Exercise 3

Konstantinos Vakalopoulos 12223236

2022-11-17

Premilinary work

First, the ISLR package was installed

```
install.packages("ISLR")
```

and then the data was loaded with.

```
library("ISLR")
data(College,package="ISLR")
```

For addition information about the data, the following commands were used.

```
?College
str(College)
```

The observations which contain missing values and the variables Accept and Enroll were removed from the original data.

```
College <- na.omit(College) #Check for NA observations
College <- College[,-c(3,4)] #Remove the Accept and Enroll variables
```

The values of the Apps variable has been changed to log transformed.

```
College$Apps <- log(College$Apps) #log transformation
```

Finally, the data was split randomly into training and test data (about 2/3 and 1/3).

```
#Split the data into train and test
set.seed(12223236)
n <- nrow(College)
train <- sample(1:n,round(n*2/3))
test <- (1:n)[-train]
```

Question 1(a)

From the library MASS, the function lm.ridge() was used on the train data in order to apply ridge regression.

```
library(MASS)
```

```
res.ridge <- lm.ridge(Apps~., data=College, lambda=seq(0,50, by=0.05), subset=train)
```

For the ridge parameter lambda, a range from 0 to 50 with step equals to 0.05 was considered, in order to find the optimal value. In figure 1 is presented the resulting GCV against the examined lambda.

```
plot(res.ridge$lambda,res.ridge$GCV,type="l", ylab = "GCV", xlab="Lambda")
```

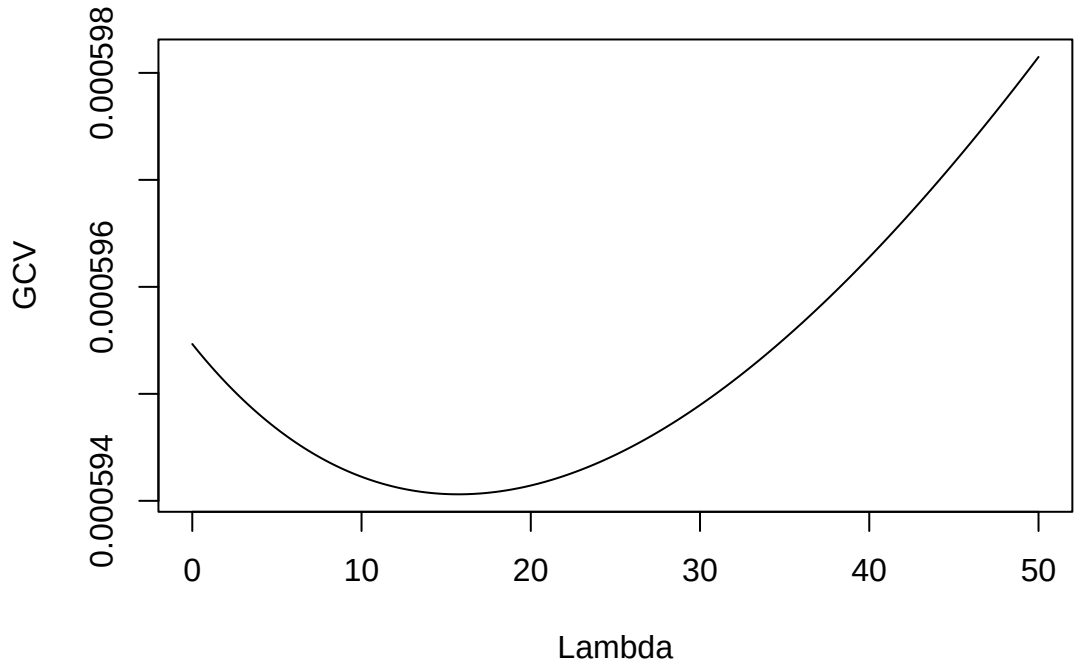

Figure 1: Lambda parameter

The optimal lambda is:

```
lambda.opt <- res.ridge$lambda[which.min(res.ridge$GCV)]
lambda.opt
```

```
## [1] 15.7
```

and the minimum GCV error is:

```
min(res.ridge$GCV)
```

```
## [1] 0.0005940612
```

Question 1(b)

The ridge regression model was created using the optimal lambda from the question 1(a).

```
res.ridge.opt <- lm.ridge(Apps~., data=College, lambda = lambda.opt,
                          subset=train) #scales variables
```

The regression coefficients are:

```
res.ridge.opt$coef #coefficients for scaled data
```

```
##  PrivateYes   Top10perc   Top25perc F.Undergrad P.Undergrad    Outstate
## -0.26958065 -0.00920632  0.08821321  0.49443271  0.03246492  0.15740295
##  Room.Board       Books    Personal         PhD    Terminal   S.F.Ratio
##  0.11461523  0.06124122  0.02063491  0.10000619  0.03160285  0.14878738
## perc.alumni      Expend   Grad.Rate
## -0.08226188  0.13052280  0.19338144
```

However these coefficients are scaled and the intercept is not included. Thus the unscaled coefficients with the intercept are:

```
ridge.coef <- coef(res.ridge.opt) #unscaled with the intercepet
ridge.coef
```

```
##                  PrivateYes     Top10perc     Top25perc   F.Undergrad
##  4.105293e+00 -5.967422e-01 -5.344671e-04  4.450536e-03  1.033235e-04
##   P.Undergrad      Outstate    Room.Board         Books      Personal
##  1.935716e-05  3.997694e-05  1.049318e-04  3.529908e-04  2.929392e-05
##           PhD      Terminal     S.F.Ratio   perc.alumni        Expend
##  5.944139e-03  2.133119e-03  3.791109e-02 -6.661536e-03  2.430616e-05
##     Grad.Rate
##  1.158254e-02
```

Question 1(c)

In order to predict the response for the test data, first the response variable was removed and then a column with 1s was added to the test data which indicates the intercept. Second, the explanatory variable "Private" was converted to binary variable. Finally, the predictions for the response variable were calculated by multiplying the test data matrix with the unscaled coefficients from the previous question.

```
test.data <- cbind(rep(1,length(test)),College[test,-2])
test.data$Private <- ifelse(test.data$Private=="Yes",1,0)
pred.ridge <- data.matrix(test.data)%*%ridge.coef
```

In figure 2 are presented the predicted and the actual values for the response variable Apps. According to the plot, the performance of the model is quite high because of the linearity of the actual and predicted values.

```
plot(College[test,"Apps"],pred.ridge, xlab="Measured",
     ylab = "Predicted", main = "Apps",
     xlim = c(5,12), ylim=c(5,12))
abline(c(0,1))
```

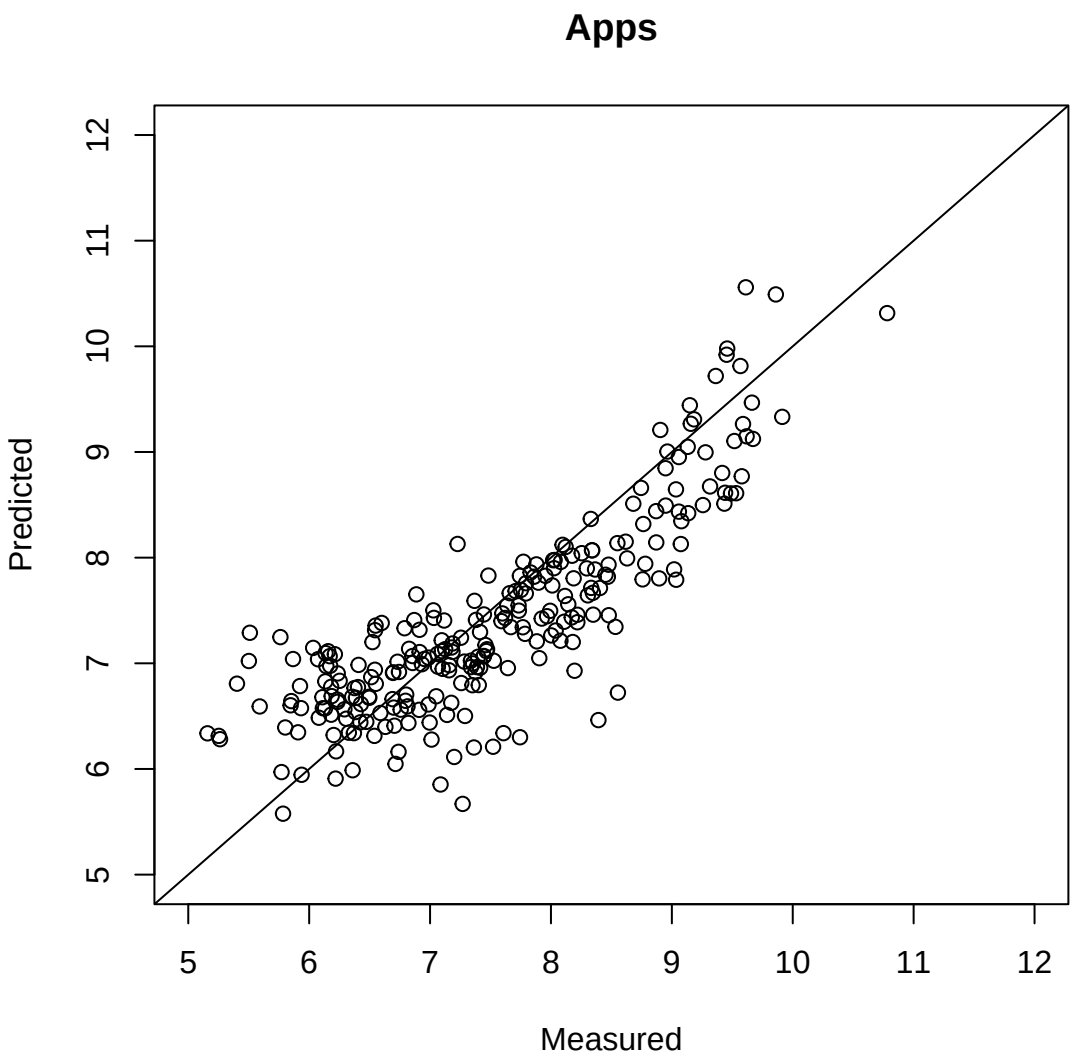

Figure 2: Predicted vs Measured

The Root Mean Square Error is:

```
mean((College[test,"Apps"]-pred.ridge)^2)
```

```
## [1] 0.3794085
```

It turns out that the performance of the model is quite competitive.

Now the RMSE was calculated for the linear model, and the models using PCR and PLS. For the linear model, the RMSE on the test data is:

```
linear.m <- lm(Apps~., data = College, subset = train)
pred.linear <- predict(linear.m,newdata = College[test,])
sqrt(mean((College$Apps[test]-pred.linear)^2))
```

```
## [1] 0.6152971
```

For the reduced linear model using PCR, the RMSE on the test data is:

```
library(pls)
```

```
## [1] 0.5587204
```

For the reduced linear model using PLS, the RMSE on the test data is:

```
pls_model <- plsr(College$Apps~., data=College, scale=TRUE, subset=train,
                  ncomp=3)
pred_pls <- predict(pls_model,newdata=College[train,],ncomp=3)
sqrt(mean((College$Apps[train]-pred_pls)^2))
```

```
## [1] 0.5480557
```

In conclusion, the best result of the RMSE was given by the ridge regression model.

Question 2(a)

For the creation of the lasso regression model, the function glmnet() for the library glmnet was used on the train data.

```
library(glmnet)
```

```
data <- College
data$Private <- ifelse(data$Private=="Yes",1,0)
res.lasso <- glmnet(data.matrix(data[train,-2]),data[train,2], alpha=1)
```

In the plot below, the results from the glmnet function are shown.

```
plot(res.lasso, ylim=c(-0.1,0.1))
```

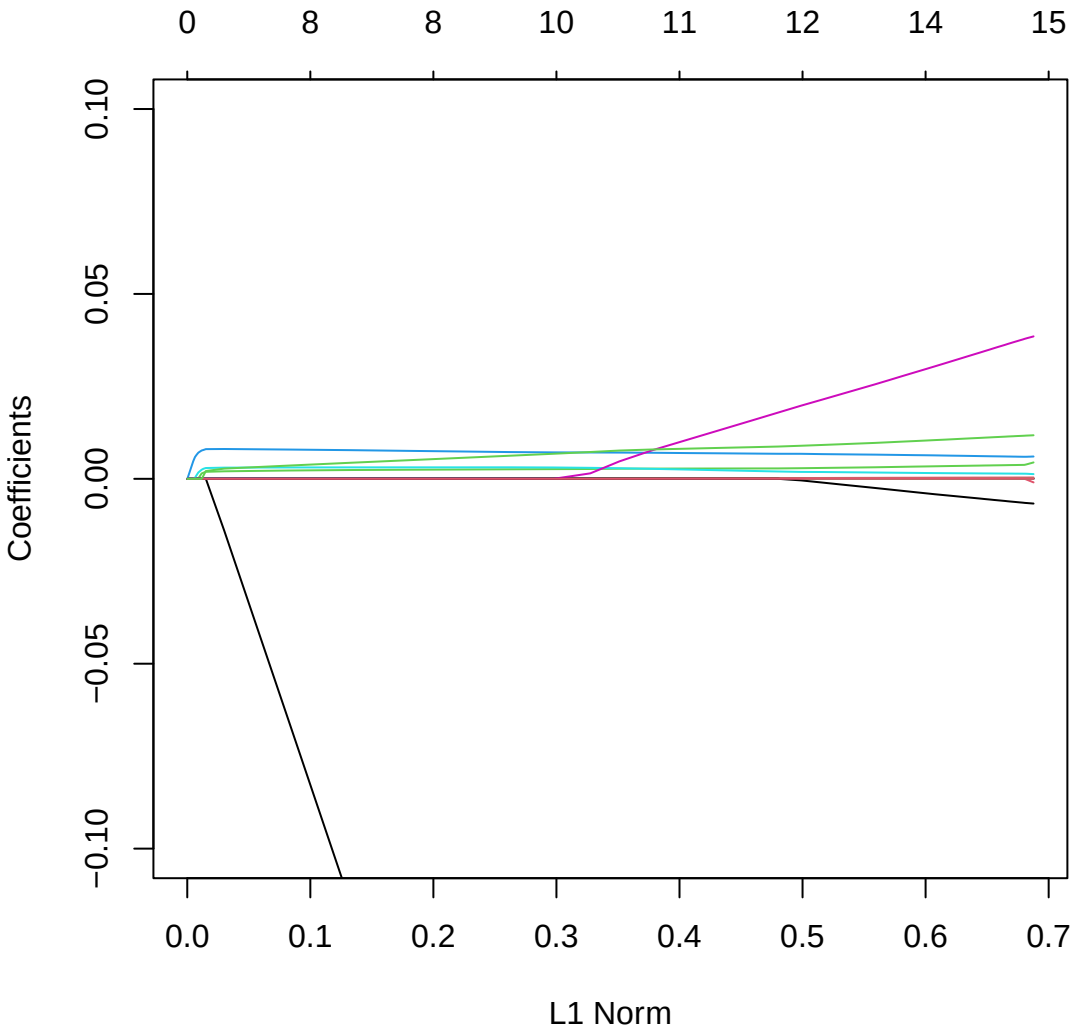

Figure 3: L1 Norm for the explanatory variables

In the y axis of the plot are the coefficients, in the bottom x axis is the L1 norm is the penalty parameter for the lasso regression. While the L1 norm is increased, the coefficients are, also, increasing away from zero. Finally, at the top x axis are the number of variables in the model. Thus, for different L1 norms or the lasso's penalty, the number of variables that are zero and non zero can be seen.

The default parameter for lambda is the nlambda. Although glmnet fits the model for 100 values of lambda by default, it stops early if %dev does not change sufficiently from one lambda to the next. In our case, we stopped for lambda equal to 72.

```
print(res.lasso)
```

```
##
## Call:  glmnet(x = data.matrix(data[train, -2]), y = data[train, 2],      alpha = 1)
##
##    Df  %Dev  Lambda
```

```
## 1   0  0.00 0.76080
## 2   1  8.98 0.69320
## 3   1 16.43 0.63160
## 4   1 22.62 0.57550
## 5   1 27.76 0.52440
## 6   1 32.02 0.47780
## 7   2 35.57 0.43540
## 8   2 40.02 0.39670
## 9   2 43.71 0.36140
## 10  2 46.78 0.32930
## 11  3 49.36 0.30010
## 12  3 51.54 0.27340
## 13  4 53.40 0.24910
## 14  4 55.18 0.22700
## 15  4 56.65 0.20680
## 16  6 58.36 0.18850
## 17  6 59.91 0.17170
## 18  6 61.20 0.15650
## 19  7 62.41 0.14260
## 20  7 63.80 0.12990
## 21  8 64.99 0.11840
## 22  8 66.02 0.10780
## 23  8 66.87 0.09826
## 24  8 67.58 0.08953
## 25  9 68.20 0.08158
## 26 10 68.82 0.07433
## 27 10 69.44 0.06773
## 28 11 70.00 0.06171
## 29 11 70.50 0.05623
## 30 11 70.92 0.05123
## 31 11 71.26 0.04668
## 32 11 71.55 0.04253
## 33 12 71.83 0.03876
## 34 12 72.09 0.03531
## 35 12 72.31 0.03218
## 36 12 72.48 0.02932
## 37 13 72.63 0.02671
## 38 13 72.76 0.02434
## 39 13 72.87 0.02218
## 40 13 72.96 0.02021
## 41 13 73.03 0.01841
## 42 14 73.09 0.01678
## 43 14 73.15 0.01529
## 44 14 73.19 0.01393
## 45 14 73.23 0.01269
## 46 14 73.26 0.01156
## 47 14 73.29 0.01054
## 48 14 73.31 0.00960
## 49 14 73.33 0.00875
## 50 14 73.34 0.00797
## 51 14 73.35 0.00726
## 52 14 73.37 0.00662
## 53 14 73.37 0.00603
## 54 14 73.38 0.00549
```

```
## 55 14 73.39 0.00500
## 56 14 73.39 0.00456
## 57 14 73.40 0.00416
## 58 14 73.40 0.00379
## 59 14 73.40 0.00345
## 60 14 73.40 0.00314
## 61 14 73.41 0.00286
## 62 14 73.41 0.00261
## 63 15 73.41 0.00238
## 64 15 73.41 0.00217
## 65 15 73.42 0.00197
## 66 15 73.42 0.00180
## 67 15 73.42 0.00164
## 68 15 73.42 0.00149
## 69 15 73.42 0.00136
## 70 15 73.42 0.00124
## 71 15 73.43 0.00113
## 72 15 73.43 0.00103
```

Regarding the alpha parameter, The alpha parameter is mixing parameter, which gets values between [0,1]. Alpha parameter is calculated using the formula (see ?glmnet). For alpha=1 we get the lasso penalty, and alpha=0 we get ridge penalty. For alpha values different from 0 and 1, we have regression called elastic net and it, actually, combines the lasso and the ridge penalty.

Question 2(b)

The function cv.glmnet() was used on the train data in order to parameters to be tuned and then to obtain the optimal ones through 10-fold cross validation.

```
res.cv <- cv.glmnet(data.matrix(data[train,-2]),data[train,2])
```

The results are plotted in the figure below.

```
plot(res.cv)
```

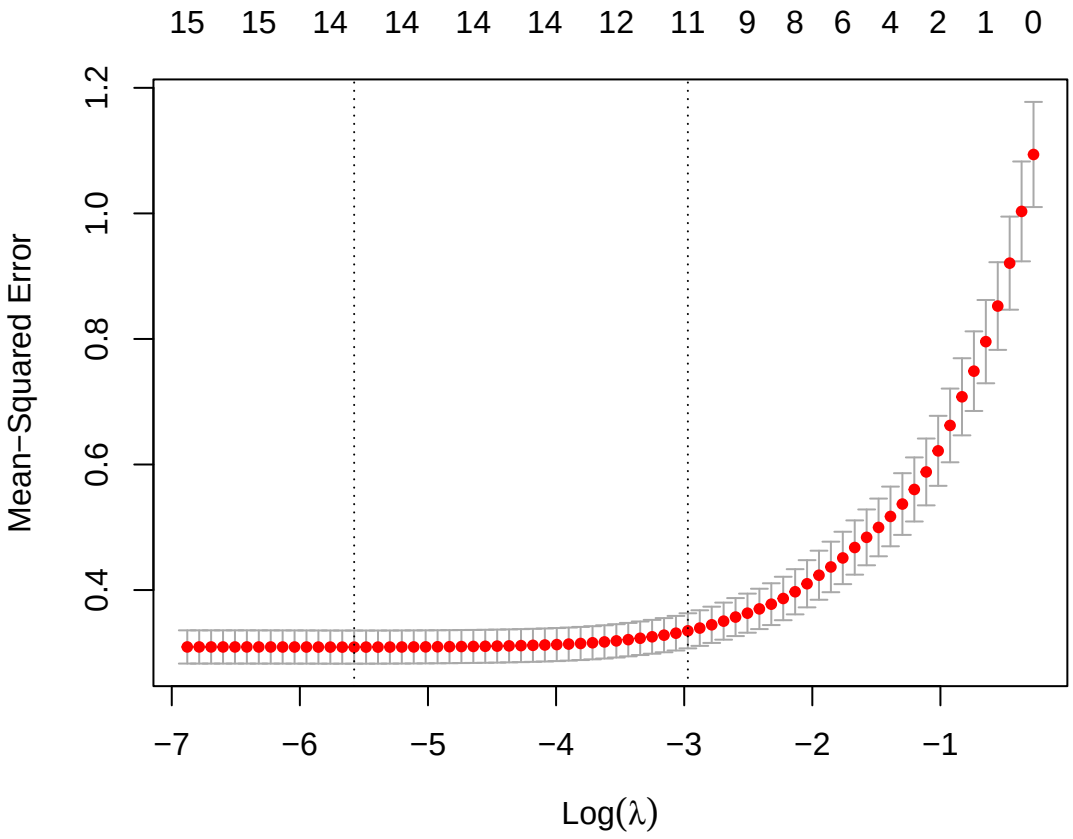

Figure 4: Cross Validation

The plot shows the cross validation mean squared error along with upper and lower standard deviation mean squared error along the lambda sequence. Also, two vertical dotted lines are created. the first line on the left, indicates the value of lambda with the minimum mean cross-validated error. However, the second dotted line indicates the optimal lambda value the minimum cross validation standard error. Finally, the smaller the log(lambda) the bigger the model until we reach to the linear full model.

The optimal tuning parameter is:

```
res.cv$lambda.1se
```

```
## [1] 0.05123279
```

and the regression coefficients are:

```
coef(res.cv,s="lambda.1se")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  4.901549e+00
## Private     -3.999977e-01
```

```
## Top10perc    .
## Top25perc    2.792322e-03
## F.Undergrad  1.160141e-04
## P.Undergrad  .
## Outstate     8.028387e-06
## Room.Board   1.110772e-04
## Books        1.540436e-04
## Personal     .
## PhD          6.890868e-03
## Terminal     2.305965e-03
## S.F.Ratio    1.331495e-02
## perc.alumni  .
## Expend       1.325844e-05
## Grad.Rate    8.369605e-03
```

Question 2(c)

The optimal model, based on the previous question, was used to predict the response for the test data.

```
pred.lasso <- predict(res.cv,newx=data.matrix(data[test,-2]),s="lambda.1se")
```

In figure 2 are presented the predicted and the actual values for the response variable Apps. According to the plot, the performance of the model is quite high because of the linearity of the actual and predicted values.

```
plot(data[test,"Apps"],pred.lasso, xlab="Measured",
     ylab = "Predicted", main = "Apps",
     xlim = c(5,12), ylim=c(5,12))
abline(c(0,1))
```

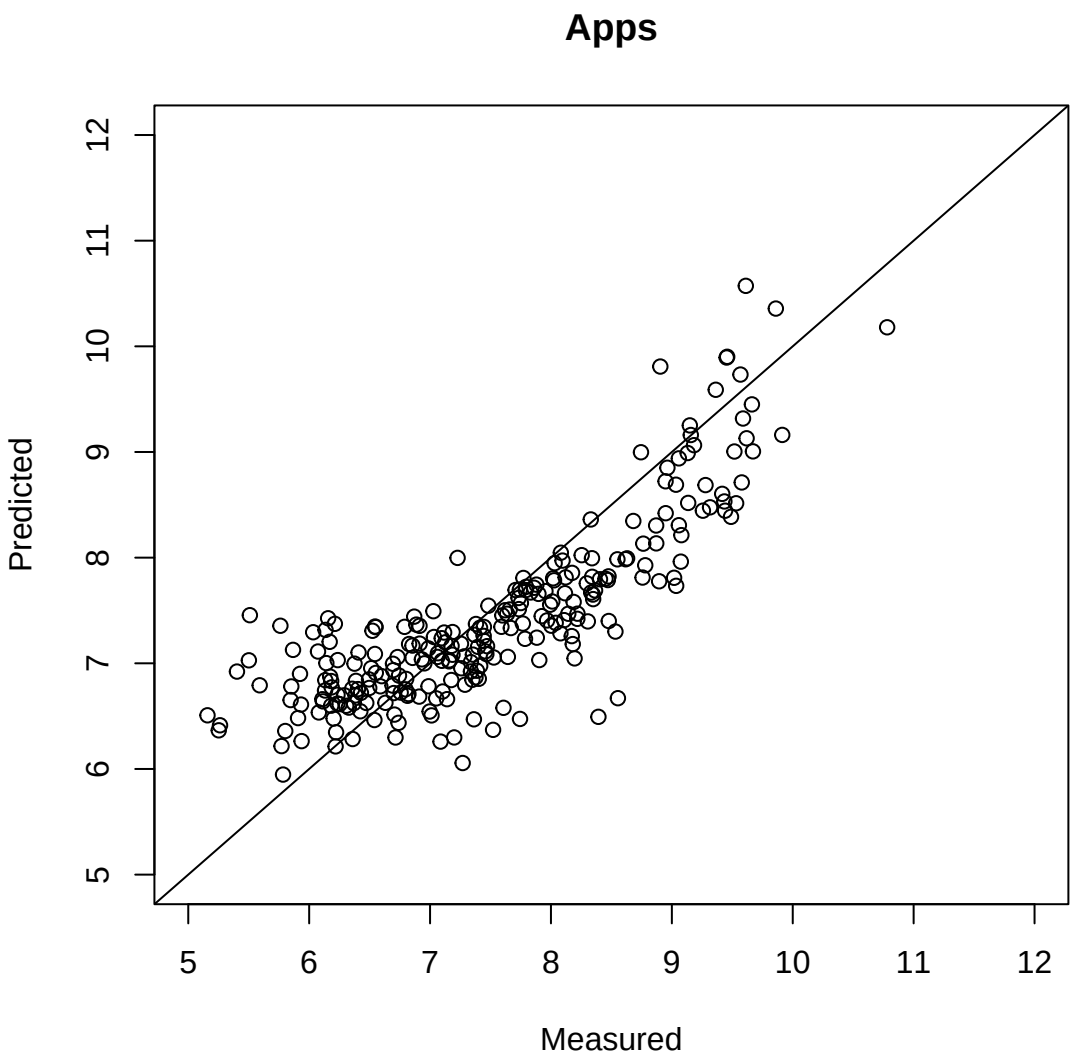

Figure 5: Predicted vs Measured

The Root Mean Squared Error for the Lasso regression is:

```
mean((data[test,"Apps"]-pred.lasso)^2)
```

```
## [1] 0.4042784
```

and the Ridge regression RMSE is:

```
mean((College[test,"Apps"]-pred.ridge)^2)
```

```
## [1] 0.3794085
```

Therefore, the Ridge Regression RMSE is slightly lower than the Lasso regression RMSE.