Assignment #5:

By: Valon Tika

Introduction:

The purpose of this assignment is to build on the existing assignments that have been created for MSDS 410. The goal will be to create a predictive modelling framework to explore the ability to use automated variable selection for variable identification and predictive accuracy of modelling efforts. We will also assess the comparison between the statistical model validation and business model validation.

Tasks:

Section 1: Sample Definition and Data Split

The data set that will be used will be defined only within the Ames, Iowa market. There are 2,930 records that were recorded as sold in Ames from 2006 to 2010. There are 82 total variables (23 ordinal, 14 discrete, and 20 continuous variables) stored within this data set.

The problem that this assignment will try to solve is by predicting the housing market sales price within this specific set of data. The first step of this will be to do data sampling by using drop conditions and other sub-setting methods to narrow down our training set of data.

Section 1.1: Sample Definition

We will continue with the eligible population set from the start of the course and what we used in assignment there where we used single family homes as our subset of data to perform our modeling efforts. This is also considered as our drop conditions when filtering out the records that we don't need for our analysis. To recap, the goal will be to try and figure out which properties have a normal sale condition, paved street, year build past 1950, a basement, and a general living area greater than 800 square feet and greater than 4,000 square feet. We first concluded a set of conditions that labeled the records for what was considered as "drop conditions" for an appropriate housing subset. Here we can determine that we will start with 1,469 homes that are eligible for further analysis.

Condition	Count
Not SFT	505
Non-normal sale	94
Street not paved	6
Built pre 1950	489
No Basement	28
LT 800 SqFT	9
GT 4000 SqFT	1
Eligible Sample	**1469**

Table 1. Drop conditions in sample definition

Section 1.2: The Train/Test Split

For splitting the data for testing and training, 70% of the data was randomly chosen to train the data and the remaining 30% was used to test the data. The total count of training population is 1037; Total count of testing population is 432

Population	Count	%
Testing	432	30
Training	1037	70
Eligible	1469	100

Table 2. Sampling count before choosing pool of variables

Section 2: Model Identification and In-Sample Model Fit

In the first attempt to create the pool of variables for , BldgType was removed as it had only one level in the defined sample and therefore it was not considered to be included in the regression analysis. In the second attempt, those variables with best significance level obtained in the upper model were chosen. In the third attempt, the variable Neighborhood was removed as it had a variance inflation factor equal to 20. The final pool of variables to feed the automated variable selection process is the following:

No	Variable	Explanation	Type
1	LotArea	Lot size in square feet	Continuous
2	HouseStyle	Style of dwelling	Nominal
3	HeatingQC	Heating quality and condition	Ordinal
4	Exterior1	Exterior covering on house	Nominal
5	TotalBsmtSF	Total square feet of basement area	Continuous
6	FirstFlrSF	First Floor square feet	Continuous
7	SecondFlrSF	Second Floor square feet	Continuous
8	FullBath	Full bathrooms above grade	Discrete
9	GarageYrBlt	Year garage was built	Discrete
10	GarageArea	Size of garage in square feet	Continuous
11	QualityIndex	OverallQual*OverallCond;	Discrete
12	TotalSqftCalc	BsmtFinSF1+BsmtFinSF2+GrLivArea	Continuous
13	HouseAge	YrSold - YearBuilt	Discrete
14	LotShape	General shape of property	Ordinal
15	YearRemodel	Remodel date	Discrete

Table 3. Pool of variables chosen to feed the automated variable selection

After the pool of variables is chosen, records containing NA values were removed to avoid execution problems with regression and predictive functions. For that reason, the sample count for training and test decrease from this point of the modelling to this new count:

Population	Count	%
Testing	425	28.9
Training	1025	69.7
NA values	19	1.3
Eligible	1469	100

Table 4. Sampling count after choosing pool of variables

Section 2.1 Forward Variable Selection

In this scenario, StepAIC selection starts with zero variables and each step that the model progresses, it will add a variable to the model. The method stepAIC performed 13 forward to reach an optimal model. Each step added a one variable to the model. The result included 13 variables in the model total after completion. YearRemodel and HeatingQC were not included in the final model when ran due to lack of strength in predictor value. The final AIC value that was produced after the model ran was 20,683.66

Section 2.2 Backward Variable Selection

The backward variable selection will take the full model that was created and work backwards to remove any unnecessary variables that are independent of the response variable. The forward variable selection does not take this into account, so working backwards to eliminate the unnecessary variables is needed. The method performed 2 steps total. The result includes 13 variables in the model. The same AIC value (20683.66) was produced and the same variables are shown compared to the forward.lm model. YearRemodel and HeatingQC were removed in the steps again.

Section 2.3 Stepwise Variable Selection

Stepwise variable selection starts with a model and does both forward and backward variable selection in search for the optimal model. The method was performed 12 steps. The result also includes 13 variables in the model. The same AIC (20,683.66) was generated and the selected variables were selected, which matched to both the backwars and forward models from earlier.

Section 2.4 Model Comparison

The three models generated with the automated variable selection method are equal, that is, forward, backward, and stepwise variable selections contain the same variables and have the same AIC. The difference in the execution of the StepAIC is the number of steps to reach the final model for each case. This is coherent as the criteria for reaching a final model in the three cases are equal and the difference remains in the initial condition of stepAIC and the direction.

Given that result, in this document, only two models will be considered: forward.lm and junk.lm, as forward.lm is similar to backwards.lm and stepwise.lm

In the forward.lm, the variance inflation factor is under 20 for all variables. In the junk model, variance inflation factor is greater than 30 for three predictors (OverallQual, OverallCond, QualityIndex) as they have a direct mathematical relationship.

Considering variance inflation factor values for indicator variables is important as the objective is to build a model that is simple, easy to understand and manageable. A high variance inflation factor value for an indicator variable means that this variable is redundant and can be removed from the model without affecting the quality of the model. Less variables in the model simplify the model and increases the easiness to work with it.

Below, we will compare the junk and forward model in terms of statistical indicators that will include R-Squred, AIC, BIC, MSE, MAE and the square root of MSE:

	forward	**junk**
R-Squared	0.8938	0.8529
AIC	23594.48	24161.62
BIC	23747.39	24196.23
MSE(*)	547374470	759492599
MAE	16788.88	20195.13
Sqrt(MSE)	23396.04	27558.89

Table 5. Model comparison with statistical indicators for the in-sample prediction

In order to compare with MAE, the square root of MSE should be applied for better evaluation between the two metrics for both the forward and junk models. (shown in last row).

$$(*)MSE = mean((sampleSale - predictedSale)^2)$$

The forward model was ranked 1st in all parameters while the junk model was ranked 2nd behind. A model can have some metrics better than others compared to other models.

Section 3: Predictive Accuracy

MAE and MSE are two metrics that are commonly used when comparing models for predictive accuracy and performance. MAE, or mean absolute error, measures the mean magnitude of error in a prediction model without considering the absolute direction. MSE, or mean squared error, measures the average of the squares of the errors. MAE is more applicable when all predictor differences have equal weight while MSE is used when large errors are not wanted. In the model to predict house pricing, the variance in error will be considered since the goal is to see how close the predictions are to the actual values. MAE would be considered as a metric in this case. Below is the MSE and MAE of the out-of-sample prediction for both models.

The summary for the prediction out-of-sample for both models:

	forward	**junk**
MSE(*)	738228467	743708101

MAE	18209.77	19631.19
Sqrt(MSE)	27170.36	27271.01

Table 6. Model comparison with statistical indicators for the out-of-sample prediction

In order to compare with MAE, the square root of MSE should be applied for better evaluation between the two metrics for both the forward and junk models. (shown in last row).

$$(*)MSE = mean((sampleSale - predictedSale)^2)$$

The forward model calculated through StepAIC is still better than the junk model according to the MSE and MAE criteria for the out-of-sample prediction. However, the difference decreases between both models, which is around $1,300 dollars of difference in the MAE for a sale price of a house.

The preference for MSE versus MAE should be when large errors is particularly undesirable. This is because, in MSE, errors are squared before they are averaged, giving relatively high weight to large errors. In this case – the prediction of home sales prices – the criticality of high errors is not as important as may be in a disease survival prediction where a high error may result in the death of persons.

Better predictive accuracy with in-sample than out-of-sample may be interpreted as the in-sample evidence is likely to be spurious (for instance, in our case, we may redo the split of test and training data to improve the types of each variable in each group). But this low accuracy for out-of-sample may simply imply that the out-of-sample tests have less power than in-sample tests. One of the reasons may be the size of the test sample or the characteristics (in our case the test sample is just 30% of the total count whereas the train sample is 70% of the total count). In the same way, a review of the test sample may be needed in our case to analyse the reasons and to improve the test results in case a more demanding targets were requested.

Section 4: Operational Validation

The analysis that has been conducted until now shows that the model using automated variable selection is the better model compared to the junk model (forward model being used to compare against the junk model since forward, backward and stepwise produced the same results). In order to implement this model into practical business use, the errors need to be identified for business to see if business requirements have been met when developing this model. The statistical errors may not provide a decent perspective on the distribution of errors. To further validate this model for operational use, the error rates have been broken down into 4 grades.

The first grade (Grade 1) represents the error when it is 0%-10% of the sales price. The second grade (Grade 2) represents the error when it is 10%-15% of the sales price. The third grade (Grade 3) represents the error when it is 15%-25% of the sales price. The fourth grade (Grade 4) represents the error when it is greater than 25% of the sales price.

Below is a table that represents the performance of the forward and junk models by percent distribution by grade for both the training and testing data for both in-sample and out-of-sample data. The forward model's performance in Grade 1 (within 10% error) is 65%-68.2% for both in-sample and out-of-sample data compared to the junk model which is only 56.6%-58%.

The summary for the results of model comparison based on the defined business policy:

(*)	**forward**		**junk**	
	Training data	Test data	Training data	Test data
Grade 1	68.2%	65.1%	56.6%	58%
Grade 2	14%	15.1%	20.5%	20.5%
Grade 3	13.9%	15.6%	15.5%	14.6%
Grade 4	3.8%	4.2%	7.3%	6.8%

Table 7. Model comparison with operational validation

(*) Numbers rounded to the first decimal digit in table 7

The forward and the junk model have Grade1 greater than 50%, therefore they are 'underwriting quality'. The junk model has worse Grade 1 performance, so it is worse than the forward model.

Conclusion

When dealing with many potential predictor variables, it can be a challenge to try and understand which variables (discrete and continuous) should be selected when trying to optimize a predictive model. Creating a sample population using drop conditions assisted greatly with this analysis for better selection criteria for training and testing data. EDA should be performed on key variables, such as response variable(s) and predictor variables with a strong correlation to the response variable based on a correlation matrix. For specifying a model, it takes both technical and non-technical skill sets to better define and test. This involves industry expertise, technical experience with tools such as R, and statistical and mathematical aptitude to understand which analyses to use when modelling the data out. Model interpretation and validation should be done carefully and analysed thoroughly for better understanding of what the model is doing. There also needs to be a consensus on understanding the actual business needs when building models out. Business requirements documentation and technical documentation need to be aligned and married along the development lifecycle so there are no gaps in what the business expects and what is developed. Interpreting model predictions and results to business will also help with understanding operational validation of models that are built.

Sample definition and data split was done by first identifying drop conditions to select only single-family housing along with creating a 70/30 training testing split of that data for model training and testing model performance.

Model identification and in-sample model fit was performed by doing forward, backward and stepwise variable selection. All three model performed the same, which led to using the forward model to compare again a junk model that was created for this analysis. In both cases, when comparing models using predictive accuracy and

operational validation criteria, the forward model ranks better than the junk. Both in-sample and out-of-sample MAE and MSE seem relatively closer to the sales price for the forward model compared to the junk model. In regards to operational validation, roughly 65-68 percent of the data were below the 10% error threshold versus the 56-58 percent equivalent in the junk model.

Appendix (R Code)

```
library(MASS)
library(car)

ames.df =
read.csv("C:/Users/vtika/Downloads/assignment5/ames_housing_data.csv",
header = TRUE,stringsAsFactors = TRUE)

########################################
###determining our sample set of data###
########################################

##Narrowing down to single family homes and removing houses where
GrllivArea is less than
##800 and greater than 4000

ames.df$dropCondition <- ifelse(ames.df$BldgType!='1Fam','01: Not
SFR',
    ifelse(ames.df$SaleCondition!='Normal','02: Non-Normal Sale',
    ifelse(ames.df$Street!='Pave','03: Street Not Paved',
    ifelse(ames.df$YearBuilt <1950,'04: Built Pre-1950',
    ifelse(ames.df$TotalBsmtSF <1,'05: No Basement',
    ifelse(ames.df$GrLivArea <800,'06: LT 800 SqFt',
    ifelse(ames.df$GrLivArea >4000,'07: GT 4000 SqFt',
    '99: Eligible Sample')
    ))))))

table(ames.df$dropCondition)

# Eliminate all observations that are not part of the eligible sample
population;
elig.pop <- subset(ames.df,dropCondition=='99: Eligible Sample');

##setting training/testing sets
set.seed(123)
elig.pop$u <- runif(n=dim(elig.pop)[1],min=0,max=1);

# Define these two variables for later use;
elig.pop$QualityIndex <- elig.pop$OverallQual*elig.pop$OverallCond;
elig.pop$TotalSqftCalc <-
elig.pop$BsmtFinSF1+elig.pop$BsmtFinSF2+elig.pop$GrLivArea;
elig.pop$HouseAge <- elig.pop$YrSold - elig.pop$YearBuilt

# Create train/test split;
train.df <- subset(elig.pop, u<0.70);
test.df <- subset(elig.pop, u>=0.70);

# Checking the data split. The sum of the parts should equal the
whole.
total_ames_count = dim(elig.pop)[1]
total_training = dim(train.df)[1]
total_testing = dim(test.df)[1]
testing_and_training = dim(train.df)[1]+dim(test.df)[1]

print(paste("Total count of eligible population is ",
total_ames_count))
print(paste("Total count of eligible population is ",
testing_and_training))
print(paste("Total count of training population is ", total_training))
print(paste("Total count of testing population is ", total_testing))
```

```
#First model: first tentative
drop.list <- c('SalePrice','GarageCond','SaleType', 'HouseAge',
'TotalSqftCalc','QualityIndex','FirstFlrSF','SecondFlrSF','TotalBsmtSF
','GarageArea','GarageYrBlt','GarageArea','Zoning','FullBath','LotArea
','LotShape','Neighborhood','Exterior1','HouseStyle','OverallCond','Ye
arRemodel');

#Second model: Using those with higher rate in the upper model
drop.list <-
c('SalePrice','LotArea','Neighborhood','HouseStyle1','OverallCond','Ex
terior1','TotalBsmtSF','FirstFlrSF','SecondFlrSF','FullBath','GarageYr
Blt','GarageArea','QualityIndex','TotalSqftCalc','HouseAge','LotShape'
,'YearRemodel')

#Third model: removing Neighborhood as it givesa VIF value equal to 20
and replacing OverallCond with HeatingQC
drop.list <-
c('SalePrice','LotArea','HouseStyle','HeatingQC','Exterior1','TotalBsm
tSF','FirstFlrSF','SecondFlrSF','FullBath','GarageYrBlt','GarageArea',
'QualityIndex','TotalSqftCalc','HouseAge','LotShape','YearRemodel')


train.clean <-train.df[,(names(elig.pop) %in% drop.list)];
#have to clean the NAs in the train clean to avoid the stepIAC
function crashes
train.clean <- na.omit(train.clean)

# Define the upper model as the FULL model
upper.lm <- lm(SalePrice ~ .,data=train.clean);
summary(upper.lm)
# Define the lower model as the Intercept model
lower.lm <- lm(SalePrice ~ 1,data=train.clean);
# Need a SLR to initialize stepwise selection
sqft.lm <- lm(SalePrice ~ TotalSqftCalc,data=train.clean);
summary(sqft.lm)

# Note: There is only one function for classical model selection in R
- stepAIC();
# stepAIC() is part of the MASS library.
# The MASS library comes with the BASE R distribution, but you still
need to load it;
# Done at the beginning of this script
# Call stepAIC() for variable selection
forward.lm <-
stepAIC(object=lower.lm,scope=list(upper=formula(upper.lm),lower=~1),d
irection=c('forward'));
summary(forward.lm)
forward.lm
backward.lm <- stepAIC(object=upper.lm,direction=c('backward'));
summary(backward.lm)

stepwise.lm <-
stepAIC(object=sqft.lm,scope=list(upper=formula(upper.lm),lower=~1),di
rection=c('both'));
summary(stepwise.lm)

#here do not need to clean NAs
```

```
junk.lm <- lm(SalePrice ~ OverallQual + OverallCond + QualityIndex +
GrLivArea + TotalSqftCalc, data=train.df)
summary(junk.lm)


sort(vif(forward.lm),decreasing=TRUE)
sort(vif(backward.lm),decreasing=TRUE)
sort(vif(stepwise.lm),decreasing=TRUE)
vif(forward.lm)
vif(backward.lm)
vif(stepwise.lm)
vif(junk.lm)

#R-sqare obtained from summary(model).

#AIC and BIC: the lower the better
AIC(forward.lm)
AIC(junk.lm)

#BIC
BIC(forward.lm)
BIC(junk.lm)

#mean squared error
mean((train.clean$SalePrice - predict(forward.lm))^2)
mean((train.df$SalePrice - predict(junk.lm))^2)

#mean absolute error
mean(abs(train.clean$SalePrice - predict(forward.lm)))
mean(abs(train.df$SalePrice - predict(junk.lm)))

#the model junk includes additional variables that were removed in the
model
drop.list <- c(drop.list,"OverallQual","OverallCond","GrLivArea")

#remove not used columns following the models drop.llist
test.clean <-test.df[,(names(elig.pop) %in% drop.list)];
#have to clean the NAs in the test clean as well to avoid the predict
function to complain
test.clean <- na.omit(test.clean)


#predict complains that it Exterior1 has a value not considered in the
model. That is cos the train data selected did not contain that type
'ImStucc'. Therefore remove it.
length(test.clean[test.clean$Exterior1=='ImStucc',]$Exterior1)
#1 record to remove
test.clean<-test.clean[test.clean$Exterior1!='ImStucc',]

forward.test <- predict(forward.lm,newdata=test.clean);
junk.test <- predict(junk.lm,newdata=test.clean);

#mean squared error
mean((test.clean$SalePrice - forward.test)^2)
mean((test.clean$SalePrice - junk.test)^2)

#mean absolute error
mean(abs(test.clean$SalePrice - forward.test))
mean(abs(test.clean$SalePrice - junk.test))
```

```
#Training Data
# Abs Pct Error
forward.pct <- abs(forward.lm$residuals)/train.clean$SalePrice;
junk.pct <- abs(junk.lm$residuals)/train.df$SalePrice;
# Assign Prediction Grades;
forward.PredictionGrade <- ifelse(forward.pct<=0.10,'Grade 1:
[0.0.10]', ifelse(forward.pct<=0.15,'Grade 2: (0.10,0.15]',
ifelse(forward.pct<=0.25,'Grade 3: (0.15,0.25]','Grade 4: (0.25+]')))
forward.trainTable <- table(forward.PredictionGrade)
forward.trainTable/sum(forward.trainTable)

junk.PredictionGrade <- ifelse(junk.pct<=0.10,'Grade 1: [0.0.10]',
ifelse(junk.pct<=0.15,'Grade 2: (0.10,0.15]',
ifelse(junk.pct<=0.25,'Grade 3: (0.15,0.25]','Grade 4: (0.25+]')))
junk.trainTable <- table(junk.PredictionGrade)
junk.trainTable/sum(junk.trainTable)

# Test Data
# Abs Pct Error
forward.testPCT <- abs(test.clean$SalePrice-
forward.test)/test.clean$SalePrice;
#Comment both lines as the model is the same for forward, backward and
stepwise
#backward.testPCT <- abs(test.df$SalePrice-
backward.test)/test.df$SalePrice;
#stepwise.testPCT <- abs(test.df$SalePrice-
stepwise.test)/test.df$SalePrice;
junk.testPCT <- abs(test.clean$SalePrice-
junk.test)/test.clean$SalePrice;

# Assign Prediction Grades;
forward.testPredictionGrade <- ifelse(forward.testPCT<=0.10,'Grade 1:
[0.0.10]', ifelse(forward.testPCT<=0.15,'Grade 2: (0.10,0.15]',
ifelse(forward.testPCT<=0.25,'Grade 3: (0.15,0.25]', 'Grade 4:
(0.25+]')))
forward.testTable <-table(forward.testPredictionGrade)
forward.testTable/sum(forward.testTable)

junk.testPredictionGrade <- ifelse(junk.testPCT<=0.10,'Grade 1:
[0.0.10]', ifelse(junk.testPCT<=0.15,'Grade 2: (0.10,0.15]',
ifelse(junk.testPCT<=0.25,'Grade 3: (0.15,0.25]', 'Grade 4:
(0.25+]')))
junk.testTable <-table(junk.testPredictionGrade)
junk.testTable/sum(junk.testTable)
```