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_rmna_OneHotEncoded_housing.csv")
```

```
## Rows: 60988 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)

#other seed option: 55
```

```
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] 0.3819910 0.3866847 0.3818091 0.3818539 0.3817692 0.3816391 0.3818805
##   [8] 0.3822441 0.3820391 0.3818881 0.3821449 0.3822375 0.3821215 0.3820269
##  [15] 0.3817746 0.3825800 0.3820126 0.3824151 0.3819084 0.3822907 0.3825017
##  [22] 0.3821946 0.3830368 0.3826419 0.3825969 0.3822848 0.3820288 0.3824855
##  [29] 0.3827575 0.3824257 0.3821518 0.3826651 0.3823532 0.3818950 0.3823139
##  [36] 0.3832637 0.3825507 0.3832153 0.3820236 0.3822351 0.3824832 0.3822097
##  [43] 0.3867209 0.3859170 0.3827300 0.3834580 0.3824072 0.3826913 0.3821408
##  [50] 0.3833983 0.3830063 0.3838197 0.3872135 0.3826293 0.3826205 0.3829507
##  [57] 0.3823217 0.3825963 0.3823087 0.3832901 0.3822967 0.3829000 0.3836954
##  [64] 0.3825559 0.3820472 0.3828722 0.3832370 0.3825336 0.3828524 0.3828466
##  [71] 0.3825189 0.3828357 0.3825098 0.3825055 0.3828206 0.3822299 0.3828113
##  [78] 0.3828068 0.3831716 0.3831672 0.3835762 0.3827903 0.3831546 0.3824689
##  [85] 0.3831469 0.3831432 0.3831396 0.3827688 0.3831326 0.3831293 0.3827592
##  [92] 0.3827562 0.3827533 0.3827504 0.3831140 0.3827448 0.3835131 0.3831057
##  [99] 0.3882215 0.3827345 0.3834994
```

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

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

```r
print(mae)
```

```
##   [1] 0.2828712 0.2888038 0.2828207 0.2829435 0.2828831 0.2827342 0.2832184
##   [8] 0.2837998 0.2835505 0.2833564 0.2837756 0.2839534 0.2838291 0.2837251
##  [15] 0.2833461 0.2846642 0.2837987 0.2844499 0.2836770 0.2843019 0.2846814
##  [22] 0.2841976 0.2856044 0.2849820 0.2849213 0.2844010 0.2840030 0.2847697
##  [29] 0.2852505 0.2846912 0.2842471 0.2851245 0.2845899 0.2838339 0.2845372
##  [36] 0.2861278 0.2849691 0.2860671 0.2840887 0.2844435 0.2848753 0.2844124
##  [43] 0.2914316 0.2902272 0.2853119 0.2864807 0.2847648 0.2852603 0.2843273
##  [50] 0.2864039 0.2857894 0.2870322 0.2921622 0.2851725 0.2851598 0.2857050
##  [57] 0.2846463 0.2851246 0.2846300 0.2862581 0.2846148 0.2856344 0.2868744
##  [64] 0.2850643 0.2842034 0.2855952 0.2861826 0.2850300 0.2855658 0.2855575
##  [71] 0.2850069 0.2855415 0.2849926 0.2849858 0.2855193 0.2845277 0.2855055
##  [78] 0.2854988 0.2860853 0.2860786 0.2867109 0.2854739 0.2860599 0.2849265
##  [85] 0.2860488 0.2860434 0.2860382 0.2854410 0.2860282 0.2860233 0.2854262
##  [92] 0.2854215 0.2854169 0.2854124 0.2860009 0.2854037 0.2866212 0.2859885
##  [99] 0.2933541 0.2853873 0.2865958
```

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

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

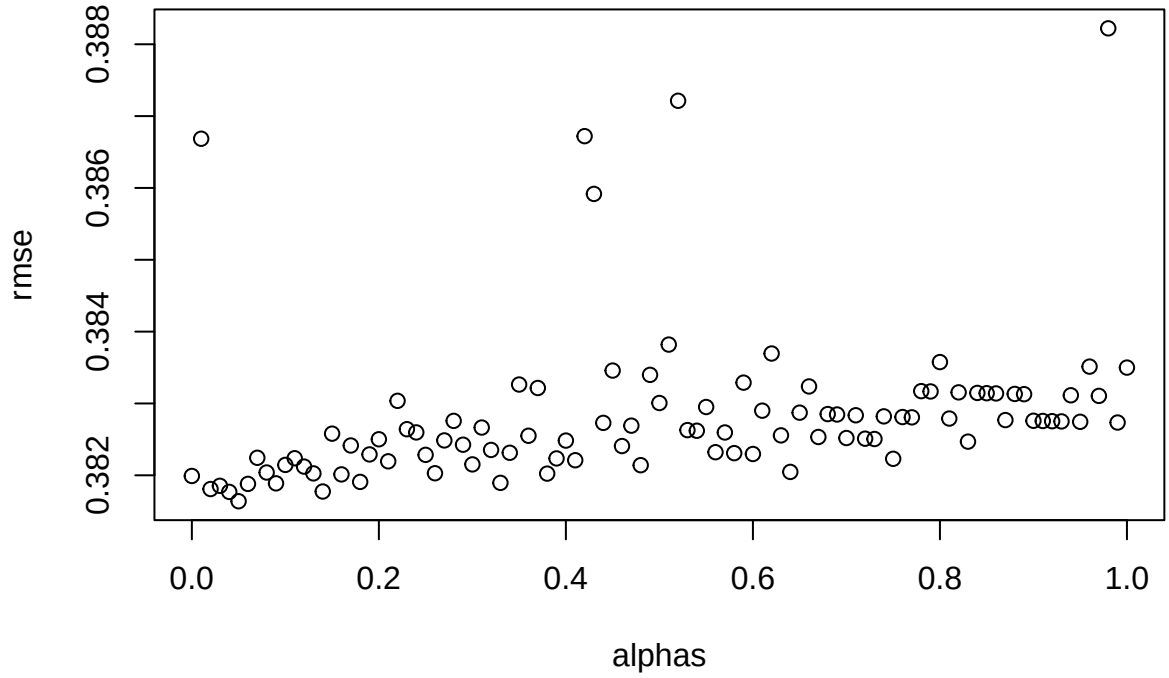
```r
print(mape)
```

```
##   [1] 3.206367 3.272894 3.205828 3.207201 3.206527 3.204860 3.210275 3.216778
##   [9] 3.213988 3.211816 3.216505 3.218527 3.217177 3.216047 3.211827 3.226684
##  [17] 3.216971 3.224335 3.215656 3.222722 3.227016 3.221597 3.237447 3.230459
##  [25] 3.229794 3.223954 3.219499 3.228135 3.233569 3.227280 3.222295 3.232190
##  [33] 3.226176 3.217684 3.225602 3.243581 3.230495 3.242916 3.220595 3.224596
##  [41] 3.229474 3.224263 3.303458 3.290109 3.234436 3.247709 3.228273 3.233880
##  [49] 3.223355 3.246869 3.239883 3.254045 3.311519 3.232927 3.232790 3.238961
##  [57] 3.226995 3.232409 3.226821 3.245281 3.226659 3.238197 3.252329 3.231756
##  [65] 3.222033 3.237774 3.244460 3.231385 3.237454 3.237364 3.231135 3.237192
##  [73] 3.230980 3.230906 3.236953 3.225732 3.236803 3.236731 3.243403 3.243329
##  [81] 3.250555 3.236462 3.243127 3.230266 3.243007 3.242949 3.242893 3.236107
##  [89] 3.242784 3.242732 3.235947 3.235897 3.235847 3.235799 3.242490 3.235705
##  [97] 3.249588 3.242357 3.324532 3.235529 3.249306
```

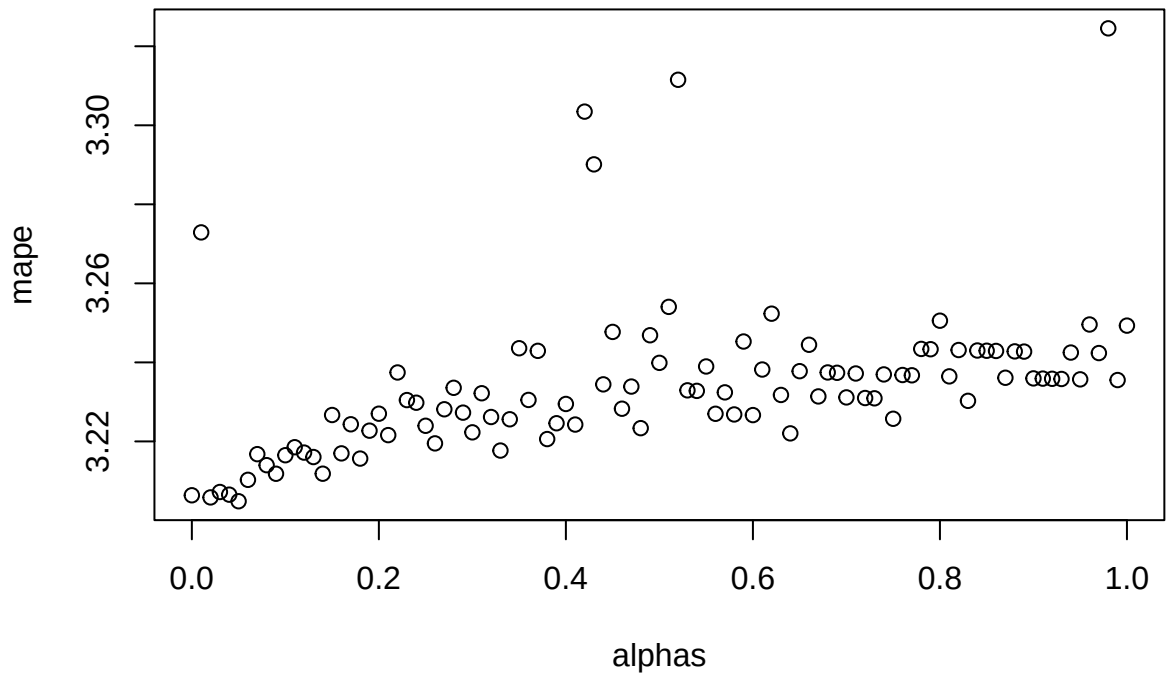
```r
which.min(abs(mape))
```

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

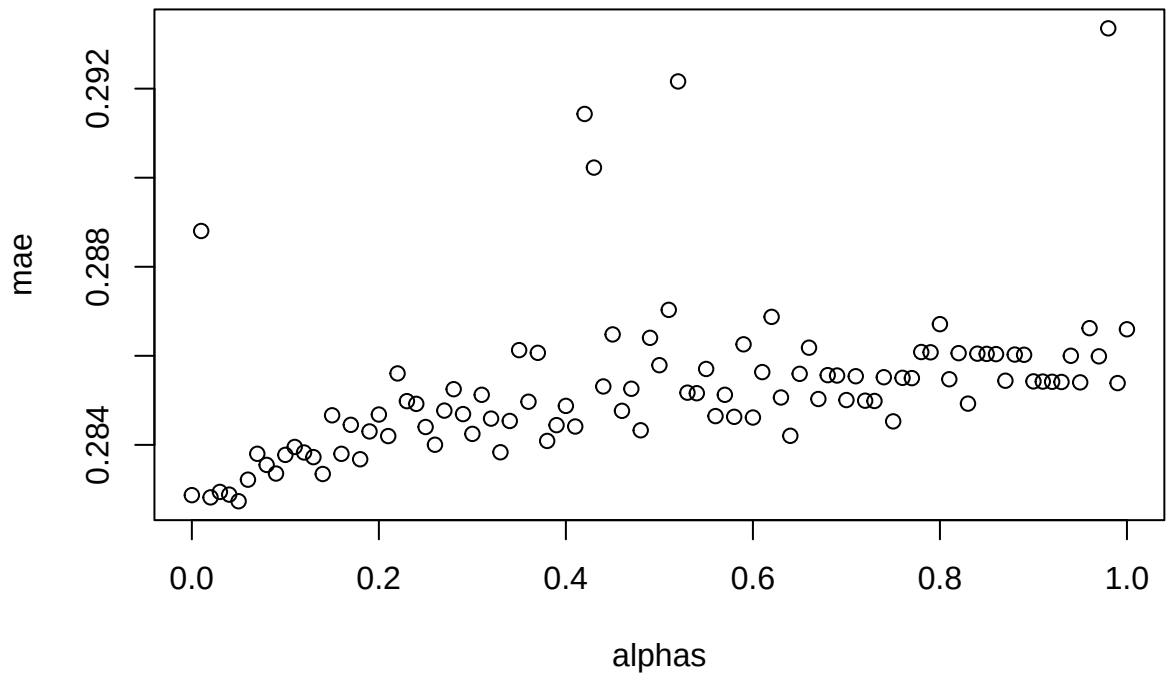
```r
plot(alphas,rmse)
```




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

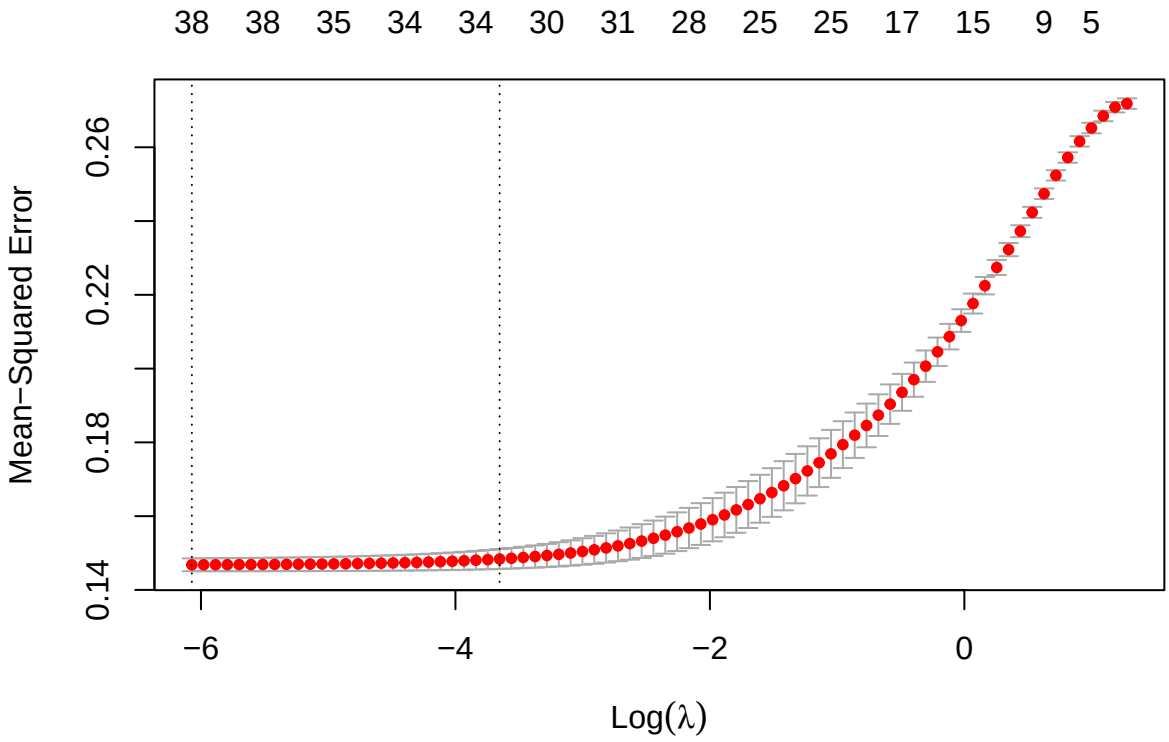



```
plot(alphas, mae)
```



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] 0.02591575
```

```
fit$lambda.min
```

```
## [1] 0.002307059
```

s comes from lambda.1se

```
# Compute RMSE, MAE, MAPE and R_squared in training dataset
predicted_prices<- predict(fit, newx = X_matrix,s=0.08663284)

  # 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] 0.3860388
```

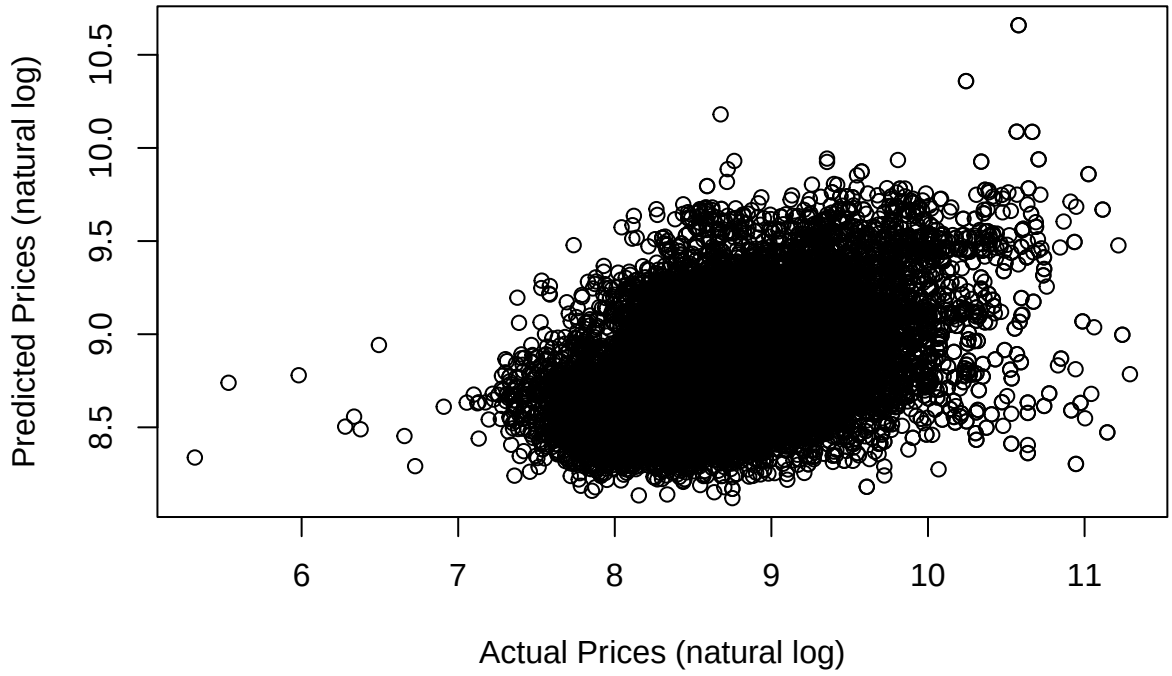
```
  mae
```

```
## [1] 0.2890695
```

```
  mape
```

```
## [1] 3.276075
```

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




```
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] 0.4517984
```