Model Results

Evan Schwartz, Ethan Wood, Max Van Fleet

2024-11-14

Loading data and Libraries

```
library(readr)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(leaps)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(ggplot2)
library(lattice)

data = read_csv("/Users/evanschwartz/664Project/Data/final_data_tranform_clean.csv", show_col_types = F
```

```
## New names:
## * `` -> `...1`
```

```
#Last Minute data cleaning, some economic vairables removed due to high collinearity
data = dplyr::select(data, -c(1,4,6, 8,9, 10, 12, 13))
```

K-folds for data=Here we prepare the training data by shuffling and preforming a 90%-10% training-test split.

```
# 90/10 data
set.seed(664)

shuffleddata = data[sample(nrow(data)), ]
data_naive_shuff <- dplyr::select(shuffleddata, c(1, 2, 3, 6, 7,10, 28:59, 70:77,))


# Calculate the index for the 90-10 split
index90 = floor(0.9 * nrow(shuffleddata))

train = shuffleddata[1:index90,]
train_naive = data_naive_shuff[1:index90,]
# Create the 10% testing data
test = shuffleddata[(index90 + 1):nrow(shuffleddata),]
test_naive = data_naive_shuff[(index90 + 1):nrow(data_naive_shuff),]

#of the 90 data, 7 fold cross validation

folds = createFolds(train$loglatestPrice, k = 7, list = TRUE)
```

AIC and BIC

```
MSE.bic = vector()
bic.num = vector()
MSE.aic = vector()
aic.num = vector()

for (j in 1:7) {
  #setting respective fold to either training or testing
  train_data = train[-folds[[j]], ] #eliminate columns in testing data
  test_data = train[folds[[j]], ]

  #doing best subset selection on our data
  regfit.train = regsubsets(loglatestPrice ~ ., data = train_data,  method= "backward" )
  reg.summary = summary(regfit.train)

  #selecting which subset is the best, specifically selecting how many variables it has
  bic.number = which.min(reg.summary$bic)
  bic.num[j] = bic.number

  #creating the model with the respective number of coeffficents
  bic.mdl = coef(regfit.train, bic.number)

  # testing the models that are created
  test.mat2 = model.matrix(loglatestPrice ~ ., data = test_data)
  bic.pred = test.mat2[, names(bic.mdl)] %*% bic.mdl


  # Start with the full model
```

```r
  full.model = lm(loglatestPrice ~ ., data = train_data)

  # Perform backward stepwise selection using AIC
  #stepwise.model = stepAIC(full.model, direction = "backward", trace = FALSE)

  #predict with the
  #aic.pred <- predict(stepwise.model, newdata = test_data)
  #aic.num[j] <- length(coef(stepwise.model))


  #computing the MSE for each model
  MSE.bic[j] = mean(( test_data$loglatestPrice - bic.pred)**2)
  #MSE.aic[j] = mean((test_data$loglatestPrice - aic.pred)**2)
}
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in =
## force.in, : 1 linear dependencies found

## Reordering variables and trying again:

## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in =
## force.in, : 2 linear dependencies found

## Reordering variables and trying again:

## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in =
## force.in, : 2 linear dependencies found

## Reordering variables and trying again:

## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in =
## force.in, : 1 linear dependencies found

## Reordering variables and trying again:
```

```r
paste("Estimated MSE for BIC is ", sum(MSE.bic))
```

```
## [1] "Estimated MSE for BIC is  0.835780797232455"
```

```r
#paste("Estimated MSE for AIC is ", sum(MSE.aic))

bic.num
```

```
## [1] 9 9 8 9 8 9 8
```

BIC says model with _____ variables, AIC says model with _____ variables

LASSO & Ridge

LASSO

```r
set.seed(1234)

vectorMSElasso = vector()
for (j in 1:7) {


  train_data = train[-folds[[j]], ]
  test_data = train[folds[[j]], ]
```

```
  train_data_x = as.matrix(train_data[,-6])
  train_data_y = as.matrix(train_data[, 6])
  test_data_x = as.matrix(test_data[,-6])
  test_data_y = as.matrix(test_data[, 6])

  grid = 10^seq(10, -2, length = 100)

  lasso.mod.cv = cv.glmnet(train_data_x, train_data_y, alpha = 1, lambda = grid, nfolds = 5)

  bestlambda = lasso.mod.cv$lambda.min

  lasso.fulldata = glmnet(train_data_x, train_data_y, alpha = 1, lambda = grid)
  lasso.coef = predict(lasso.fulldata , type = "coefficients",s = bestlambda)

  lasso.predict = predict(lasso.fulldata, s = bestlambda, newx = test_data_x)
  mse.lasso = mean((test_data_y - lasso.predict)^2)
  vectorMSElasso[j] = mse.lasso
}

mean(vectorMSElasso)
```

```
## [1] 0.1132495
```

Ridge

```
set.seed(1234)

vectorMSEridge = vector()
for (j in 1:7) {


  train_data = train[-folds[[j]], ]
  test_data = train[folds[[j]], ]

  train_data_x = as.matrix(train_data[,-6])
  train_data_y = as.matrix(train_data[, 6])
  test_data_x = as.matrix(test_data[,-6])
  test_data_y = as.matrix(test_data[, 6])

  grid = 10^seq(10, -2, length = 100)

  lasso.mod.cv = cv.glmnet(train_data_x, train_data_y, alpha = 0, lambda = grid, nfolds = 5)

  bestlambda = lasso.mod.cv$lambda.min

  lasso.fulldata = glmnet(train_data_x, train_data_y, alpha = 0, lambda = grid)
  lasso.coef = predict(lasso.fulldata , type = "coefficients",s = bestlambda)

  lasso.predict = predict(lasso.fulldata, s = bestlambda, newx = test_data_x)
  mse.lasso = mean((test_data_y - lasso.predict)^2)
  vectorMSEridge[j] = mse.lasso
}
mean(vectorMSEridge)
```

```
## [1] 0.110535
```

```r
#lowest mean squared error was Ridge, so we will run this over entire training dataset
train_data_x = as.matrix(train[,-6])
train_data_y = as.matrix(train[, 6])
test_data_x = as.matrix(test[,-6])
test_data_y = as.matrix(test[, 6])

grid = 10^seq(10, -2, length = 100)

ridge.mod.cv = cv.glmnet(train_data_x, train_data_y, alpha = 0, lambda = grid, nfolds = 5)


bestlambda = ridge.mod.cv$lambda.min


lasso.fulldata = glmnet(train_data_x, train_data_y, alpha = 0, lambda = grid)
ridge.coef = predict(lasso.fulldata , type = "coefficients",s = bestlambda)


ridge.predict = predict(lasso.fulldata, s = bestlambda, newx = test_data_x)
mse.ridge = mean((test_data_y - ridge.predict)^2)
paste("mean squared error on test data for ridge is ", mse.ridge)
```

```
## [1] "mean squared error on test data for ridge is  0.0980150942822328"
```

```r
paste("best lambda for ridge is ", bestlambda)
```

```
## [1] "best lambda for ridge is  0.01"
```

The Optimal model

Using our various models we produced the opitmal model by selecting the parameters ridge identified as the most valuable. We use a number of thresholds for eliminating variables and identify the elbow at which the reduction in MSE becomes negligible.

```r
thresholds = seq(1.0e-07,5.0e-01, length.out = 100)
mse_values = c(length(thresholds))
num_non_zero_coefs = c(length(thresholds))  # Vector to store number of non-zero coefficients

# Loop over thresholds and calculate MSE for each using normal linear regression
for (i in 1:length(thresholds)) {
  # Extract coefficients, excluding the intercept
  ridge.coef = coef(lasso.fulldata, s = bestlambda)
  coef_thresholded = as.numeric(ridge.coef[-1])  # Remove intercept

  # Apply thresholding
  coef_thresholded[abs(coef_thresholded) < thresholds[i]] = 0

  # Identify selected predictors
  selected_predictors = which(coef_thresholded != 0)
  num_non_zero_coefs[i] = length(selected_predictors)

  if (length(selected_predictors) > 0) {
    # Subset training and test data matrices
```

```r
    reduced_train_data_x = train_data_x[, selected_predictors, drop = FALSE]
    reduced_test_data_x = test_data_x[, selected_predictors, drop = FALSE]

    # Convert to data frames for lm()
    reduced_train_data_x_df = cbind(as.data.frame(reduced_train_data_x), loglatestPrice = train_data_y)
    reduced_test_data_x_df = cbind(as.data.frame(reduced_test_data_x), loglatestPrice = test_data_y)

    #print(names(reduced_train_data_x_df))  # Should include 'loglatestPrice'

    #print("BREAK BREAK BREAK ")

    #print(names(reduced_test_data_x_df))  # Should include 'loglatestPrice'

    # Fit linear regression model
    lm.reduced = lm(loglatestPrice ~ ., data = reduced_train_data_x_df)
    lm.predict = predict(lm.reduced, newdata = reduced_test_data_x_df)

    # Compute MSE
    mse_values[i] = mean((reduced_test_data_x_df$loglatestPrice - lm.predict)^2)
  } else {
    mse_values[i] = NA  # No predictors selected
  }
}
#via analyzing the graph we get the following optimal threshold
opt.threshold = NULL
opt.threshold.num = NULL

# Loop through the MSE values and find the threshold just before MSE exceeds 0.2
for (i in 2:length(mse_values)) {
  if (mse_values[i] > 0.15 && mse_values[i - 1] <= 0.15) {
    opt_threshold = thresholds[i - 1]
    opt_threshold_num = i-1
    break
  }
}

# Print the threshold just before MSE exceeds 0.21
paste('We thus having that by finding the elbow joint, the optimal threshold is about', opt_threshold,
```

```
## [1] "We thus having that by finding the elbow joint, the optimal threshold is about 0.09595967676767
```

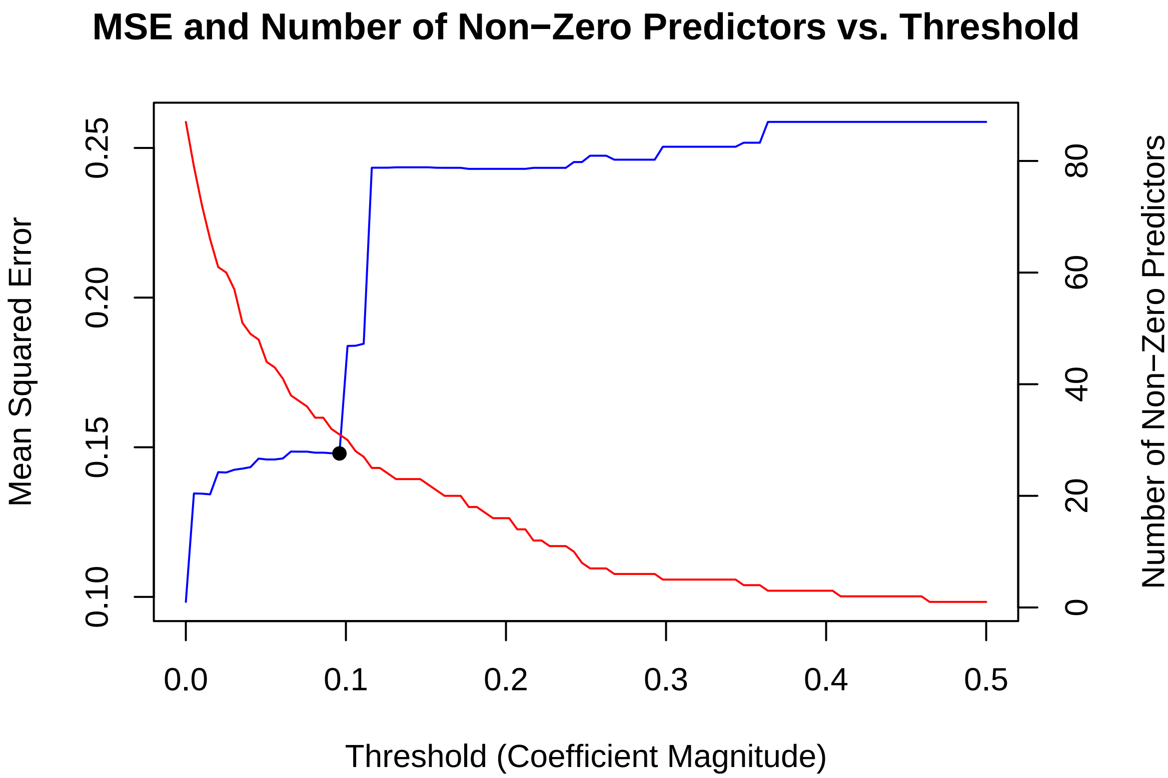
```r
# Adjust margins to create more space for the second y-axis label
par(mar = c(5, 4, 4, 6))

# Plot MSE and number of non-zero predictors on the same plot
plot(thresholds, mse_values, type = "l", col = "blue",
     xlab = "Threshold (Coefficient Magnitude)",
     ylab = "Mean Squared Error",
     main = "MSE and Number of Non-Zero Predictors vs. Threshold")

# Add a black dot at a specific index (e.g., opt_index)
points(thresholds[opt_threshold_num], mse_values[opt_threshold_num], col = "black", pch = 16)  # pch =
```

```r
# Add the second y-axis for the number of non-zero predictors
par(new = TRUE)
plot(thresholds, num_non_zero_coefs, type = "l", col = "red",
     axes = FALSE, xlab = "", ylab = "")

# Label the right axis
axis(side = 4)
mtext("Number of Non-Zero Predictors", side = 4, line = 3)
```




```r
print(thresholds[opt_threshold_num])
```

```
## [1] 0.09595968
```

```r
print(mse_values[opt_threshold_num])
```

```
## [1] 0.1479377
```

```r
# Reset plot parameters
par(mfrow = c(1, 1))
```

Analyzing the results from the vector

Comparison with Naive Model

As noted in our report we now consider a model based on what some relevant field experts consider to be the main features most buyers use when valuing houses.

```r
# Fit the normal linear regression model for naive data
lm_model_naive = lm(train_naive$loglatestPrice ~ ., data = train_naive)

naive_predictions <- predict(lm_model_naive, newdata = test_naive)
```

```
mse_naive <- mean((test_naive$loglatestPrice - naive_predictions)^2)

paste('Our coefficients form a linear model with MSE', mse_values[opt_threshold_num], 'and the naive mo
```

```
## [1] "Our coefficients form a linear model with MSE 0.147937667263368 and the naive model produces MS
```

Residual Plots

```
# Residual plot for the naive model
par(mfrow = c(1, 2))  # Set up two plots side by side

plot(naive_predictions, test_naive$loglatestPrice - naive_predictions,
     main = "Residuals for Naive Model",
     xlab = "Predicted Values",
     ylab = "Residuals")
abline(h = 0, col = "red")

# Fit the optimal model
coef_thresholded = ridge.coef
coef_thresholded = as.numeric(ridge.coef[-1])  # Remove intercept

coef_thresholded[abs(coef_thresholded) < thresholds[opt_threshold_num]] = 0

opt_predictors = which(coef_thresholded != 0)

opt_reduced_train_data_x = train[, opt_predictors, drop = FALSE]
opt_reduced_test_data_x = test[, opt_predictors, drop = FALSE]

opt_reduced_train_data_x_df = cbind(as.data.frame(opt_reduced_train_data_x), loglatestPrice = train_data
opt_reduced_test_data_x_df = cbind(as.data.frame(opt_reduced_test_data_x), loglatestPrice = test_data_y

lm_model_optimal = lm(loglatestPrice ~ ., data = opt_reduced_train_data_x_df)
opt_predictions = predict(lm_model_optimal, newdata = opt_reduced_test_data_x_df)

paste("MSE: ", mean((opt_reduced_test_data_x_df$loglatestPrice - opt_predictions)^2))
```

```
## [1] "MSE:  0.12685782528971"
```

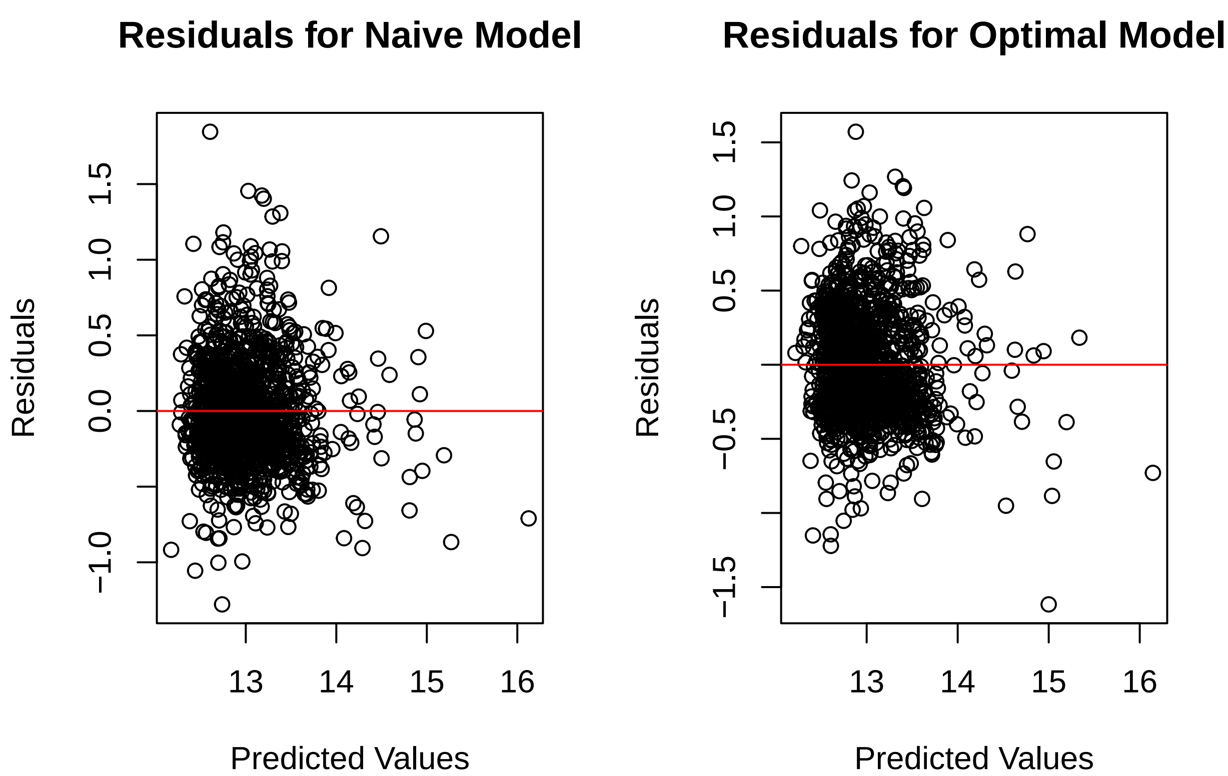
```
summary(lm_model_optimal)
```

```
##
## Call:
## lm(formula = loglatestPrice ~ ., data = opt_reduced_train_data_x_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.86104 -0.24378 -0.04368  0.20947  2.42344
##
## Coefficients:
##                             Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                1.206e+01  1.159e-02 1040.480  < 2e-16 ***
## avgSchoolRating            6.688e-02  2.047e-03   32.671  < 2e-16 ***
## logavgSchoolDistance      -1.538e-01  6.520e-03  -23.596  < 2e-16 ***
## livingMult                 2.251e-03  3.172e-05   70.951  < 2e-16 ***
```

```
## month12                      5.212e-02  1.300e-02    4.008 6.15e-05 ***
## propertyTaxRate2.01         -9.879e-02  1.872e-02   -5.278 1.33e-07 ***
## parkingSpaces4               7.725e-02  1.765e-02    4.377 1.21e-05 ***
## parkingSpaces8               1.439e-01  1.551e-01    0.928   0.3535
## parkingSpaces10              1.819e-01  1.900e-01    0.957   0.3384
## numPriceChanges4            -1.759e-02  1.122e-02   -1.567   0.1171
## numPriceChanges5            -8.002e-02  1.367e-02   -5.854 4.93e-09 ***
## numPriceChanges6            -1.476e-01  1.626e-02   -9.073  < 2e-16 ***
## numPriceChanges7            -2.399e-01  1.976e-02  -12.140  < 2e-16 ***
## numPriceChanges8            -1.699e-01  2.431e-02   -6.991 2.86e-12 ***
## numPriceChanges9            -1.825e-01  3.076e-02   -5.933 3.06e-09 ***
## numPriceChanges11           -1.372e-02  4.762e-02   -0.288   0.7732
## numPriceChanges12           -4.992e-02  4.663e-02   -1.071   0.2843
## numPriceChanges13           -8.743e-02  5.611e-02   -1.558   0.1192
## numPriceChanges14           -3.060e-02  7.181e-02   -0.426   0.6701
## numPriceChanges17            5.368e-02  1.898e-01    0.283   0.7773
## numPriceChanges19            4.857e-02  2.190e-01    0.222   0.8245
## numOfAppliances9            -4.717e-02  5.184e-02   -0.910   0.3629
## numOfPatioAndPorchFeatures3  1.306e-01  1.578e-02    8.276  < 2e-16 ***
## numOfPatioAndPorchFeatures5  1.123e-01  8.496e-02    1.321   0.1864
## numOfPatioAndPorchFeatures6  3.433e-01  1.897e-01    1.809   0.0704 .
## numOfPatioAndPorchFeatures7  2.726e-02  2.694e-01    0.101   0.9194
## numOfSecurityFeatures4       7.460e-02  4.632e-02    1.611   0.1073
## numOfSecurityFeatures6      -5.671e-02  2.682e-01   -0.211   0.8325
## numOfWaterfrontFeatures1     5.690e-01  1.096e-01    5.191 2.12e-07 ***
## numOfStories2               -1.296e-01  8.129e-03  -15.945  < 2e-16 ***
## numOfStories3                3.606e-01  3.684e-02    9.789  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3791 on 12658 degrees of freedom
## Multiple R-squared:  0.4812, Adjusted R-squared:  0.4799
## F-statistic: 391.3 on 30 and 12658 DF,  p-value: < 2.2e-16
```

```r
optimal_predictions = predict(lm_model_optimal, newdata = opt_reduced_test_data_x_df)

# Plot residuals for the optimal model
plot(optimal_predictions, test_data_y - optimal_predictions,
     main = "Residuals for Optimal Model",
     xlab = "Predicted Values",
     ylab = "Residuals")
abline(h = 0, col = "red")
```




```r
# Reset the plot layout
par(mfrow = c(1, 1))
```

Whole Regression Model

```r
train_data_x_df = cbind(as.data.frame(train_data_x), loglatestPrice = train_data_y)
test_data_x_df = cbind(as.data.frame(test_data_x), loglatestPrice = test_data_y)

lm_model_whole = lm(loglatestPrice ~., data = train_data_x_df)

predictions = predict(lm_model_whole, newdata = test_data_x_df)

paste('MSE: ',mean((test_data_x_df$loglatestPrice - predictions)^2))
```

```
## [1] "MSE:  0.098338136534761"
```

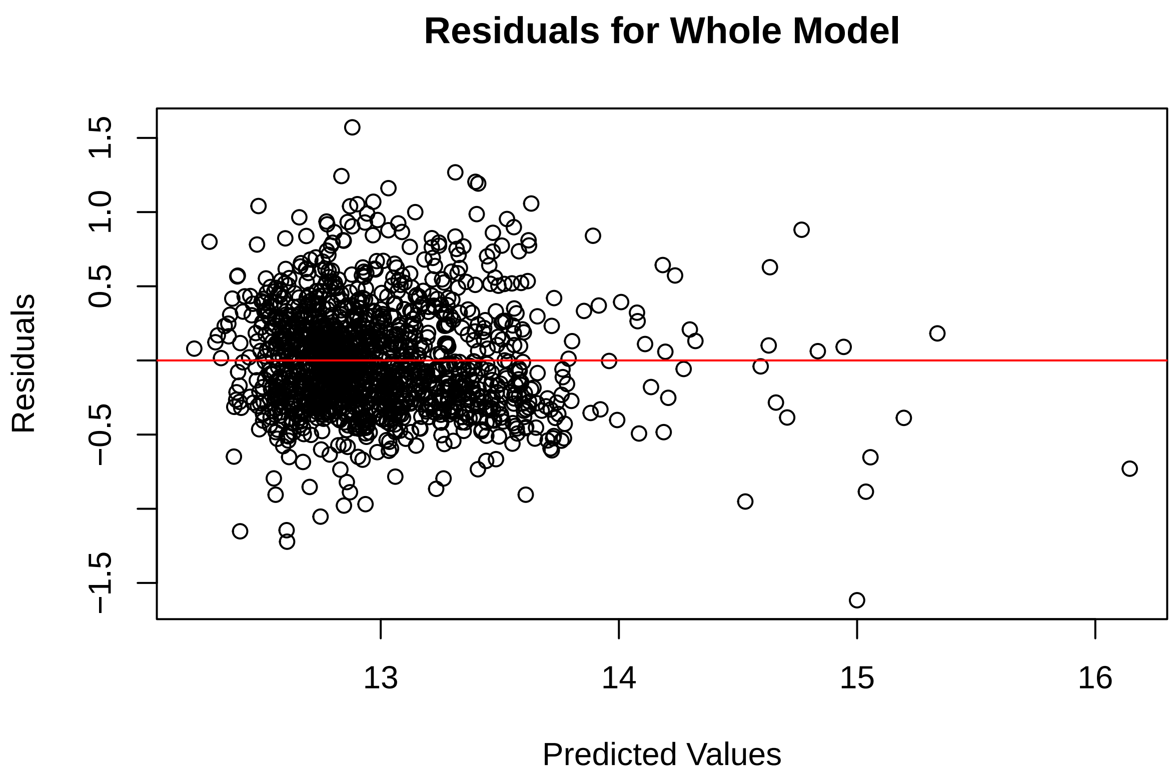
```r
summary(lm_model_whole)
```

```
##
## Call:
## lm(formula = loglatestPrice ~ ., data = train_data_x_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.91937 -0.18793 -0.01875  0.16828  2.29747
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   1.582e+01  4.224e-01  37.447  < 2e-16 ***
```

```
## yearBuilt                  -1.459e-03  1.993e-04  -7.324 2.55e-13 ***
## avgSchoolRating             1.045e-01  2.259e-03  46.262  < 2e-16 ***
## gdp_3_month_lag             2.308e-07  5.712e-08   4.041 5.35e-05 ***
## unemployment_1_month_lag    8.677e-04  1.840e-03   0.472 0.637175
## mortgage_1_month_lag       -4.561e-02  9.508e-03  -4.797 1.63e-06 ***
## loglotSizeSqFt              1.079e-01  6.782e-03  15.916  < 2e-16 ***
## logavgSchoolDistance       -7.648e-02  6.192e-03 -12.350  < 2e-16 ***
## logavgSchoolSize           -3.199e-01  1.419e-02 -22.544  < 2e-16 ***
## livingMult                  2.162e-03  3.386e-05  63.834  < 2e-16 ***
## hasAssociationTRUE         -1.793e-01  8.398e-03 -21.351  < 2e-16 ***
## hasCoolingTRUE             -3.795e-02  2.629e-02  -1.444 0.148852
## hasSpaTRUE                  5.807e-02  1.195e-02   4.858 1.20e-06 ***
## hasViewTRUE                 3.200e-02  7.536e-03   4.246 2.19e-05 ***
## month2                      6.698e-03  1.858e-02   0.361 0.718437
## month3                      1.645e-02  1.772e-02   0.929 0.353165
## month4                      9.884e-03  1.750e-02   0.565 0.572315
## month5                     -7.598e-03  1.760e-02  -0.432 0.665998
## month6                     -2.899e-03  1.748e-02  -0.166 0.868248
## month7                     -3.396e-04  1.735e-02  -0.020 0.984385
## month8                      1.523e-02  1.748e-02   0.871 0.383602
## month9                      2.694e-02  1.776e-02   1.517 0.129343
## month10                    -3.457e-03  1.772e-02  -0.195 0.845321
## month11                     9.234e-03  1.812e-02   0.510 0.610408
## month12                     3.121e-02  1.811e-02   1.723 0.084859 .
## propertyTaxRate2.01        -2.691e-01  1.856e-02 -14.503  < 2e-16 ***
## propertyTaxRate2.21        -2.474e-01  1.281e-02 -19.310  < 2e-16 ***
## parkingSpaces1              6.183e-02  1.284e-02   4.816 1.49e-06 ***
## parkingSpaces2              1.110e-02  7.502e-03   1.480 0.138957
## parkingSpaces3              2.196e-02  1.384e-02   1.587 0.112533
## parkingSpaces4              1.816e-02  1.651e-02   1.100 0.271303
## parkingSpaces5              1.047e-01  4.264e-02   2.454 0.014137 *
## parkingSpaces6             -5.468e-02  3.838e-02  -1.425 0.154236
## parkingSpaces7              5.574e-02  1.182e-01   0.471 0.637300
## parkingSpaces8              1.930e-02  1.355e-01   0.142 0.886754
## parkingSpaces9             -2.610e-01  1.915e-01  -1.363 0.172900
## parkingSpaces10             8.762e-02  1.661e-01   0.528 0.597821
## parkingSpaces12             2.296e-01  1.933e-01   1.188 0.234880
## numPriceChanges2            1.405e-02  8.102e-03   1.734 0.083031 .
## numPriceChanges3           -4.138e-03  9.401e-03  -0.440 0.659834
## numPriceChanges4           -3.905e-02  1.059e-02  -3.686 0.000229 ***
## numPriceChanges5           -1.117e-01  1.262e-02  -8.855  < 2e-16 ***
## numPriceChanges6           -1.845e-01  1.477e-02 -12.491  < 2e-16 ***
## numPriceChanges7           -2.777e-01  1.776e-02 -15.639  < 2e-16 ***
## numPriceChanges8           -2.274e-01  2.166e-02 -10.498  < 2e-16 ***
## numPriceChanges9           -2.216e-01  2.722e-02  -8.140 4.34e-16 ***
## numPriceChanges10          -1.965e-01  3.122e-02  -6.293 3.21e-10 ***
## numPriceChanges11          -7.408e-02  4.184e-02  -1.771 0.076639 .
## numPriceChanges12          -1.246e-01  4.103e-02  -3.036 0.002405 **
## numPriceChanges13          -1.602e-01  4.924e-02  -3.253 0.001144 **
## numPriceChanges14          -1.398e-01  6.340e-02  -2.205 0.027483 *
## numPriceChanges15          -1.688e-01  7.903e-02  -2.137 0.032651 *
## numPriceChanges16           8.366e-02  1.105e-01   0.757 0.448990
## numPriceChanges17          -1.061e-02  1.660e-01  -0.064 0.949045
## numPriceChanges18          -2.184e-01  2.350e-01  -0.929 0.352789
```

```
## numPriceChanges19            6.670e-02  1.912e-01   0.349 0.727166
## numPriceChanges20            2.064e-01  3.312e-01   0.623 0.533148
## numPriceChanges22            8.341e-02  3.336e-01   0.250 0.802586
## numPriceChanges23           -2.713e-02  3.311e-01  -0.082 0.934707
## numOfAppliances1             1.832e-02  1.680e-02   1.090 0.275572
## numOfAppliances2             4.516e-02  1.479e-02   3.055 0.002258 **
## numOfAppliances3             4.653e-02  1.379e-02   3.374 0.000744 ***
## numOfAppliances4             6.457e-02  1.399e-02   4.617 3.94e-06 ***
## numOfAppliances5             7.601e-02  1.629e-02   4.666 3.10e-06 ***
## numOfAppliances6             6.460e-02  1.876e-02   3.444 0.000574 ***
## numOfAppliances7             2.166e-02  1.837e-02   1.179 0.238224
## numOfAppliances8             5.943e-02  2.161e-02   2.751 0.005958 **
## numOfAppliances9            -2.570e-02  4.702e-02  -0.547 0.584663
## numOfAppliances10            2.065e-01  3.318e-01   0.622 0.533645
## numOfPatioAndPorchFeatures1 -1.102e-02  9.977e-03  -1.104 0.269592
## numOfPatioAndPorchFeatures2  3.890e-02  1.148e-02   3.387 0.000708 ***
## numOfPatioAndPorchFeatures3  5.007e-02  1.608e-02   3.114 0.001847 **
## numOfPatioAndPorchFeatures4  1.033e-01  2.918e-02   3.541 0.000400 ***
## numOfPatioAndPorchFeatures5  1.870e-02  7.528e-02   0.248 0.803782
## numOfPatioAndPorchFeatures6  2.456e-01  1.661e-01   1.478 0.139355
## numOfPatioAndPorchFeatures7 -1.059e-01  2.384e-01  -0.444 0.656966
## numOfPatioAndPorchFeatures8  4.640e-01  3.427e-01   1.354 0.175707
## numOfSecurityFeatures1      -1.421e-02  9.160e-03  -1.551 0.120876
## numOfSecurityFeatures2       1.556e-02  1.236e-02   1.259 0.208018
## numOfSecurityFeatures3       7.713e-03  1.883e-02   0.410 0.682084
## numOfSecurityFeatures4       7.617e-02  4.162e-02   1.830 0.067281 .
## numOfSecurityFeatures5       1.575e-01  8.645e-02   1.822 0.068519 .
## numOfSecurityFeatures6      -6.103e-02  2.348e-01  -0.260 0.794934
## numOfWaterfrontFeatures1     3.539e-01  9.598e-02   3.687 0.000227 ***
## numOfWaterfrontFeatures2     7.728e-01  1.358e-01   5.692 1.29e-08 ***
## numOfStories2               -2.675e-02  7.537e-03  -3.548 0.000389 ***
## numOfStories3                3.484e-01  3.255e-02  10.705  < 2e-16 ***
## numOfStories4                3.921e-01  2.368e-01   1.656 0.097780 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3305 on 12601 degrees of freedom
## Multiple R-squared:  0.6074, Adjusted R-squared:  0.6047
## F-statistic: 224.1 on 87 and 12601 DF,  p-value: < 2.2e-16
```

```r
# Plot residuals for the optimal model
plot(optimal_predictions, test_data_y - optimal_predictions,
     main = "Residuals for Whole Model",
     xlab = "Predicted Values",
     ylab = "Residuals")
abline(h = 0, col = "red")
```


Residuals for Whole Model
Residuals
Predicted Values
1.5
1.0
0.5
−0.5
−1.5
13
14
15
16