elastic net

2024-11-20

"final_full_rmna_OneHotEncoded_housing.csv" (no outliers removed)

"final_rmna_OneHotEncoded_housing.csv" (price outliers removed)

```
library(readr)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
setwd("~/Desktop/625 final project")
housing_data <- read_csv("final_full_rmna_OneHotEncoded_housing.csv")
```

```
## Rows: 60994 Columns: 44
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (44): Transaction_Other, Transaction_Resale, Transaction_New Property, T...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
set.seed(34)

#seed options: 55, 34
```

```
attach(housing_data)

#transform price into log_price
housing_data$log_price=log(housing_data$`Price (in rupees)`)
X <- housing_data[, -which(names(housing_data) == "Price (in rupees)" | names(housing_data) == "log_pri
Y <- housing_data$log_price
X_matrix <- as.matrix(X)
Y_matrix <- as.matrix(Y)
```

```r
# Initialize storage
alphas <- seq(0, 1, by = 0.01)
elastic_net <- list()  # A list to store models
best_lambda <- numeric(length(alphas))  # Numeric vector for best lambda values
predicted_prices <- list()  # A list to store predictions
results <- list()  # A list to store data frames with actual and predicted values
rmse <- numeric(length(alphas))   # Numeric vector for RMSE values
mae <- numeric(length(alphas))   # Numeric vector for MAE values
mape=numeric(length(alphas))  # Numeric vector for MAPE values

for (i in seq_along(alphas)) {
  alpha=alphas[i]
  # Fit elastic net model with alpha = i/20
  elastic_net[[i]] <- cv.glmnet(X_matrix, Y_matrix, alpha = alpha)

  # Store the best lambda value (we choose lambda where MSE is smallest) #lambda.min
  best_lambda[i] <- elastic_net[[i]]$lambda.1se

  # Predict on the test set using the best lambda
  predicted_prices[[i]] <- predict(elastic_net[[i]], newx = X_matrix, s = "lambda.1se")

  # Combine actual and predicted values
  results[[i]] <- data.frame(Actual = Y_matrix, Predicted = as.vector(predicted_prices[[i]]))
  # Compute RMSE, MAE, MAPE
  rmse[i] <- sqrt(mean((results[[i]]$Actual - results[[i]]$Predicted)^2))
  mae[i] <- mean(abs(results[[i]]$Actual - results[[i]]$Predicted))
  mape[i] <- mean(abs((results[[i]]$Actual - results[[i]]$Predicted) / results[[i]]$Actual)) * 100
}
```

```r
# Results for RMSE, MAE and MAPE
print(rmse)
```

```
##   [1] 0.3855821 0.3888134 0.3854056 0.3856297 0.3855306 0.3852394 0.3886506
##   [8] 0.3852936 0.3854536 0.3854855 0.3857370 0.3867147 0.3857147 0.3856222
##  [15] 0.3859868 0.3864778 0.3867286 0.3855543 0.3890328 0.3861460 0.3860869
##  [22] 0.3870655 0.3862760 0.3865587 0.3872847 0.3861427 0.3895947 0.3858166
##  [29] 0.3860433 0.3863103 0.3862801 0.3900663 0.3859447 0.3865318 0.3865059
##  [36] 0.3861595 0.3864577 0.3858569 0.3864166 0.3863979 0.3863803 0.3860596
##  [43] 0.3881044 0.3870724 0.3860216 0.3893912 0.3900949 0.3862796 0.3869934
##  [50] 0.3866049 0.3862468 0.3859528 0.3862280 0.3865580 0.3869211 0.3869109
##  [57] 0.3865293 0.3865206 0.3884375 0.3865041 0.3864964 0.3864889 0.3868500
##  [64] 0.3868426 0.3868355 0.3896223 0.3864541 0.3868157 0.3864424 0.3864368
##  [71] 0.3867978 0.3867922 0.3861035 0.3871941 0.3864113 0.3864066 0.3871771
##  [78] 0.3867624 0.3871666 0.3882120 0.3863854 0.3871521 0.3876344 0.3867377
##  [85] 0.3876242 0.3871345 0.3909929 0.3900970 0.3876053 0.3867167 0.3871142
##  [92] 0.3875919 0.3875877 0.3871036 0.3871003 0.3900210 0.3863342 0.3863316
##  [99] 0.3875653 0.3899880 0.3875580
```

```r
which.min(abs(rmse))
```

```
## [1] 6
```

```r
print(mae)
```

```
##   [1] 0.2833366 0.2876325 0.2832731 0.2836855 0.2836181 0.2831885 0.2884449
##   [8] 0.2834210 0.2837441 0.2838287 0.2842460 0.2858316 0.2843000 0.2841965
##  [15] 0.2847922 0.2856338 0.2860648 0.2842069 0.2898366 0.2852135 0.2851336
##  [22] 0.2867887 0.2855057 0.2859912 0.2872173 0.2853217 0.2907912 0.2848085
##  [29] 0.2851840 0.2856661 0.2856240 0.2915335 0.2850601 0.2860763 0.2860414
##  [36] 0.2854547 0.2859739 0.2849574 0.2859160 0.2858893 0.2858639 0.2853126
##  [43] 0.2886414 0.2869944 0.2852562 0.2905738 0.2916623 0.2857240 0.2868923
##  [50] 0.2862769 0.2856767 0.2851714 0.2856472 0.2862045 0.2867950 0.2867809
##  [57] 0.2861623 0.2861502 0.2891273 0.2861273 0.2861165 0.2861060 0.2866953
##  [64] 0.2866846 0.2866742 0.2909468 0.2860551 0.2866451 0.2860380 0.2860298
##  [71] 0.2866186 0.2866102 0.2854592 0.2872361 0.2859917 0.2859846 0.2872119
##  [78] 0.2865649 0.2871969 0.2887208 0.2859520 0.2871758 0.2878824 0.2865271
##  [85] 0.2878680 0.2871502 0.2928018 0.2915646 0.2878396 0.2864962 0.2871178
##  [92] 0.2878148 0.2878069 0.2871019 0.2870969 0.2914284 0.2858706 0.2858664
##  [99] 0.2877643 0.2913683 0.2877503
```

```r
which.min(abs(mae))
```

```
## [1] 6
```

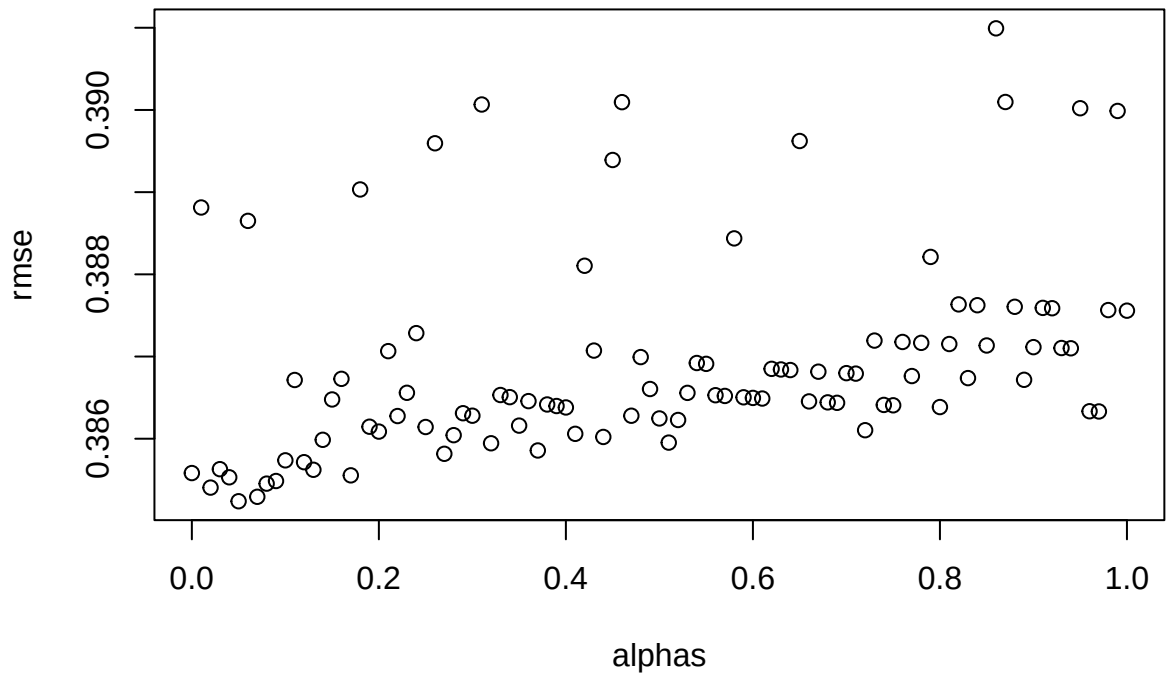
```r
print(mape)
```

```
##   [1] 3.209803 3.257968 3.209123 3.213756 3.213000 3.208178 3.267229 3.210788
##   [9] 3.214401 3.215347 3.220016 3.237867 3.220692 3.219570 3.226301 3.235789
##  [17] 3.240661 3.219811 3.283270 3.231166 3.230289 3.248960 3.234518 3.240006
##  [25] 3.253879 3.232500 3.294342 3.226757 3.230993 3.236444 3.235983 3.302865
##  [33] 3.229650 3.241121 3.240739 3.234134 3.240002 3.228549 3.239370 3.239078
##  [41] 3.238801 3.232586 3.270349 3.251658 3.231972 3.292220 3.304281 3.237282
##  [49] 3.250544 3.243546 3.236768 3.231064 3.236445 3.242755 3.249482 3.249329
##  [57] 3.242297 3.242165 3.276064 3.241918 3.241801 3.241687 3.248396 3.248280
##  [65] 3.248167 3.296291 3.241137 3.247851 3.240951 3.240864 3.247562 3.247472
##  [73] 3.234415 3.254627 3.240451 3.240375 3.254364 3.246979 3.254201 3.271556
##  [81] 3.240022 3.253972 3.262052 3.246570 3.261895 3.253695 3.316739 3.303044
##  [89] 3.261586 3.246235 3.253342 3.261312 3.261224 3.253170 3.253115 3.301515
##  [97] 3.239143 3.239098 3.260757 3.300841 3.260602
```

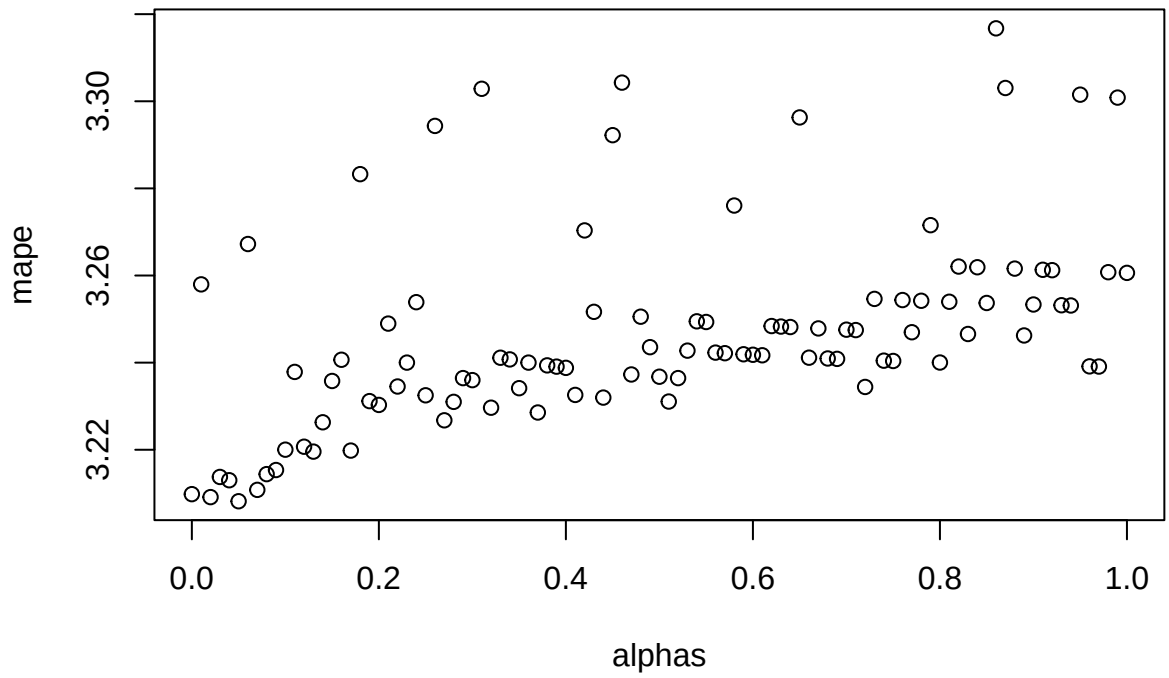
```r
which.min(abs(mape))
```

```
## [1] 6
```

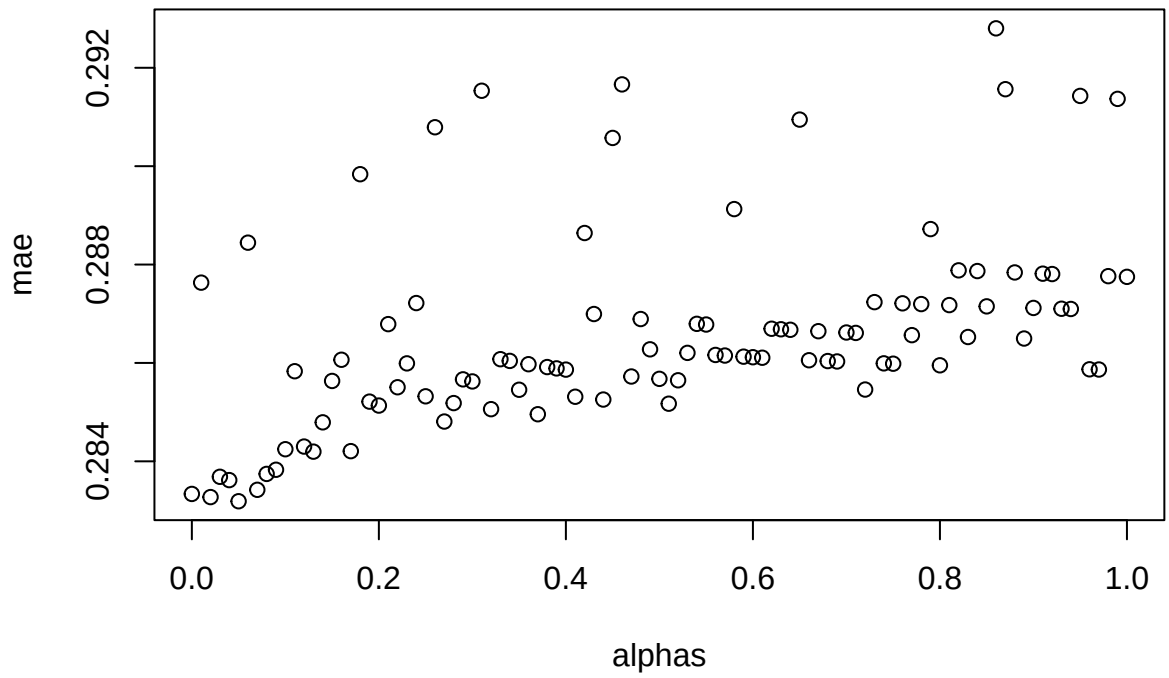
```r
plot(alphas,rmse)
```




```
plot(alphas,mape)
```

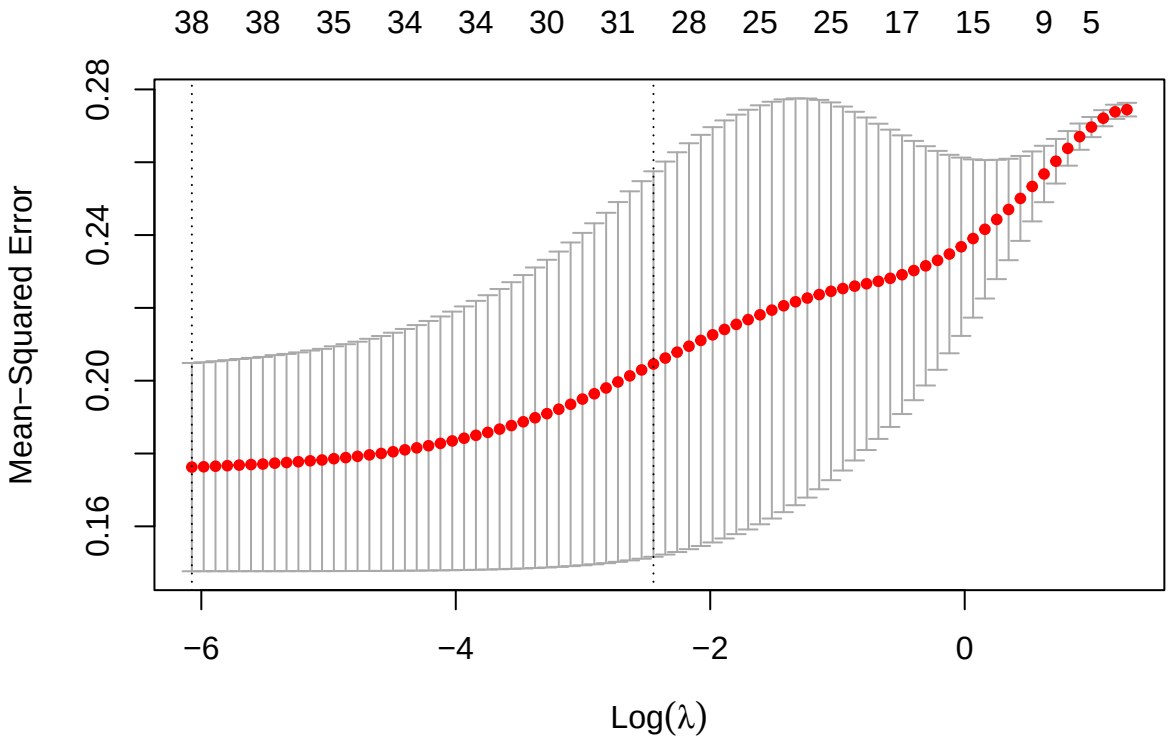



```
plot(alphas, mae)
```



Is alpha = 0 a weird value?

```
fit= cv.glmnet(X_matrix, Y_matrix, alpha = 0.05)
plot(fit)
```




```
# two methods to choose lambda
fit$lambda.1se
```

```
## [1] 0.08663284
```

```
fit$lambda.min
```

```
## [1] 0.002301044
```

s comes from lambda.1se

```
# Compute RMSE, MAE, MAPE and R_squared in training dataset
predicted_prices<- predict(fit, newx = X_matrix,s=0.03750129)

  # Combine actual and predicted values
results<- data.frame(Actual = Y_matrix, Predicted = as.vector(predicted_prices))

  rmse<- sqrt(mean((results$Actual - results$Predicted)^2))
  mae<- mean(abs(results$Actual - results$Predicted))
  mape<- mean(abs((results$Actual - results$Predicted) / results$Actual)) * 100
  rmse
```

```
## [1] 0.3857504
```

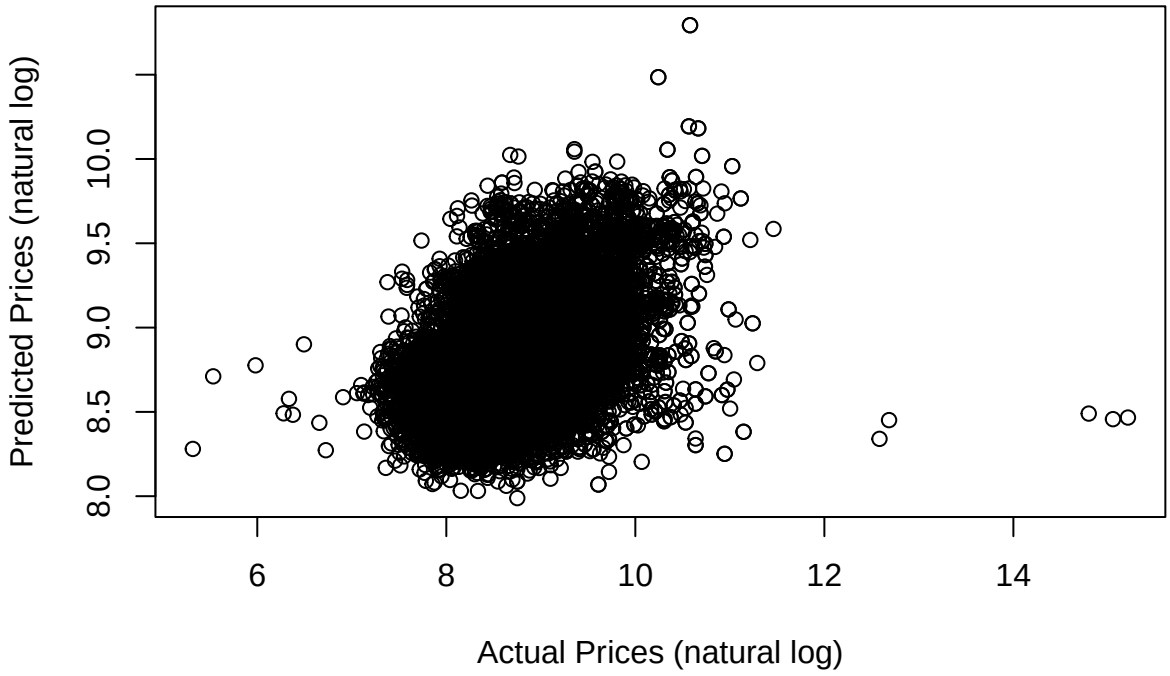
```r
  mae
```

```
## [1] 0.2840488
```

```r
  mape
```

```
## [1] 3.217813
```

```r
plot(Y_matrix, predicted_prices, xlab = "Actual Prices (natural log)", ylab = "Predicted Prices (natural
```




```r
y_mean <- mean(Y_matrix)
SS_total <- sum((Y_matrix - y_mean)^2)
SS_residual <- sum((Y_matrix - predicted_prices)^2)
R_squared <- 1 - (SS_residual / SS_total)
print(R_squared)
```

```
## [1] 0.4574067
```