Mini Project 2: Predicting Housing Prices in Ames, Iowa

Chris Corona, Farshina Nazrul, Gak Roppongi, Nolan Walker

2022-11-03

Abstract

In this mini-project, our task is to select predictors, fit, and evaluate several different types of models for regression. The data set is from the Ames Assessor's Office house in computing assessed values for individual residential properties sold in Ames, IA, from 2006 to 2010. This dataset has 2930 observations with 82 predictors (2930 * 82). 82 predictors include 23 nomial (categorical) variables, 23 ordinal variables, 14 discrete variables, and 20 continuous variables with two additional observation identifiers. We tried to predict the price of the residential properties based the predictors. After fitting various models, we compared the OOO to determine which model has the best performance.

1 Summary

This data set presented a challenging problem with so much missing data to clean and then so many variables to sort through. To deal with the 74 variables, we approached the task in several ways. First, we tried to reduce the variables through subset selection - choosing the most important variables one at a time in a stepwise fashion. This only achieved a minor reduction and left us with a somewhat confusing interpretation. The neighborhood, square footage of different parts of the house, condition, quality, roofing material, are all important when predicting price - not much of a reduction. Next we tried two different shrinkage methods, lasso and ridge regression. Ridge regression shrinks the coefficients towards zero, but not exactly zero so this actually does not perform any reduction on the variables. Lasso on the other hand shrinks many of the coefficients exactly to zero. The best lasso model left us with just 9 non-zero coefficients which were related to area of the house, year built/remodeled, and condition. This is a much more interpretable result. And finally, we tried Principal Component Analysis which allowed us to reduce the model down to just five variables, a huge reduction. However, PCA is known for even more confusing interpretations because the principal components are derivations from all the other variables. That being said, the first two PCs could be interpreted as relating to overall quality, area of the house, and year built/remodeled. After these analyses, we can say that the most important variables in predicting the price of a house in Ames, IA between the years of 1872-2010 are area, year built/remodeled, and overall quality.

The final step in our analysis was using just one predictor, year built, to predict price using three different methods. Simple linear regression gives us simple results. The regression line does not capture the non-linearity of the data, and therefore is not a good model to predict price (particularly for very old or very new houses). Next, we tried polynomial regression which the 2nd order polynomial gave the best results. This makes sense because the curvature of the data most closely matches a parabola. And finally, we used smoothing spline to fit a model. The best smooth spline appears to greatly overfit the data as evidenced by how wiggly and flexible the fit is. In the end, the best model is the one that does the best job at fitting to the underlying truth of the data. The challenge is to not underfit or overfit to the data. This is more of an art than a science.

2 Data Preprocessing

The data set is from the Ames Assessor's Office house in computing assessed values for individual residential properties sold in Ames, IA, from 2006 to 2010. There are 2930 observations. The prices were between $12,789 and $755,000 with the mean value of $180,796.0600682594, the median value of $160,000, and the standard deviation of 79886.692356665. "Pool quality", "Miscellaneous Feature", "Alley", "Fence", "Fireplace quality", and "Lot Frontage" features were removed because most of the values in these columns were missed. We filled the missing value with the most common response for "Masonry veneer type", "Garage Condition", "Garage Quality", "Garage Finish", "Garage Year Built", "Garage Type", "Basement Condition", "Basement Quality", "Basement Exposure", "Basement Finish Type. 1", and "Basement Finish Type. 2". As a preprocessing step, we split out data into training and test sets. We performed 10 times cross validation. First 10% of the dataset was used as the test set and the other 90% was used as the training set for the first iteration, the second 10% was used as the test set and the other 90% was used as the training set for the second iteration, and we repeated this for 10 times to complete the 10 times cross validation. Since this is a regression dataset, the datasets were split according to the index.

3 Exploratory Analysis

In our Explanatory Data Analysis (EDA), we rigorously reviewed the data for any issues and reduced the number of predictors from 80 to 74. We first checked for any missing data. 6 features were removed because most of the values were missed in these columns. Missing values in 11 features were replaced with the most common response in each of the feature. We followed the instruction from the data description and removed the observations with the area more than 4000 squared feet. Next, we analyzed the variable correlation. The highest correlation between a variable and the response variable was between area and price at ~0.8. Then, we performed the forward and backward selection. However,

```
# read in the data set
library(tidyverse)
data <- read_csv("ames.csv")

# take a peak at the data
#head(data)

# immediately drop order and pid as predictors
data <- data[,-c(1,2)]

# check structure of data
#str(data)

# change strings into factors
data$MS.Zoning <- factor(data$MS.Zoning)
data$Street <- factor(data$Street)
data$Alley <- factor(data$Alley)
data$Lot.Shape <- factor(data$Lot.Shape)
data$Land.Contour <- factor(data$Land.Contour)
data$Utilities <- factor(data$Utilities)
data$Lot.Config <- factor(data$Lot.Config)
data$Land.Slope <- factor(data$Land.Slope)
data$Neighborhood <- factor(data$Neighborhood)
data$Condition.1 <- factor(data$Condition.1)
data$Condition.2 <- factor(data$Condition.2)
data$Bldg.Type <- factor(data$Bldg.Type)
```

```
data$House.Style <- factor(data$House.Style)
data$Roof.Style <- factor(data$Roof.Style)
data$Roof.Matl <- factor(data$Roof.Matl)
data$Exterior.1st <- factor(data$Exterior.1st)
data$Exterior.2nd <- factor(data$Exterior.2nd)
data$Mas.Vnr.Type <- factor(data$Mas.Vnr.Type)
data$Exter.Qual <- factor(data$Exter.Qual)
data$Exter.Cond <- factor(data$Exter.Cond)
data$Foundation <- factor(data$Foundation)
data$Bsmt.Qual <- factor(data$Bsmt.Qual)
data$Bsmt.Cond <- factor(data$Bsmt.Cond)
data$Bsmt.Exposure <- factor(data$Bsmt.Exposure)
data$BsmtFin.Type.1 <- factor(data$BsmtFin.Type.1)
data$BsmtFin.Type.2 <- factor(data$BsmtFin.Type.2)
data$Heating <- factor(data$Heating)
data$Heating.QC <- factor(data$Heating.QC)
data$Central.Air <- factor(data$Central.Air)
data$Electrical <- factor(data$Electrical)
data$Kitchen.Qual <- factor(data$Kitchen.Qual)
data$Functional <- factor(data$Functional)
data$Fireplace.Qu <- factor(data$Fireplace.Qu)
data$Garage.Type <- factor(data$Garage.Type)
data$Garage.Finish <- factor(data$Garage.Finish)
data$Garage.Qual <- factor(data$Garage.Qual)
data$Garage.Cond <- factor(data$Garage.Cond)
data$Paved.Drive <- factor(data$Paved.Drive)
data$Pool.QC <- factor(data$Pool.QC)
data$Fence <- factor(data$Fence)
data$Misc.Feature <- factor(data$Misc.Feature)
data$Sale.Type <- factor(data$Sale.Type)
data$Sale.Condition <- factor(data$Sale.Condition)

# check for missing data - lots of missing data
library(mice)
#md.pattern(data, rotate.names=TRUE)

# drop pool.qu, misc.feature, alley, fence, fireplace.qu, and lot.frontage because most of that data is
data <- data[,-c(5,8,58,73,74,75)]

# remove single observations with missing data
data <- data[-c(which(is.na(data$Bsmt.Full.Bath)), which(is.na(data$Garage.Area)), which(is.na(data$Ele

# remove single observations with missing data
data <- data[-c(1356,67,1794,2776,445,2820),]

# impute Mas.Vnr values with most common response
data[which(is.na(data$Mas.Vnr.Type)),25] <- "None"
data[which(is.na(data$Mas.Vnr.Area)),26] <- 0

# impute Garage values with most common response
data$Garage.Cond[is.na(data$Garage.Cond)]<-factor('TA')
data$Garage.Qual[is.na(data$Garage.Qual)]<-factor('TA')
data$Garage.Finish[is.na(data$Garage.Finish)]<-factor('Unf')
```

```
data$Garage.Yr.Blt[is.na(data$Garage.Yr.Blt)]<-2005
data$Garage.Type[is.na(data$Garage.Type)]<-factor(unique(data$Garage.Type)[1])

# impute Bsmt values with most common response
data[which(is.na(data$Bsmt.Cond)),which(colnames(data)=="Bsmt.Cond")] = names(which.max(table(data$Bsmt
data[which(is.na(data$Bsmt.Qual)),which(colnames(data)=="Bsmt.Qual")] = sample(c("Gd", "TA"), size = nr
data[which(is.na(data$Bsmt.Exposure)),which(colnames(data)=="Bsmt.Exposure")] = names(which.max(table(da
data[which(is.na(data$BsmtFin.Type.1)),which(colnames(data)=="BsmtFin.Type.1")] = sample(c("GLQ", "Unf"
data[which(is.na(data$BsmtFin.Type.2)),which(colnames(data)=="BsmtFin.Type.2")] = names(which.max(table

# now there is no missing data
#md.pattern(data, rotate.names=TRUE)

# per instructions from data set, remove observations with area > 4000
data <- data[-c(which(data$area > 4000)),]
```

4 Model development and performance evaluation

Subset Selection

Best Subset and step-wise selection are two model selection methods that reduce the number of terms in a model, by comparing AICs of models with different number of terms. Best subset compares all models with 1 to N number of terms, where N is the total number of variables in the data set, against each other and returns the best model for each value of n. This approach is computationally impractical for data sets where there are a large number of features, as it scales exponentially with the number of features. Step-wise selection is a model selection approach that instead takes into account the performance of the model at each step after adding or removing a term from the model, for forward or backward selection respectively. It returns the model with terms that only increased the performance at the addition or removal at each step of a term, that stopped when performance started to decrease relative to the last best performing model.

```
# fit a simple linear regression model
library(caret)
m1 <- lm(price~., data=data) # full model
train_control <- trainControl(method = "cv",
                              number = 10)
model<-train(price ~ .,data=data,method='lm',trControl=train_control) # full model trained using cross
print(model)
reduced.m1<-step(m1,direction = "backward",trace=0) # backward selection
s1<-summary(reduced.m1)
m1.model<- as.formula(paste0('price ~ ',paste(attr(s1$terms, "term.labels"), collapse=" + ")))
model.1 <- train(m1.model, data = data,
                 method = "lm",
                 trControl = train_control)
print(model.1) # backward selected model trained with cross validation
reduced.m2<-step(m1,direction ='forward',trace = 0) # forward selection
s2<-summary(reduced.m2)
m2.model<- as.formula(paste0('price ~ ',paste(attr(s2$terms, "term.labels"), collapse=" + ")))
model.2 <- train(m2.model, data = data,
                 method = "lm",
                 trControl = train_control)
print(model.2)
```

```
library(caret)
library(mltools)
library(data.table)

newdata <- one_hot(as.data.table(data)) # one hot encode all of the categorical variables
m1.onehot <- lm(price~., data=newdata) # full model
model.onehot.full<-train(price ~ .,data=newdata,method='lm',trControl=train_control) # cross validation
print(model.onehot.full)
reduced.m1.onehot<-step(m1.onehot,direction = "backward",trace=0) # backward selection
s1.onehot<-summary(reduced.m1.onehot)
m1.onehot.model<- as.formula(paste0('price ~ ',paste(attr(s1.onehot$terms, "term.labels"), collapse=" +
model.onehot.1 <- train(m1.onehot.model, data = newdata,
               method = "lm",
               trControl = train_control)
print(model.onehot.1)
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
load("reducedModels")
load("reducedModelCV")
#print(model.1) # this is the trained model using 10 fold cv from backward selection
#print(model.onehot.1) # this is the trained model using 10 fold cv from backward selection on one hot
m1 <- lm(price~., data=data)
train_control <- trainControl(method = "cv",
                              number = 10)
model<-train(price ~ .,data=data,method='lm',trControl=train_control)
#print(model) # this is just the trained model using cross validation without one hot encoding or backw
```

```
load("reducedOnehotSummary")
idx <- order(coef(s1.onehot)[,4])  # sort out the p-values
out <- coef(s1.onehot)[idx,]
print(out[out[,4]<.05,c(1,4)]) # coefficients from model sorted by p vals
```

```
##                             Estimate      Pr(>|t|)
## Roof.Matl_ClyTile      -6.541570e+05 3.734999e-115
## area                    4.899541e+01  4.637292e-66
## Neighborhood_StoneBr    4.712845e+04  2.603557e-35
## Overall.Qual            7.496563e+03  3.343027e-33
## Neighborhood_NridgHt    3.111856e+04  4.878000e-33
## Neighborhood_NoRidge    3.858718e+04  4.019759e-32
## BsmtFin.SF.1            3.070232e+01  8.188474e-32
## Misc.Val               -8.727790e+00  5.785672e-27
```

```
## Overall.Cond             5.600154e+03  3.474018e-26
## Bldg.Type_1Fam           2.426120e+04  1.311599e-25
## Neighborhood_Somerst     2.413950e+04  1.272028e-22
## Kitchen.Qual_Ex          2.256745e+04  3.589225e-21
## Bsmt.Exposure_Gd         1.813739e+04  8.666704e-21
## Bsmt.Qual_Ex             1.964933e+04  1.412907e-19
## Condition.2_PosN        -1.111750e+05  5.654908e-19
## Lot.Area                 6.096945e-01  7.031330e-16
## BsmtFin.SF.2             3.290237e+01  1.797417e-15
## Neighborhood_GrnHill     1.263057e+05  1.820325e-14
## Exter.Qual_Ex            2.387125e+04  1.273928e-13
## Roof.Matl_CompShg       -6.064824e+04  2.610131e-12
## Condition.1_Norm         9.791548e+03  2.700137e-12
## Year.Built               2.407314e+02  1.643738e-10
## Neighborhood_Crawfor     1.747126e+04  2.492725e-10
## Exterior.1st_BrkFace     1.686944e+04  4.098810e-10
## Bsmt.Unf.SF              1.340007e+01  1.598979e-08
## House.Style_1Story       1.043999e+04  1.821048e-08
## Roof.Matl_WdShake       -6.412097e+04  2.295463e-08
## Screen.Porch             4.447029e+01  2.428694e-08
## Pool.Area                6.824306e+01  1.207118e-07
## Bsmt.Exposure_Av         7.407956e+03  2.526550e-07
## Mas.Vnr.Area             1.920143e+01  2.571985e-07
## Sale.Type_New            8.790949e+03  3.230372e-06
## `Roof.Matl_Tar&Grv`     -4.861543e+04  5.654868e-06
## Condition.2_PosA         5.465662e+04  1.394226e-05
## Sale.Condition_Abnorml -7.826559e+03  1.427642e-05
## Fireplaces               3.693406e+03  1.453961e-05
## Neighborhood_Edwards    -8.206289e+03  5.269414e-05
## Land.Contour_Bnk        -9.393813e+03  8.221926e-05
## X2nd.Flr.SF              1.381680e+01  8.519175e-05
## Condition.1_PosN         1.620394e+04  1.129208e-04
## Garage.Qual_Ex           6.981785e+04  3.256009e-04
## Lot.Config_CulDSac       6.572861e+03  5.786654e-04
## Neighborhood_NPkVill     1.907924e+04  6.843556e-04
## Land.Slope_Mod           2.477835e+04  7.913510e-04
## Bedroom.AbvGr           -2.621373e+03  8.395246e-04
## Neighborhood_BrDale      1.822136e+04  9.115680e-04
## Land.Contour_Low        -1.216109e+04  1.257092e-03
## Bldg.Type_2fmCon         1.322837e+04  2.110695e-03
## Exterior.1st_PreCast     7.144042e+04  2.691044e-03
## Exterior.1st_MetalSd     4.046507e+03  2.916924e-03
## Garage.Cond_Ex          -5.537691e+04  3.104841e-03
## BsmtFin.Type.2_LwQ      -8.159345e+03  3.190949e-03
## Full.Bath                4.027983e+03  3.309251e-03
## Mas.Vnr.Type_CBlock     -6.737885e+04  3.561786e-03
## Garage.Area              1.418483e+01  3.606275e-03
## MS.Zoning_RL             5.102231e+03  3.628292e-03
## Heating.QC_Ex            3.270043e+03  3.780935e-03
## House.Style_2.5Fin      -2.587233e+04  3.910683e-03
## BsmtFin.Type.2_Rec      -7.601937e+03  4.028942e-03
## Mas.Vnr.Type_BrkFace    -3.702238e+03  4.061867e-03
## Roof.Style_Flat         -2.540958e+04  5.501468e-03
## Kitchen.AbvGr           -1.028292e+04  5.824926e-03
```

```
## Sale.Type_Con           2.803748e+04  6.586380e-03
## Functional_Maj1        -1.501322e+04  6.830701e-03
## Functional_Min2        -7.743513e+03  7.984123e-03
## Garage.Yr.Blt           7.107056e+01  9.443831e-03
## Functional_Mod         -1.053419e+04  9.825312e-03
## Neighborhood_MeadowV    1.220903e+04  1.008264e-02
## Neighborhood_NWAmes    -6.114098e+03  1.196597e-02
## Garage.Qual_Gd          1.228178e+04  1.260600e-02
## Year.Remod.Add          8.497792e+01  1.353905e-02
## Neighborhood_BrkSide    6.650979e+03  1.362639e-02
## Bldg.Type_Twnhs        -7.787479e+03  1.379081e-02
## Heating_OthW           -4.047283e+04  1.391204e-02
## Street_Grvl            -1.762185e+04  1.425590e-02
## Garage.Finish_RFn      -2.574093e+03  1.563387e-02
## Condition.2_RRAe        5.849103e+04  1.572026e-02
## Land.Slope_Gtl          1.763194e+04  1.690963e-02
## Exter.Qual_Fa           1.101013e+04  1.840714e-02
## BsmtFin.Type.1_LwQ     -4.805857e+03  2.193252e-02
## Bldg.Type_Duplex        1.082977e+04  2.302073e-02
## Roof.Matl_Roll         -5.497535e+04  2.577453e-02
## Lot.Config_FR2         -5.813948e+03  2.579927e-02
## Functional_Maj2        -1.739224e+04  2.668491e-02
## Functional_Sal         -3.780071e+04  2.670581e-02
## Neighborhood_Blmngtn    1.105346e+04  2.719552e-02
## Neighborhood_Greens     1.927611e+04  3.018307e-02
## Garage.Cars             3.047177e+03  3.198328e-02
## BsmtFin.Type.2_BLQ     -6.519595e+03  3.220404e-02
## Neighborhood_Mitchel   -5.111395e+03  3.482812e-02
## Functional_Sev         -3.550133e+04  3.514866e-02
## Land.Contour_HLS        5.346337e+03  3.613726e-02
## BsmtFin.Type.1_Rec     -3.367571e+03  3.666833e-02
## Exterior.2nd_VinylSd    2.688500e+03  3.779772e-02
## Lot.Shape_IR1          -2.092068e+03  3.974367e-02
## BsmtFin.Type.1_GLQ      2.718602e+03  4.295348e-02
## Sale.Condition_AdjLand  1.406146e+04  4.749378e-02
```

Table 1: List of coefficients and p-values from best model using backward selection and one-hot encoding

A Simple linear model was fit to the data using all of the variables and backwards selection was performed, performance was evaluated using cross validation. The model obtained using backwards selection had a lower RMSE and used 53 of the 72 variables in the data(RMSE 34524 vs 35120). However the model still had 172 terms, since some of the variables were categorical with multiple levels each getting their own term in the model. To try to reduce the number of terms, the categorical variables were one hot encoded, and backwards selection was performed on the full model with all of the one hot encoded variables, this resulted in a model with fewer terms, 123, and the lowest RMSE of any model tested(RMSE=26755 vs 35120). There were over 40 terms in the reduced model with p-values below .001, most with very large coefficients, so the best model is one that with a lot of terms. Some of the most significant terms were, in order, clay tile roofing(categorical), area, stone brook neighborhood(categorical), Overall quality, Finished Basement Square Foot, Value of miscellaneous features, single family building type(categorical), Excellent Kitchen Quality(categorical), and so on. While many of the terms describe similar features, the types of terms that increase value are the ones for total area and quality of each floor/room/area, the location within and which neighborhood the house is in, having various features like a pool, a fireplace, a finished basement/garage, the Year built/remodeled/additions added and the type/size of area outside the house. Interestingly, almost all of the coefficients for terms that describe roofing material are negative.

Lasso and Ridge Regression

Both Lasso and Ridge are shrinkage or regularization methods that estimates the coefficients of 'not-so-useful' predictors towards zero (ridge) or exactly zero (lasso) based on the shrinkage penalty factor, λ. In this project, we used the 10-fold Cross validation for model performance evaluation. However, we only used the numeric predictors. A plot of the cross-validated prediction RMSE for each value of λ is shown in the Figures for both ridge and lasso. The best λ for lasso was selected as 1151.4 and their corresponding RMSE was 36490.15, whereas the best λ for ridge was selected as 3727.59 and their corresponding RMSE was 35716.38.

Lasso had 9 non-zero coefficients. Most of them are related to the total area, year built or remodeled, quality of the house, room areas etc. The coefficients that were 'dropped' were mostly related to basement, porch, garage and month or year sold. Ridge, by definition, has all non-zero coefficients, but some of them were very close to zero.

In between the two methods, ridge had overall lowest prediction RMSE because it is essentially using more predictors or features, so it has a little bit better estimation than lasso. On the other hand, lasso is computationally and storage-wise much more efficient as it requires fewer number of predictors (reduces the dimension).

```
# only keep numeric columns
num_cols <- unlist(lapply(data, is.numeric))
data_num <- data[ , num_cols]
x <- data_num[ , -2]
x <- as.matrix(x)
y <- data_num[ , 2]
y <- unlist(y)
# lasso -> alpha = 1
# ridge -> alpha = 0
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.3
```

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

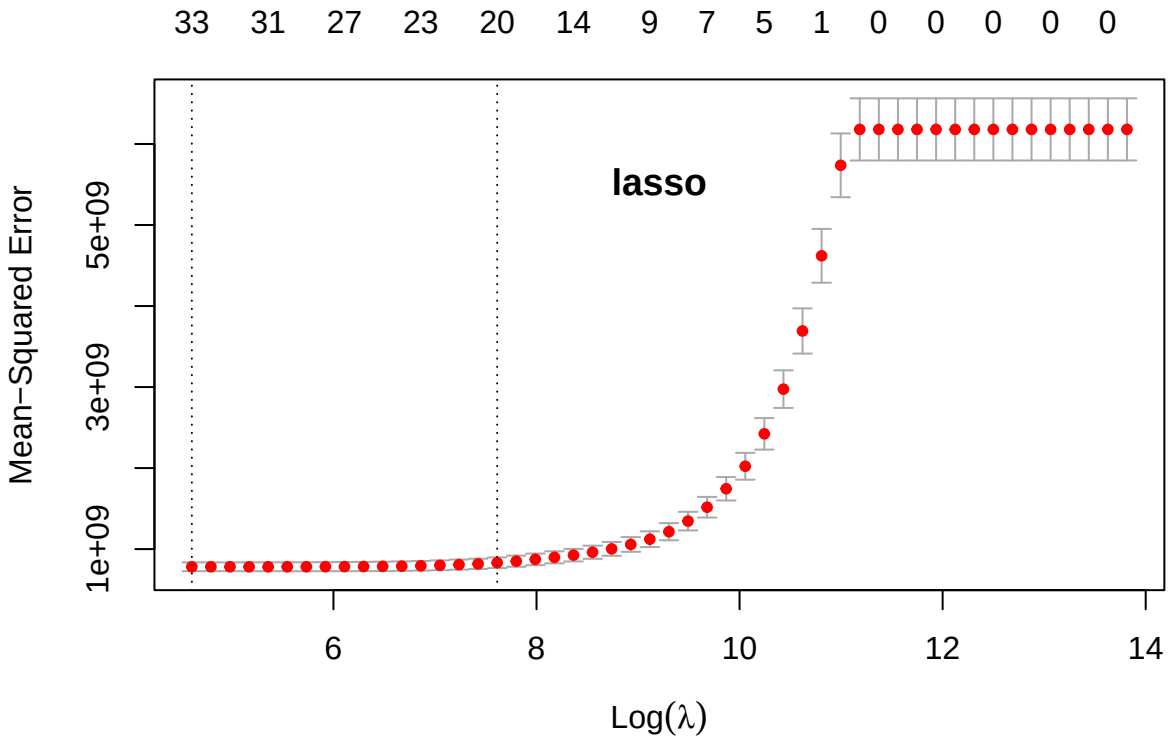
```
## Warning: package 'Matrix' was built under R version 4.1.3
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-4
```

```
# LASSO::::::::::::::::::::::::::::
lambdas_lasso = 10^seq(from = 2, to = 6, length.out = 50)
cv_lasso = cv.glmnet(x, y, nfolds = 10, alpha = 1, lambda = lambdas_lasso, type.measure = c("mse"))
plot(cv_lasso)
title("lasso", line = -3)
```

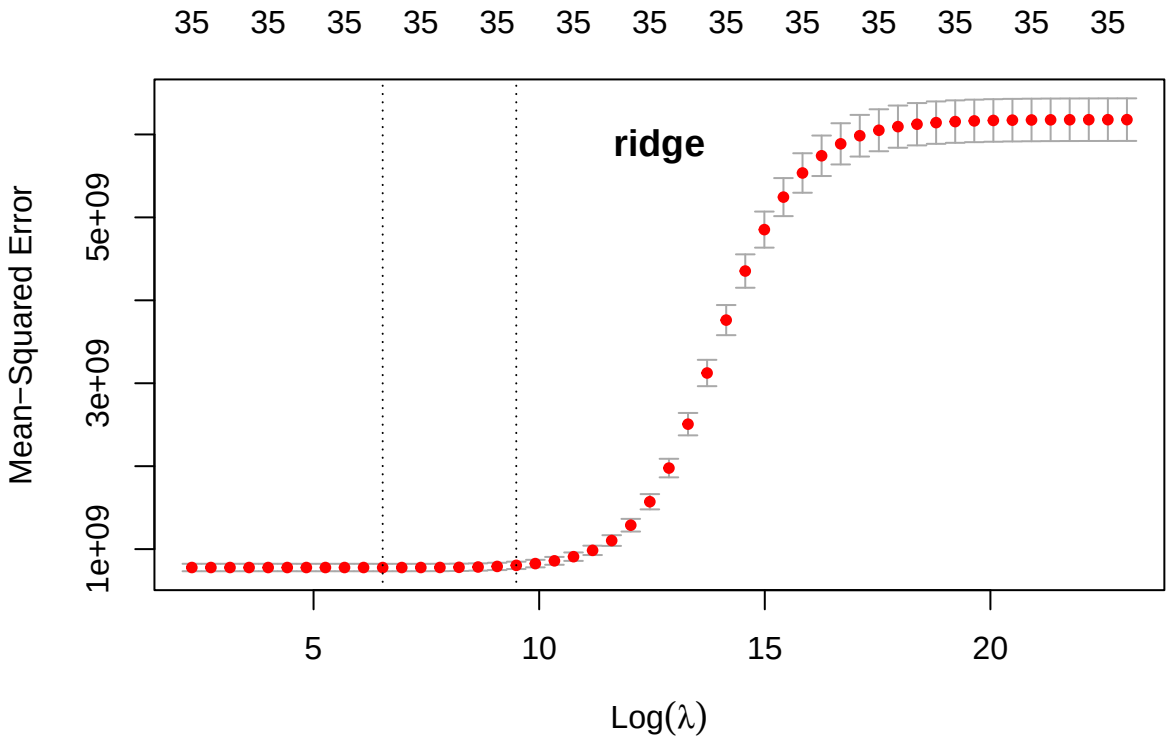



```
lambda_out_lasso = cv_lasso$lambda.min
rmse_best_lasso <- sqrt(min(cv_lasso$cvm))
#sprintf("Best Lambda for lasso: %s , corresponding RMSE: %s ", round(lambda_out_lasso, digits = 2),  r

co_lasso<-coef(cv_lasso,s = "lambda.1se")
inds<-which(co_lasso!=0)
variables_lasso<-row.names(co_lasso)[inds]
variables_lasso<-variables_lasso[!(variables_lasso %in% '(Intercept)')];
#cat("Chosen lasso variables: \n")
#print(variables_lasso)
```

Figure 1: Plot of RMSE across various lambda using lasso shrinkage method

```
# RIDGE::::::::::::::::::::::::::::
lambdas_ridge = 10^seq(from = 1, to = 10, length.out = 50)
cv_ridge = cv.glmnet(x, y, nfolds = 10, alpha = 0, lambda = lambdas_ridge, type.measure = c("mse"))
plot(cv_ridge)
title("ridge", line = -2)
```




```
lambda_out_ridge = cv_ridge$lambda.min
rmse_best_ridge <- sqrt(min(cv_ridge$cvm))
#sprintf("Best Lambda for ridge: %s , corresponding RMSE: %s ", round(lambda_out_ridge, digits = 2),  r
co_ridge<-coef(cv_ridge,s = "lambda.1se")
inds<-which(co_ridge!=0)
variables_ridge<-row.names(co_ridge)[inds]
variables_ridge<-variables_ridge[!(variables_ridge %in% '(Intercept)')];
#cat("Chosen ridge variables: \n")
#print(variables_ridge)
```

Figure 2: Plot of RMSE across various lambda using ridge regression shrinkage method

Principal Components Analysis

For this data set, we carried out a Principal Components Analysis (PCA). PCA can only be performed on quantitative variables, of which there are 35 in the cleaned data set. We split the data set into a train and test set, then ran the `princomp` function on the 35 quantitative predictors and the response. From this function, we are able to get the PC loadings and scores from all 35 principle components. We built 35 regression models each using cumulative PC scores as predictors (ie model 1 uses PC1 only, model 2 uses PC1 and PC2, model 3 uses PC1, PC2, and PC3, etc). With each of these models, we predicted the price for the test set (after calculating the PC scores for the test set using the loadings). The RMSE was computed for each of these models. We repeated these steps on 10 folds for k-fold cross-validation. A plot of the cross-validated RMSE for each of the 35 models is shown in Figure 3. I would recommend 5 PCs because the slope of the RMSE drops off significantly after that. 5 PCs is the best balance of dimensionality reduction and improvement in RMSE. The minimum RMSE achieved was with 32 PCs, however we have only reduced

the dimensionality by three predictors - hardly any improvement. The whole effort of PCA is to reduce dimensionality. If we choose not to reduce the dimensionality by very much, there is little point. Looking at the PCA loadings, we can determine (somewhat) which variables are the most important. We will only focus on the first 2 PCs since these two together explain the most variation (~14%). The first principal component puts the most emphasis on the overall quality, total area of the house, garage size, number of full bathrooms, year built/remodeled, basement and 1st floor square footage. These are all typical factors that home buyers value. The second principal component puts the most emphasis on the second floor square footage, number of bedrooms/rooms above ground, total area, number of half bathrooms, and kitchen size. Interestingly, there are also variables that weigh negatively on this second PC: variables dealing with basement size. So we can think of the 2nd PC as favoring houses without basements. Figure 4 shows a biplot of the different variables and how they relate to the first two principal components. The first PC has many variables associated with it that are not at all associated with the 2nd PC and are therefore too hard to read. But we can clearly see the variables associated with the 2nd PC (and not with the 1st PC) such as X2nd.Flr.SF, Bedroom.AbvGr, and TotRms.AbvGrd.

```
# first remove categorical data
dataPCA <- data[,-c(4,6:16,21:25,27:33,35,39:42,52,54,56,58,61:63,73:74)]

# then normalize the data
dataPCA <- data.frame(scale(dataPCA))

# variables to split data into train and test sets
n <- dim(dataPCA)[1]
k <- 10
n_k <- n/k
indices <- 1:n
split <- sample(indices, n, replace=F)

# function for computing PCA scores
pcaScores <- function(data, loadings, numPCs) {
  scores <- matrix(, nrow=dim(data)[1], ncol=numPCs)
  for(i in 1:numPCs) {
    scores[,i] <- as.matrix(data[]) %*% loadings[,i]
  }
  return(scores)
}

# initialize RMSE variables
# initialize RMSE variables
rmse1 <- 0
rmse2 <- 0
rmse3 <- 0
rmse4 <- 0
rmse5 <- 0
rmse6 <- 0
rmse7 <- 0
rmse8 <- 0
rmse9 <- 0
rmse10 <- 0
rmse11 <- 0
rmse12 <- 0
rmse13 <- 0
rmse14 <- 0
rmse15 <- 0
```

```
rmse16 <- 0
rmse17 <- 0
rmse18 <- 0
rmse19 <- 0
rmse20 <- 0
rmse21 <- 0
rmse22 <- 0
rmse23 <- 0
rmse24 <- 0
rmse25 <- 0
rmse26 <- 0
rmse27 <- 0
rmse28 <- 0
rmse29 <- 0
rmse30 <- 0
rmse31 <- 0
rmse32 <- 0
rmse33 <- 0
rmse34 <- 0
rmse35 <- 0
# cross validation
for(i in 1:k) {
  # train/test split
  train <- dataPCA[-split[(((i-1)*n_k)+1):(i*n_k)],]
  test <- dataPCA[split[(((i-1)*n_k)+1):(i*n_k)],]

  # apply PCA
  PCA <- princomp(~. -price, data=train)

  # calculate PC scores for train set
  train <- train %>% mutate(PC1 = PCA$scores[,1],
                            PC2 = PCA$scores[,2],
                            PC3 = PCA$scores[,3],
                            PC4 = PCA$scores[,4],
                            PC5 = PCA$scores[,5],
                            PC6 = PCA$scores[,6],
                            PC7 = PCA$scores[,7],
                            PC8 = PCA$scores[,8],
                            PC9 = PCA$scores[,9],
                            PC10 = PCA$scores[,10],
                            PC11 = PCA$scores[,11],
                            PC12 = PCA$scores[,12],
                            PC13 = PCA$scores[,13],
                            PC14 = PCA$scores[,14],
                            PC15 = PCA$scores[,15],
                            PC16 = PCA$scores[,16],
                            PC17 = PCA$scores[,17],
                            PC18 = PCA$scores[,18],
                            PC19 = PCA$scores[,19],
                            PC20 = PCA$scores[,20],
                            PC21 = PCA$scores[,21],
                            PC22 = PCA$scores[,22],
                            PC23 = PCA$scores[,23],
```

```
                            PC24 = PCA$scores[,24],
                            PC25 = PCA$scores[,25],
                            PC26 = PCA$scores[,26],
                            PC27 = PCA$scores[,27],
                            PC28 = PCA$scores[,28],
                            PC29 = PCA$scores[,29],
                            PC30 = PCA$scores[,30],
                            PC31 = PCA$scores[,31],
                            PC32 = PCA$scores[,32],
                            PC33 = PCA$scores[,33],
                            PC34 = PCA$scores[,34],
                            PC35 = PCA$scores[,35])

# calculate PC scores and add to test data frame as new variables
testScores <- pcaScores(test[,-2], PCA$loadings, 35)
testNewData <- tibble(PC1 = testScores[,1],
                      PC2 = testScores[,2],
                      PC3 = testScores[,3],
                      PC4 = testScores[,4],
                      PC5 = testScores[,5],
                      PC6 = testScores[,6],
                      PC7 = testScores[,7],
                      PC8 = testScores[,8],
                      PC9 = testScores[,9],
                      PC10 = testScores[,10],
                      PC11 = testScores[,11],
                      PC12 = testScores[,12],
                      PC13 = testScores[,13],
                      PC14 = testScores[,14],
                      PC15 = testScores[,15],
                      PC16 = testScores[,16],
                      PC17 = testScores[,17],
                      PC18 = testScores[,18],
                      PC19 = testScores[,19],
                      PC20 = testScores[,20],
                      PC21 = testScores[,21],
                      PC22 = testScores[,22],
                      PC23 = testScores[,23],
                      PC24 = testScores[,24],
                      PC25 = testScores[,25],
                      PC26 = testScores[,26],
                      PC27 = testScores[,27],
                      PC28 = testScores[,28],
                      PC29 = testScores[,29],
                      PC30 = testScores[,30],
                      PC31 = testScores[,31],
                      PC32 = testScores[,32],
                      PC33 = testScores[,33],
                      PC34 = testScores[,34],
                      PC35 = testScores[,35])

# fit linear models using PCs
mPCA1 <- lm(price~PC1, data=train)
```

```
mPCA2 <- lm(price~PC1+PC2, data=train)
mPCA3 <- lm(price~PC1+PC2+PC3, data=train)
mPCA4 <- lm(price~PC1+PC2+PC3+PC4, data=train)
mPCA5 <- lm(price~PC1+PC2+PC3+PC4+PC5, data=train)
mPCA6 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6, data=train)
mPCA7 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7, data=train)
mPCA8 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8, data=train)
mPCA9 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9, data=train)
mPCA10 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10, data=train)
mPCA11 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11, data=train)
mPCA12 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12, data=train)
mPCA13 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13, data=train)
mPCA14 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14, data=train)
mPCA15 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15, data=train)
mPCA16 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16, data=train
mPCA17 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17, data=
mPCA18 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18, d
mPCA19 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA20 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA21 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA22 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA23 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA24 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA25 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA26 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA27 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA28 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA29 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA30 <- lm(price ~ PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18
mPCA31 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA32 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA33 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA34 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC
mPCA35 <- lm(price~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10+PC11+PC12+PC13+PC14+PC15+PC16+PC17+PC18+PC

# predict on all models
pred1 <- predict(mPCA1, newdata=testNewData[,1])
pred2 <- predict(mPCA2, newdata=testNewData[,c(1:2)])
pred3 <- predict(mPCA3, newdata=testNewData[,c(1:3)])
pred4 <- predict(mPCA4, newdata=testNewData[,c(1:4)])
pred5 <- predict(mPCA5, newdata=testNewData[,c(1:5)])
pred6 <- predict(mPCA6, newdata=testNewData[,c(1:6)])
pred7 <- predict(mPCA7, newdata=testNewData[,c(1:7)])
pred8 <- predict(mPCA8, newdata=testNewData[,c(1:8)])
pred9 <- predict(mPCA9, newdata=testNewData[,c(1:9)])
pred10 <- predict(mPCA10, newdata=testNewData[,c(1:10)])
pred11 <- predict(mPCA11, newdata=testNewData[,c(1:11)])
pred12 <- predict(mPCA12, newdata=testNewData[,c(1:12)])
pred13 <- predict(mPCA13, newdata=testNewData[,c(1:13)])
pred14 <- predict(mPCA14, newdata=testNewData[,c(1:14)])
pred15 <- predict(mPCA15, newdata=testNewData[,c(1:15)])
pred16 <- predict(mPCA16, newdata=testNewData[,c(1:16)])
pred17 <- predict(mPCA17, newdata=testNewData[,c(1:17)])
```

```
pred18 <- predict(mPCA18, newdata=testNewData[,c(1:18)])
pred19 <- predict(mPCA19, newdata=testNewData[,c(1:19)])
pred20 <- predict(mPCA20, newdata=testNewData[,c(1:20)])
pred21 <- predict(mPCA21, newdata=testNewData[,c(1:21)])
pred22 <- predict(mPCA22, newdata=testNewData[,c(1:22)])
pred23 <- predict(mPCA23, newdata=testNewData[,c(1:23)])
pred24 <- predict(mPCA24, newdata=testNewData[,c(1:24)])
pred25 <- predict(mPCA25, newdata=testNewData[,c(1:25)])
pred26 <- predict(mPCA26, newdata=testNewData[,c(1:26)])
pred27 <- predict(mPCA27, newdata=testNewData[,c(1:27)])
pred28 <- predict(mPCA28, newdata=testNewData[,c(1:28)])
pred29 <- predict(mPCA29, newdata=testNewData[,c(1:29)])
pred30 <- predict(mPCA30, newdata=testNewData[,c(1:30)])
pred31 <- predict(mPCA31, newdata=testNewData[,c(1:31)])
pred32 <- predict(mPCA32, newdata=testNewData[,c(1:32)])
pred33 <- predict(mPCA33, newdata=testNewData[,c(1:33)])
pred34 <- predict(mPCA34, newdata=testNewData[,c(1:34)])
pred35 <- predict(mPCA35, newdata=testNewData[,c(1:35)])

# calculate RMSE on all models
rmse1 <- rmse1 + sqrt(sum((pred1-test[,2])^2)/n_k)
rmse2 <- rmse2 + sqrt(sum((pred2-test[,2])^2)/n_k)
rmse3 <- rmse3 + sqrt(sum((pred3-test[,2])^2)/n_k)
rmse4 <- rmse4 + sqrt(sum((pred4-test[,2])^2)/n_k)
rmse5 <- rmse5 + sqrt(sum((pred5-test[,2])^2)/n_k)
rmse6 <- rmse6 + sqrt(sum((pred6-test[,2])^2)/n_k)
rmse7 <- rmse7 + sqrt(sum((pred7-test[,2])^2)/n_k)
rmse8 <- rmse8 + sqrt(sum((pred8-test[,2])^2)/n_k)
rmse9 <- rmse9 + sqrt(sum((pred9-test[,2])^2)/n_k)
rmse10 <- rmse10 + sqrt(sum((pred10-test[,2])^2)/n_k)
rmse11 <- rmse11 + sqrt(sum((pred11-test[,2])^2)/n_k)
rmse12 <- rmse12 + sqrt(sum((pred12-test[,2])^2)/n_k)
rmse13 <- rmse13 + sqrt(sum((pred13-test[,2])^2)/n_k)
rmse14 <- rmse14 + sqrt(sum((pred14-test[,2])^2)/n_k)
rmse15 <- rmse15 + sqrt(sum((pred15-test[,2])^2)/n_k)
rmse16 <- rmse16 + sqrt(sum((pred16-test[,2])^2)/n_k)
rmse17 <- rmse17 + sqrt(sum((pred17-test[,2])^2)/n_k)
rmse18 <- rmse18 + sqrt(sum((pred18-test[,2])^2)/n_k)
rmse19 <- rmse19 + sqrt(sum((pred19-test[,2])^2)/n_k)
rmse20 <- rmse20 + sqrt(sum((pred20-test[,2])^2)/n_k)
rmse21 <- rmse21 + sqrt(sum((pred21-test[,2])^2)/n_k)
rmse22 <- rmse22 + sqrt(sum((pred22-test[,2])^2)/n_k)
rmse23 <- rmse23 + sqrt(sum((pred23-test[,2])^2)/n_k)
rmse24 <- rmse24 + sqrt(sum((pred24-test[,2])^2)/n_k)
rmse25 <- rmse25 + sqrt(sum((pred25-test[,2])^2)/n_k)
rmse26 <- rmse26 + sqrt(sum((pred26-test[,2])^2)/n_k)
rmse27 <- rmse27 + sqrt(sum((pred27-test[,2])^2)/n_k)
rmse28 <- rmse28 + sqrt(sum((pred28-test[,2])^2)/n_k)
rmse29 <- rmse29 + sqrt(sum((pred29-test[,2])^2)/n_k)
rmse30 <- rmse30 + sqrt(sum((pred30-test[,2])^2)/n_k)
rmse31 <- rmse31 + sqrt(sum((pred31-test[,2])^2)/n_k)
rmse32 <- rmse32 + sqrt(sum((pred32-test[,2])^2)/n_k)
rmse33 <- rmse33 + sqrt(sum((pred33-test[,2])^2)/n_k)
```

```
  rmse34 <- rmse34 + sqrt(sum((pred34-test[,2])^2)/n_k)
  rmse35 <- rmse35 + sqrt(sum((pred35-test[,2])^2)/n_k)
}

# average rmse over the kfolds
avg_rmse <- numeric(length=20)
avg_rmse[1] <- rmse1/k
avg_rmse[2] <- rmse2/k
avg_rmse[3] <- rmse3/k
avg_rmse[4] <- rmse4/k
avg_rmse[5] <- rmse5/k
avg_rmse[6] <- rmse6/k
avg_rmse[7] <- rmse7/k
avg_rmse[8] <- rmse8/k
avg_rmse[9] <- rmse9/k
avg_rmse[10] <- rmse10/k
avg_rmse[11] <- rmse11/k
avg_rmse[12] <- rmse12/k
avg_rmse[13] <- rmse13/k
avg_rmse[14] <- rmse14/k
avg_rmse[15] <- rmse15/k
avg_rmse[16] <- rmse16/k
avg_rmse[17] <- rmse17/k
avg_rmse[18] <- rmse18/k
avg_rmse[19] <- rmse19/k
avg_rmse[20] <- rmse20/k
avg_rmse[21] <- rmse21/k
avg_rmse[22] <- rmse22/k
avg_rmse[23] <- rmse23/k
avg_rmse[24] <- rmse24/k
avg_rmse[25] <- rmse25/k
avg_rmse[26] <- rmse26/k
avg_rmse[27] <- rmse27/k
avg_rmse[28] <- rmse28/k
avg_rmse[29] <- rmse29/k
avg_rmse[30] <- rmse30/k
avg_rmse[31] <- rmse31/k
avg_rmse[32] <- rmse32/k
avg_rmse[33] <- rmse33/k
avg_rmse[34] <- rmse34/k
avg_rmse[35] <- rmse35/k

# plot RMSE vs number of PCs
plot(1:35, avg_rmse, type="l", main="PCA Cross Validation", xlab="number of PCs", ylab="Test RMSE")
```

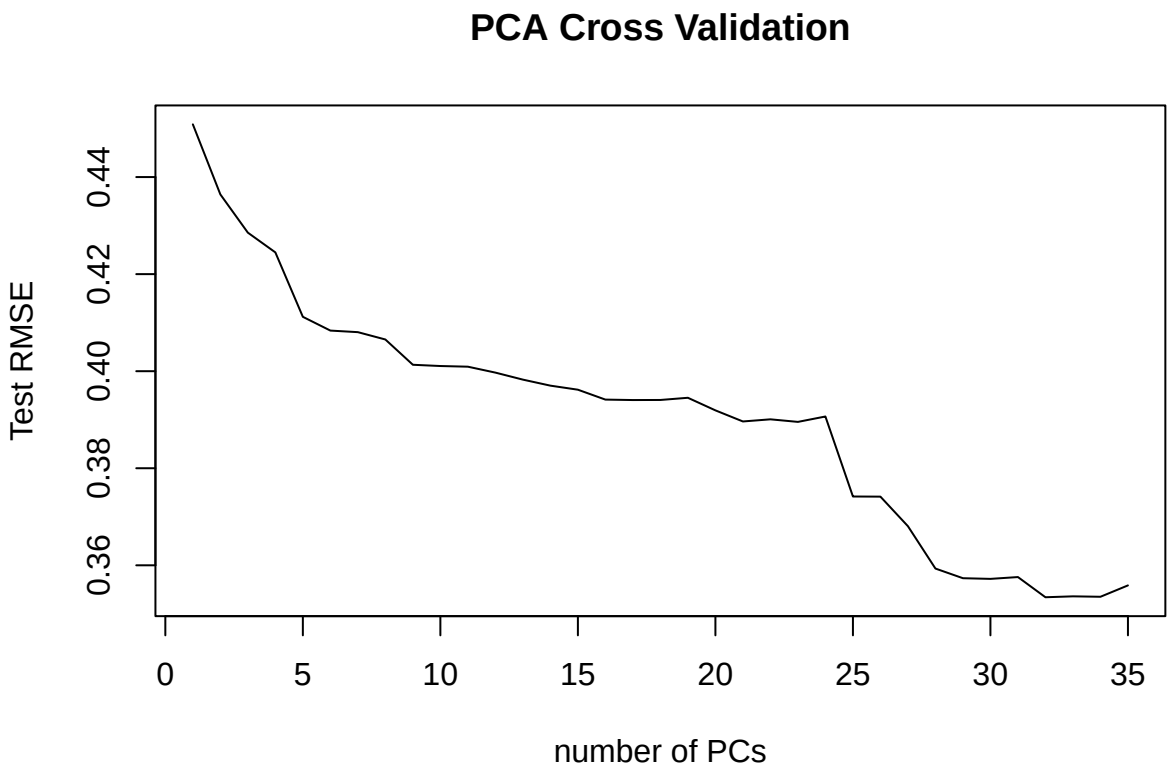


Figure 3: Plot of 10-fold cross-validation of PCA regression.

```
biplot(PCA, col=c("darkgrey","blue"), asp=1)
```

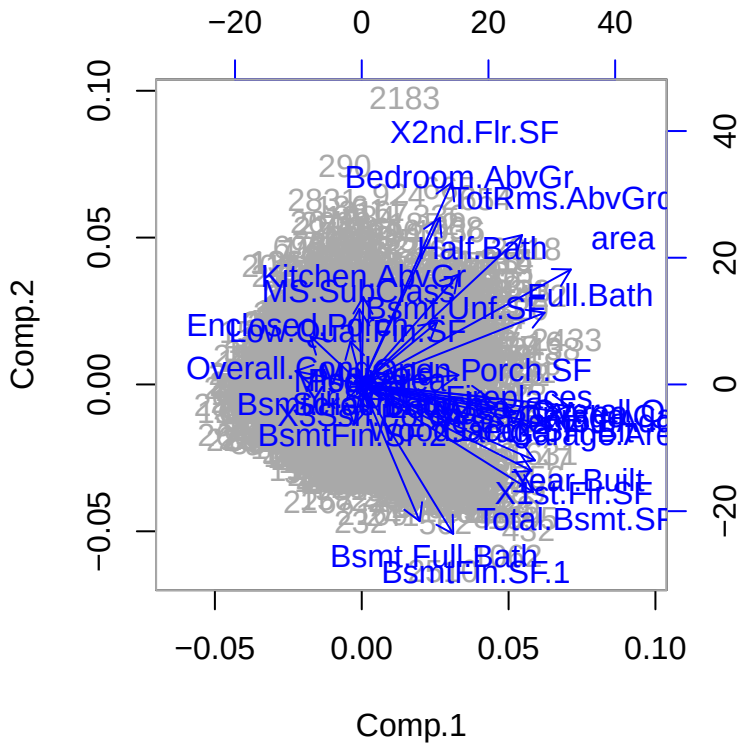


Figure 4: Biplot of first two principal components

Simple Linear Model

A linear model with only year built was fit against price. Ten fold cross validation was performed and as expected, the RMSE was a lot higher than for any of the other linear models fit with more predictors.

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
data<-data[sample(nrow(data)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(data)),breaks=10,labels=FALSE)
error<- 1:10
#Perform 10 fold cross validation
for(i in 1:10){
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==i,arr.ind=TRUE)
```

```
    testData <- data[testIndexes, ]
    trainData <- data[-testIndexes, ]
    simple.model<-lm(price~Year.Built,data=trainData)
    predicted.price<-predict(simple.model,testData)
    error[i]<-rmse(predicted.price,testData$price)
}
#print(mean(error))
plot(testData$Year.Built,testData$price, xlab="Year Built", ylab="Price")
lines(testData$Year.Built,predicted.price,col="red")
legend("topleft",legend=c("linear model prediction","test data"),col=c("red","black"),pch=c('_','o'))
```

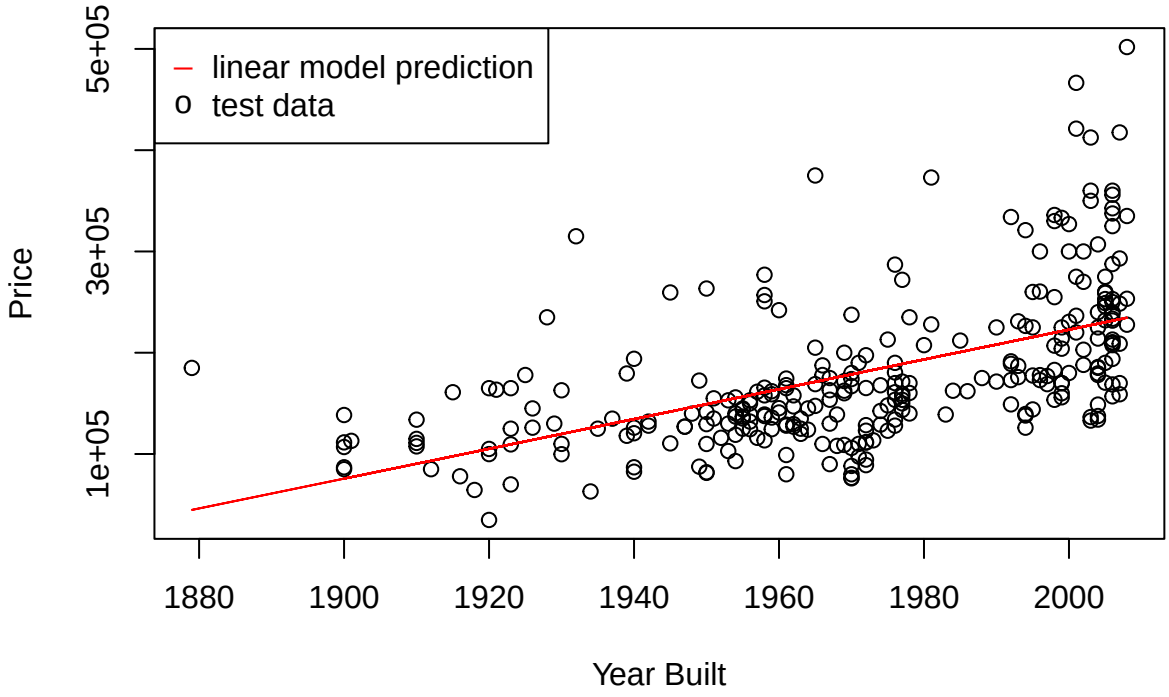


Figure 5: Plot of simple linear regression `price ~ year built`

Polynomial

We used the year the house was built as the only predictor to fit various polynomials to the data set. First we split the data set into a train and test set. On the training set, we built 5 models each with `Year.Built` and adding the next order polynomial of `Year.Built` to subsequent models up to 5th order (ie model 1 `price ~ Year.Built`, model 2 `price ~ Year.Built + Year.Built^2`, and so on). We used these models to predict on the test set and computed the RMSE. We repeated these steps on 10 folds and averaged the RMSE for k-fold cross-validation. I would recommend the 2nd order polynomial because the slope of the RMSE drops off significantly after that. There is minimal improvement in RMSE with higher order polynomials. Also with higher orders, there is the risk of overfitting to the training data. A plot of the cross-validated RMSE for each of the 5 models is shown in Figure 6. The 1st-5th order polynomial fits are shown in Figure 7. The 1st order polynomial (linear) does not capture the curvature of the data. The 2nd order polynomial is able

to capture this curvature well. Beyond the 2nd order there is not much difference, the lines are crowded together. Notice the 4th order polynomial appears to be missing - it is not, the 5th order polynomial predicts the exact same values and therefore plots perfectly on top, covering the 4th order.

```
# split data into train and test sets
n <- dim(data)[1]
k <- 10
n_k <- n/k
indices <- 1:n
split <- sample(indices, n, replace=F)

rmse1 <- 0
rmse2 <- 0
rmse3 <- 0
rmse4 <- 0
rmse5 <- 0
testCols <- c(19) # define the predictors necessary for your model

for(i in 1:k) {
  train <- data[-split[(((i-1)*n_k)+1):(i*n_k)],]
  test <- data[split[(((i-1)*n_k)+1):(i*n_k)],]

  # fit polynomial models up to 5th order
  m1 <- lm(price~Year.Built, data=train)
  m2 <- lm(price~Year.Built+I(Year.Built^2), data=train)
  m3 <- lm(price~Year.Built+I(Year.Built^2)+I(Year.Built^3), data=train)
  m4 <- lm(price~Year.Built+I(Year.Built^2)+I(Year.Built^3)+I(Year.Built^4), data=train)
  m5 <- lm(price~Year.Built+I(Year.Built^2)+I(Year.Built^3)+I(Year.Built^4)+I(Year.Built^5), data=train)

  # add polynomial variables to data frame
  testNewData <- tibble(Year.Built = test$Year.Built,
                        Year.Built2 = test$Year.Built^2,
                        Year.Built3 = test$Year.Built^3,
                        Year.Built4 = test$Year.Built^4,
                        Year.Built5 = test$Year.Built^5)

  # predict with all the models
  pred1 <- predict(m1, newdata=testNewData[,1])
  pred2 <- predict(m2, newdata=testNewData[,c(1:2)])
  pred3 <- predict(m3, newdata=testNewData[,c(1:3)])
  pred4 <- predict(m4, newdata=testNewData[,c(1:4)])
  pred5 <- predict(m5, newdata=testNewData[,c(1:5)])

  # calculate rsme for all the models
  rmse1 <- rmse1 + sqrt(sum((pred1-test[,2])^2)/n_k)
  rmse2 <- rmse2 + sqrt(sum((pred2-test[,2])^2)/n_k)
  rmse3 <- rmse3 + sqrt(sum((pred3-test[,2])^2)/n_k)
  rmse4 <- rmse4 + sqrt(sum((pred4-test[,2])^2)/n_k)
  rmse5 <- rmse5 + sqrt(sum((pred5-test[,2])^2)/n_k)
}
# average rmse over the kfolds
avg_rmse <- numeric(length=5)
avg_rmse[1] <- sum(rmse1)/k
avg_rmse[2] <- sum(rmse2)/k
```

```
avg_rmse[3] <- sum(rmse3)/k
avg_rmse[4] <- sum(rmse4)/k
avg_rmse[5] <- sum(rmse5)/k

# plot RMSE vs number of PCs
plot(1:5, avg_rmse, type="l", main="Polynomial Cross Validation", xlab="order of polynomial", ylab="Tes
```

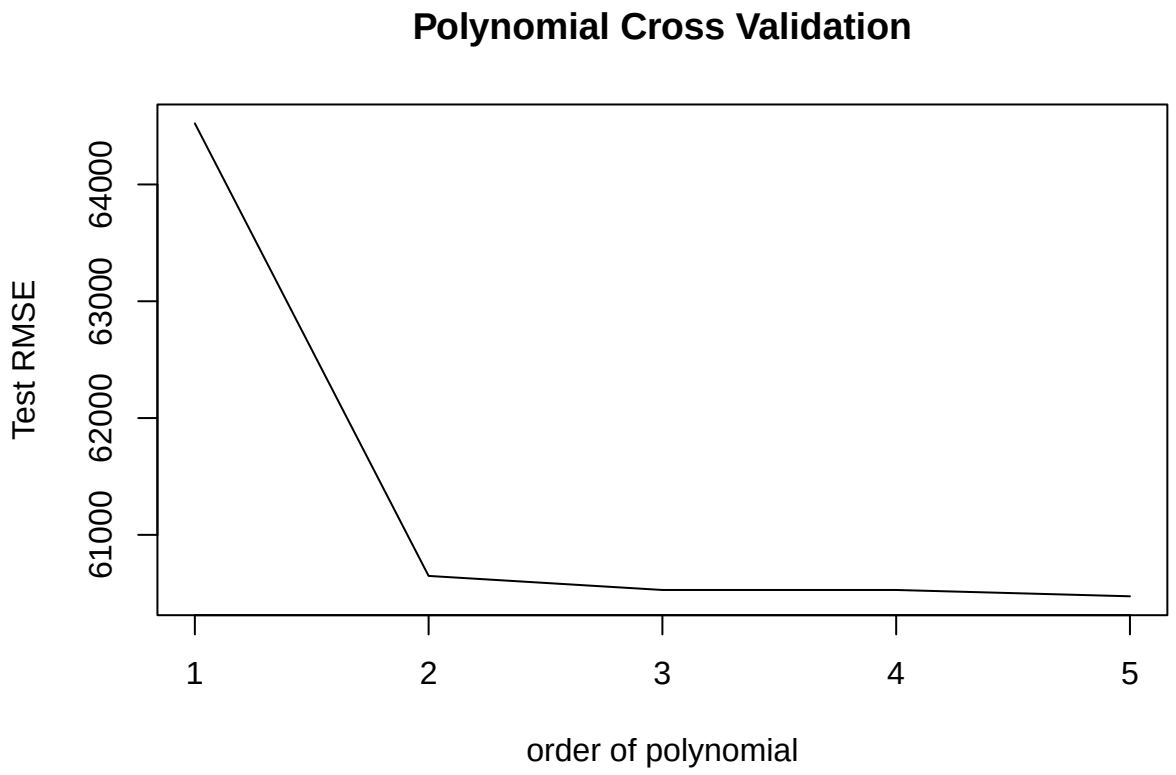


Figure 6: Plot of 10-fold cross-validation of polynomial regression

```
plot(data$Year.Built, data$price, main="Price vs. Year Built fit with various order polynomials", xlab="
x <- seq(1872, 2010, 1)
xNew <- tibble(Year.Built = x,
               Year.Built2 = x^2,
               Year.Built3 = x^3,
               Year.Built4 = x^4,
               Year.Built5 = x^5)
y1 <- predict(m1, newdata=xNew[,1])
y2 <- predict(m2, newdata=xNew[,1:2])
y3 <- predict(m3, newdata=xNew[,1:3])
y4 <- predict(m4, newdata=xNew[,1:4])
y5 <- predict(m5, newdata=xNew[,1:5])
lines(x, y1, col="red")
lines(x, y2, col="blue")
lines(x, y3, col="green")
lines(x, y4, col="magenta")
lines(x, y5, col="orange")
```

```
legend("topleft",
       title="polynomial order",
       legend=c("1", "2", "3", "4", "5"),
       col=c("red", "blue", "green", "magenta", "orange"),
       pch=15)
```

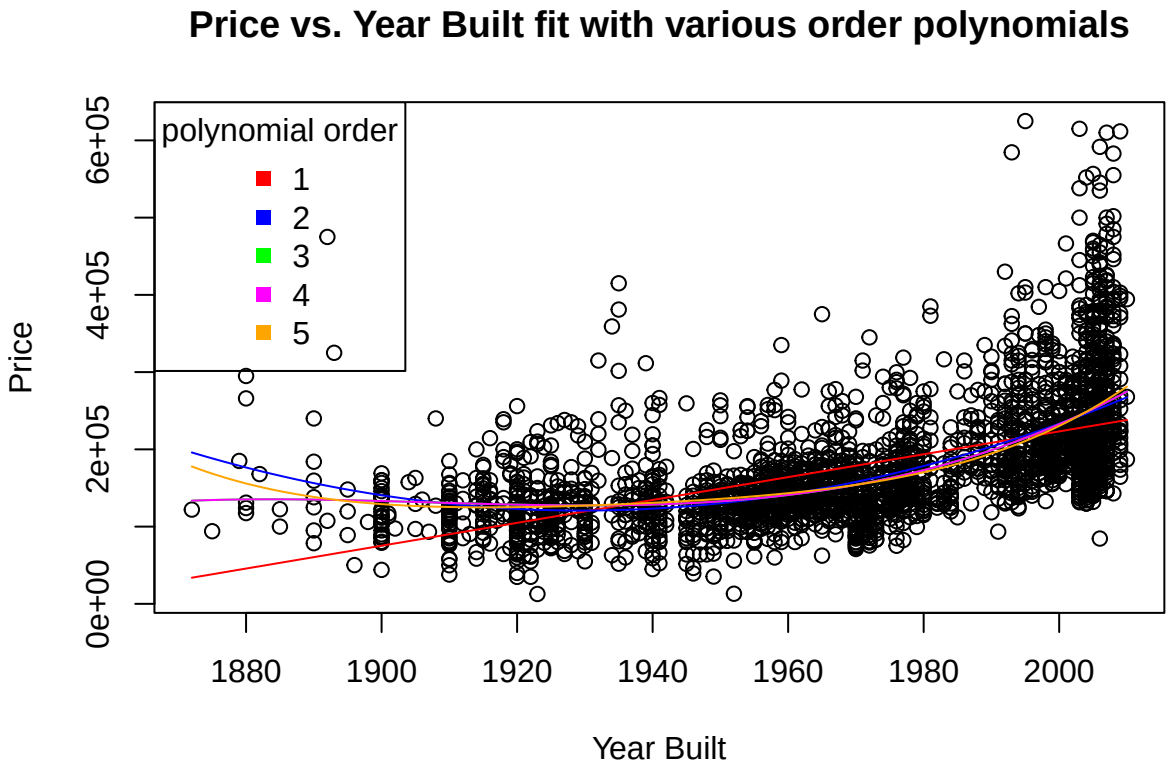


Figure 7: Plot of price vs year built and fit of 1st-5th order polynomials

Smoothing Spline

Splines are a type of interpolation or function estimation techniques. Based on a set of noisy observations, it fits the data into piece-wise polynomial curves called basis functions, in such a way that minimizes the total error between the observed data and the estimated data points. Similar to lasso and ridge regularization, the smoothing spline approach also has a tuning parameter λ that helps penalize the variability in the basis function. λ effectively controls the degree of freedom and hence the number of knots. A larger value of λ forces the function to be smoother, but at the cost of underfitting, whereas $\lambda \to 0$ causes overfitting. Therefore, we tried 50 different values of λ within a logarithmic range from 10^{-10} to 10^{4}.

In this project, instead of leave-one-out cross-validation (LOOCV), we used the 10-fold Cross validation to be consistent with the other approaches. A plot of the cross-validated prediction RMSE for each value of λ is shown in Figure . The lowest prediction RMSE was obtained corresponding to $\lambda = 5.18 \times 10^{-5}$. Figure shows the best fit smoothing spline of year built vs price. It seems that after 1940, the spline follows the trend of the scatter plot. But before that, the data are significantly sparse so there are some variations in the curve. Intuitively, the total number of houses were much less (or we have access to very small amount of data for the years before 1940), and there were some exceptionally high-priced houses during that time.

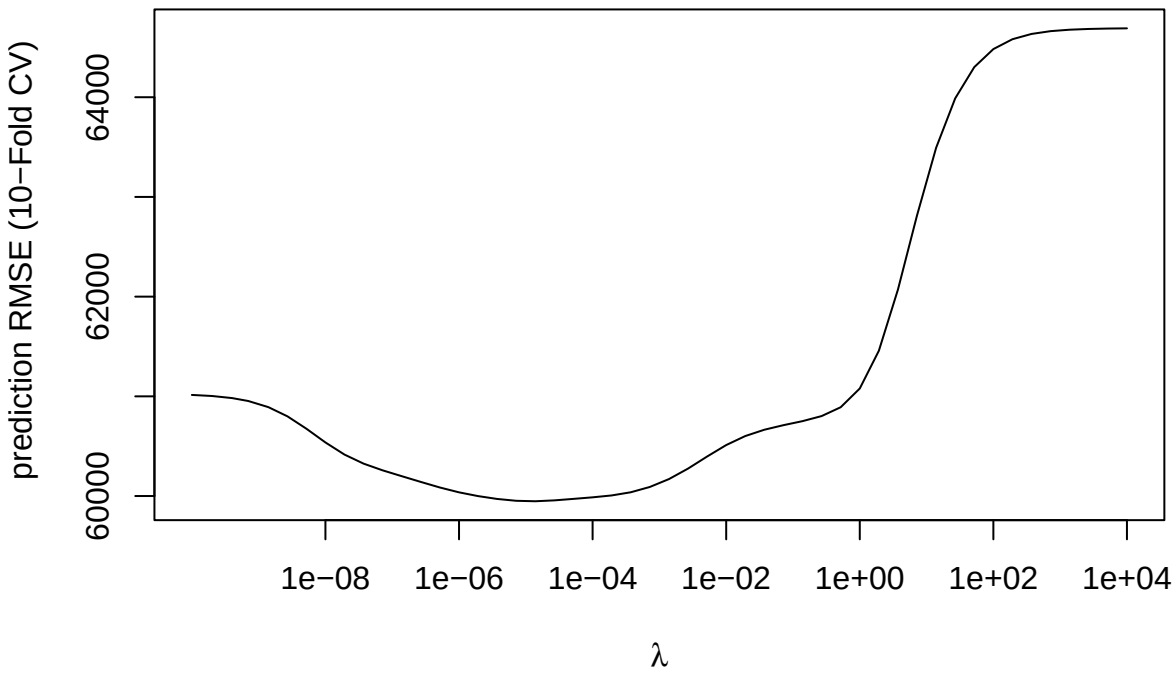
```r
set.seed(125)
# make dataframe with only year built
newdata = data.frame(data$price)
newdata = cbind(newdata, data$Year.Built)
colnames(newdata) = c("price","Year.Built")
# create partitions for CV
n = nrow(newdata)
nFolds = 10
CVfolds = sample( rep( 1:nFolds , length=n ) , n )
x = newdata$Year.Built
y = newdata$price
# initiate final RMSE matrices
RMSE_train = matrix(0, 1,nFolds)
RMSE_test = matrix(0, 1,nFolds)
# define grid-search range for lambda
lambdas = 10^seq(from = -10, to = 4, length.out = 50)
lambdas = round(lambdas, digits = 10)
RMSE_array = matrix(0, 1, length(lambdas)) # create an array to store the prediction RMSE
RMSE_array_index = 1
for(l in lambdas) # l = value of lambda
{

  for( fold in 1:nFolds )
  {
    xtrain = x[ CVfolds!=fold ] # Train is everything that isn't the fold itself
    ytrain = y[ CVfolds!=fold ]
    xtest = x[ CVfolds==fold ] # Test is  the current fold (k) being used
    ytest = y[ CVfolds==fold ]

    # train
    smth_spl_mdl = smooth.spline(xtrain , ytrain, lambda = l)
    # plot(smth_spl_mdl, xlab = "Year.Built", ylab = "price") #plots the splines themselves but not nee
    # RMSE_train[fold] = sqrt(mean(( ytrain - smth_spl_mdl$y )^2)) # stores the training RMSE but not n

    # test/prediction
    pred = predict(smth_spl_mdl, xtest)
    RMSE_test[fold] = sqrt(mean(( ytest - pred$y )^2))
  }
  RMSE_array[RMSE_array_index] = mean(RMSE_test)
  RMSE_array_index = RMSE_array_index + 1

}
# plot RMSE_array w.r.t. lambda
plot(lambdas, RMSE_array, xlab = expression(lambda) , ylab = " prediction RMSE (10-Fold CV)", log = 'x'
```




```
best_lambda_ss = lambdas[which(RMSE_array==min(RMSE_array))]
#print(best_lambda_ss)
```

Figure 8: Plot of 10-fold cross-validation of smoothing spline

```
# plot best smooth spline model
best_smth_spl_mdl = smooth.spline(x , y, lambda = lambdas[which(RMSE_array==min(RMSE_array))])
plot(xtrain, ytrain, col="black")
lines(best_smth_spl_mdl, col="red",lwd=2)
```

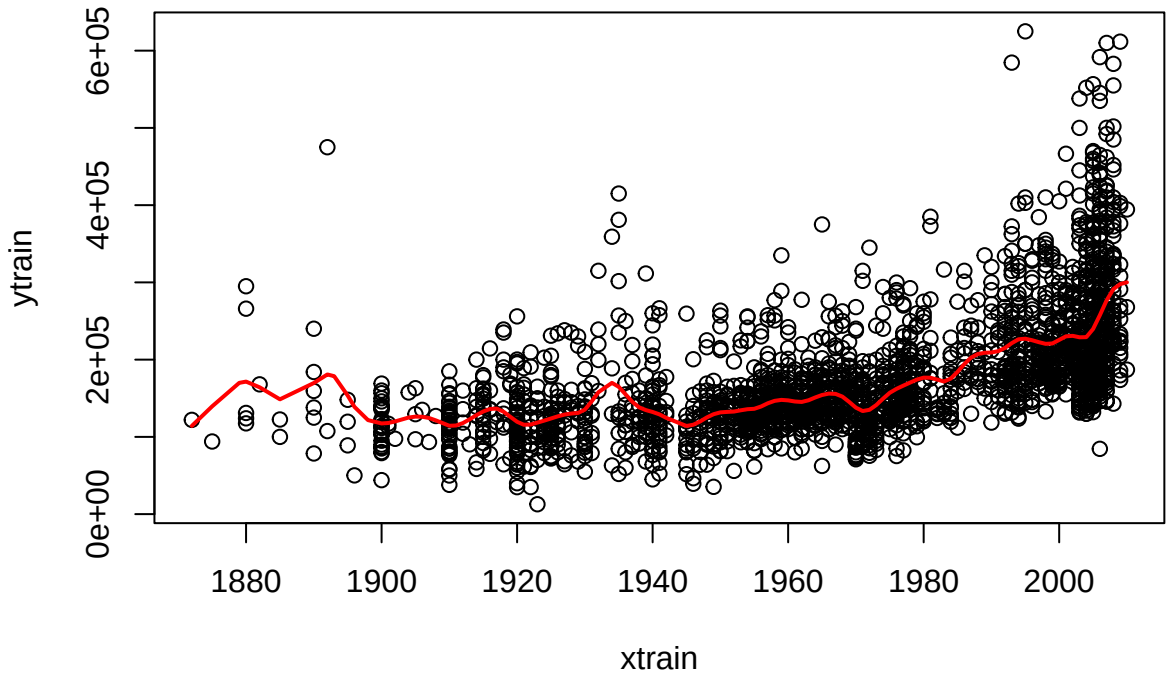


Figure 9: Plot of best smoothing spline model

References

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York :Springer, 2013.