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 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). 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 prediction 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.

head(colSums(is.na(total)))

## Id MSSubClass MSZoning LotFrontage LotArea Street   
## 0 0 4 486 0 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' #4 NAs replaced by mode (RL)   
total$MasVnrArea[is.na(total$MasVnrArea)] <- 0 #replace NAs with 0  
total$GarageArea[is.na(total$GarageArea)] <- 412 #1 NA replaced by average size of detached garages

### 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.

total$MSSubClass <- as.character(total$MSSubClass)  
total$MoSold <- as.character(total$MoSold)  
total$YrSold <- as.character(total$YrSold)

For quality and condition evaluation scores, it makes more sense to set them as numerical variables. One example, ExterQual, is shown below. For additional columns, including ExterCond, BsmtCond, BsmtQual, HeatingQC, KitchenQual, FireplaceQu and GarageQual, see RMarkdown file for details. Note that for some columns, we combined some subcategories 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 categorical variables, we combined some subcategories that contain few data points (less than 10). It is possible that they might be only present in test dataset and lead to prediction failure.

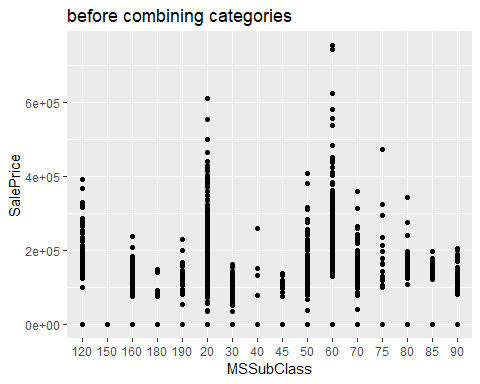
First, we find the subcategories with fewer than 10 samples.

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

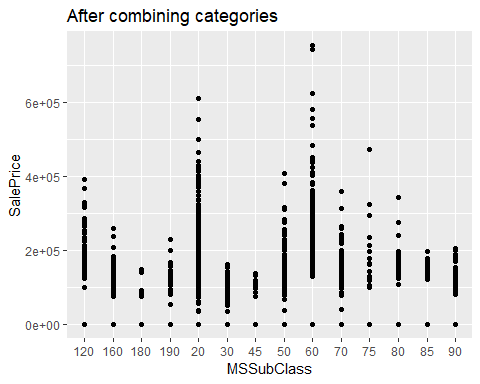
## [1] "MSSubClass150" "MSSubClass40" "UtilitiesNoSeWa" "Condition1RRNe"   
## [5] "Condition1RRNn" "Condition2PosA"

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 MSSubClasses and combine subclass 40 and 150 with 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.

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)]#nymerical  
categorical\_cols = total[, !sapply(total, is.numeric)]#categorical  
dummy = data.frame(model.matrix(~.-1, categorical\_cols))#dummy   
total1 = cbind(dummy, numeric\_cols)#regenerate full table

### 5. Generating correlation martix, listing and removing highly correlated predictive variables (cutoff = 0.9)

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

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

### 6. Recovering train and test datasets

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

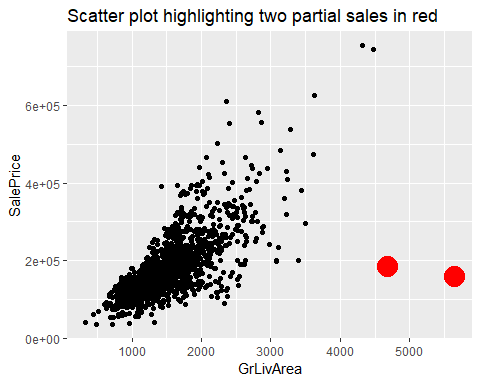
### 7. looking for potential outliers in train dataset

We first identified 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 with high leverage and influence (id =524 and 1299, 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 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 have exceptionally low sale prices.

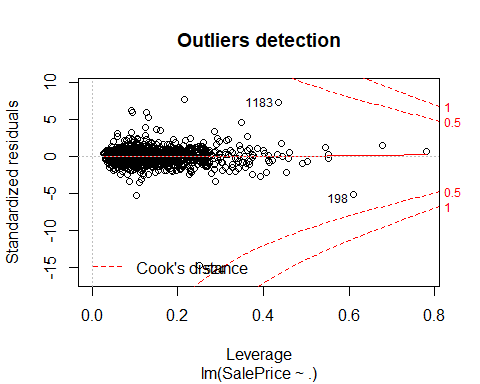
corr = cor(train)  
table = as.data.frame(as.table(corr))  
table = table[order(table$Freq, decreasing =TRUE),]  
corr1 = subset(table, table$Var1=='SalePrice')  
head(corr1, 3)#Top predictors for saleprice

## Var1 Var2 Freq  
## 39601 SalePrice SalePrice 1.0000000  
## 31840 SalePrice OverallQual 0.7909816  
## 35223 SalePrice GrLivArea 0.7086245

#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)



#Build a preliminary linear model and look for outliners  
Pre\_model = lm(SalePrice~., data= train)  
plot(Pre\_model, which = 5, caption = NULL, main = 'Outliers detection')



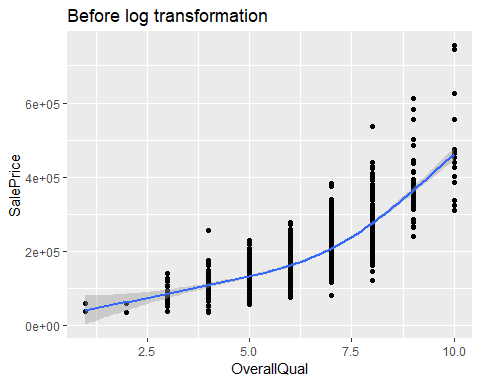
train = train[train$GrLivArea<4500, ]

### 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.

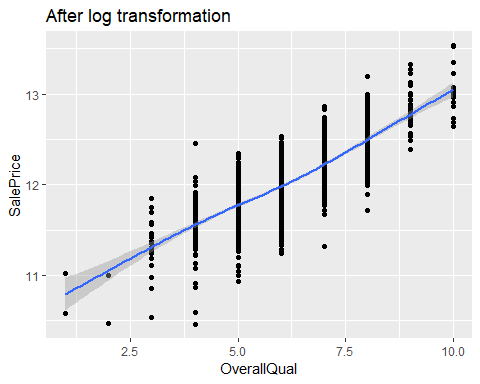
ggplot(train, aes(OverallQual, SalePrice))+geom\_point()+geom\_smooth()+ggtitle('Before log transformation')

## `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

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

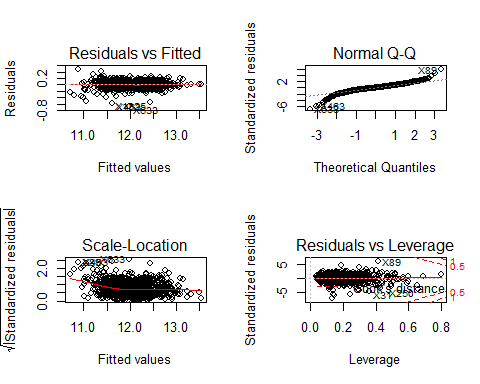
### 2. setup training controls

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

### 3. fit OLS model and address the assumptions

We first fitted full regression model and assessed its residual plots for assumptions of linear regression. First of all, we have no reason to suspect that sales are not independent from each other and reflect the house’s value. Second, except a few 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 at both end of theoretical quantiles. Fourth, many predictors are indeed linearly correlated with the response variable, SalePrice. In general, we think that linear regression models are likely sufficient for the dataset.

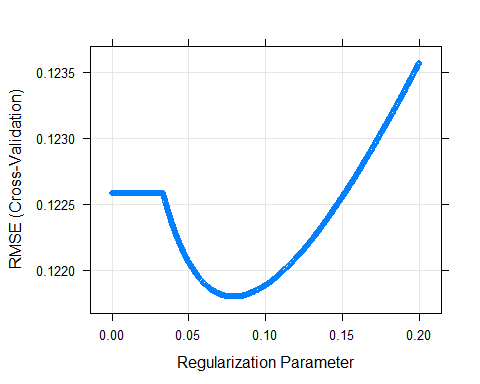
full\_model = train(SalePrice~., data=train\_train, method = 'lm', trControl = fit\_control)  
par(mfrow = c(2,2))  
plot(full\_model$finalModel)



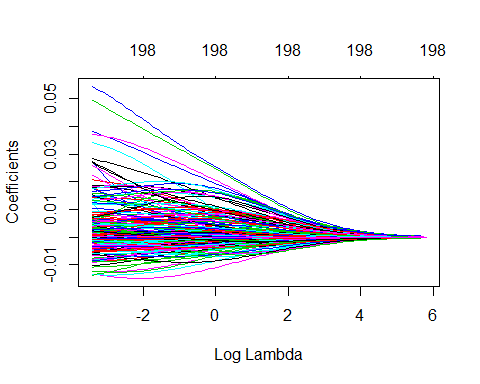
### 4. fit the ridge model

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)



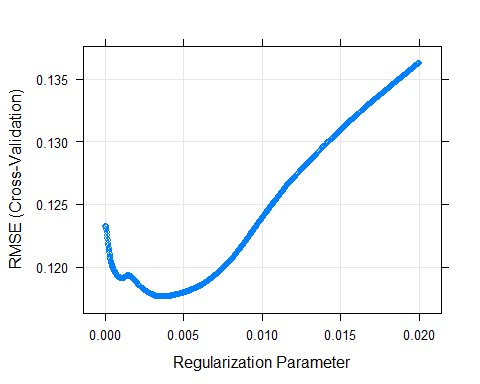
plot(ridge\_model$finalModel, 'lambda')



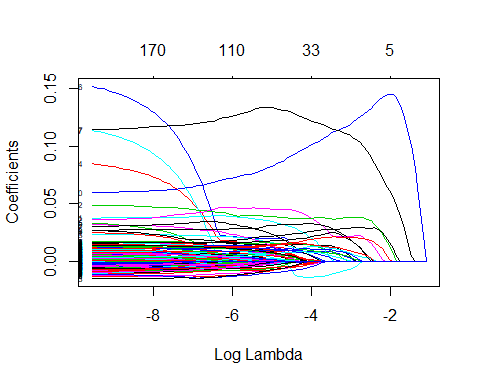
### 5. fit the lasso model

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



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"

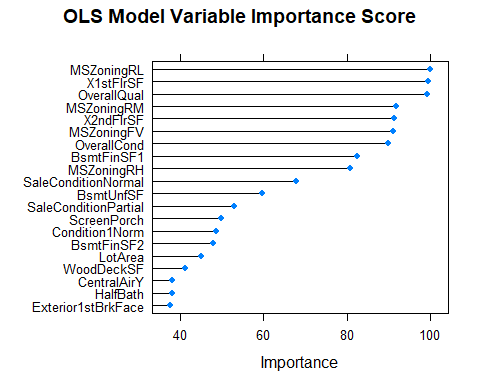
### 6. Comparing the models

We compare the models’ mean squared error (MSE) and R-Squared scores after they are applied on the same test dataset that we set aside earlier. From the results, we conclude that both regularized models have significantly higher prediction accuracy then OLS model because their MSE scores are lower. In addition, all three models have similar lists of important predictors shown by their variable importance charts.

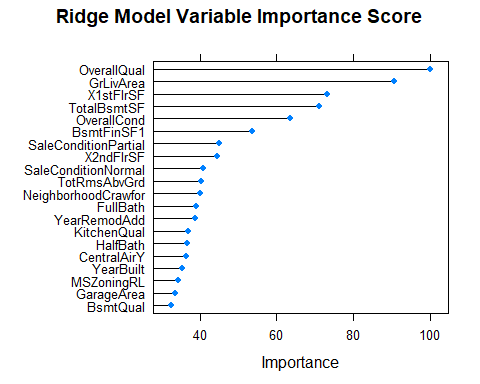
predict1 = predict(full\_model, train\_test)  
OLS = postResample(predict1, train\_test$SalePrice)  
predict2 = predict(ridge\_model, train\_test)  
Ridge = postResample(predict2, train\_test$SalePrice)  
predict3 = predict(lasso\_model, train\_test)  
Lasso = postResample(predict3, train\_test$SalePrice)  
table = rbind(OLS, Ridge, Lasso)  
table#table showing test MSE and R squared

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

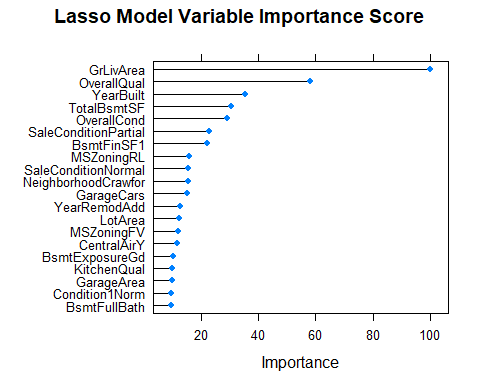
plot(varImp(full\_model), top=20, main='OLS Model Variable Importance Score')



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 present less of a challenge, 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.