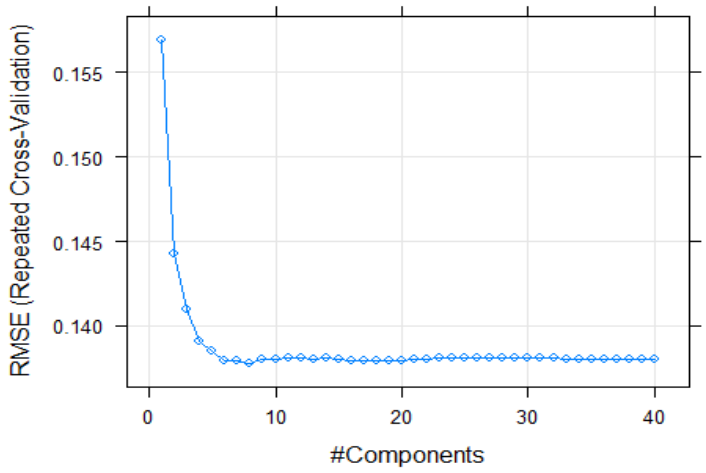
The Cross Validation technique used for all Models is a repeated k-fold Cross Validation, where k = 10 and it is repeated 10 times.

```
ctrl = trainControl(method="repeatedcv", number = 10, repeats = 10)
```

Partial Least Squares

The caret package is used to train pls.fit with a tuneLength of 42.

```
pls.fit = train(ViolentCrimesPerPop ~ ., data = crimeTrain, method = "pls",trControl=ctrl, tuneLength = 42)

plot(pls.fit)
```



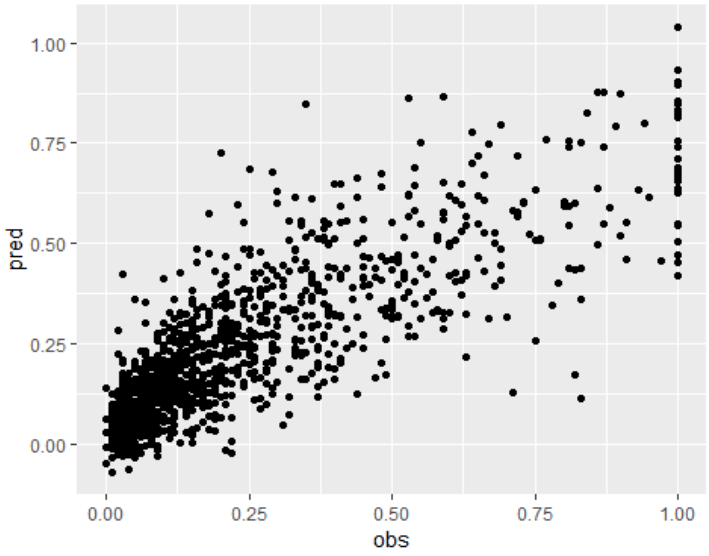

```
pls.fit$bestTune

##   ncomp
## 8     8
```

By looking at the chart and bestTune of pls.fit, ncomp = 8 is most optimal for this model.

```
pls.fit = train(ViolentCrimesPerPop ~ ., data = crimeTrain, method = "pls",trControl=ctrl, tun
eGrid = expand.grid(ncomp = pls.fit$bestTune))

pls.fit

##   RMSE       Rsquared   MAE
##   0.1377132  0.6588015  0.09703326
## Tuning parameter 'ncomp' was held constant at a value of 8
```

To check the prediction, we look at the prediction on the crimeTrain dataset and then plot the observed ViolentCrimesPerPop vs prediction

```
pls.predict = predict(pls.fit,crimeTrain)
ggplot(data = data.frame(obs = crimeTrain$ViolentCrimesPerPop, pred = pls.predict )) + geom_po
int(aes(x = obs, y = pred))
```

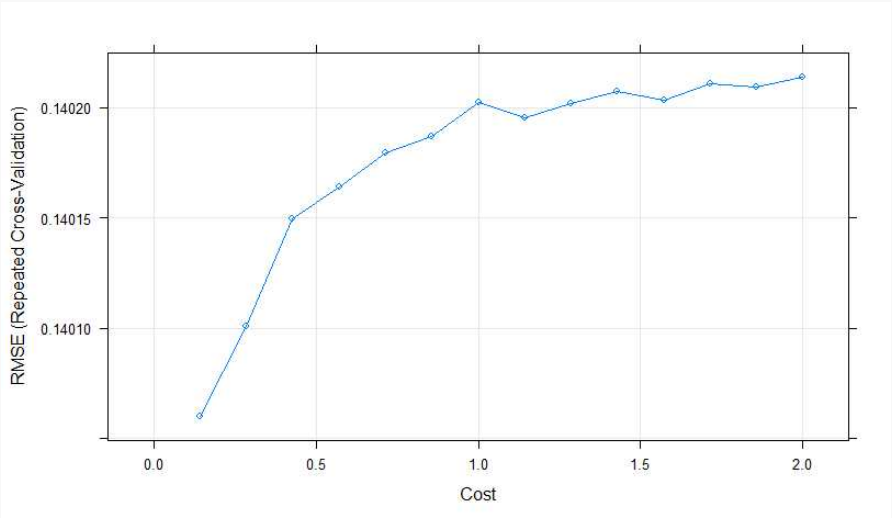


The data moves mostly in in linear line to the right which shows that the model is sound.

Support Vector Machine

Set grid2 to tune the data of the cost from 0 to 2 with a length of 15 and set this to tune grid in train. (Note: do not run this, this takes a while to run)

```
grid2 <- expand.grid(C = seq(0,2,length=15)) #runtime is long

# using a linear kernel
svm.lm <- train(data=crimeTrainNoState, ViolentCrimesPerPop ~ ., method = "svmLinear",
    preProc = c("center","scale"), trControl=ctrl ,tuneGrid = grid2)
```

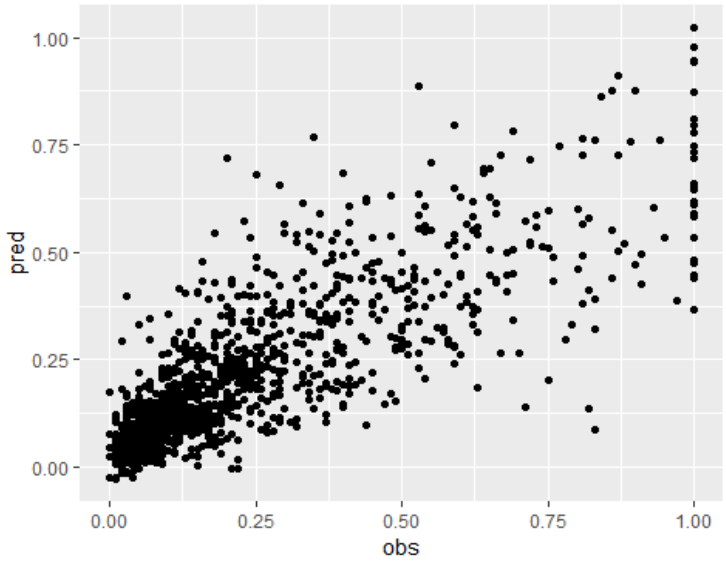


By plotting the cost vs the RMSE for the SVM linear model, we see that C = 0.1 gives the more optimal model. So svm.lm is trained on C = 0.1.

```
svm.lm <- train(data=crimeTrain,
                  ViolentCrimesPerPop ~ .,
                  method = "svmLinear",   # linear kernel
                  preProc = c("center","scale"),  # Center and scale data
                  trControl=ctrl ,tuneGrid = expand.grid(C = .1))

svm.lm

## Support Vector Machines with Linear Kernel
##   RMSE       Rsquared   MAE
##   0.1397551  0.6552908  0.09292356
## Tuning parameter 'C' was held constant at a value of 0.1
```

To check the prediction like above, we look at the prediction on the crimeTrain dataset and then plot the observed ViolentCrimesPerPop vs. prediction

```
x = data.matrix(crimeTrain[,-which(colnames(crimeTrain) == "ViolentCrimesPerPop")])

# bestTune c = .1,
svmLm.predict = predict(svm.lm,x)

ggplot(data = data.frame(obs = crimeTrain$ViolentCrimesPerPop, pred = svmLm.predict )) + geom_
point(aes(x = obs, y = pred))
```



The data moves mostly in in linear line to the right which shows that the model is sound.

Ridge Regression

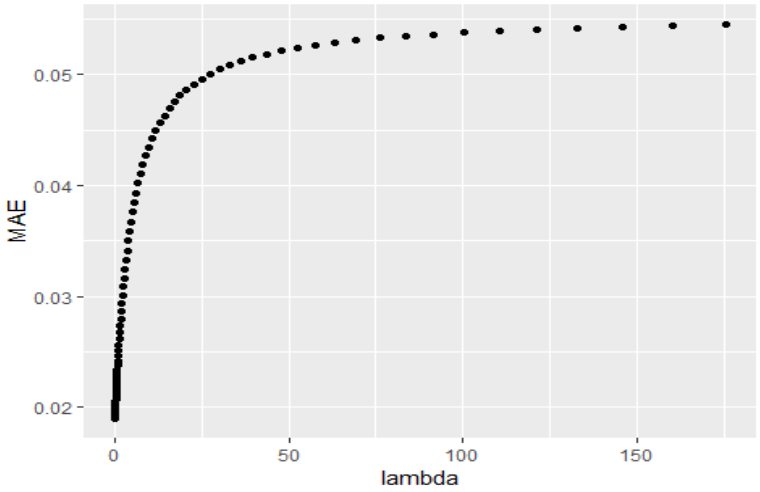
We use cvr.glmnet to run repeated k-fold cross validation on ridge regression where k = 10 and is repeated 10 times.

```
## ridge regression
ridge.cv = cvr.glmnet(x,y, family = "gaussian", alpha = 0, nfolds = 10, ncv = 10)
lambdaOpt = ridge.cv$lambda[which.min(ridge.cv$cvm)]
```

After running the model, the minimal lambda found which would be most optimal is 0.0192.

When plotting the lambda vs the Mean Absolute Error, we see that MAE goes to 0 while lambda approaches 0.

```
# plot for tuning
ggplot(data = data.frame(MAE = ridge.cv$cvm, lambda = ridge.cv$lambda), aes(x = lambda, y = MA
E)) + geom_point()
```

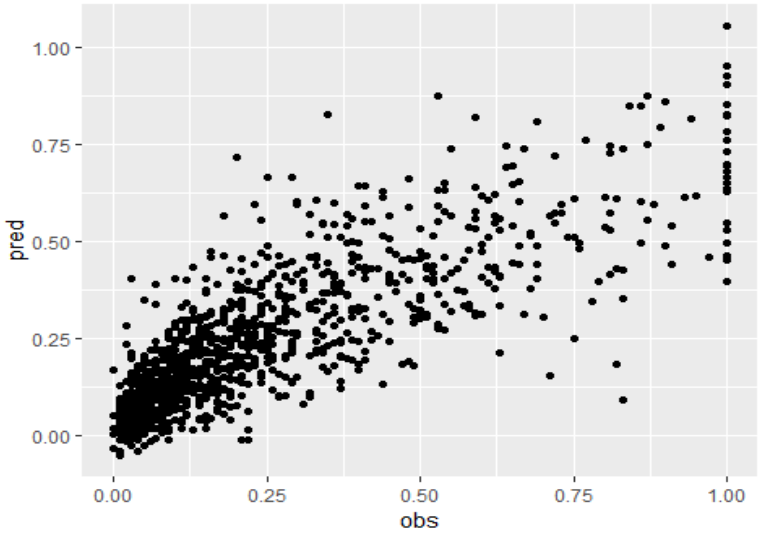


A new model is made with glmnet with lambda = 0.0192.

```
# fit model
ridge.model = glmnet(x = x,y = y, lambda = lambdaOpt, family = "gaussian", alpha = 0)
```

In order to check the prediction like above, we look at the prediction on the crimeTrain dataset and then plot the observed ViolentCrimesPerPop vs. prediction

```
# make predictions on training data
ridge.predict = predict.glmnet(object = ridge.model, newx = x)

# compare training and test
observedPredicted.df = data.frame(Observed = crimeTrain$ViolentCrimesPerPop,Predicted = ridge.
predict)
colnames(observedPredicted.df) = c("obs", "pred")
ggplot(data = observedPredicted.df) + geom_point(aes(x = obs, y = pred))
```



The data moves mostly in in linear line to the right which shows that the model is sound.

Finally, we use defaultSummary to find RMSE and RSquared

```
# rmse, Rsquared, MAE
ridge.perf = defaultSummary(observedPredicted.df)
ridge.perf

##       RMSE   Rsquared        MAE
## 0.13428571 0.67424699 0.09342158
```

Lasso

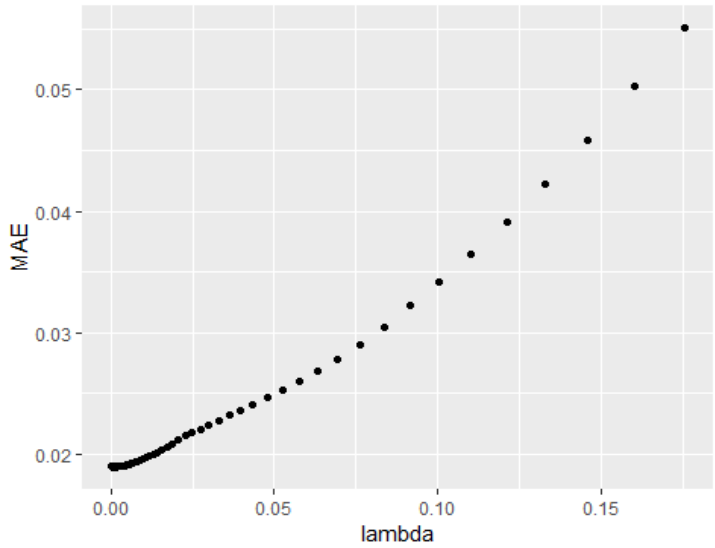
We use cvr.glmnet to run repeated k-fold cross validation on ridge regression where k = 10 and is repeated 10 times.

```
lasso.cv = cvr.glmnet(x,y, family = "gaussian", alpha = 1, nfolds = 10, ncv = 10)

lambdaOptLasso = lasso.cv$lambda[which.min(lasso.cv$cvm)]
```

After running the model, the minimal lambda found which would be most optimal is 0.000875

When plotting the lambda vs the Mean Absolute Error, we see that MAE goes to 0 while lambda approaches 0.

```
# plot for tuning
ggplot(data = data.frame(MAE = lasso.cv$cvm, lambda = lasso.cv$lambda), aes(x = lambda, y = MA
E)) + geom_point()
```

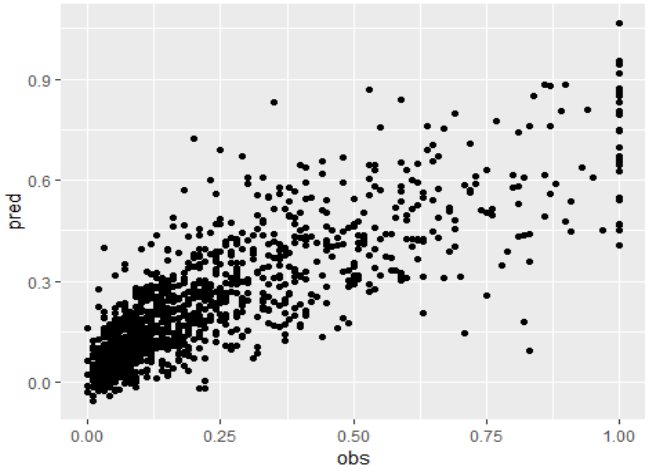


A new model is made with glmnet with lambda = 0.000875.

```
# fit model
lasso.model = glmnet(x = x,y = y, lambda = lambdaOptLasso, family = "gaussian", alpha = 1)
```

To check the prediction like above, we look at the prediction on the crimeTrain dataset and then plot the observed ViolentCrimesPerPop vs. prediction

```
# make predictions on training data
lasso.predict = predict.glmnet(object = lasso.model, newx = x)

# compare training and test
observedPredictedLasso.df = data.frame(Observed = crimeTrain$ViolentCrimesPerPop,Predicted = l
asso.predict)
colnames(observedPredictedLasso.df) = c("obs", "pred")

# observed vs predicted for lasso
ggplot(data = observedPredictedLasso.df) + geom_point(aes(x = obs, y = pred))
```



The data moves mostly in in linear line to the right which shows that the model is sound.

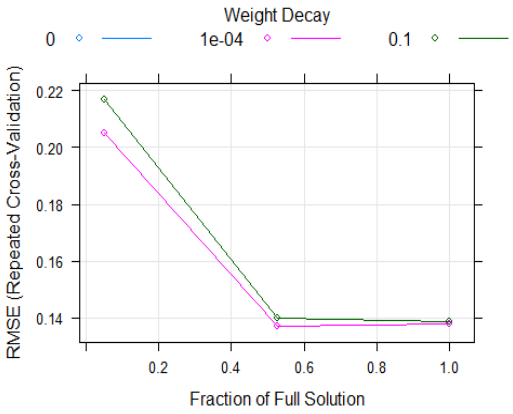
Finally, we use defaultSummary to find RMSE and RSquared

```
# rmse, r_squared, MAE
lasso.perf = defaultSummary(observedPredictedLasso.df)
lasso.perf

##       RMSE   Rsquared        MAE
## 0.13366256 0.67702189 0.09311159
```

Elastic Net

We create a grid that goes from fraction from 0.2 to 0.8 and lambda from 0 to .1 all for a length of 100.

```
# did sparse search, best tune was fraction = .525, lambda = 1e-04
alphaGrid = expand.grid(fraction = seq(0.2,0.8,length.out = 100), lambda = seq(0,.1,length.out
= 100))
eNet.model = train(data = crimeTrain, ViolentCrimesPerPop ~ ., method = "enet", trControl = ct
rl)

# cross validation
plot(eNet.model) # cross validation
```

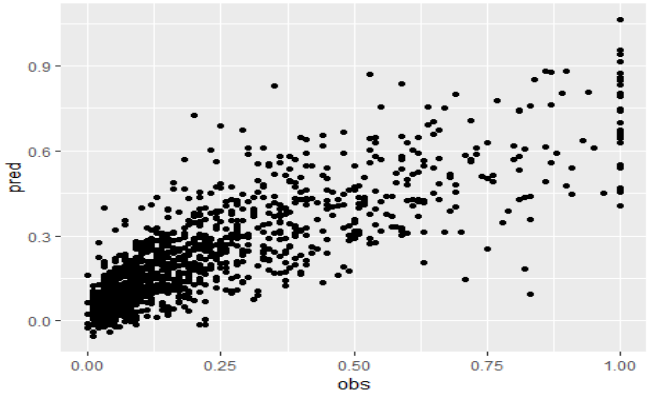



```
eNet.model$bestTune # show best parameters

##   fraction lambda
## 5    0.525  1e-04
```

When we look at the chart and the best tune, we see that fraction = 0.525 and lambda = .0001 is most optimal

To check the prediction like above, we look at the prediction on the crimeTrain dataset and then plot the observed ViolentCrimesPerPop vs. prediction

```
# set predictors and response
x = data.matrix(crimeTrain[,-which(colnames(crimeTrain) == "ViolentCrimesPerPop")])

# make predictions
eNet.predict = predict(eNet.model, x)
obsPredEnet.df = data.frame(obs = crimeTrain$ViolentCrimesPerPop, pred = eNet.predict)

# plot observed vs predicted
ggplot(data = obsPredEnet.df) + geom_point(aes(x = obs, y = pred))
```



The data moves mostly in in linear line to the right which shows that the model is sound.

Finally, we use defaultSummary to find RMSE and RSquared

```
# calculate r-squared, RMSE, MAE
eNet.perf = defaultSummary(obsPredEnet.df)
eNet.perf

##       RMSE   Rsquared        MAE
## 0.13375758 0.67657099 0.09312979
```

Problem 1c

Using any regression technique or combination of techniques you prefer, predict the predict the community-level per capita violent crimes.

This is the concept on one the models: the linear model methods.

Since lasso, elastic Net, and Ridge Regression all have the value of lambda near 0, that we can view this as a linear model. From here, we decided to make a new linear model with a new cleaned data set. So crimeTrain2 is created with the raw data with all variables with the average of missing values greater than 0% removed from the data set. X, state, and community are removed too since they are either too unique or factors that will cause near zero variance.

```
crimeTrain2 = read.csv("C:\\Users\\zackw\\Documents\\Classes\\Intelligent Data Analytics\\HW\\
HW6\\crimeTrain.csv")
crimeTrain2 = crimeTrain2[,!(colMeans(is.na(crimeTrain)) > 0)]
crimeTrain2 = crimeTrain2[,-1:-3]
```

We made a linear model with all variables.

```
linMod.fit = lm(data = crimeTrain2, ViolentCrimesPerPop ~.)
```

Next for more feature selection, we used step function with all variables to move backwards to see which variables would have the most effcient AIC and RSS. We save this as linMod.backward.

```
linMod.backward = step(linMod.fit, direction = "backward")
```

The model step returns is *ViolentCrimesPerPop ~ population + racepctblack + agePct12t29 + numbUrban + pctUrban + pctWWage + pctWFarmSelf + pctWInvInc + pctWRetire + whitePerCap + indianPerCap + HispPerCap + PctPopUnderPov + PctLess9thGrade + PctEmploy + PctEmplManu + PctOccupManu + PctOccupMgmtProf + MalePctDivorce + MalePctNevMarr + FemalePctDiv + PersPerFam + PctKids2Par + PctWorkMom + PctIlleg + PctImmigRec5 + PctImmigRec8 + PctImmigRec10 + PctRecentImmig + PctRecImmig8 + PctRecImmig10 + PctSpeakEnglOnly + PctNotSpeakEnglWell + PersPerOccupHous + PersPerRentOccHous + PctPersOwnOccup + PctPersDenseHous + PctHousLess3BR + MedNumBR + PctHousOccup + PctHousOwnOcc + PctVacantBoarded + PctVacMore6Mos + OwnOccLowQuart + RentLowQ + MedRent + MedRentPctHousInc + MedOwnCostPctInc + MedOwnCostPctIncNoMtg + NumStreet + PctForeignBorn + PctSameCity85 + PctUsePubTrans*

We use this to set an upper bound when we use the step function again for the forward direction.

```
range = list(upper=~population + racepctblack +
               agePct12t29 + numbUrban + pctUrban + pctWWage + pctWFarmSelf +
               pctWInvInc + pctWRetire + whitePerCap + indianPerCap + HispPerCap +
               PctPopUnderPov + PctLess9thGrade + PctEmploy + PctEmplManu +
               PctOccupManu + PctOccupMgmtProf + MalePctDivorce + MalePctNevMarr +
               FemalePctDiv + PersPerFam + PctKids2Par + PctWorkMom + PctIlleg +
               PctImmigRec5 + PctImmigRec8 + PctImmigRec10 + PctRecentImmig +
               PctRecImmig8 + PctRecImmig10 + PctSpeakEnglOnly + PctNotSpeakEnglWell +
               PersPerOccupHous + PersPerRentOccHous + PctPersOwnOccup +
               PctPersDenseHous + PctHousLess3BR + MedNumBR + PctHousOccup +
               PctHousOwnOcc + PctVacantBoarded + PctVacMore6Mos + OwnOccLowQuart +
               RentLowQ + MedRent + MedRentPctHousInc + MedOwnCostPctInc +
               MedOwnCostPctIncNoMtg + NumStreet + PctForeignBorn + PctSameCity85 +
               PctUsePubTrans, lower=~.)

linMod.forward = step(lm(data = crimeTrain2, ViolentCrimesPerPop~1),range, direction="forward"
)
```

To make a model with the fewer variables, we will be using linMod.forward. Looking at the summary of linMod.forward, we remove the Pr(>|t|) values greater than .05. Then look at the summary of the new model and repeating the process of removing variables until all variables have Pr(>|t|) values less than .05.

By that standard we remove *FemalePctDiv, MalePctDivorce, PctSpeakEnglOnly, PctForeignBorn, MalePctNevMarr, MedRentPctHousInc, PctNotSpeakEnglWell, PctUsePubTrans, and MedOwnCostPctInc*. We now have a model:

lm(formula = ViolentCrimesPerPop ~ PctIlleg + PctPersDenseHous + PctHousOccup + pctUrban + racepctblack + NumStreet + PctKids2Par + PctWorkMom + agePct12t29 + PctEmploy + MedOwnCostPctIncNoMtg + PctVacantBoarded + PctVacMore6Mos + pctWWage + pctWRetire + PctPopUnderPov + RentLowQ + MedRent + whitePerCap + PctEmplManu, data = crimeTrain2)

```
summary(linMod.forward)

lm(formula = ViolentCrimesPerPop ~ PctIlleg + PctPersDenseHous +
    PctHousOccup + pctUrban + racepctblack + NumStreet + PctKids2Par +
    PctWorkMom + agePct12t29 + PctEmploy + MedOwnCostPctIncNoMtg +
    PctVacantBoarded + PctVacMore6Mos + pctWWage + pctWRetire +
    PctPopUnderPov + RentLowQ + MedRent + whitePerCap + PctEmplManu,
    data = crimeTrain2)
```

From here we did cross validation with this linear model relation. The CV technique was repeated k-fold where k = 10 and was repeated 100.

```
ctrl.lm <- trainControl(method="repeatedcv", number=10, repeats = 100)

lm.model = train(ViolentCrimesPerPop ~ PctIlleg + PctPersDenseHous + PctHousOccup + pctUrban +
racepctblack +
                   NumStreet + PctKids2Par + PctWorkMom + agePct12t29 + PctEmploy + MedOwnCost
PctIncNoMtg +
                   PctVacantBoarded + PctVacMore6Mos + pctWWage + pctWRetire +
                   PctPopUnderPov + RentLowQ + MedRent + whitePerCap + PctEmplManu,
                 data = crimeTrain, method = "lm",trControl = ctrl.lm)
```

This gives a model with an Rsquare = 0.6758 and RMSE = 0.1341.