Ames house price prediction using lasso and ridge regression models

MSDS6372 Applied Statistics Project 1 Team 1 Part 1

June, 2019

## **Introduction**

One of the biggest pitfalls for building prediction models is the risk of overfitting the training dataset, resulting in its poor prediction accuracy. The reason for overfitting is that with increasing number of predictors, the model not only fits the signal, but also tried to fit the noise in the training dataset. Mainly, there are two ways to prevent overfitting, validation/cross-validation and regularization. The idea of validation is very straight forward. Before fitting a model, a small portion of training data is randomly setted aside and used for evaluating the accuracy of the model trained on the rest of data. Any model with low training error, but has very high testing error is a typical overfitted model. In general, complicated models with many predictors are at higher risk of becoming overfitting. Thus, the key to prevent overfitting is to keep the model simple (less predictors) and make the weights (coefficients) of predictors small. One popular way to do that is by including a regularization term in the model. Two shrinkage approaches, ridge and lasso, are the most popular regularization methods.

In the ridge model, a L2 regularization term (also called L2 norm or ridge penalty), is introduced into the model to keep the coefficients from getting too large. Similarly, for the lasso model, a L1 regularization term (also called L1 norm or Lasso penalty), is included in the model to force some coefficients to zero. Both models achieve at least two objectives, one is to keep the weights of predictors from getting too large, thus lowering the variation of prediction. The other is that both models automatically perform feature selection by forcing the weights of less important predictors close or equal to zero. In order to choose the proper amount of penalty terms for ridge and lasso models, cross validation are used to find the best tuning parameter that results in the least amount of cross validation error.

## **Data Description**

The Ames house price dataset on Kaggle (<https://www.kaggle.com/c/house-prices-advanced-regression-techniques>) is one of the most popular dataset for regression models. It is an updated and modern version of the traditional Boston Housing dataset. The training and testing (for competition) datasets each have data on about 1,500 houses sold from 2006 to 2010 in Ames, IA. Besides the response variable, SalePrice, the datasets contain about 80 predictive variables associated with the price of a typical house, such as its location (Neighborhood), size (Gross living area), quality and condition (OverallQual and OverallCond). And the predictors are a rich mixture of both numerical and categorical variables. The purpose of the project is to generate several predictive linear regression models, including OLS (ordinary least square model) and OLS with regularization (ridge and lasso models), and compare them in terms of predictive accuracy).

## **Exploratory Data Analysis**

### 1. Dealing with missing data

**After loading and combining the train and test datasets, we first checked for missing data in the combined dataset. For linear regression, it is critical to deal with missing data because any NAs will inevitablly lead to the failure for model training and prediction of test data.**

colSums(is.na(total))

## Id MSSubClass MSZoning LotFrontage LotArea   
## 0 0 4 486 0   
## Street Alley LotShape LandContour Utilities   
## 0 2721 0 0 2   
## LotConfig LandSlope Neighborhood Condition1 Condition2   
## 0 0 0 0 0   
## BldgType HouseStyle OverallQual OverallCond YearBuilt   
## 0 0 0 0 0   
## YearRemodAdd RoofStyle RoofMatl Exterior1st Exterior2nd   
## 0 0 0 1 1   
## MasVnrType MasVnrArea ExterQual ExterCond Foundation   
## 24 23 0 0 0   
## BsmtQual BsmtCond BsmtExposure BsmtFinType1 BsmtFinSF1   
## 81 82 82 79 1   
## BsmtFinType2 BsmtFinSF2 BsmtUnfSF TotalBsmtSF Heating   
## 80 1 1 1 0   
## HeatingQC CentralAir Electrical X1stFlrSF X2ndFlrSF   
## 0 0 1 0 0   
## LowQualFinSF GrLivArea BsmtFullBath BsmtHalfBath FullBath   
## 0 0 2 2 0   
## HalfBath BedroomAbvGr KitchenAbvGr KitchenQual TotRmsAbvGrd   
## 0 0 0 1 0   
## Functional Fireplaces FireplaceQu GarageType GarageYrBlt   
## 2 0 1420 157 159   
## GarageFinish GarageCars GarageArea GarageQual GarageCond   
## 159 1 1 159 159   
## PavedDrive WoodDeckSF OpenPorchSF EnclosedPorch X3SsnPorch   
## 0 0 0 0 0   
## ScreenPorch PoolArea PoolQC Fence MiscFeature   
## 0 0 2909 2348 2814   
## MiscVal MoSold YrSold SaleType SaleCondition   
## 0 0 0 1 0   
## SalePrice   
## 0

**For several columns with lots of NAs, such as Alley and FireplaceQu, the NAs are actually a subcategory by themsleves, we replace these NAs with their own category ‘None’.**

total$MiscFeature[is.na(total$MiscFeature)] <- 'None'#MiscFeature  
total$FireplaceQu[is.na(total$FireplaceQu)] <- 'None'#FireplaceQu  
total$Fence[is.na(total$Fence)] <- 'None'#Fence  
total$Alley[is.na(total$Alley)] <- 'None'#Alley  
total$PoolQC[is.na(total$PoolQC)] <- 'None'#Alley

**For columns with few NAs, we replaced them by their mean or 0 (numerical variables), mode (categorical variables). A few examples are included below. In particular, for LotFrontage, NAs are replaced by average lotfrontage in their own neighborhood, assuming that house are more similar in the same neighborhood. For dealing with additional columns with missing data, see RMarkdown code for details**

total$MSZoning[is.na(total$MSZoning)] <- 'RL' #For MSZoning, 4 NA replaced by its mode (RL, 76.4%)   
total$MasVnrArea[is.na(total$MasVnrArea)] <- 0 #MAsVnrArea, replace NA with 0  
total$GarageArea[is.na(total$GarageArea)] <- 412 #GarageArea 1 NA replaced by the average size of detached garages  
  
#LotFrontage NAs replaced by their neighborhood averages ('mean' column in frontage)  
total1 <- total[!is.na(total$LotFrontage),]  
Frontage <- total1%>%group\_by(Neighborhood)%>%summarize(mean = as.integer(mean(LotFrontage)))  
for (neighborhood in Frontage$Neighborhood){  
 total$LotFrontage[is.na(total$LotFrontage)&total$Neighborhood == neighborhood] <- Frontage$mean[Frontage$Neighborhood==neighborhood]  
}

### 2. Change some numerical variables into categorical and *vis versa*

**For MSSubClass, MoSold and YrSold, it makes more sense to set them as categorical variables.**

#set the following three numerical variables as characters   
total$MSSubClass <- as.character(total$MSSubClass)  
total$MoSold <- as.character(total$MoSold)  
total$YrSold <- as.character(total$YrSold)

**For quality and condition evaluations, it makes more sense to set them as numerical variables. One example ExterQual is shown below. For additional columns, include ExterCond, BsmtCond, BsmtQual, HeatingQC, KitchenQual, FireplaceQu and GarageQual, see RMarkdown file for details. Note that for some columns, we combined some categories because they have few data points.**

#set categorical variable ExterQual as numerical   
total$ExterQual[total$ExterQual == 'Fa'] <- 1  
total$ExterQual[total$ExterQual == 'TA'] <- 2  
total$ExterQual[total$ExterQual == 'Gd'] <- 3  
total$ExterQual[total$ExterQual == 'Ex'] <- 4  
total$ExterQual <- as.numeric(total$ExterQual)

### 3. For some additonal categorical variables, we combined some subcategories. Because some categories have few data points (less than 10), which can present challenges for prediction and cross validation. It is possible that they might be only present in test dataset and lead to prediction failure.\_\_

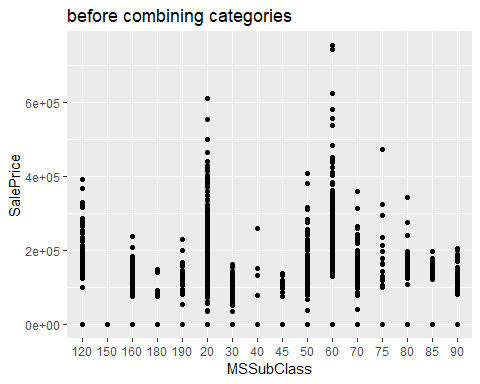
**First, we find the subcategories with fewer than 10 samples.**

categorical\_cols = total[, !sapply(total, is.numeric)]#table with only categorical variables  
dummy = data.frame(model.matrix(~.-1, categorical\_cols))#one hot encode create dummy variables  
colnames(dummy[, colSums(dummy)<10])#subcategories contain less than 10 samples

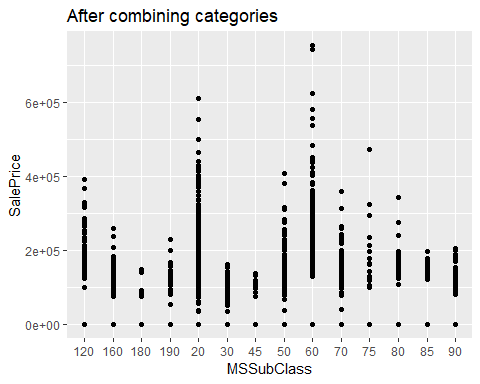
## [1] "MSSubClass150" "MSSubClass40" "UtilitiesNoSeWa"   
## [4] "Condition1RRNe" "Condition1RRNn" "Condition2PosA"   
## [7] "Condition2PosN" "Condition2RRAe" "Condition2RRAn"   
## [10] "Condition2RRNn" "HouseStyle2.5Fin" "RoofStyleShed"   
## [13] "RoofMatlMembran" "RoofMatlMetal" "RoofMatlRoll"   
## [16] "RoofMatlWdShake" "RoofMatlWdShngl" "Exterior1stAsphShn"  
## [19] "Exterior1stBrkComm" "Exterior1stCBlock" "Exterior1stImStucc"  
## [22] "Exterior1stStone" "Exterior2ndAsphShn" "Exterior2ndCBlock"   
## [25] "Exterior2ndOther" "Exterior2ndStone" "FoundationWood"   
## [28] "HeatingGrav" "HeatingOthW" "HeatingWall"   
## [31] "ElectricalFuseP" "ElectricalMix" "FunctionalMaj2"   
## [34] "FunctionalSev" "MiscFeatureOthr" "MiscFeatureTenC"   
## [37] "SaleTypeCon" "SaleTypeConLI" "SaleTypeConLw"   
## [40] "SaleTypeOth"

**Next, we can either delete all these subcategories (especially if it is not feasible to wrangle them manually), or deal with them individually by combining them with other subcategories with similar SalePrice distribution. An example how to do that is shown below. We plot the distribution of SalePrice against all MSSubClasses and combine subclass 40 and 150 with subclass 160 (they share similar distribution). We performed similar procedures for addtional 12 categorical variables (not shown, see Rmarkdown file for details). In addition, we dropped a couple of variables, Id and Utilities, because of their lack of prediction value.**

#combined MSSubClass40 and 150 with MSSubClass160 according the following plot  
ggplot(total, aes(MSSubClass, SalePrice))+geom\_point()+ggtitle("before combining categories")



total$MSSubClass[total$MSSubClass=='40'|total$MSSubClass=='150']= '160'  
ggplot(total, aes(MSSubClass, SalePrice))+geom\_point()+ggtitle('After combining categories')



### 4. Generate all dummy variables using one hot encode

numeric\_cols = total[, sapply(total, is.numeric)]#separate numerical and categorical variables   
categorical\_cols = total[, !sapply(total, is.numeric)]  
dummy = data.frame(model.matrix(~.-1, categorical\_cols))#one hot encode create dummy variables   
total1 = cbind(dummy, numeric\_cols)#regenerate the full table

### 5. Generate correlation martix, list and remove highly correlated variables (cutoff=0.9)

corr = cor(total1)  
table = as.data.frame(as.table(corr))  
table = table[order(table$Freq),]  
subset(table, abs(Freq) > 0.9 & table$Var1 != table$Var2)

## Var1 Var2 Freq  
## 15552 RoofStyleHip RoofStyleGable -0.9562473  
## 15974 RoofStyleGable RoofStyleHip -0.9562473  
## 28823 GarageCond GarageTypeNone -0.9206019  
## 29035 GarageCond GarageFinishNone -0.9206019  
## 42960 GarageTypeNone GarageCond -0.9206019  
## 42961 GarageFinishNone GarageCond -0.9206019  
## 42815 GarageCond GarageQual 0.9012176  
## 43026 GarageQual GarageCond 0.9012176  
## 2405 HouseStyleSLvl MSSubClass80 0.9584278  
## 15276 MSSubClass80 HouseStyleSLvl 0.9584278  
## 17477 Exterior2ndMetalSd Exterior1stMetalSd 0.9697117  
## 19587 Exterior1stMetalSd Exterior2ndMetalSd 0.9697117  
## 701 BldgType2fmCon MSSubClass190 0.9751185  
## 13572 MSSubClass190 BldgType2fmCon 0.9751185  
## 18116 Exterior2ndVinylSd Exterior1stVinylSd 0.9759649  
## 20226 Exterior1stVinylSd Exterior2ndVinylSd 0.9759649  
## 17051 Exterior2ndCmentBd Exterior1stCemntBd 0.9834109  
## 19161 Exterior1stCemntBd Exterior2ndCmentBd 0.9834109  
## 34300 SaleConditionPartial SaleTypeNew 0.9865730  
## 35566 SaleTypeNew SaleConditionPartial 0.9865730  
## 22802 BsmtFinType2None BsmtExposureNone 0.9935554  
## 23650 BsmtFinType2None BsmtFinType1None 0.9935554  
## 24912 BsmtExposureNone BsmtFinType2None 0.9935554  
## 24916 BsmtFinType1None BsmtFinType2None 0.9935554  
## 2822 BldgTypeDuplex MSSubClass90 1.0000000  
## 13794 MSSubClass90 BldgTypeDuplex 1.0000000  
## 22796 BsmtFinType1None BsmtExposureNone 1.0000000  
## 23640 BsmtExposureNone BsmtFinType1None 1.0000000  
## 28757 GarageFinishNone GarageTypeNone 1.0000000  
## 28968 GarageTypeNone GarageFinishNone 1.0000000

#remove highly correlated variables  
highCorr = findCorrelation(corr, cutoff=.9)  
total2 = total1[,-c(highCorr)]

### 6. Recovering train and test datasets

train <- total2[total$SalePrice != 10, ]  
test <- total2[total$SalePrice == 10, ]

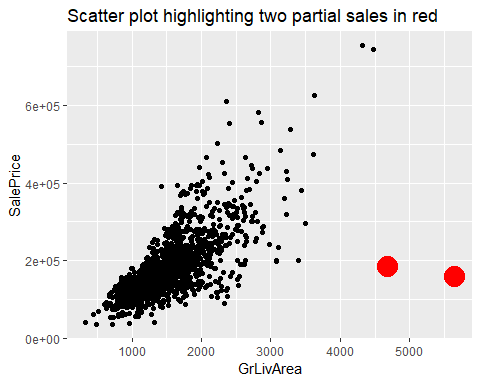
### 7. looking for potential outliers in train dataset

**We first idenitfied the predictors that are highly correlated with the response variable SalePrice and plotted one of strong predictor GrLivArea against SalePrice and visually identified two potential datapoints (id =524 and 1299) with high leverage and influence (highlighted in red). We further fitted a preliminary full linear regression model and examined its residual vs leverage plot. Indeed, the same two data points have very large residuals, high leverage and cook’s D score (notice that Id= 1299 is actually literally off the chart!). In addition, further investigation confirmed that both are partial sales and likely not reflective their true sale prices. We removed these two outliers in part because of they have the largest living area (GrLivArea > 4500), yet both of them have exceptionally low sale prices.**

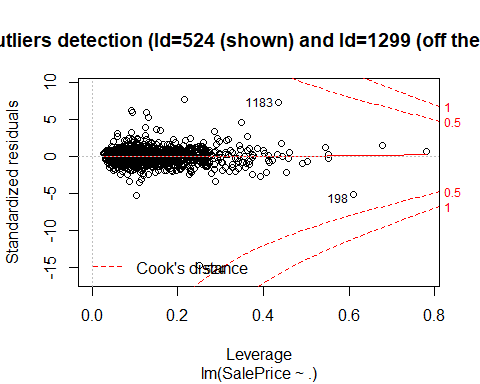
#identify the highest corerlated predictors for the response variable SalePrice  
corr = cor(train)  
table = as.data.frame(as.table(corr))  
table = table[order(table$Freq, decreasing =TRUE),]  
corr1 = subset(table, table$Var1=='SalePrice')  
head(corr1, 10)#Top 10 predictors

## Var1 Var2 Freq  
## 39601 SalePrice SalePrice 1.0000000  
## 31840 SalePrice OverallQual 0.7909816  
## 35223 SalePrice GrLivArea 0.7086245  
## 32835 SalePrice ExterQual 0.6826392  
## 36616 SalePrice KitchenQual 0.6595997  
## 37611 SalePrice GarageCars 0.6404092  
## 37810 SalePrice GarageArea 0.6234314  
## 33233 SalePrice BsmtQual 0.6229247  
## 34228 SalePrice TotalBsmtSF 0.6135806  
## 34626 SalePrice X1stFlrSF 0.6058522

#plot Saleprice vs one of its top predictors, GrLivArea, to visually identify outliers   
ggplot(train, aes(GrLivArea, SalePrice))+geom\_point()+ggtitle('Scatter plot highlighting two partial sales in red')+geom\_point(data=train[train$GrLivArea>4500, ], aes(GrLivArea, SalePrice), colour="red", size=7)



#Another way to look at their leverage and influence by fitting a preliminary full linear regression model and examine its residual plots  
Pre\_model = lm(SalePrice~., data= train)  
plot(Pre\_model, which = 5, caption = NULL, main = 'Outliers detection (Id=524 (shown) and Id=1299 (off the chart)')



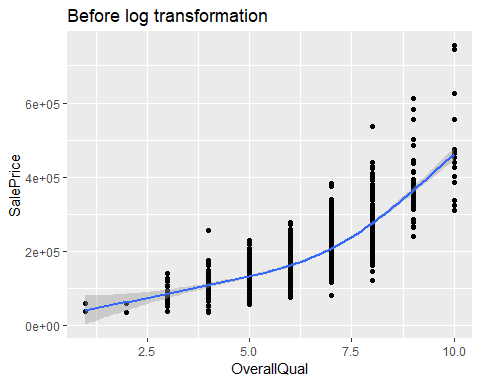
#removing two large houses with exceptionally low sale price (presales, not reflective their true value)  
train = train[train$GrLivArea<4500, ]

### 8. log transformation of SalePrice

**plotting SalePrice vs several of its strong predictors, such as OverallQual shown below, demonstrates a distictive curve, suggesting that logorithmic transformation of SalePrice might help fit the regression model better. Indeed, after SalePrice log transformation, the two variables appear to have a linear relationship.**

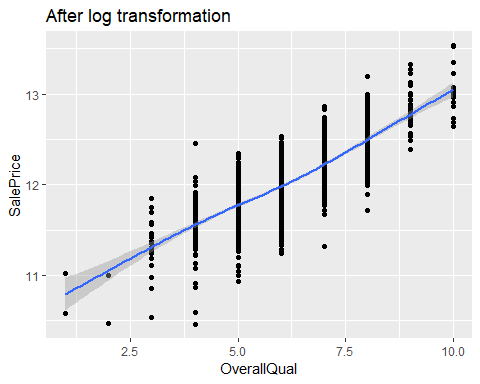
ggplot(train, aes(OverallQual, SalePrice))+geom\_point()+geom\_smooth()+ggtitle('Before log transformation')

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



train$SalePrice = log(train$SalePrice)#logtranformation of response variable SalePrice  
ggplot(train, aes(OverallQual, SalePrice))+geom\_point()+geom\_smooth()+ggtitle('After log transformation')

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



## **Fitting models**

**After extensive data cleanup and transformation, we are ready to train regression models for predicting house prices. We are going to train three models: a regular linear regression model, also called ordinary least square (OLS) model, ridge and lasso regression model.**

### 1. First we set aside a test dataset by spliting the train dataset into train\_train and train\_test (8:2)

set.seed(99)  
index = createDataPartition(train$SalePrice, p=.8, list = F)  
train\_train = train[index, ]  
train\_test = train[-index, ]

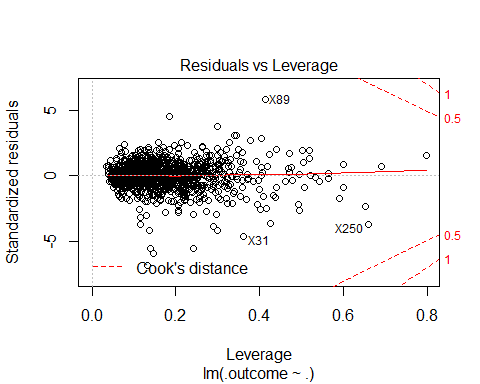
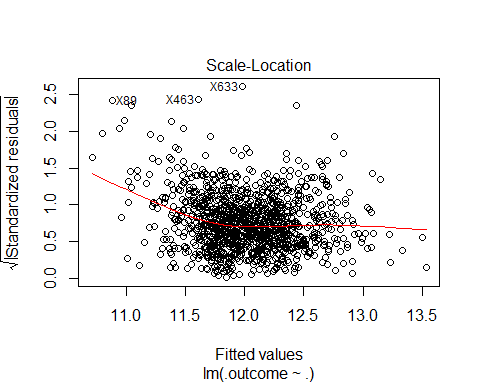
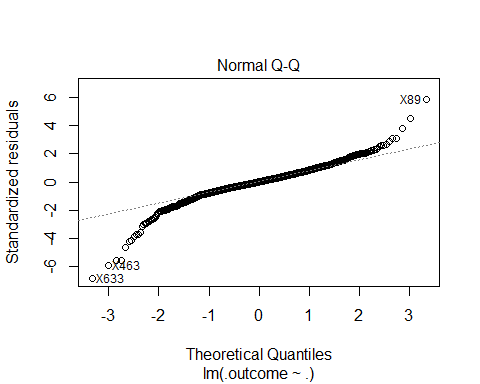
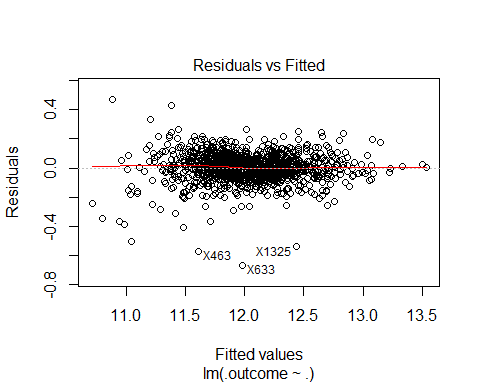
### 2. setup training controls

fit\_control = trainControl(method = 'cv', number = 10)

### 3. fit the OLS model

**We first fitted full regression model, also called ordinary least square (OLS) model and assessed its residual plots for assumptions of linear regression. First of all, we have no reason to suspect that all sales are independent from each other and likely reflect their true value. Second, with the exception of houses with low predicted values (fitted values < 11.5), the variance is more or less constant. Third, from the QQ-plot, the assumption of normal distribution of residuals are less than ideal, significant deviation from linearity exists in both end of theoretical quantiles. Fourth, many predictors are indeed linearly correlated with the response variable, SalePrice. Thus, the linear regression model is likely a sufficient fit for the dataset.**

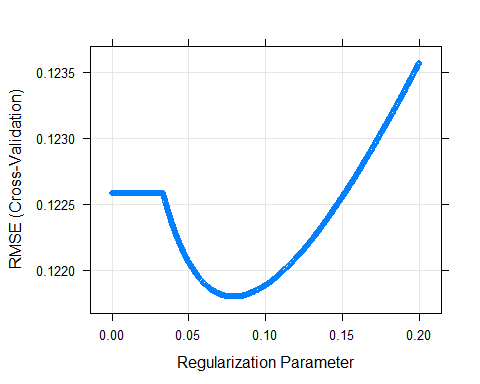
full\_model = train(SalePrice~., data=train\_train, method = 'lm', trControl = fit\_control)  
plot(full\_model$finalModel)#residual plots



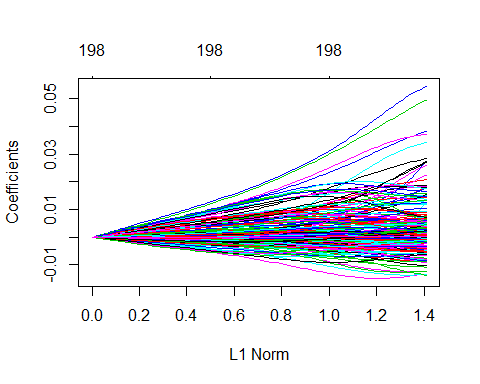
### 4. fit the ridge model

**Next, we fitted the ridge model and used cross validation to find the best value for the tuning parameter lambda. From the results, we noticed that the best tuned lambda value is rather small (), suggesting that only a small penalty is necessary for the reguarization. In other words, this is a rather rich model with many strong predictors, consistent with our correlation martrix analyses and ordinary least square model above.**

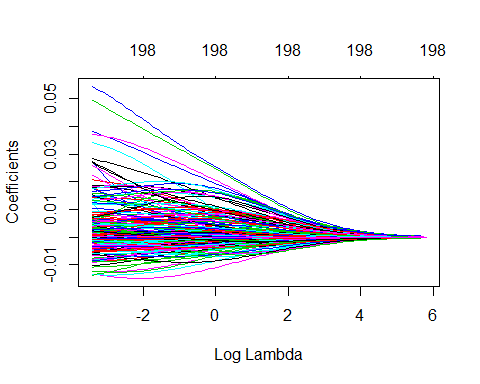
ridge\_model = train(SalePrice~., train\_train, method='glmnet', preProc = c("center", "scale"), tuneGrid = expand.grid(alpha=0, lambda = seq(0.0001, .2, length = 1000)), trControl=fit\_control)  
plot(ridge\_model)



plot(ridge\_model$finalModel)



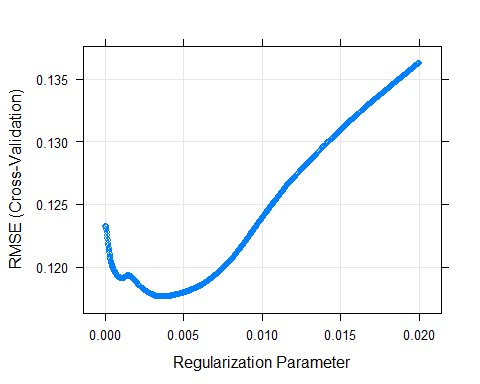
plot(ridge\_model$finalModel, 'lambda')



### 5. fit the lasso model

**We also fitted the asso model and used cross validation to choose the best tuning parameter lambda, which results in smallest MSE. We similarly noticed that the best tuning lambda value is rather small, suggesting that the training dataset contains a rich set of strong predictors and only small penalty term is necessary to achieve the best regularization.**

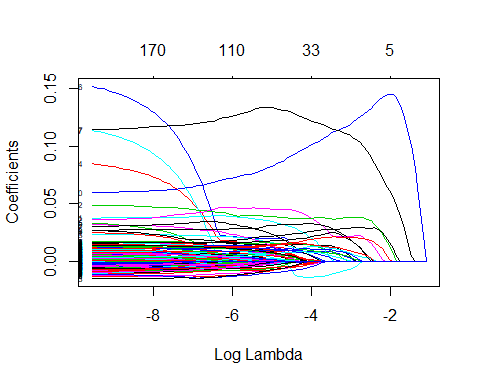
lasso\_model = train(SalePrice~., train\_train, method='glmnet', preProc = c("center", "scale"), tuneGrid = expand.grid(alpha=1, lambda = seq(0.00001, 0.02, length = 1000)), trControl=fit\_control)  
plot(lasso\_model)



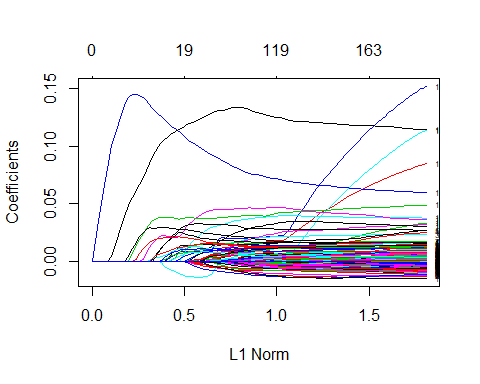
coef = coef(lasso\_model$finalModel, lasso\_model$bestTune$lambda)  
paste('The number of predictors with non-zero coefficients in the Lasso model: ', sum(coef!=0))

## [1] "The number of predictors with non-zero coefficients in the Lasso model: 92"

plot(lasso\_model$finalModel, 'lambda', label = TRUE)



plot(lasso\_model$finalModel, 'norm', label = TRUE)



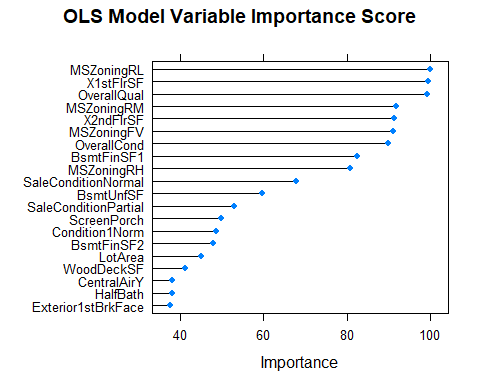
### 6. Comparing the models

**We compare mean squared errors (MSE) of the fitted models when they are applied on the same test dataset we set aside earlier. From the results, we conclude that both regularzed models have significant higher prediction accuracy then OLS model because their MSE scores are lower then the of OLS model.**

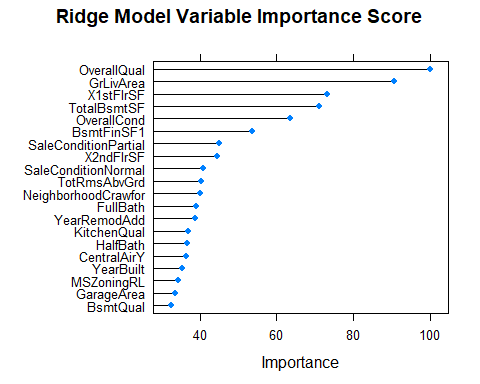
predict1 = predict(full\_model, train\_test)  
OLS = postResample(predict1, train\_test$SalePrice)  
predict2 = predict(ridge\_model, train\_test)  
Ridge = postResample(predict2, train\_test$SalePrice)  
predict3 = predict(lasso\_model, train\_test)  
Lasso = postResample(predict3, train\_test$SalePrice)  
table = rbind(OLS, Ridge, Lasso)  
table

## RMSE Rsquared MAE  
## OLS 0.11004399 0.9209650 0.07925006  
## Ridge 0.10414333 0.9271675 0.07637990  
## Lasso 0.09813157 0.9353413 0.07180475

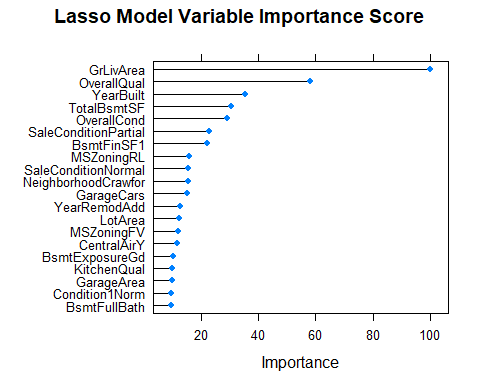
par(mfrow=c(1,3))  
plot(varImp(full\_model), top=20, main='OLS Model Variable Importance Score')



plot(varImp(ridge\_model, scale = T), top = 20, main='Ridge Model Variable Importance Score')



plot(varImp(lasso\_model), top =20, main= 'Lasso Model Variable Importance Score')



## **Conclusion**

**From our analysis, we conclude that Ames house price is a very rich dataset in the sense that it contains a number of strong predictors of house price in Ames, IA, such as GrLivArea (the sum of ist and 2nd floor square footages), Zoning and OverallQual. All three models correctly identified them as the most important factors for determining the sale price of a house. While OLS and Ridge models incorporates all predictors in their models, Lasso uses L1 penalty to limit both the number and range of predictors. The best tuned Lasso models incorporate on average 90 predictors, which represent slightly less than half of all available predictors. Like mentioned above, our analysis also indicate that even for a rich dataset where noises represent less of a problem, regularization combined with cross validation is still critical to prevent overfitting thus improve prediction accuracy. In our Ridge and Lasso models, including a small amount of penalty term both sigificantly improves the MSE and scores of prediction, further confirming that regularization is the key to prevent building overly complicated models that fit both signal and noise in training dataset.**