Ames Iowa House Prediction Project

Jordan Kassof and Jose Torres

2/11/2018

Contents

Introduction

Buying a house is one of the largest financial decisions many people will make. So many factors go into someone's decision, but it can be hard to really explain why one house "felt right" and another didn't. We want to quantify the factors that add up to someone making the decision to purchase a house.

Data Description

We will be using the Ames, Iowa individual residential property sales data set freely available on Kaggle.com. The data set contains 2,930 observations with 79 explanatory variables. All observations occur between 2006 and 2010. Given the geography dependent nature of home value, the results of below analyses can't be applied nationally. For more information on the data, or to download it yourself, visit https://www.kaggle.com/c/house-prices-advanced-regression-techniques.

See the codebook.txt file in the github repository for complete information about all variables.

Exploratory Analysis

Comments on various things noticed as exploring the data and general idea of what the cleaning was. Reference appendix for entire cleaning functions and scripts.

Questions of Interest

We focused on two approaches, one geared towards model performance and one towards model interpretability by one of the parties involves in a home purchase.

Interpretable Models

For the interpretable model approach, rather than just taking a handful of easily understood parameters and building a model, we wanted to take a different approach. There are many adages when it comes to home buying, we wanted to see which are the most true. The three ideas about what drives the price of a house that we looked at are as follows:

- Location, location, location!
  - For this model, we used parameters that are related to the physical location of the property. For example, neighborhood, zoning, frontage, lot size, etc.
- It's all about the curb appeal
  - For this model, we used parameters related to the external appearance of the property. For example, house style, roof style, external veneer materials, etc.
- It's what's on the inside that counts
  - For this model, we used parameters related to the internals of the property, the bones if you will. For example, the foundation, the electrical and heating system, etc.

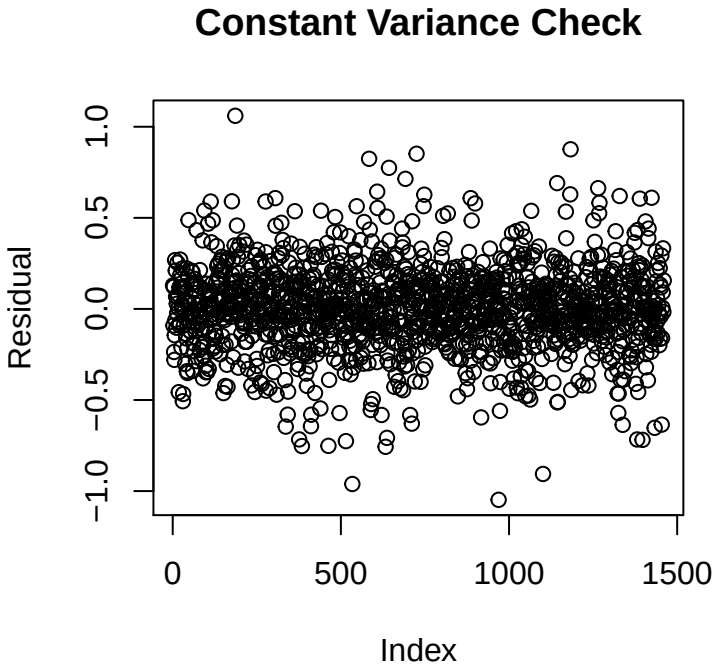
We also included the Sale Condition variable in all three models because the context of the sale seems like way too key of a factor to leave out of any model that is meant to be easily interpreted.

Model Selection

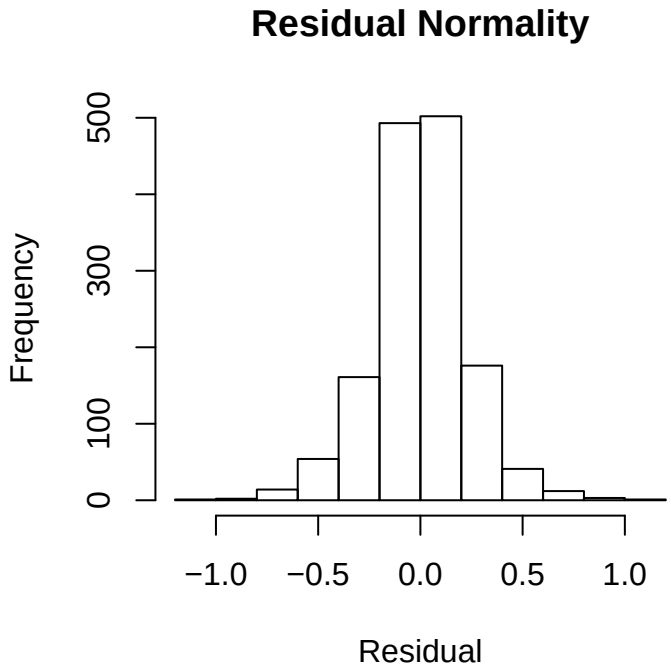
After running our three models, let's take a look at some diagnostics. After picking a model based on diagnostics, we will examine assumptions and parameters.

ModelName	adj.r.squared	AIC	BIC	df
location	0.6567534	-52.13605	185.7426	44
inside	0.6296121	81.06299	440.5240	67
outside	0.5964634	178.29170	384.4532	38

Based on R^2, AIC, and BIC, the best model appears to be the location model. Let's examine some diagnostic plots to make sure the assumptions of linear regression are met.



In the plot above, it appears there is not strong evidence against the assumption of constant variance in our residuals. I see a few points that have potentially high leverage, but nothing too egregious so we can proceed.

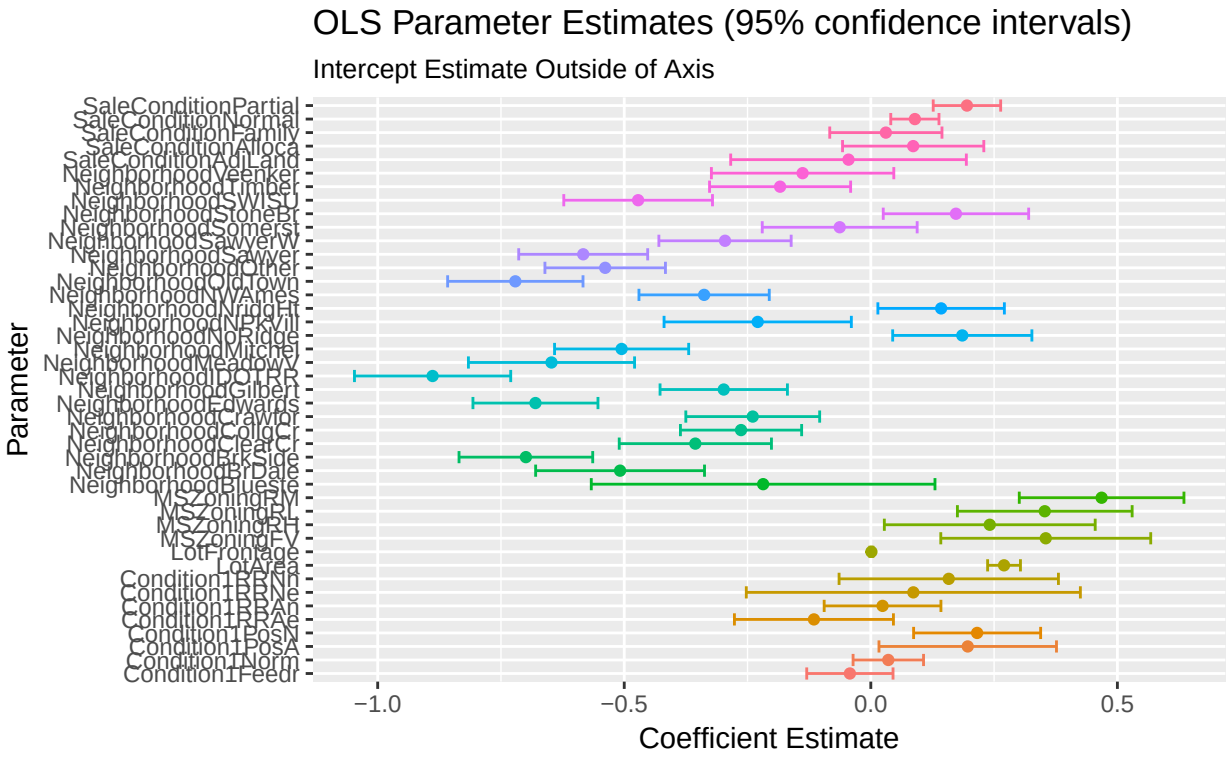


This histogram shows a symmetric distribution, and does not provide strong evidence against normality.

Our final assumption of independent observations, we will assume the data collection was conducted in a way that will provide independent observations. This assumption is probably the weakest as property values within a given city (and more so within a given neighborhood) are fairly dependent on each other. We will proceed with caution.

Parameter Interpretation

Below we will take a cursory look at all parameter estimates, then discuss a few of the parameters with the strongest influence on sale price.



The neighborhood variable has quite a lot of levels, but we can get an idea here for all the parameters estimates and a 95% confidence interval. Let's take a closer look at the five variables with the largest coefficient estimates.

term	estimate	std.error	statistic	p.value	conf.low	conf.high
MSZoningRM	0.4680163	0.0851141	5.498690	0.0000000	0.3010530	0.6349797
MSZoningFV	0.3547451	0.1085082	3.269294	0.0011041	0.1418910	0.5675992
MSZoningRL	0.3527361	0.0903168	3.905542	0.0000984	0.1755670	0.5299053
LotArea	0.2701473	0.0169594	15.929064	0.0000000	0.2368790	0.3034155
MSZoningRH	0.2411929	0.1089905	2.212972	0.0270585	0.0273928	0.4549931

It appears that the zoning variable has the strongest effect on price, as does the size of the lot. The RM Level of the zoning variable means "Residential Medium Density." Bearing in mind that we did a log transform to the sale price (which makes this a log-linear model), we can estimate that a property in this zoning area increases the median sale price by a multiplicative factor of $e^{.458} = 1.5809$. Similarly, zoning classification FV (floating village residential) indicates an $e^{.354} = 1.4248$ multiplier to the median sale price. Our most influential continuous variable, the area of the lot the proper is built on, gives a $e^{.27} = 1$ multiplier to the median sale price for each unit (acre) increase.

Predictive Models

The goal of this section is to make as performant a model as possible. We are not trying to be interpretable or parsimonious, we are primarily optimizing on Kaggle score.

Model Selection

Type of Selection Assumptions Comparing Competing Models AIC, BIC, adj R2 Interval CVPress External Cross Validation Kaggle Score

---

Conclusion

In the battle of the property value tropes, it turns out that the old adage of "location, location, location" is right afterall. We compared models that focused on location, curb appeal, and the interior construction of a property, and found that on all measures, the location centric model was most predictive. This is useful for the parties in a real estate transaction because they know that playing up the location of a property can lead to a higher sale price. On the flipside, if you are looking for a home, it is good to know that you can likely get a good deal on a otherwise very nice property if you are williing to live outside of the premier neighborhoods.

Appendix: Code for all analyses

The below commented code provides the steps taken in order to create, analyze, and select our models focused on interpretability.

```
# Load libraries
# If you don't have one, you will have to run install.packages('library')
library(dplyr)
library(purrr)
library(broom)

train <- read.csv("data/train1_clean.csv")

# Select variables related to the location of a property
location <- train %>%
  select(MSZoning,
         LotFrontage,
         LotArea,
         Neighborhood,
         Condition1,
         SaleCondition,
         SalePrice)
# Select variables related to the external appearance of a property
outside <- train %>%
  select(LotConfig,
         BldgType,
         HouseStyle,
         RoofStyle,
         Exterior1st,
         Exterior2nd,
         MasVnrType,
         MasVnrArea,
         ExterQual,
         ExterCond,
         PavedDrive,
         SaleCondition,
```

```
         SalePrice)

# Select variables related to the internal
inside <- train %>%
  select(Foundation,
         BsmtFinType1,
         BsmtFinType2,
         Heating,
         HeatingQC,
         CentralAir,
         Electrical,
         Fireplaces,
         SaleCondition,
         SalePrice)

# Combine our 3 datasets into a list for easy functional mapping
model_dfs <- list(location, outside, inside)

# Creat a helper function that will generate a model formula
# and run a standard OLS regression of SalePrice against all
# variables for each of our 3 datasets
model_fit <- function(x) {
  model_formula <- formula("SalePrice ~ .")
  lm(model_formula, data = x)
}

# Map our model fitting function above to all 3 datasets
models <- map(model_dfs, model_fit) %>%
  set_names(c("location", "outside", "inside"))

# Add NAmes as a level because it shows up in the test
# data and not the training data
models$location$xlevels$Neighborhood <-
  c(models$location$xlevels$Neighborhood, "NAmes")

# Map the broom::tidy function across our models to get parameter
# estimates in a tidy data frame
model_params <- map(models, tidy)

# Map the broom::glance function across our models to
# get model diagnostics in a tidy data frame
model_diags <- map(models, glance)

# After looking at the below, we can see that location seems to
# be the best model, will continue analysis with that model
bind_cols(
  data.frame(ModelName = c("location", "inside", "outside")),
  bind_rows(model_diags)
)
# Give selected model its own variable for easy reference
loc_lm <- models$location

# Check Assumptions
# Constant variance looks good, one suspicious point but not too
```

```
# bad
plot(loc_lm$residuals)

# Symmetric mostly normal distribution, assumption ok
hist(loc_lm$residuals)

# Read in the test data and get rid of the SalePrice column
# which we added to facilitate prediction in SAS
test <- read.csv("data/test_clean.csv")
test_x <- test[, -which(names(test) == "SalePrice")]

# Map the predict function aross our 3 models to generate
# predictions for submitting to kaggle
preds <- list()
preds <- map(models, predict, newdata = test_x)
```

The below code provides the functions we used for cleaning up the data set.

```
#Libraries

#Functions
factor.Adjust = function(data,adj,narep=FALSE) {
  for (i in seq(nrow(adj))) {
    x = as.character(unlist(unname(adj[i,])))
    feature = x[1]
    replace = x[2]
    with = x[3]

    if (!narep) {

      if (is.factor(data[,feature])) {
        feature.NewLevels = gsub(replace,with,levels(data[,feature]),fixed=TRUE)
        levels(data[,feature]) = feature.NewLevels
      } else
        data[data[,feature]==as.numeric(replace),feature] = as.numeric(with)
    } else {
      my.Class = class(data[,feature])
      data[is.na(data[,feature]),feature] = with
      class(data[,feature]) = my.Class

    }
  }
  return(data)
}

indicator.Add = function(data,indices) {
  for (i in seq(length(indices))) {
    new.Col = names(indices)[i]
    data[,new.Col] = 0
    data[indices[[i]],new.Col] = 1
  }
  return(data)
}

impute.bySampling = function(data) {
```

```
  data.Complete = data[!is.na(data)]
  na.Indices = which(is.na(data))

  sample.Size = length(na.Indices)
  data[na.Indices] = sample(data.Complete,size=sample.Size,replace=TRUE)
  return(data)
}
```

The below code provides the script that calls the above cleaner functions, and does various other data cleaning.

```
# Import data

train <- read.csv('data/train1.csv')
test <- read.csv('data/test.csv')

#Drop unwanted columns
train_drop = c('LandContour','Utilities','LandSlope','Condition2',
                 'SaleType','Street','RoofMatl','X3SsnPorch', 'Functio.1l')

test_drop = c('LandContour','Utilities','LandSlope','Condition2',
                 'SaleType','Street','RoofMatl','X3SsnPorch','PoolQC', 'Fence',
                 'MiscFeature', 'Alley', 'Functional')

train <- train[, !(names(train) %in% train_drop)]
test  <- test[, !(names(test) %in% test_drop)]

# Fix column names
names(train)[names(train)=='X1stFlrSF'] = 'FirstFlrSF'
names(train)[names(train)=='X2ndFlrSF'] = 'SecFlrSF'
names(train)[names(train)=='Kitche.1bvGr'] = 'KitchenAbvGr'
names(train)[names(train)=='Functio.1l'] = 'Functional'
names(train)[names(train)=='ScreenPorch'] = 'ScreenPorchSF'

names(test)[names(test)=='X1stFlrSF'] = 'FirstFlrSF'
names(test)[names(test)=='X2ndFlrSF'] = 'SecFlrSF'
names(test)[names(test)=='Functio.1l'] = 'Functional'
names(test)[names(test)=='ScreenPorch'] = 'ScreenPorchSF'

# Create None factor level for FireplaceQu to represent the NA values
levels(test$FireplaceQu) = c(levels(test$FireplaceQu),'None')

#Identify source of missing data and reassign values
#The custom function factor.adjust will handle both categorical and
# continuous variables

level.Fix =  rbind.data.frame(
  c('MSZoning','C (all)','C'),
  c('LotFrontage','-1','0'),
  c('Neighborhood','-1mes','Other'),
  c('MasVnrType','-1','None'),
  c('MasVnrArea','-1','0'),
  c('BsmtQual','-1','None'),
  c('BsmtCond','-1','None'),
  c('BsmtExposure','-1','None'),
```

```
  c('BsmtFinType1','-1','None'),
  c('BsmtFinType2','-1','None'),
  c('Electrical','-1','SBrkr'),
  c('FireplaceQu','-1','None'),
  c('GarageType','-1','None'),
  c('GarageYrBlt','-1','0'),
  c('GarageFinish','-1','None'),
  c('GarageQual','-1','None'),
  c('GarageCond','-1','None')
)

level.Fix2 =  rbind.data.frame(
  c('MSZoning','C (all)','C'),
  c('Neighborhood','-1mes','Other')
)

#Assign missing values to existing factor levels

level.Fix3 = rbind.data.frame(
  c('MSZoning','NA','RL'),
  c('Exterior1st','NA','VinylSd'),
  c('Exterior2nd','NA','VinylSd'),
  c('MasVnrType','NA','None'),
  c('MasVnrArea','NA','0'),
  c('BsmtQual','NA','TA'),
  c('BsmtCond','NA','TA'),
  c('BsmtExposure','NA','No'),
  c('BsmtFinSF1','NA','0'),
  c('BsmtFinSF2','NA','0'),
  c('BsmtFinType2','NA','Unf'),
  c('BsmtUnfSF','NA','0'),
  c('TotalBsmtSF','NA','0'),
  c('BsmtHalfBath','NA','0'),
  c('KitchenQual','NA','TA'),
  c('FireplaceQu','NA','None'),
  c('GarageCars','NA','2'),
  c('GarageQual','NA','TA'),
  c('GarageCond','NA','TA')
)

names(level.Fix) = c('Feature','Replace','With')

train = factor.Adjust(data=train,adj=level.Fix)
test = factor.Adjust(data=test,adj=level.Fix2)
test = factor.Adjust(data=test,adj=level.Fix3,narep=TRUE)



# Create a predicted SalePrice for the train. Use this line if exporting to
# SAS
test$SalePrice = -1

# Indicator variables which are required for correct parameterization of
# continuous variables which have no value for certain homes
```

```
# e.g. 81 homes have no garage, so we use hasGarage x GarageYrBlt
# instead of dropping GarageYrBlt because of the homes with value 0

new.Indicators.Indices =
  list(
    'idxhasG' = which(train$GarageYrBlt != 0),
    'idxhasMV' = which(train$MasVnrArea != 0),
    'idxhasFB1' = which(train$BsmtFinSF1 != 0),
    'idxhasFB2' = which(train$BsmtFinSF2 != 0),
    'idxhasB' = which(train$TotalBsmtSF != 0),
    'idxhasSF' = which(train$SecFlrSF != 0),
    'idxhasPool' = which(train$PoolArea != 0),
    'idxhasLF' = which(train$LotFrontage != 0),
    'idxhasLQF' = which(train$LowQualFinSF != 0),
    'idxhasWD' = which(train$WoodDeckSF != 0),
    'idxhasOP' = which(train$OpenPorchSF != 0),
    'idxhasEP' = which(train$EnclosedPorch != 0),
    'idxhasSP' = which(train$ScreenPorchSF != 0),
    'idxhasMV' = which(train$MiscVal != 0)
  )

new.Indicators.Indices2 =
  list(
    'idxhasG' = which(test$GarageYrBlt != 0),
    'idxhasMV' = which(test$MasVnrArea != 0),
    'idxhasFB1' = which(test$BsmtFinSF1 != 0),
    'idxhasFB2' = which(test$BsmtFinSF2 != 0),
    'idxhasB' = which(test$TotalBsmtSF != 0),
    'idxhasSF' = which(test$SecFlrSF != 0),
    'idxhasPool' = which(test$PoolArea != 0),
    'idxhasLF' = which(test$LotFrontage != 0),
    'idxhasLQF' = which(test$LowQualFinSF != 0),
    'idxhasWD' = which(test$WoodDeckSF != 0),
    'idxhasOP' = which(test$OpenPorchSF != 0),
    'idxhasEP' = which(test$EnclosedPorch != 0),
    'idxhasSP' = which(test$ScreenPorchSF != 0),
    'idxhasMV' = which(test$MiscVal != 0)
  )

train = indicator.Add(data=train,indices = new.Indicators.Indices)
test = indicator.Add(data=test,indices = new.Indicators.Indices2)


#Imputing by sampling
test$LotFrontage = impute.bySampling(data=test$LotFrontage)
test$BsmtFinType1 = impute.bySampling(data=test$BsmtFinType1)
test$BsmtFullBath = impute.bySampling(data=test$BsmtFullBath)
test$GarageType = impute.bySampling(data=test$GarageType)
test$GarageYrBlt = impute.bySampling(data=test$GarageYrBlt)
test$GarageFinish = impute.bySampling(data=test$GarageFinish)
test$GarageArea = impute.bySampling(data=test$GarageArea)

# Variable Transformations
```

```
log.Transform = c('LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1',
                  'TotalBsmtSF', 'FirstFlrSF', 'SecFlrSF', 'GrLivArea',
                  'WoodDeckSF', 'OpenPorchSF', 'MiscVal', 'SalePrice')

log.Transform2 = c('LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1',
                  'TotalBsmtSF', 'FirstFlrSF', 'SecFlrSF', 'GrLivArea',
                  'WoodDeckSF', 'OpenPorchSF', 'MiscVal')

# Log transform skewed variables
train[,log.Transform] = log(train[,log.Transform] + .001)
test[,log.Transform2] = log(test[,log.Transform2] + .001)


# Merge training and test files, including '.' for empty salePrices for SAS to predict
merge = rbind.data.frame(train,test)

write.csv(train,'data/train1_clean.csv',row.names=FALSE)
write.csv(test,'data/test_clean.csv',row.names=FALSE)
write.csv(merge,'data/merge_clean.csv',row.names=FALSE)


# Below code can be uncommented and run if desired. It generates a PDF with
# histograms and scatter/box plots (against sale price) for each variable in the data set.

# pdf('plots.pdf')
# feature.Levels = lapply(X=train,table)
# for (i in seq(feature.Levels)) {
# f.num = i
# barplot(feature.Levels[[f.num]],main=names(feature.Levels[f.num]))
# plot(x=train[,names(feature.Levels[f.num])],y=log(train$SalePrice),
#      xlab=names(feature.Levels[f.num]))
# }
# dev.off()
# fit = lm(log(SalePrice) ~ train[,names(feature.Levels[f.num])],data=train)
#
```

The below code provides the SAS code used for generating and evaluating our SAS regressions used for finding the most predictive model.