STATS 790
Written Report for Final Project

Amandeep Sidhu (400076920)

21 April, 2023

Contents

Introduction

In this final project, the data set used for our techniques will be the built in "Boston Housing" data set that is available in R and Python (Harrison Jr. and Rubinfeld 1978). According to the description provided by the original data collector, it was collected in the 1970s by Harrison and Rubinfeld during the census. This data set includes 506 observations with 13 predictors that aim to predict the value of the houses in the area of Boston. These predictors include: CRIM (crime rate), AGE (proportion of owner-occupied units built prior to 1940), DIS (weighted distances to employment centres) and LSTAT (the lower status/class of the population in Boston). The target variable (denoted as 'MEDV') is the median value of the homes in thousands of dollars. This data set was first used in a paper titled "Housing Prices and the Demand for Clean Air" (Harrison Jr. and Rubinfeld 1978) as the demand for clean air rose with pollution becoming a predominant issue in modern society. As a naive approach, Harrison and Rubinfeld used a simple linear regression model (least squares method) to estimate the parameters. On top of this, they simply used a variety of different combinations of the predictors present in attempt to improve model performance. The final model they used included all 13 predictor variables which yielded an adjusted R^2 value of 0.728.

The statistical problem at hand here is to use regression to predict the house prices for houses in Boston. More specifically, we will implement Ridge Regression, LASSO and Elastic Net techniques on this dataset to address this regression problem in both R and Python to try and compare their time and memory for a series of benchmark problems. We will benchmark each algorithm in R and Python to compare how fast each technique is and their relative accuracy (using RMSE, R^2 and MAE) to discuss potential trade-offs between the techniques. Note, that because the dataset is relatively small to modern standards, the approaches implemented by our study here will be sufficient for analysis. It is important to note that although a technique may be less computationally expensive, it may not be worth a decline in accuracy (the opposite holds true as well).

We will also be extending this application by trying to implement a warm start for those algorithms discussed above in order to obtain the optimal parameters for each model. As well as implementing hyper-parameter tuning using methods discussed in literature such as (Pocs 2021) mentions. The goal of this tuning is to optimize the results we obtain from model fits.

Methodology

A warm start is essentially a technique that aims to reduce the running time of the iterative methods in its model fitting process (Chu et al. 2015). It uses the solution of a different optimization as a starting point (rather than a cold approach that uses some arbitrary starting point for each iteration). Ideally, one would choose a starting point that is very close to the optimal point, but generally multiple starting points are investigated. A possible method of doing so is by selecting a constant C_min, which balances the training loss and regularization term according to Chu et al. (2015). The objective is to find the C where the CV accuracy is good. This will be the optimal point.

Method 1: Ridge Regression

The first method to be implemented on this data set is called Ridge Regression. Ridge regression is a model fitting method that can be used to analyze data and performs L2 regularization (Ashok 2022). The estimates equation for ridge regression is then: $\beta^{\hat{ridge}} = argmin_\beta\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|^2\}$ (Hastie, Tibshirani, and Friedman 2008). The penalty term aims to prevent multicollinearity and reduce model complexity by coefficient shrinkage. This results in an avoidance of overfitting and all the parameters are kept As a result, we will need to tune this specific parameter to find the optimal value as this will significant impact the results of our model fitting. In order to use a warm start approach, the ridge function has a 'start' argument that aims to do this by using a previously defined ridge regression model. This process will be repeated in Python as well.

Method 2: LASSO

The second method to be implemented on this data set is called LASSO regression. Like ridge regression, LASSO regression is also a regularization technique. The primary point of this technique is that it uses shrinkage which corresponds to shrinking the data values towards a central point as a mean (Kumar 2023). As this article also points out, the LASSO procedure encourages simpler models (i.e. with fewer parameters present). However, as opposed to ridge regression, LASSO uses L1 regularization techniques. The estimates equation for LASSO is then: $\beta^{\hat{lasso}} = argmin_\beta\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\}$ (Hastie, Tibshirani, and Friedman

2008). The term λ here indicates the shrinkage penalty. This will be tuned to attain optimal estimates. Once again, within the 'glmnet' package, the lasso function can be defined to have a 'start' argument of a previously defined model. This will be the starting point for our warm start approach. This process will be repeated in Python as well.

Method 3: Elastic Net

The final method to be implemented on this data set is called Elastic Net. In general terms, Elastic Net regression combines penalties from both LASSO and ridge techniques to try and regularize regression models (Reid 2022). In other words, it tries to learn from both LASSO and ridge regression to try and improve on its own model creation. This technique may seem rather redundant to do given that both LASSO and ridge regression are already being implemented, however we wish to also observe the timing and memory storage differences across multiple programming languages. A method that combines both LASSO and Ridge regression may be pose interesting results for this reason. Moreover, as Reid (2022) explains, elastic net improves on LASSO's limitation of taking few samples for high dimensional data through the inclusion of "n" number of variables until saturation. Once again, a warm start will be implemented to try and find the optimal point. This process will be repeated in Python as well.

Description of Comparison Criterion

As this is a regression problem by nature, standard criterion such as RMSE, R^2 and even MAE will be calculated to compare the three methods. In addition, we will also use timing metrics (such as R's benchmark tool) in order to find the computation time for the algorithms to run. This will also be computed in Python.

Results

To avoid clutter in this report, I have included the coding portion in the appendix of the report.

Exploratory Data Analysis (EDA)

Before we actually fit the model using the methods discussed, let us first explore the data in order to see if any pre-processing or scaling is required. Upon initial inspection, our target variable is

‘MEDV’ and the remaining 13 variables will serve as the predictors in model creation. To observe the first six rows of the data set, refer to the Figure **1** in the appendix section of the report. Overall, the values appear to be normal for the context of their variable (i.e. the nitrogen counts are in part per million rather than their explicit form). There appears to are no missing values. To observe the spread of the variables, we will observe the boxplot.

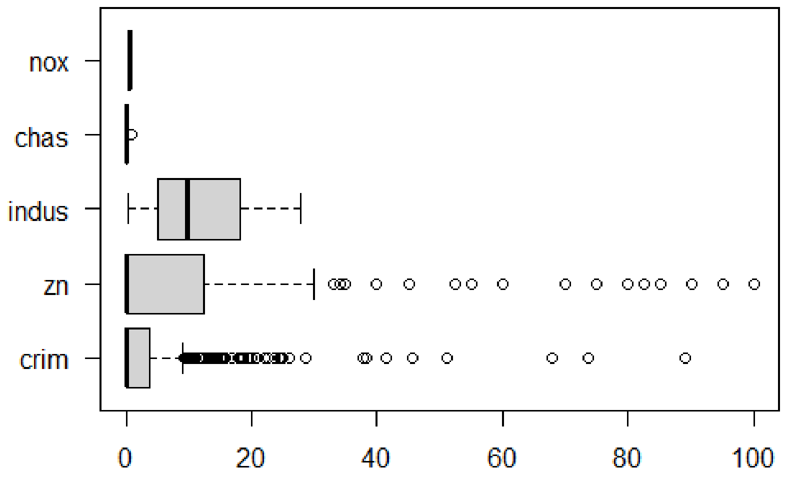

Figure 2: Boxplot of the first five variables present

For the most part, there is a good spread for the variables as a whole as the box manages to capture them adequately. We have only included the first five variables as the boxplot would be too crowded to interpret (in the appendix, simply change the code to observe the other variables spread for the boxplot). Overall, there may be an issue in terms of outliers for the crim and zn predictors, but we will keep them in for now as we are unclear on how they may affect other predictors.

In order to observe the effect of the predictors on each other, let us create a correlation plot to observe potential relationships between the variables. Figure **3** shows a correlation plot between the variables present.

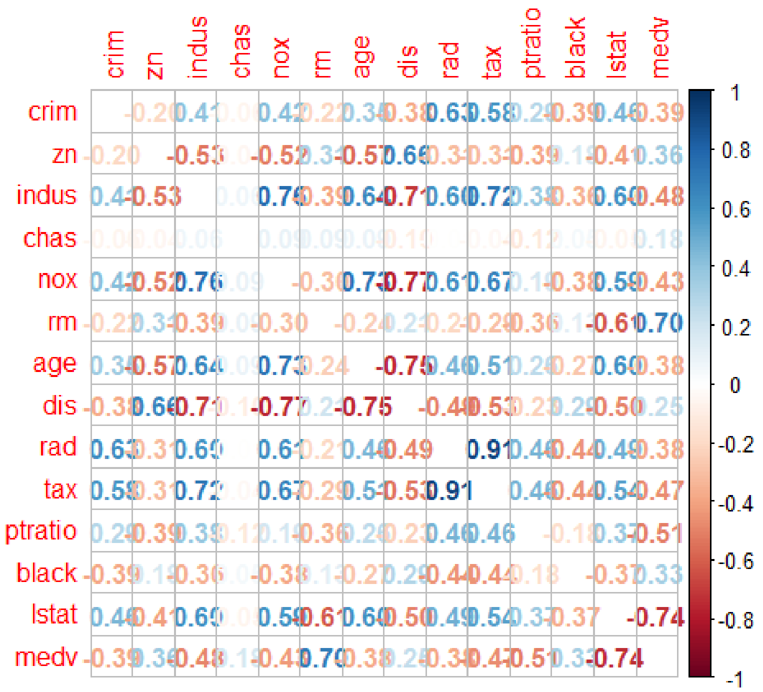

Figure 3: Correlation Matrix of the Variables Present

A darker shade of blue indicates a stronger positive relationship, while a darker red indicates a stronger negative relationship. Some observations of this plot include, the crime variable is strongly correlated with the variables rad and tax. This suggests that if either of these variables increase then so too will the crime variable. Moreover, if we observe the nox variable, we see this is strongly associated with the indus in a positive relationship but a negative relationship with the distance (of employment centres). Remember that the "indus" variable denotes the industrial area so it would make sense to have a higher nitrogen oxide count there. As discussed before, we will keep the variables with potential outliers (that may exist upon inspection of the boxplot in Figure **1**) in the data set as they seem to have a strong correlation with other variables in the data set.

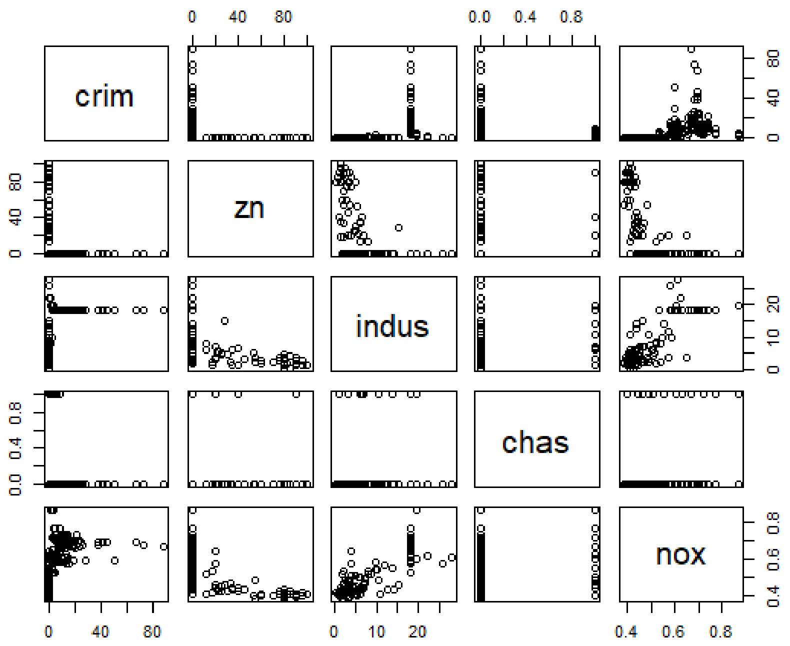

Figure 4: Pairs Plot of the First Five Variables Present

In Figure **4**, we observe a pairsplot of the first five predictor variables involved in the dataset. As we saw in Figure **3**, we see relatively weak patterns except for the relationship between the 'indus' and 'nox' variables which makes sense as we expect high nitrogen counts in industrial areas. As a result, since we see that the variables are not all independent of one another, removing one may impact another. Thus, we will keep all the variables for model fitting.

Now, let us visualize the data with the help of a scatterplot of dependent variables versus the median value of houses (i.e. our "MEDV" attribute). As discussed by Subrahmanya (2018), we can use 'ggplot2' to attain this important visual.

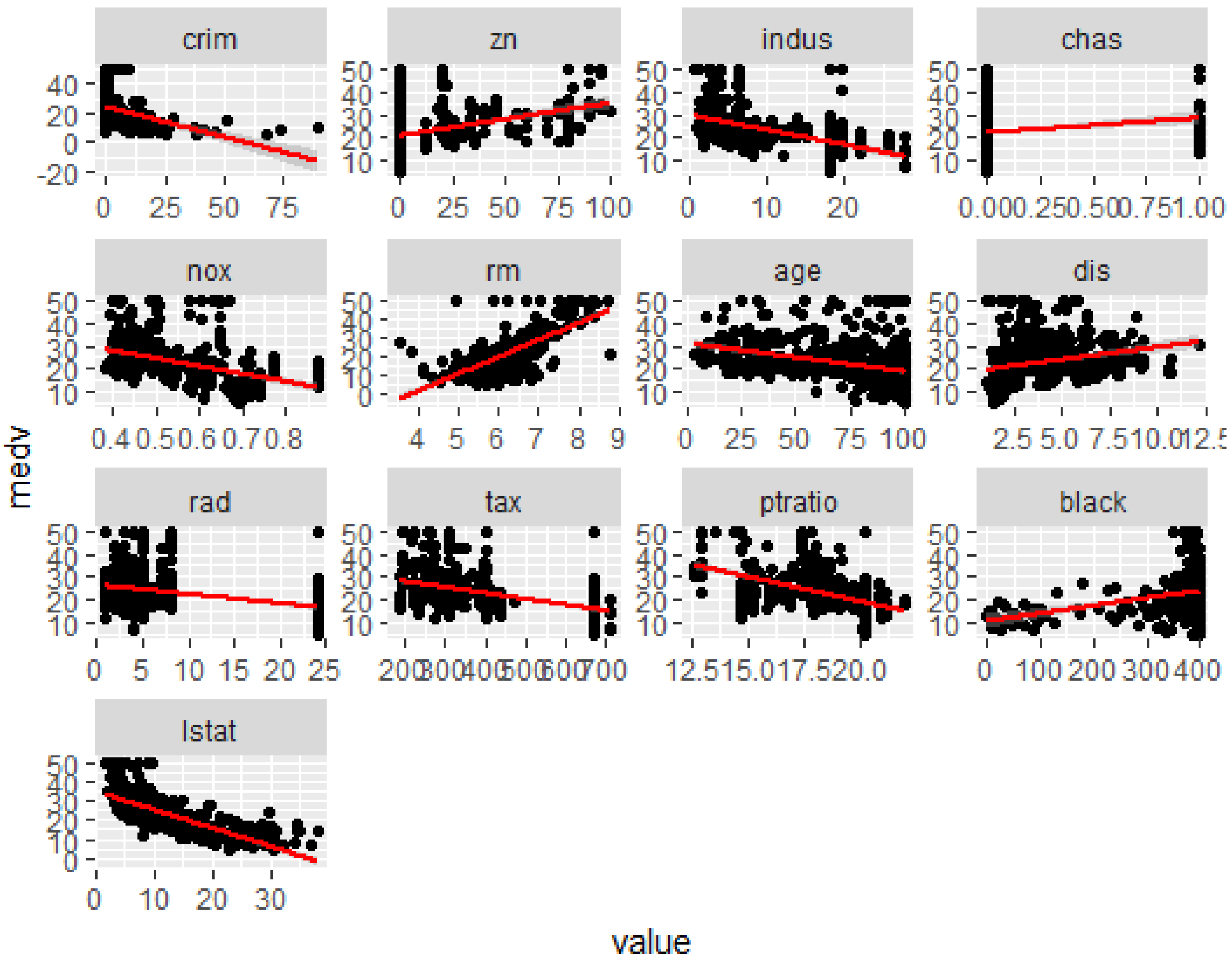

Figure 5: Scatterplot versus the MEDV present

As we can see in Figure **5**, we can observe the relationship between a single predictor variable and the MEDV variable (i.e. the target variable). Some observations include that the per capita crime (crim) has a negative relationship with the MEDV target variable, while the rm variable has a positive relationship with MEDV. These results are also in agreement of our correlation matrix in Figure **3**, but that figure gives the numerical value of the slope where the correlation is scaled.

Results of the Regression Techniques in R and Python

Once again, to observe the coding portion of this project, refer to the appendix section for this. Please note, to ensure that the train/test sets were identical in both R and Python, we created them in R and exported them to allow access for these two datasets in Python. This will erase any potential issues that may arise when fitting the model.

Model Implemented	RMSE	MAE	R^2	Time (seconds)
Ridge (in R)	4.693	3.450	0.730	0.05
Lasso (in R)	4.696	3.449	0.729	0.04
Elastic Net (in R)	4.717	3.474	0.726	0.05
Ridge (in Python)	4.742	3.317	0.737	0.02
Lasso (in Python)	4.976	3.472	0.710	0.035
Elastic Net (in Python)	4.934	3.459	0.715	0.05

Table 1: Results of Model Implementation

Note, in all 3 models we tuned λ but for elastic net we tuned α as well since we need to obtain the optimal parameter. For elastic net, the optimal *alpha* appears to be 0.3. Let us now plot the results with the help of the 'ggplot2' library.

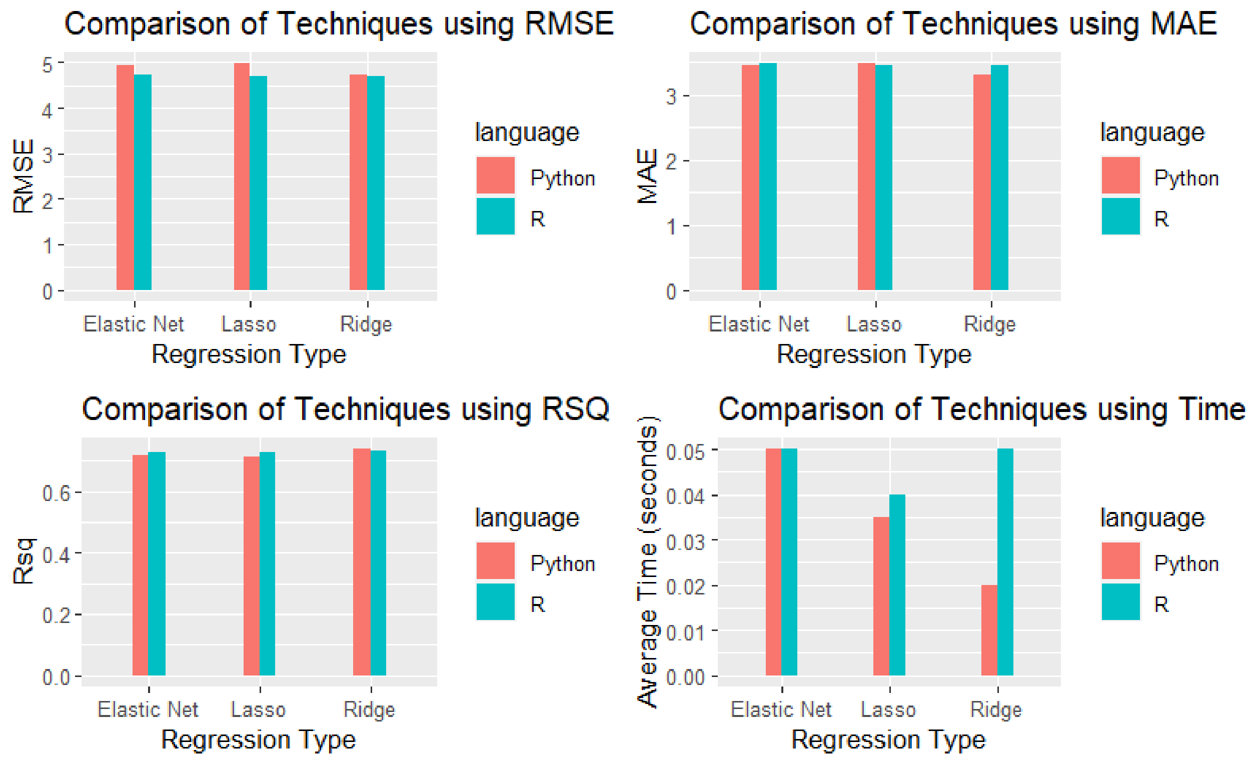

Figure 6: Plot of Results for Python Versus R

Conclusions

In terms of model performance (i.e. observing the metrics in Table **1** and Figure **6**), it appears that all of them do relatively similar. In both R and Python, we observe that the RMSE settles around 4.7 (which corresponds to the standard deviation of the residuals) and because this is good we know that the model is fit adequately. We observe that the MAE settles around 3.5 and again a smaller value here is good. Next, R^2 seems to have an overall score of around 0.7 which explains variation in the data by the model.

As we can again see in Table **1** and Figure **6**, the metrics appear to be relatively close together (with small differences moving from Python to R). In both languages, we observe that the order in terms of best to worst performance is ridge regression, followed by lasso and then Elastic Net. Normally, this is the case when there are a good amount of predictors and they are all relevant (and we know this after our brief EDA). Recall that Elastic Net's design allows it to combine the penalties of lasso and ridge which would be useful had some of the predictors not been significant (Brownlee 2020). This aligns with our results in Table **1**. This may be the result of a different parameter definition, as we tried to keep the optimization process the same across both programming languages. However, observe the difference in the time it takes to compute the process for Elastic Net. There appears to be a difference, but this may due to the fact not enough trials were run to get a better representation. When more trials were run and an average time was take using the R console, we observed that the mean time settled around 0.04-0.05 for all the methods.

Overall, we observe that with the help of a warm-start, that all 3 regression techniques perform relatively similar whether in R and Python as expected. The subtle differences may be due to how the individual techniques are defined in R and Python. For example, if you observe the code, the parameter α is defined as the *lambda* for python when using the Lasso() and Ridge() functions (Brownlee 2020). This can be confusing when building the grid. However, the Elastic Net tuning remains the same.

References

Ashok, Prashanth. 2022. *What Is Ridge Regression?* https://www.mygreatlearning.com/blog/what-is-ridge-regression/.

Brownlee, Jason. 2020. *How to Develop Elastic Net Regression Models in Python.* https://machinelearningmastery.com/elastic-net-regression-in-python/.

Chu, Bo-Yu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 2015. *Warm Start for Parameter Selection of Linear Classifiers.* https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/warm-start/warm-start.pdf.

Harrison Jr., David, and David L. Rubinfeld. 1978. *Hedonic Housing Prices and the Demand for Clean Air.* https://www.law.berkeley.edu/files/Hedonic.PDF.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2008. *The Elements of Statistical Learning.* Springer.

Kumar, Dinesh. 2023. *A Complete Understanding of LASSO Regression.* https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/.

Pocs, Mate. 2021. *Hyperparameter Tuning in Lasso and Ridge Regressions.* https://towardsdatascience.com/hyperparameter-tuning-in-lasso-and-ridge-regressions-70a4b158ae6d.

Reid, Walter. 2022. *Elastic Net- a Regression Method That Performs Variable Selection and Regularization Simultaneously.* https://corporatefinanceinstitute.com/resources/data-science/elastic-net/.

Subrahmanya, Rashmi. 2018. *Analysis of Boston Housing Data Using Linear Regression, Trees and GAM.* https://rpubs.com/Rashmi_Subrahmanya/371719.

Appendix

```
library(MASS)
library(corrplot)
library(dplyr)
library(ggplot2)
library(glmnet)
library(caret)
library(ggplot2)
data("Boston")
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```
set.seed(400076920)

#Split into 75-25 accordingly
train_ind <- createDataPartition(Boston$medv, p = 0.75, list = FALSE)
train <- Boston[train_ind,]
```

```
test <- Boston[-train_ind,]


#export the train and test data to a file that will be read into python!
write.csv(train, "C:/Users/User/Documents/SidhuAmandeep_Final_Project_790/train_data.csv",
          row.names = FALSE)
write.csv(test, "C:/Users/User/Documents/SidhuAmandeep_Final_Project_790/test_data.csv",
          row.names = FALSE)
```

Figure 1: First six rows of the Boston Housing Data Set

```
#code for the correlation matrix
corrplot(cor(Boston), method = "number", diag = FALSE)
```

```
#code for creating the boxplot of the first five variables
boxplot(Boston[c(1,2,3,4,5)], horizontal= TRUE, las= 1)
```

```
pairs(Boston[c(1,2,3,4,5)])
```

```
#code for the scatterplots vs dependent variable


#reshape the data
data <- reshape2::melt(Boston, id.vars = "medv")


#create the plot
ggplot(data, aes(x = value, y = medv)) +
  geom_point() +
  stat_smooth(method = "lm", se = TRUE, col = "red") +
  facet_wrap(~ variable, scales = "free") +
  theme_gray()
```

```
#code for implementation of ridge regression


#define predictor and response/target variable for the cv.glment
```

```
 x<- model.matrix(medv ~. , data = train)[,-1]
 y <- train$medv

 #create the ridge regression with a warm start/ and find timing
 time_ridge <-system.time({
   ridge_model <- cv.glmnet(x,y, alpha=0, nfolds = 10,
                            lambda=seq(0.001, 0.1, length=100), standardize=TRUE,
                            keep=TRUE, type.measure="mse", warm.start=TRUE)

 })

 #default
 ridge_model <- cv.glmnet(x,y, alpha=0, nfolds = 10,
                            lambda=seq(0.001, 0.1, length=100), standardize=TRUE,
                            keep=TRUE, type.measure="mse", warm.start=TRUE)

 #now, use the model to make predictions
x_test <- model.matrix(medv~., data = test)[,-1]
y_test <- test$medv
y_pred <- predict(ridge_model, newx = x_test, s = ridge_model$lambda.min)

#calculate comparison criterion


rmse_ridge <- sqrt(mean((y_pred-y_test)^2))
mae_ridge <- mean(abs(y_pred - y_test))
rsq_ridge <- cor(y_pred, y_test)^2
```

```
#code implementation for lasso regression

#define predictor and response/target variable for the cv.glment
 x<- model.matrix(medv ~. , data = train)[,-1]
 y <- train$medv
```

```r
#find optimal lambda
lambdas <- seq(0.001, 0.1, length=100)

#create the lasso regression with a warm start/ and find timing
time_lasso <-system.time({
 lasso_model <- cv.glmnet(x,y, alpha=1, nfolds = 10,
                        lambda=lambdas, standardize=TRUE,
                        keep=TRUE, type.measure="mse", warm.start=TRUE)

})

 #now, use the model to make predictions
x_test <- model.matrix(medv~., data = test)[,-1]
y_test <- test$medv
y_pred <- predict(lasso_model, newx = x_test, s = lasso_model$lambda.min)

#calculate comparison criterion

rmse_lasso <- sqrt(mean((y_pred-y_test)^2))
mae_lasso <- mean(abs(y_pred - y_test))
rsq_lasso <- cor(y_pred, y_test)^2
```

```r
#code implementation for elastic net

#create the en regression with a warm start/ and find timing
#the alpha = 0.5 is the split between L1 and L2 regularization of the model
#remember that EN is the combination of the two!
lambdas <- seq(0, 1, by = 0.1)
alphas <- seq(0, 1, by = 0.1)

#find best model parameters
```

```r
elastic_net_model<- lapply(alphas, function(a) {
  cv.glmnet(x, y, alpha = a, lambda = lambdas,
            type.measure="mse", warm.start=TRUE)
})

#for (i in 1:11) {print(min(elastic_net_model[[i]]$cvm))} to print errors
#lowest error appears to be alpha = 0.3 - use this for final model

 time_en <-system.time({
   en_model <- cv.glmnet(x,y, alpha=0.3, nfolds = 10,
                          lambda=lambdas,type.measure="mse", warm.start=TRUE)

 })

 #now, use the model to make predictions
x_test <- model.matrix(medv~., data = test)[,-1]
y_test <- test$medv
y_pred <- predict(en_model, newx = x_test, s = en_model$lambda.min)

#calculate comparison criterion

rmse_en <- sqrt(mean((y_pred-y_test)^2))
mae_en <- mean(abs(y_pred - y_test))
rsq_en <- cor(y_pred, y_test)^2
```

```r
library(gridExtra)
#code for the plots created~
data_rmse <- data.frame(
  name= c("Ridge", "Lasso", "Elastic Net"),
  language = c("R","R","R", "Python", "Python", "Python"),
  values = c(4.693, 4.696, 4.717, 4.742, 4.976, 4.934)
)
```

```r
data_mae <- data.frame(
  name= c("Ridge", "Lasso", "Elastic Net"),
  language = c("R","R","R", "Python", "Python", "Python"),
  values = c(3.450, 3.449, 3.474, 3.317, 3.472, 3.459)
)

data_rsq <- data.frame(
  name= c("Ridge", "Lasso", "Elastic Net"),
  language = c("R","R","R", "Python", "Python", "Python"),
  values = c(0.730, 0.729, 0.726, 0.737, 0.710, 0.715)
)

data_time <- data.frame(
  name= c("Ridge", "Lasso", "Elastic Net"),
  language = c("R","R","R", "Python", "Python", "Python"),
  values = c(0.05, 0.04, 0.05, 0.02, 0.035, 0.05)
)

p1 <- ggplot(data_rmse, aes(x=name, y = values, fill = language)) +
  geom_bar(stat="identity", width = 0.3, position = position_dodge()) +
  labs(title = "Comparison of Techniques using RMSE", x = "Regression Type",
       y = "RMSE")

p2 <- ggplot(data_mae, aes(x=name, y = values, fill = language)) +
  geom_bar(stat="identity", width = 0.3, position = position_dodge()) +
  labs(title = "Comparison of Techniques using MAE",x = "Regression Type",
       y = "MAE")

p3 <- ggplot(data_rsq, aes(x=name, y = values, fill = language)) +
  geom_bar(stat="identity", width = 0.3, position = position_dodge()) +
  labs(title = "Comparison of Techniques using RSQ", x = "Regression Type",
      y = "Rsq")

p4 <- ggplot(data_time, aes(x=name, y = values, fill = language)) +
  geom_bar(stat="identity", width = 0.3, position = position_dodge()) +
  labs(title = "Comparison of Techniques using Time", x = "Regression Type",
      y = "Average Time (seconds)")

p <- grid.arrange(p1, p2, p3, p4, ncol= 2)
```

```
# Note, I implemented this code in spyder but did not run it here

import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge, Lasso, ElasticNet, ElasticNetCV
from sklearn.datasets import fetch_openml
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV, RepeatedKFold
import time


#read in the previously created data tables (ensure the train and test data
#same in R and python!)

train = pd.read_csv("C:/Users/User/Documents/SidhuAmandeep_Final_Project_790/train_data.csv")
test = pd.read_csv("C:/Users/User/Documents/SidhuAmandeep_Final_Project_790/test_data.csv")


param_grid = {'alpha' : np.arange(0,1,0.01)}
X_train = train.iloc[:, 0:13]
y_train = train["medv"]
X_test = train.iloc[:, 0:13]
y_test = train["medv"]
```

```
#fit the data initially for a starting point
#find optimal lambda

ridge = Ridge()

grid_search = GridSearchCV(estimator=ridge, param_grid = param_grid, cv = 10,
                           scoring = "neg_mean_squared_error")
grid_search.fit(X_train, y_train)

lambda_opt = grid_search.best_params_['alpha']

ridge_warm_started = Ridge(alpha = lambda_opt)

#time the fitting for the warm start process/
start_time = time.time()
ridge_warm_started.fit(X_train, y_train)
end_time = time.time()



#Compute the metrics
y_pred = ridge_warm_started.predict(X_test)
rmse_ridge = mean_squared_error(y_test, y_pred, squared = False)
mae_ridge = mean_absolute_error(y_test, y_pred)
rsq_ridge = r2_score(y_test, y_pred)
time_ridge = end_time - start_time

#Repeat the process for Lasso

lasso = Lasso()
```

```
grid_search = GridSearchCV(lasso, param_grid = param_grid, cv = 10,
                           scoring = "neg_mean_squared_error", n_jobs = -1)
grid_search.fit(X_train, y_train)

lambda_opt = grid_search.best_params_['alpha']

#ideal parameter is 0.014 like R

start_time = time.time()

lasso = Lasso(alpha = lambda_opt, warm_start=True)
lasso.fit(X_train, y_train)

end_time = time.time()

#Compute the metrics
y_pred = lasso.predict(X_test)
rmse_lasso = mean_squared_error(y_test, y_pred, squared = False)
mae_lasso = mean_absolute_error(y_test, y_pred)
rsq_lasso = r2_score(y_test, y_pred)
time_lasso = end_time - start_time

#Repeat it again for Elastic Net

en = ElasticNet()

grid = dict()
grid['alpha'] = np.arange(0,1.1, 0.1)
grid['l1_ratio'] = np.arange(0,1, 0.01)

#tuning alpha we  observe that the optimal value is 0.3 again
```

```
#optimize alpha and lambda here
grid_search = GridSearchCV(en, grid, scoring = "neg_mean_squared_error",
cv = 10, n_jobs = -1)
results = grid_search.fit(X_train, y_train)

#results.best_params_
#appears best values are alpha = 0.3, lambda = 0

start_time = time.time()
en = ElasticNet(l1_ratio=0, alpha = 0.3)
en.fit(X_train, y_train)

end_time = time.time()

#Compute the metrics
y_pred = en.predict(X_test)
rmse_en = mean_squared_error(y_test, y_pred, squared = False)
mae_en = mean_absolute_error(y_test, y_pred)
rsq_en = r2_score(y_test, y_pred)
time_en = end_time - start_time
```