Assignment #5: Automated Variable Selection, Multicollinearity, and Predictive Modeling (100 points)

<u>Data:</u> The data for this assignment is the Ames, Iowa housing data set. This data will be made available by your instructor.

<u>Assignment Instructions:</u>

In this assignment we will begin building regression models for the home sale price (the raw home sale price SalePrice, not any transformation of the home sale price). We will set up a predictive modeling framework, explore the use of automated variable selection techniques for model identification, assess the predictive accuracy of our model using cross-validation, and compare and contrast the difference between a statistical model validation and an application (or business) model validation.

(1) Define the Sample Population

- Define the appropriate sample population for your statistical problem. Hint: As it says in the two sentences one inch above this line, we are building regression models for the response variable SalePrice. Are all properties the same? Would we want to include an apartment building in the same sample as a single family residence? Would we want to include a warehouse or a shopping center in the same sample as a single family residence? Would we want to include condominiums in the same sample as a single family residence?

- Define your sample using 'drop conditions'. Create a waterfall for the drop conditions and include it in your report so that it is clear to any reader what you are excluding from the data set when defining your sample population.

(2) The Predictive Modeling Framework

- A defining feature of predictive modeling is assessing model performance out-of-sample. We will use uniform random number to split the sample into a 70/30 train/test split. With a train/test split we now have two data sets: one for in-sample model development and one for out-of-sample model assessment.

```
# Set the seed on the random number generator so you get the same split every time that
# you run the code.
set.seed(123)
my.data$u <- runif(n=dim(my.data)[1],min=0,max=1);
```

```
# Define these two variables for later use;
my.data$QualityIndex <- my.data$OverallQual*my.data$OverallCond;
my.data$TotalSqftCalc <- my.data$BsmtFinSF1+my.data$BsmtFinSF2+my.data$GrLivArea;

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


# Check your data split. The sum of the parts should equal the whole.
# Do your totals add up?
dim(my.data)[1]
dim(train.df)[1]
dim(test.df)[1]
dim(train.df)[1]+dim(test.df)[1]
```

- Our 70/30 training/test split is the most basic form of cross-validation. We will 'train' each model by estimating the models on the 70% of the data identified as the training data set, and we will 'test' each model by examining the predictive accuracy on the 30% of the data. In R will estimate our models using the `lm()` function, and we will be able to apply those linear models using the R function `predict()`. You will want to read the R help page for the R function `predict()`. In particular pay attention to the `newdata` argument. Your test data set is your new data.

Show a table of observation counts for your train/test data partition in your data section.

(3) Model Identification by Automated Variable Selection

- Create a pool of candidate predictor variables. This pool of candidate predictor variables needs to have 15-20 predictor variables. The variables should be a mix of discrete and continuous variables. Include a well-designed list or table of your pool of candidate predictor variables in your report. **NOTE: If you need to create additional predictor variables, then you will want to create those predictor variables before you perform the train/test split outlined in (2). Also note that we will be using our two variables QualityIndex and TotalSqftCalc in this section.**

The easiest way to use variable selection in R is to use some R tricks. If you have small data sets (small number of columns), then these tricks are not necessary. However, if you have large data sets (large number of columns), then these tricks are NECESSARY in order to use variable selection in R effectively and easily. Trick #1: we need to create a data frame that only contains our response variable and the predictor variables that we want to include as our pool of predictor variables. We will do this by creating a drop list and using the drop list to shed the unwanted columns from `train.df` to create a ‘clean’ data frame.

```
drop.list <- ('SID','PID','LotConfig','Neighborhood','HouseStyle','YearBuilt','YearRemodel',
   'Exterior1','BsmtFinSF1','BsmtFinSF2','CentralAir','YrSold','MoSold','SaleCondition',
   'u','train','I2010','BsmtFullBath','BsmtHalfBath','FullBath','HalfBath',
   'FireplaceInd1','FireplaceInd2','OverallQual','OverallCond','PoolArea','GrLivArea');

train.clean <-train.df[,!(names(my.data) %in% drop.list)];
```

- Model Identification: Using the training data find the 'best' models using automated variable selection using the techniques: forward, backward, and stepwise variable selection using the R function `stepAIC()` from the `MASS` library. Identify (list) each of these three models individually. Name them `forward.lm`, `backward.lm`, and `stepwise.lm`.

Note that variable selection using `stepAIC()` requires that we specify the upper and lower models in the `scope` argument. We want to perform a 'full search' or an 'exhaustive search', and hence we need to specify the upper model as the Full Model containing every predictor variable in the variable pool (or in our clean data frame!), and the lower model as the Intercept Model. Both of these models are easy to specify in R.

Trick #2: specify the upper model and lower models using these R shortcuts.

```
# 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 that all of these models use the `train.clean` data set. We will use these three models to initialize and provide the formula needed for the `scope` argument. Trick #3: use the R function `formula()` to pass your shortcut definition of the Full Model to the `scope` argument in `stepAIC()`. Be sure to read the help page for `stepAIC()` to understand the `scope` argument and its default value.

```
# 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;
library(MASS)

# Call stepAIC() for variable selection
forward.lm <- stepAIC(object=lower.lm,scope=list(upper=formula(upper.lm),lower=~1),
   direction=c('forward'));
summary(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),
   direction=c('both'));
summary(stepwise.lm)
```

Note that we do not specify any data sets when we call `stepAIC()`. The data set is passed along with the initializing model in the `object` argument.

In addition to these three models identified using variable selection we will include a fourth model for model comparison purposes. We will call this model `junk.lm`. The model is appropriately named. Do we know why we are calling this model junk? Note that this model will use the `train.df` data frame since I shed all of these columns off of `train.df` when I created `train.clean`.
Remember that `train.df` and `train.clean` are essentially the same data set, `train.df` just has more columns than `train.clean` so it is perfectly okay to compare models fit on `train.df` with models fit on `train.clean`.

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

Before we go any further we should consider if we like these models. One issue with using variable selection on a pool that contains highly correlated predictor variables is that the variable selection algorithm will select the highly correlated pairs. (Hint: do we have correlated predictor variables in the junk model?)

Compute the VIF values for the variable selection models. If the models selected highly correlated pairs of predictors that you do not like, then go back, add them to your drop list, and re-perform the variable selection before you go on with the assignment. The VIF values do not need to be ideal, but if you have a very large VIF value (like 20, 30, 50 etc.), then you should consider removing a variable so that your variable selection models are not junk too.

Should we be concerned with VIF values for indicator variables? Why or why not?

```
# Compute the VIF values
library(car)
sort(vif(forward.lm),decreasing=TRUE)
sort(vif(backward.lm),decreasing=TRUE)
sort(vif(stepwise.lm),decreasing=TRUE)
```

Did the different variable selection procedures select the same model or different models? Display the final estimated models and their VIF values for each of these four models in your report.

- Model Comparison: Now that we have our final models, we need to compare the in-sample fit and predictive accuracy of our models. For each of these four models compute the adjusted R-Squared, AIC, BIC, mean squared error, and the mean absolute error for each of these models for the training sample. Each of these metrics represents some concept of 'fit'. In addition to the values provide the rank for each model in each metric. If a model is #2 in one metric, then is it #2 in all metrics? Should we expect each metric to give us the same ranking of model 'fit'.

(4) Predictive Accuracy

In predictive modeling we are interested in how well our model performs (predicts) out-of-sample. That is the point of predictive modeling. For each of the four models compute the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) for the test sample. Which model fits the best based on these criteria? Did the model that fit best in-sample predict the best out-of-sample? Should we have a preference for the MSE or the MAE? What does it mean when a model has better predictive accuracy in-sample then it does out-of-sample?

Here is an example of how you use the `predict()` function to score your model on your out-of-sample data.

```
forward.test <- predict(forward.lm,newdata=test.df);
```

(5) Operational Validation

- We have validated these models in the statistical sense, but what about the business sense? Do MSE or MAE easily translate to the development of a business policy? Typically, in applications we need to be able to hit defined cut-off points, i.e. we set a policy that we need to be p% accurate. Let's define a variable called PredictionGrade, and consider the predicted value to be 'Grade 1' if it is within ten percent of the actual value, 'Grade 2' if it is not Grade 1 but within fifteen percent of the actual value, Grade 3 if it is not Grade 2 but within twenty-five percent of the actual value, and 'Grade 4' otherwise.

Here is a code snippet to show you how we will produce the prediction grades.

```
# Training Data
# Abs Pct Error
forward.pct <- abs(forward.lm$residuals)/train.clean$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)


# Test Data
# Abs Pct Error
forward.testPCT <- abs(test.df$SalePrice-forward.test)/test.df$SalePrice;
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.df$SalePrice-junk.test)/test.df$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)
```

Produce these prediction grades for the in-sample training data and the out-of-sample test data. Note that we want to show these tables in distribution form, not counts. Distribution form is more informative and easier for your reader (and you!) to understand, hence we have normalized the table object.

How accurate are the models under this definition of predictive accuracy? How do these results compare to our predictive accuracy results? Did the model ranking remain the same?

Note: The GSEs (Fannie Mae and Freddie Mac) rate an AVM model as ‘underwriting quality’ if the model is accurate to within ten percent more than fifty percent of the time. Are any of your models ‘underwriting quality’?

<u>Assignment Document:</u>

All assignment reports should conform to the standards and style of the report template provided to you. Results should be presented and discussed in an organized manner with the discussion in close proximity of the results. The report should not contain unnecessary results or information. The document should be submitted in pdf format. Name your file Assignment5_LastName.pdf.

Here is a reasonable section outline for this assignment report.

Section 1: Sample Definition and Data Split

- Section 1.1: Sample Definition
    - Provide the waterfall of your drop conditions with counts.
- Section 1.2: The Train/Test Split
    - Show a table of counts for your train/test split.

Section 2: Model Identification and In-Sample Model Fit

- Section 2.1 Forward Variable Selection
- Section 2.2 Backward Variable Selection
- Section 2.3 Stepwise Variable Selection
- Section 2.4 Model Comparison

Section 3: Predictive Accuracy

Section 4: Operational Validation