House Prices

Jason Smith

January 24, 2020

Table of Contents

# Load the data set

train\_raw = read.csv("train.csv")  
test\_raw = read.csv("test.csv")

# Summary of Missing Data

dim(train\_raw)

## [1] 1460 81

missing\_per = sapply(train\_raw, function(x) round(sum(is.na(x))/1460, 2) )  
missing\_count = sapply(train\_raw, function(x) sum(is.na(x)) )  
  
missing\_per[order(missing\_per, decreasing = T)]

## PoolQC MiscFeature Alley Fence FireplaceQu   
## 1.00 0.96 0.94 0.81 0.47   
## LotFrontage GarageType GarageYrBlt GarageFinish GarageQual   
## 0.18 0.06 0.06 0.06 0.06   
## GarageCond BsmtQual BsmtCond BsmtExposure BsmtFinType1   
## 0.06 0.03 0.03 0.03 0.03   
## BsmtFinType2 MasVnrType MasVnrArea Id MSSubClass   
## 0.03 0.01 0.01 0.00 0.00   
## MSZoning LotArea Street LotShape LandContour   
## 0.00 0.00 0.00 0.00 0.00   
## Utilities LotConfig LandSlope Neighborhood Condition1   
## 0.00 0.00 0.00 0.00 0.00   
## Condition2 BldgType HouseStyle OverallQual OverallCond   
## 0.00 0.00 0.00 0.00 0.00   
## YearBuilt YearRemodAdd RoofStyle RoofMatl Exterior1st   
## 0.00 0.00 0.00 0.00 0.00   
## Exterior2nd ExterQual ExterCond Foundation BsmtFinSF1   
## 0.00 0.00 0.00 0.00 0.00   
## BsmtFinSF2 BsmtUnfSF TotalBsmtSF Heating HeatingQC   
## 0.00 0.00 0.00 0.00 0.00   
## CentralAir Electrical X1stFlrSF X2ndFlrSF LowQualFinSF   
## 0.00 0.00 0.00 0.00 0.00   
## GrLivArea BsmtFullBath BsmtHalfBath FullBath HalfBath   
## 0.00 0.00 0.00 0.00 0.00   
## BedroomAbvGr KitchenAbvGr KitchenQual TotRmsAbvGrd Functional   
## 0.00 0.00 0.00 0.00 0.00   
## Fireplaces GarageCars GarageArea PavedDrive WoodDeckSF   
## 0.00 0.00 0.00 0.00 0.00   
## OpenPorchSF EnclosedPorch X3SsnPorch ScreenPorch PoolArea   
## 0.00 0.00 0.00 0.00 0.00   
## MiscVal MoSold YrSold SaleType SaleCondition   
## 0.00 0.00 0.00 0.00 0.00   
## SalePrice   
## 0.00

#summary(train\_raw)

missing\_per\_test = sapply(test\_raw, function(x) round(sum(is.na(x))/1460, 2) )  
missing\_per\_test[order(missing\_per\_test, decreasing = T)]

## PoolQC MiscFeature Alley Fence FireplaceQu   
## 1.00 0.96 0.93 0.80 0.50   
## LotFrontage GarageType GarageYrBlt GarageFinish GarageQual   
## 0.16 0.05 0.05 0.05 0.05   
## GarageCond BsmtQual BsmtCond BsmtExposure BsmtFinType1   
## 0.05 0.03 0.03 0.03 0.03   
## BsmtFinType2 MasVnrType MasVnrArea Id MSSubClass   
## 0.03 0.01 0.01 0.00 0.00   
## MSZoning LotArea Street LotShape LandContour   
## 0.00 0.00 0.00 0.00 0.00   
## Utilities LotConfig LandSlope Neighborhood Condition1   
## 0.00 0.00 0.00 0.00 0.00   
## Condition2 BldgType HouseStyle OverallQual OverallCond   
## 0.00 0.00 0.00 0.00 0.00   
## YearBuilt YearRemodAdd RoofStyle RoofMatl Exterior1st   
## 0.00 0.00 0.00 0.00 0.00   
## Exterior2nd ExterQual ExterCond Foundation BsmtFinSF1   
## 0.00 0.00 0.00 0.00 0.00   
## BsmtFinSF2 BsmtUnfSF TotalBsmtSF Heating HeatingQC   
## 0.00 0.00 0.00 0.00 0.00   
## CentralAir Electrical X1stFlrSF X2ndFlrSF LowQualFinSF   
## 0.00 0.00 0.00 0.00 0.00   
## GrLivArea BsmtFullBath BsmtHalfBath FullBath HalfBath   
## 0.00 0.00 0.00 0.00 0.00   
## BedroomAbvGr KitchenAbvGr KitchenQual TotRmsAbvGrd Functional   
## 0.00 0.00 0.00 0.00 0.00   
## Fireplaces GarageCars GarageArea PavedDrive WoodDeckSF   
## 0.00 0.00 0.00 0.00 0.00   
## OpenPorchSF EnclosedPorch X3SsnPorch ScreenPorch PoolArea   
## 0.00 0.00 0.00 0.00 0.00   
## MiscVal MoSold YrSold SaleType SaleCondition   
## 0.00 0.00 0.00 0.00 0.00

# Data Cleaning

## Step 1: Remove variables with more than 10% missing data

## Step 2: Use Random Forest Imputation on training and test sets for variables less than 10%

library(missForest)

## Loading required package: randomForest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

## Loading required package: foreach

## Loading required package: itertools

## Loading required package: iterators

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':  
##   
## combine

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

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

# # Comment out the code so do not run this multiple times. Uncomment to run in need to.  
#   
# set.seed(2020)  
#   
# # Data processing for training set.  
# data\_raw\_train = train\_raw  
# data\_raw2\_train = select(data\_raw\_train, -PoolQC, -MiscFeature,  
# -Alley, -Fence, -FireplaceQu,  
# -LotFrontage, -SalePrice)  
# data\_imp\_train = missForest(data\_raw2\_train)  
# training = data\_imp\_train$ximp  
#   
# set.seed(2020)  
#   
# # Data processing for test set.  
# data\_raw\_test = test\_raw  
# data\_raw2\_test = select(data\_raw\_test, -PoolQC, -MiscFeature,  
# -Alley, -Fence, -FireplaceQu,  
# -LotFrontage)  
# data\_imp\_test = missForest(data\_raw2\_test)  
# test = data\_imp\_test$ximp  
#   
#   
# # Export the imputed training and test data sets so we can read them in directly  
# # and not have to redo the random forest imputation every we run the code.  
#   
# # Uncomment to Export the data.  
# write.csv(training, "training\_imputed.csv", row.names = F)  
# write.csv(test, "test\_imputed.csv", row.names = F)

# Read in the data from here.   
  
training = read.csv("training\_imputed.csv")  
test = read.csv("test\_imputed.csv")

# Function to Seperate training data set into train and validation sets.  
  
# The splitting function  
split.data = function(data, train.prop, set.seed=NA){  
   
 if(!is.na(set.seed)){set.seed(set.seed)}  
   
 train.idx = sample(1:dim(data)[1], round(dim(data)[1]\*train.prop), replace = F)  
 test.idx = setdiff(1:dim(data)[1], train.idx)  
   
 train.set = data[train.idx,]  
 test.set = data[test.idx,]  
   
 return(list(train=train.set, test= test.set))  
   
}

# Separating the data into training and validation sets

# Choosing the proportion to Separate into training and validation sets. We used 50% and 70%.   
  
train.prop = 0.7  
  
# Reattach the y variable SalePrice back to the dataset.   
  
training$SalePrice = train\_raw$SalePrice  
  
# Spliting the dataset.   
  
data\_slipt = split.data(data = training, train.prop = train.prop, set.seed=2020)  
training\_set = data\_slipt$train; validation\_set = data\_slipt$test  
  
# Creating the data sets with only numeric variables.   
  
#Training dataset  
num\_col = sapply(training\_set, is.numeric)  
training\_set\_num = training\_set[, num\_col][,-1]  
dim(training\_set\_num)

## [1] 1022 36

#validation dataset  
num\_col = sapply(validation\_set, is.numeric)  
validation\_set\_num = validation\_set[, num\_col][,-1]  
dim(validation\_set\_num)

## [1] 438 36

# Fitting Elastic-Net Regularized General Linear Models

require(glmnet)

## Loading required package: glmnet

## Loading required package: Matrix

## Loaded glmnet 3.0-2

# Scale the data and Create the Data matrix X  
  
train.data <- model.matrix(SalePrice~.,as.data.frame(scale(training\_set\_num)))  
test.data <- model.matrix(SalePrice~.,as.data.frame(scale(validation\_set\_num)))  
labels.train <- training\_set\_num$SalePrice  
labels.test <- validation\_set\_num$SalePrice   
  
x = train.data   
y <- labels.train  
  
  
# Fitting the glm.   
rr.mod <- glmnet(x,y,family="gaussian",alpha=1)   
  
# We perform cross-validation.  
cv.rr <- cv.glmnet(x,y,family="gaussian",alpha=1)   
  
# THis is the smallest value of lambda.   
lambda = cv.rr$lambda.min  
lambda

## [1] 1249.371

# If choose this lambda then we can get the estimates.   
coef.min <- coef(cv.rr, s = "lambda.min") # Here s is lambda. Tells to use minmum labmbda from cv.rr.   
  
# Below we print to see the glmnet estimated coefficient values.  
coef.min

## 37 x 1 sparse Matrix of class "dgCMatrix"  
## 1  
## (Intercept) 179108.9716  
## (Intercept) .   
## MSSubClass -6494.1279  
## LotArea 1966.9996  
## OverallQual 26001.5836  
## OverallCond 2338.2851  
## YearBuilt 7096.8407  
## YearRemodAdd 4817.4673  
## MasVnrArea 4847.9276  
## BsmtFinSF1 1761.1336  
## BsmtFinSF2 .   
## BsmtUnfSF .   
## TotalBsmtSF 3426.5582  
## X1stFlrSF 206.4688  
## X2ndFlrSF .   
## LowQualFinSF .   
## GrLivArea 25905.7134  
## BsmtFullBath 4657.7432  
## BsmtHalfBath .   
## FullBath 1856.7212  
## HalfBath .   
## BedroomAbvGr -1882.3642  
## KitchenAbvGr -1086.0826  
## TotRmsAbvGrd .   
## Fireplaces 3177.5727  
## GarageYrBlt .   
## GarageCars 8911.0613  
## GarageArea .   
## WoodDeckSF 3564.5557  
## OpenPorchSF .   
## EnclosedPorch .   
## X3SsnPorch .   
## ScreenPorch 2245.6198  
## PoolArea .   
## MiscVal .   
## MoSold .   
## YrSold .

# We will get the predicted house sale prices for the test data using the fitted glmnet model.   
  
predictions1 = as.numeric(predict(rr.mod, newx=test.data, s= cv.rr$lambda.min))  
  
  
# Fit the classical regression models with all the variables, then using only numeric variables  
# and lastly no variables which is the null model and is simply the average house sale price. Our models   
# should at least perform better than the null model.   
  
model\_regularReg\_full = lm(SalePrice ~., data = training\_set[,-1]) # Regression: All the variables  
model\_regularReg\_num = lm(SalePrice ~ ., data = training\_set\_num) # Regression: Numerical variables  
model\_null = lm(SalePrice ~ 1, data = training\_set\_num) # Null model: calculates the average house price  
  
# Function to get rmse   
  
rmse\_fun = function(model, testdata) {  
   
 # Test   
 #model = model\_regularReg\_full  
 #testdata = validation\_set  
   
 test\_labels = as.numeric(testdata[,"SalePrice"])  
 pred = predict(model, newx=select(testdata, -SalePrice))  
 rmse = sqrt(mean((test\_labels - pred)^2))  
 return(rmse)  
   
}  
  
  
# Get the rmse for the different regression models.   
  
rmse\_regularReg\_null = rmse\_fun(model\_null, validation\_set\_num)

## Warning in test\_labels - pred: longer object length is not a multiple of shorter  
## object length

rmse\_regularReg\_full = rmse\_fun(model\_regularReg\_full, validation\_set)

## Warning in test\_labels - pred: longer object length is not a multiple of shorter  
## object length

rmse\_regularReg\_num = rmse\_fun(model\_regularReg\_num, validation\_set\_num)

## Warning in test\_labels - pred: longer object length is not a multiple of shorter  
## object length

rmse\_glmnet = sqrt(mean((predictions1 - labels.test)^2))  
  
# Just get the raw rmse values for the different models.   
paste0("rmse\_regularReg\_null: ", round(rmse\_regularReg\_null, 2), ", rmse\_glmnet: ", round(rmse\_glmnet, 2), ", rmse\_regularReg\_num: ", round(rmse\_regularReg\_num, 2), ", rmse\_regularReg\_full: ", round(rmse\_regularReg\_full, 2)) # rmse\_glmnet is the lowest

## [1] "rmse\_regularReg\_null: 80447.43, rmse\_glmnet: 32522.54, rmse\_regularReg\_num: 108239.07, rmse\_regularReg\_full: 112357.21"

# Here we divide the rmse's by the rmse of the null model to clearly see which models perform better  
# than the null model.   
  
c(rmse\_regularReg\_null, rmse\_glmnet, rmse\_regularReg\_num, rmse\_regularReg\_full)/rmse\_regularReg\_null

## [1] 1.0000000 0.4042707 1.3454633 1.3966538

# Fitting Partial Least Square Regression.

# DO THE PARTIAL LEAST SQUARES REGRESSIONS.   
  
library(pls)

##   
## Attaching package: 'pls'

## The following object is masked from 'package:stats':  
##   
## loadings

n.score.vec <- c(5, 6, 7, 8, 10, 11, 13, 16) # of score vecotrs to try  
  
pdf(paste0('pls\_mse\_plots\_train\_perc70.pdf'), height=15, width=10)  
par(mfrow=c(2,1))  
  
for (score.idx in 1:length(n.score.vec)) {  
 n.comp = n.score.vec[score.idx]  
 print(n.comp)  
   
 #Use plsr built in cross validation  
 pls.fit <- plsr(SalePrice ~ ., data=training\_set\_num, ncomp = n.comp,  
 validation="CV",scale=T)  
 plot(MSEP(pls.fit))  
   
}

## [1] 5

## [1] 6

## [1] 7

## [1] 8

## [1] 10

## [1] 11

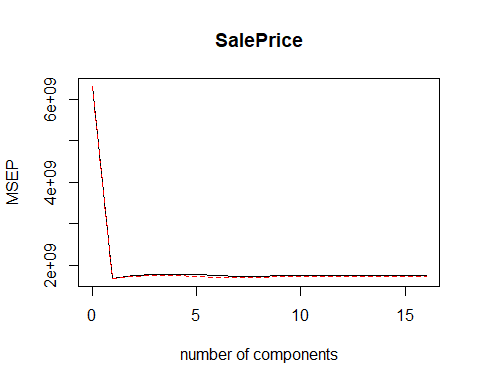
## [1] 13

## [1] 16

dev.off()

## png   
## 2

#Plot the MSE  
plot(MSEP(pls.fit))



#Save the plot of the MSE  
pdf(paste0('pls\_mse\_comp16\_plots\_train\_perc70.pdf'))  
plot(MSEP(pls.fit))  
dev.off()

## png   
## 2

# Save the cross-validation MSE corresponding to the number of componenets.   
# The number of components that is greater than 2 and that produces the lowest MSE will  
# be chosen as the optimum number of components to be used for the partial least square regression.  
  
sink("pls16comp\_summary\_train\_perc70.txt")  
print(summary(pls.fit))

## Data: X dimension: 1022 35   
## Y dimension: 1022 1  
## Fit method: kernelpls  
## Number of components considered: 16  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 79468 41009 41721 42066 42115 42015 41696  
## adjCV 79468 40978 41507 41778 41775 41673 41378  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 41609 41675 41735 41760 41751 41746 41739  
## adjCV 41297 41355 41407 41429 41421 41417 41411  
## 14 comps 15 comps 16 comps  
## CV 41740 41738 41738  
## adjCV 41411 41410 41409  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 20.17 24.44 30.18 34.60 40.37 44.40 47.38  
## SalePrice 74.52 77.69 78.53 79.16 79.38 79.47 79.52  
## 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps 14 comps  
## X 50.71 53.21 55.21 57.23 59.30 61.18 63.21  
## SalePrice 79.56 79.57 79.58 79.58 79.58 79.58 79.58  
## 15 comps 16 comps  
## X 64.61 66.34  
## SalePrice 79.58 79.58  
## NULL

sink()  
  
  
# Using the above training results we choose the optimum number of components below as opt.comp to be used for the final pls model.   
  
# We found for 50% and 70% of data used for training that the optimum number of components are opt.comp = 4 and opt.comp = 7 respectively. As it has the smallest MSE after 2 components, and we do not want number of components to be less than 3.   
  
######################  
  
#We will now set opt.comp = 7.   
  
opt.comp = 7  
  
# We will fit the plsr using ncomp = opt.comp.   
  
plsr.opt.te = plsr(SalePrice~., data=validation\_set\_num, ncomp = opt.comp, validation="none",scale=T)  
  
# Extract the estimated Sale Prices from the fitted plsr model.   
predict\_plsr = plsr.opt.te$fitted.values   
  
# Calculate the RMSE for the plsr.   
rmse\_plsr\_opt.comp\_7\_train\_per70 = sqrt(mean((labels.test - predict\_plsr)^2))  
rmse\_plsr\_opt.comp\_7\_train\_per70

## [1] 29449.38

#rmse\_plsr\_opt.comp\_4\_train\_per50  
  
# We can see that rmse\_plsr\_opt.comp\_4\_train\_per50 = 28396.45 and rmse\_plsr\_opt.comp\_7\_train\_per70 27295.49. Hence, 7 components is better to be used.

# Fitting the Random Forest Regression.

# Do the Random Forest Regression.   
  
require(randomForest)  
require(dplyr)  
  
# Fit the random forest regression using the default values of the hyper parameters.   
rf = randomForest(SalePrice ~., data = training\_set )  
  
# Get the predicted house sale prices using the fitted random forest.   
pred\_rf = predict(rf, newdata = select(validation\_set, -SalePrice))   
  
# Obtain the rmse for the random forst model.   
rmse\_rf\_train\_perc70 = sqrt(mean((labels.test - pred\_rf)^2))  
  
rmse\_rf\_train\_perc70

## [1] 26955.31

#rmse\_rf\_train\_perc50  
  
# We find that the rmse for random forest rmse for 50% is 25547.81  
# and that the random forest rmse for 70% is 24155.2