Project 1: Predict the Housing Prices in Ames (PART I)

You are asked to analyze the housing data collected on residential properties sold in Ames, Iowa between 2006 and 2010.

Download the dataset, "Ames_data.csv", from the Resouces page. The dataset has 2930 rows (i.e., houses) and 83 columns. Column 1 is "PID", the Parcel identification number, the last column is the response variable, "Sale_Price", and the remaining 81 columns are explanatory variables describing (almost) every aspect of residential homes.

The goal is to predict the final price of a home with those explanatory variables.

Source

De Cock, D. (2011). "Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project," Journal of Statistics Education, Volume 19, Number 3. http://ww2.amstat.org/publications/jse/v19n3/decock.pdf

Check variable description at
https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt

This data set has been used in a Kaggle competition (https://www.kaggle.com/c/house-prices-advanced-regression-techniques). You can check how others analyze this data and try some sample code on Kaggle. Note that our data set has two more explanatory variables, "Longitude" and "Latitude", than the one on Kaggle.

What you need to submit?

Before the deadline (Thursday, Oct 18, 11:30pm, Pacific Time) please submit the following two items to the corresponding assignment box on Compass:

- R/Python code
  (.R or .py or zip) that takes a training data and a test data as input and outputs three submission files (details are given below);

- A report
  (3 pages maximum, pdf only) that provides the details of your code, e.g., pre-processing, some technical details or implementation details (if not trivial) of the models you use, etc.

  In addition, report the accuracy (see evaluation metric given below), running time of your code and the computer system you use (e.g., Macbook Pro, 2.53 GHz, 4GB memory, or AWS t2.large). You **DO NOT** need to submit the part of the code related to the evaluation you conduct.

How we evaluate your code?

Name your main file as **mymain.R**. If you have multiple R files, upload the zip file. After unzipping your file, we will run the command "source(mymain.R)" in a directory, in which there are only two files: train.csv and test.csv.

- **train.csv**: about 70% of the whole data "Ames_data.csv" with exactly the same 83 columns;
- **test.csv**: the remaining 30% of the whole data without the last column "Sale_Price", i.e., it has 82 columns.

Build **TWO** prediction models. Always include a tree-based ensemble model, e.g., randomForest, and/or boosting tree.

After running your Rcode, we should see **TWO txt files** in the same directory named "mysubmission1.txt" and "mysubmission2.txt". Each submission file correspoonds to a prediction on the test data.

- **Submission File Format.** The file should have the following format (do not forget the **comma** between PID and Sale_Price):

```
PID,  Sale_Price
528221060,  169000.7
535152150,  14523.6
533130020,  195608.2
```

- **Evaluation Metric.** Submission are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted price and the logarithm of the observed sales price. Our evaluation R code looks like the following:

```
# Assume the Y value for the test data is stored in a two-column
# data frame named "test.y":
# col 1: PID
# col 2: Sale_Price

pred <- read.csv("mysubmission1.txt")
names(test.y)[2] <- "True_Sale_Price"
pred <- merge(pred, test.y, by="PID")
sqrt(mean((log(pred$Sale_Price) - log(pred$True_Sale_Price))^2))
```

  The evaluation process for Python code is the same.

- **Performance Target.** Full credit for submissions with **one RMSE less than 0.132**. Extra credit (2pt) for one accuracy less than **0.120**.

Frequently Asked Questions

- *Will you give us the training and test dataset?*

  The training and test data used for our validation will not be given to students. As mentioned on the project page, you should test your code by conducting a self-evaluation, in which you randomly split the whole data into training and test sets (70% vs. 30%).

- *Should we download the training/test datasets from Kaggle and upload our prediction on Kaggle for evaluation?*

  No, you do not need to download any datasets from Kaggle. The whole data set "Ames_data.csv" is available on the Resources page; you form your own training and test sets by random splitting (70% vs. 30%).

- *Should we include the split data part in the function we write? The variable we use in the function should be only training data or both training and testing data?*

  No, you do not need to include the split data part in your code. Your code should start with loading the train.csv and test.csv. Build your models based on train.csv, and then output your predictions for houses in test.csv. Store your output in txt files

following the Submission File Format described on the project page.

- Are the training and test data already in the Global Environment or we have to read them in?

  You have to read them in. The starting point of your R script may look like this

  ```
  # read in data
  train <- read.csv("train.csv", ...)
  test <- read.csv("test.csv", ...)
  ```

- *Do we need to do diagnostics? Is it important for this data set? If it is, should we delete the extreme values?*

  Diagnostics or any pre-processing for missing values and extreme values are done by you. If you find out that a pre-processing procedure plays an important role in prediction, you should include it in the code and also describe it in the report.

  Please keep in mind that you are NOT asked to report detailed EDA (exploratory data analysis), but to build predictive models based on this data set.

- *The test data could have new levels that do not appear in the training data. This happens frequently with categorical variables that have infrequent levels, when I randomly split the data into two parts. How to handle this situation?*

  Usually, there is no need to consider all levels of a categorical variable, especially the levels with small frequencies (i.e., rare levels). You can, for example, only code the top K most frequent levels, and merge all the remaining levels as "Other". Now any unseen levels in the test data are regarded as "Other."

  "Condition_2" and "Utilities" tend to have just one level in the training data, since the other levels, except a dominate level, are rare. If a categorical variable has only one level in the training data, it's confounded with the intercept. Just ignore that variable when building your model.

  "Overall_Qual" is a ordered categorical variable randing from 1 to 10 based on the data description. Suppose level 7 is missing in the training data, then for a test house with level 7, you can form a prediction with level 6 and one with level 8, and then average them. Or you can merge some levels, say, <=3 to be low, 4 to 7 to be med, and >=8 to be high.

  For "Year_Built", you can create some bins, such as "before 1950", "1951-1965", etc. How to choose the cut-off points? You can build a tree model with the original response variable, i.e., the housing price, as response, and one predictor "Year_Built". You can require the number of leaf nodes for that tree to be a relatively small number, say 12. Then, each leaf node corresponds to an interval/bin of Year_Built. (Ignore this suggestion if you don't know how to fit a tree model; you can use this approach for future projects.)

  You can also view a new level as a missing data problem (now missing in the test not training), and then impute the missing value, e.g., 1) form a prediction with the missing replaced by the most frequent level, 2) form an averaged prediction with the missing replaced by several levels (top K most frequent levels or all levels if the number of levels is relatively small), or 3) replace the missing value by imputation. Some students recommend this R package "mice" for imputation.

- *Apply PCA? This Kaggle post [link] used the first 36 PC's for regression. How to implement it in R?*

  Below we translate the Python code (on Kaggle) to R. Note that the code is based on the Ames data on Kaggle, which is just part of the Ames data (and may be of different format), so the the result from a PCA on our Ames data might be different.

```
train = read.csv('./input/train.csv')
labels = train$SalePrice
test = read.csv('./input/test.csv')
data = rbind(train[, which(! colnames(train) %in% 'SalePrice')], test)
ids = test$Id

head(train)

# Count the number of rows in train
nrow(train)

# Count the number of rows in total
nrow(data)

# Count the number of NaNs each column has.
nans = colSums(is.na(data))
nans[which(nans > 0)]

# Remove columns with more than a thousand missing values
# Will remove the Id's after one-hot-encoding
data = data[, which(! colnames(data) %in% c('Alley', 'Fence',
        'MiscFeature', 'PoolQC', 'FireplaceQu'))]

# Count the column types
table(sapply(data, class))


all_columns = colnames(data)
non_categorical = c("LotFrontage", "LotArea", "MasVnrArea",
                    "BsmtFinSF1", "BsmtFinSF2", "BsmtUnfSF",
                    "TotalBsmtSF", "1stFlrSF", "2ndFlrSF",
                    "LowQualFinSF", "GrLivArea", "GarageArea",
                   "WoodDeckSF", "OpenPorchSF", "EnclosedPorch",
                   "3SsnPorch", "ScreenPorch","PoolArea", "MiscVal")
categorical = setdiff(all_columns, non_categorical)

# One Hot Encoding and nan transformation
data = model.matrix(Id ~ . -1, model.frame(~ ., data = data,
              na.action = na.pass))  # Id removed from data

getMode = function(x){
    x.values = table(x)
    as.numeric(names(x.values)[which.max(x.values)])
}

imp = function(x){
    na.id = which(is.na(x))
    if (length(na.id) > 0){
        x[na.id] = getMode(x)
    }
    x
}

data = apply(data, 2, imp)
```

```
# Log transformation
data = log(data)
labels = log(labels)

# Change -inf to 0 again
data[which(data == -Inf, arr.ind = TRUE)] = 0

# Apply PCA
pca = prcomp(data)

# variance explained by PC's
cumsum(pca$sdev^2) / sum(pca$sdev^2)
```

- *The RMSE of the logarithm of price is infinity/nan for some observations. What should I do?*

  Recall that `log(y)` is only defined for `y > 0`. If you train a model to predict `y` without any constraints it is possible that your model may predict a zero or negative value. These zero or negative predictions will cause infinities and nans to pop up in the metric calculations.

  Although not as prominent in the Ames dataset, this can commonly happen when you switch from optimizing for RMSE to MAE (Mean-Absolute-Error). In the case of zero-inflated regression the intercept only model will predict the median, which is often times zero.

  To solve this issue it is good practice to predict `log(y)` or in this case `log(Sales_Price)` instead of the untransformed target. In other words, what matters in your modeling is `log(Sales_Price)`. You do not build a model to predict price and then plug a log of it into the evaluation metric. You build a model to predict the logarithm of the target and then use normal RMSE to evaluate the model's performance.

  If you decide to predict `log(Sales_Price)`, then you will need to transform the target back using the exponential -- `exp(pred)` -- when submitting your results.
