Modeling Apartment Sale Prices in Queens

Final project for Math 390 Data Science at Queens College

May 24, 2020

By Tziporah Horowitz

Abstract

The goal of this paper is to create a predictive model for the sale price of apartments in Queens, New York. Using data on the Queens housing market between February 2016 and February 2017, three predictive models were made via the OLS, Regression Tree, and Random Forest algorithms. By identifying key explanatory features, the three models are obtained and discussed in detail. Of the three, only the Random Forest model is accurate enough to be used in the real world.

1. Introduction

As one of four outer-boroughs of New York City and the most ethnically diverse urban area in the world, Queens County sees a large variation in real estate sale prices each year. Real estate sites like Zillow.com try to estimate the sale price of houses and apartments in a given location, but they are often inaccurate and do not apply to Queens. Using data obtained from the Multiple Listing Service of Long Island, this paper aims to build a mathematical model that can predict future sale prices of Queens apartments.

Mathematical models are abstractions of reality, expressed as functions of measurable data and are used to explain or predict future events of phenomena. Although models inherently contain error due to the nature of unknown causation, they can be useful when metrics of a phenomenon's causal features are defined to minimize that error. Even with properly defined features, the model will not be meaningful without a hypothesis set ($\mathcal{H}$) to allow for various algorithms that can form predictions. Using the *Regression Tree*, *OLS*, and *Random Forest* algorithms, this paper defines three models for predicting the sale price of apartments below \$1,000,000 in Queens.

2. The Data

The data obtained from www.MLSLI.com via Amazon's MTurk has 2,330 rows and 55 columns of character, logical, and numerical type. After importing the data to RStudio running R 3.6.3, the columns created by MTurk's download process were immediately dropped, leaving 31 character and numerical features. Of the 2,330 rows, only 528 contained values of sale price and could be used for prediction. Before dropping the unusable NA values, other columns were cleaned and imputed where necessary. After the data wrangling process, additional data from public.opendatasoft.com was merged via left join on the original dataset to provide three more columns of meaningful geographical data for each zip code.

2.2. Featurization

From the 34 columns in the new dataset, `sale_price` was removed and set aside as `y`, the response variable. Of the remaining columns, 17 were selected for featurization; 6 binary or categorical and 11 numeric.

The categorical variables include `dining_room_type`, `fuel_type`, `kitchen_type`, `cats_allowed`, `dogs_allowed`, `condo_coop`, and `season`; `dogs_allowed` was not included in the linear model because it presented near collinearities with `cats_allowed`. `dining_room_type` is a factor of four categories: combo, formal, dining area, and other. `fuel_type` is also a factor of four categories: oil, gas, electric, and other. `cats_allowed` and `dogs_allowed` are a binary variables: 1 if the pet is allowed in the building and 0 otherwise. `coop_condo` is also a binary variable: 1 if the apartment is a co-op and 0 if it is a condo. All of the categorical variables except for `season` were from the original dataset. `season` was creating by extracting the month of sale from `date_of_sale` and placing them in four bins: winter, spring, summer, and fall.

The numeric variables include `parking_charges`, `sq_footage`, `num_floors_in_building`, `approx_year_built`, `community_district_num`, `num_bedrooms`, `num_full_bathrooms`, `num_total_rooms`, `total_maintenance`, `Latitude`, and `Longitude`. `parking_charges` is the total amount of parking fees associated with a given apartment, expressed on dollars. `num_floors_in_building` is the number of floors in the building, ranging from 1 to 34. `sq_footage` is the area of the apartment in ft^2. `approx_year_built` is the year that the building was in, ranging from 1915 to 2016. `community_district_num` is the community district number that the apartment is located in. `num_bedrooms` is the number of bedrooms in the apartment, ranging from 0 to 3. `num_full_bathrooms` is the number of full bathrooms in the apartment, ranging from 1 to 3. `num_total_rooms` is the number of total rooms in the apartment, ranging from 1 to 8. All variables except for `total_maintenance`, `Latitude`, and `Longitude` were from the original dataset. `total_maintenance` is a combined metric of maintenance fees, common charges, and total taxes, differing between co-ops and condos. `Latitude` and `Longitude` are the geographic coordinates from the additional location dataset, based on the postal code provided in the url of each observation.

2.3. Errors and Missingness

Prior to feature selection, there were many spelling errors in categorical variables. For example, the `cats_allowed` column originally contained three levels: "yes," "y," and "no." The column was then converted to a binary, 1 if `cats_allowed` was "yes" or "y" and 0 if it was "no." Other columns like `parking_charges` were imported as strings like "$1,000" where the "$" and "," needed to be removed to convert the variable to numeric type. Lastly, columns like `date_of_sale` had to be converted from character type to date-time so that it could be used numerically.

Once the non-missing values of the variables were made meaningful, the NA values could be dropped or imputed. NA values in each column were only omitted if there were fewer than 20 missing observations for the feature. After those observations were removed, a subset of the dataset was created by selecting the rows in which `y` was not missing. In this subset, there were 513 remaining rows of data and 6 columns that still contained missing values. The `pct_tax_deductibl` column was removed from the dataset completely because there were too many NA values. Missingness dummies for the 5 columns with missing values were added to the feature set after imputation to see if there is meaning in the missing data. The missing values were imputed in the full dataset using the `missForest` library.

3. Modeling

Using a subset of the imputed dataset concatenated with the missingness dummies where `y` is not missing, three models were built to comparatively predict the sale price of Queens apartments.

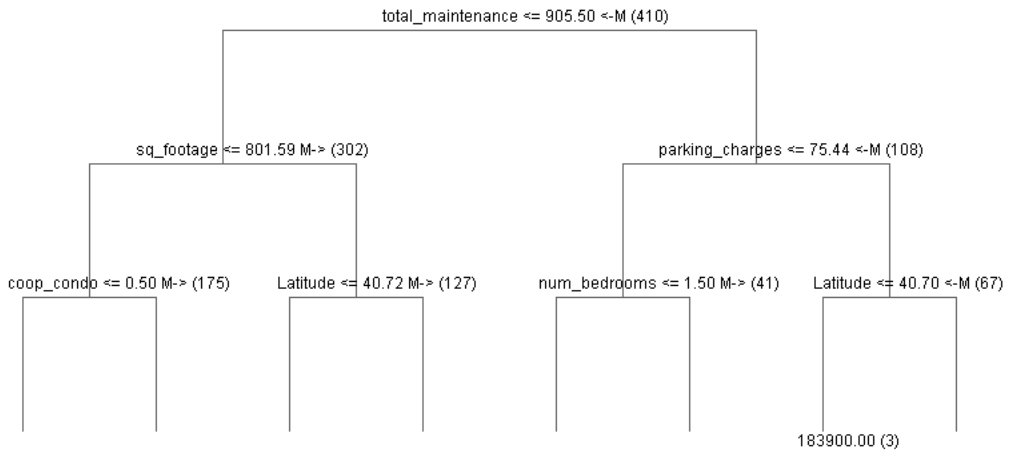
total_maintenance <= 905.50 <-M (410)
sq_footage <= 801.59 M-> (302)
parking_charges <= 75.44 <-M (108)
coop_condo <= 0.50 M-> (175)
Latitude <= 40.72 M-> (127)
num_bedrooms <= 1.50 M-> (41)
Latitude <= 40.70 <-M (67)
183900.00 (3)

Figure 1: First three layers of the regression tree model

3.1 Regression Tree Modeling

Regression trees are used to create step-wise models for prediction using split rules. The algorithm iteratively considers every possible orthogonal-to-axis split and selects the split rule with the smallest Weighted Sum of Squared Error $\left(SSE_{weighted} = \frac{n_l SSE_l + n_r SSE_r}{n_l + n_r}\right.$ where l is the left of the split and r is the right$\left.\right)$ for node construction. Using the *Regression Tree* algorithm, a model was constructed with 21 layers, 157 leaves, and 313 nodes.

Figure 1. shows the first three layers of the tree. Starting from the root node, the most important split rule is `total_maintenance` less than or equal to \$905.50. This is likely because apartments with higher maintenance fees often sell at lower prices. In an apartment with lower maintenance fees, the next most significant feature is `sq_footage` less than or equal to 801.59 ft^2. In general, apartments with smaller square footage cost less than apartments with significantly more; however, because there are many other factors determining sale price, it is not a rule of thumb. If the square footage is below the split, the next split is on `coop_condo`, likely due to both the difference in price and maintenance fees between apartment types. Co-ops tend to be cheaper per square foot but have both property tax and maintenance costs while condos only have common charges. Following the apartment type, the tree splits into two nodes: `Longitude` less than or equal to $-73.82°$ for condos and Latitude less than or equal to $40.72°$ for co-ops. The apartment's geographical location plays an important role in predicting sale price as apartments closer to Manhattan often cost more than those farther away. Furthermore, apartments in low-income geographical areas cost less than those in high-income neighborhoods. Latitude less than or equal to $40.72°$ is also used as a split rule when sq_footage is greater than 801.59 ft^2. Following this split, the tree breaks off into sq_footage less than or equal to 940.56 ft^2 on the left and approx_year_built before 1986 on the right.

On the right side of the tree in Figure 1., the first split rule after total_maintenance greater than \$905.50 is parking_charges less than or equal to \$75.44. This is likely because higher parking fees tend to raise the sale prices of apartments. If parking fees are low, the next split is num_bedrooms less than 2, as sale price tends to increase with the number of bedrooms in an apartment. Following num_bedrooms, the tree splits into approx_year_built before 1957 on the left and Longitude less than or equal to $-73.84°$. Geographical location also plays a large role on the right of parking_charges, with Latitude less than or equal to $40.70°$ leading to the tree's first leaf, a predicted sale price of \$183,900. However, if Latitude is greater than $40.70°$, the tree continues to split on sq_footage less than or equal to 580 ft^2. Although there are too many nodes for the human eye to interpret, observing the regression tree can help one understand which features are most important.

3.2 Linear Modeling

Linear models are useful because they provide meaningful interpretations and error metrics. Table 1. shows the regression outputs of the model, `y ~ 0 + X_train`. The linear model has an in-sample R^2 of 95.78% and an RMSE of ±\$55,500. The in-sample error metrics suggest that the model predicts well; however, when comparing the standard error of the residuals in-sample (\$53,721.85) and out-of-sample (\$126,121.10), the model largely overfits, indicating that it will not perform well in the future.

According to the in-sample model, sale price decreases by \$0.76 for each dollar of maintenance fees when all other variables are held constant. This enforces the theory stated earlier that apartments with higher maintenance fees sell at lower prices[1]. The model similarly asserts the assumption that the larger the square footage of the apartment, the higher the sale price tends to be. When all other variables are held constant, sale price increases by \$136.09 for each square foot. The linear model also suggests that apartment type plays a large role in predicting sale price; if the apartment is a co-op, sale price decreases by \$168,053.00 when all other variables are held constant. The geographic estimates indicate that sale price increases by \$889,756.81 per degree of latitude and decreases \$108,243.26 per degree of longitude[2]. Like the regression tree model, this model suggests that sale prices increase significantly by \$361.87 for each dollar of parking fees.

3.3 Random Forest Modeling

Random Forest is a non-parametric algorithm that utilizes *Bootstrap Aggregating* or bagging to accurately make predictions out-of-sample. Bagging iteratively takes m random samples of approximately $\frac{2}{3}$ of the original dataset with replacement and creates a new model with each sample. The algorithm then calculates the average of the models and its Mean Squared Error ($MSE = \sigma^2 + \rho\mathbb{E}_x\left[\text{Var}[g_{(m)}]\right]$ where ρ is the average correlation between two trees, each built with a different bootstrap sample). *Random Forest* aims to de-correlate the trees during node construction by splitting on a subset of features. By applying this algorithm to the Queens housing data, one can optimize prediction while shrinking the overfit based on the out-of-bag error metrics.

4. Performance of the Random Forest Model

The Random Forest model has an out-of-bag R^2 of 84.64% and an out-of-bag RMSE of 70263.86; meaning the 95% of the time the out-of-bag predictions are within $\pm$\$70,263.86 of the true sale price. When creating a train-test split to estimate the generalization error of the model, out-of-bag

[1]It is important to note however, that while the regression tree suggests total_maintenance is an important feature, the linear model suggests it is insignificant based on a p-value of 0.86.

[2]Because geographical coordinates vary by hundredths of degrees in Queens, these coefficients do not change sale prices as drastically as they suggest.

R^2 was 83.34% in-sample and 85.9% out-of-sample. Furthermore, out-of-bag RMSE was 74749.7 in-sample and 61119.73 out-of-sample, indicating that the model slightly underfit but with low generalization error.

Table 2: Error Metrics

	R^2	RMSE
g_{train}	83.34	74749.7
g_{test}	85.9	61119.73
g_{final}	84.64	70263.86

5. Discussion

The goal of this paper was to create a model that can accurately predict apartment sale prices in the future. To do so, three models were created and discussed in detail. Although the Regression Tree model and Linear model are both inaccurate out-of-sample, they can be used to help a modeler understand the role of key explanatory variables. Although the Random Forest model could do better, it remains accurate out-of-sample and does better than Zillow's "zestimates."

While much of the limitations in the model may stem from misspecification error, the model is largely limited by the dataset as well. The dataset started with 2,330 observations, but nearly 80% of the rows had to be dropped due to the missing values of sale price. Further imputing was needed to model the remaining data points, but too much imputing can make the model dishonest. Although the dataset was usable, future enhancement to the training set, in terms of finding more data, removing unneeded variables, and finding new features can help create a better model with higher predictive abilities.

References

CivicSpace Labs. (2018, February 9). US Zip Code Latitude and Longitude. Open Data Soft. https://public.opendatasoft.com/explore/dataset/us-zip-code-latitude-and-longitude/table/?sort=-latitude.

Code Appendix

Queens Housing Data

Data Wrangling

```
pacman::p_load(skimr, dplyr, magrittr, lubridate, stringr, ggplot2)

# import data
housing =
readr::read_csv("C:\\Users\\Tziporah\\Documents\\Major\\Math\\MATH_390\\housi
ng_data_2016_2017.csv")
zip_data =
readr::read_delim("C:\\Users\\Tziporah\\Documents\\Major\\Math\\MATH_390\\us-
zip-code-latitude-and-longitude.csv", delim = ";")

# omit any column with just 1 value
cols_to_omit = c()
for (i in 1:ncol(housing)){
  if (n_unique(housing[, i]) == 1){
    cols_to_omit = c(cols_to_omit, i)
  }
}

housing %<>%
  # clean the data
  mutate(cats_allowed = ifelse(cats_allowed == "y" | cats_allowed == "yes",
1, 0),
         coop_condo = ifelse(coop_condo == "condo", 0, 1),
         dogs_allowed = ifelse(dogs_allowed == "yes89" | cats_allowed ==
"yes", 1, 0),
         garage_listed = ifelse(!is.na(garage_exists), 1, 0),
         kitchen_type = factor(ifelse(kitchen_type == "eat in" | kitchen_type
== "Eat In" |
                                 kitchen_type == "Eat in" | kitchen_type ==
"eatin", "eat in",
                         ifelse(kitchen_type == "Combo" | kitchen_type ==
"combo", "combo",
                         ifelse(kitchen_type == "efficiency" | kitchen_type ==
"efficiency kitchene" |
                                 kitchen_type == "efficiency kitchen" |
kitchen_type == "efficiemcy" |
                                 kitchen_type == "efficiency ktchen",
"efficiency", kitchen_type)))),
         fuel_type = ifelse(fuel_type == "Other" | fuel_type == "none",
"other", fuel_type),
         maintenance_cost =
as.numeric(str_remove_all(str_remove_all(maintenance_cost, "[$]"), "[,]")),
         parking_charges =
as.numeric(str_remove_all(str_remove_all(parking_charges, "[$]"), "[,]")),
```

```r
housing

# deal with common_charges/maintenence_cost
housing %<>%
  mutate(total_taxes = ifelse(coop_condo == 1 & is.na(total_taxes), 0,
total_taxes),
         maintenance_cost = ifelse(is.na(maintenance_cost), 0,
maintenance_cost),
         common_charges = ifelse(is.na(common_charges), 0, common_charges),
         total_maintenance = common_charges + total_taxes + maintenance_cost)
%>%
  select(-total_taxes, -maintenance_cost, -common_charges) %>%
  filter(!is.na(total_maintenance))


# remove NA y values
housing_sub = housing %<>%
  filter(!is.na(sale_price))
skim(housing_sub)

# make dummies for missing values
colnames(housing_sub)
y = housing_sub[, 16]
X = housing_sub[, -16]
M = tbl_df(apply(is.na(X), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(X), sep = "")
M = tbl_df(t(unique(t(M))))
M %<>% select_if(function(x){sum(x) > 0})
M


# impute on NA's
pacman::p_load(missForest)

colnames(X); skim(X)
vars_to_impute = c(11,6,16,14,15)
no_NAs = setdiff(1:22, vars_to_impute)


x = X[, c(8, no_NAs)]
for (i in vars_to_impute){
  Ximp = missForest(data.frame(x), sampsize = rep(125, ncol(x)), replace =
TRUE)$ximp
  x = cbind(X[, i], Ximp)
}
colnames(x)[1:length(vars_to_impute)] = names(X[, rev(vars_to_impute)])
Ximp = missForest(data.frame(x), sampsize = rep(75, ncol(x)), replace =
TRUE)$ximp
```

```
# Final feature selection
Ximp = cbind(Ximp, M)
Ximp %<>%
  select(-fuel_type.1, -walk_score, -WorkTimeInSeconds, -pct_tax_deductibl, -
is_missing_pct_tax_deductibl,
         -garage_listed)
skim(Ximp)

n_train = 1:410
n_test = 411:length(y)

y_train = y[n_train]
y_test = y[n_test]
X_train = Ximp[n_train, ]
X_test = Ximp[n_test, ]

pacman::p_load(YARF)
options(java.parameters = "-Xmx4000m")

tree_mod = YARFCART(X_train, y_train)
tree_mod
get_tree_num_nodes_leaves_max_depths(tree_mod)
illustrate_trees(tree_mod, max_depth = 4, open_file = TRUE,
length_in_px_per_half_split = 40)

pacman::p_load(xtable)

n_train = sample(1:513, 410)
n_test = setdiff(1:513, n_train)
y_train = y[n_train]
y_test = y[n_test]
X_train = Ximp[n_train, ]
X_test = Ximp[n_test, ]

linear_mod = lm(y_train ~ 0 + ., X_train)
summary(linear_mod)$sigma
summary(linear_mod)$r.squared
xtable(linear_mod)

yhat_oos = predict(linear_mod, X_test)

## Warning in predict.lm(linear_mod, X_test): prediction from a rank-
deficient fit
## may be misleading

oos_e = y_test - yhat_oos
sd(linear_mod$residuals)
sd(oos_e)

pacman::p_load(mlr)
```

```
mlr_data = cbind(y, Ximp)
colnames(mlr_data)[1] = "sale_price"

task = makeRegrTask(data = mlr_data, target = "sale_price")

## Warning in makeTask(type = type, data = data, weights = weights,
## blocking = blocking, : Empty factor levels were dropped for columns:
## dining_room_type,kitchen_type

mlr_mod = resample(makeLearner("regr.randomForest"), task,
makeResampleDesc("CV", iters = 10), measures = list(rmse))

## Resampling: cross-validation

## Measures:             rmse

## [Resample] iter 1:    62702.3401010

## [Resample] iter 2:    89909.5285509

## [Resample] iter 3:    70173.6232784

## [Resample] iter 4:    78371.6501371

## [Resample] iter 5:    66791.9801367

## [Resample] iter 6:    70806.7796705

## [Resample] iter 7:    65678.3470702

## [Resample] iter 8:    59945.4718041

## [Resample] iter 9:    58146.7140569

## [Resample] iter 10:   75406.9022642

##

## Aggregated Result: rmse.test.rmse=70378.1745386

##

sd(unlist(mlr_mod$measures.test))

rf_mod = YARF(Ximp, y, num_trees = 320)
rf_mod
#illustrate_trees(rf_mod, max_depth = 3, open_file = TRUE,
length_in_px_per_half_split = 30)

n_train = sample(1:513, 410)
n_test = setdiff(1:513, n_train)
y_train = y[n_train]
y_test = y[n_test]
X_train = Ximp[n_train, ]
```

```
X_test = Ximp[n_test, ]


rf_is_mod = YARF(X_train, y_train, num_trees = 320)
rf_is_mod
yhat = predict(rf_is_mod, X_test)
oos_rmse = sqrt(mean((y_test - yhat)^2))
oos_rsq = 1 - sum((y_test - yhat)^2)/sum((y_test - mean(y))^2)
oos_rmse
oos_rsq
```

Table 1: Linear Model (In-Sample)

	Estimate	Std. Error	t value	Pr(>|t|)
parking_charges	361.8722	82.7363	4.37	0.0000
sq_footage	136.0939	32.8910	4.14	0.0000
dining_room_typecombo	-42811915.3953	7652419.2110	-5.59	0.0000
dining_room_typedining area	-42843715.9091	7656467.5956	-5.60	0.0000
dining_room_typeformal	-42806314.6291	7652835.4925	-5.59	0.0000
dining_room_typeother	-42792148.1625	7653466.9771	-5.59	0.0000
num_floors_in_building	4446.3163	680.0970	6.54	0.0000
fuel_typegas	-8331.6065	23975.7382	-0.35	0.7284
fuel_typeoil	-5804.8785	24317.7857	-0.24	0.8115
fuel_typeother	6005.0612	31748.3807	0.19	0.8501
approx_year_built	-686.4603	283.9789	-2.42	0.0161
cats_allowed	18789.1958	5902.9989	3.18	0.0016
community_district_num	1888.4452	1005.4334	1.88	0.0611
coop_condo	-168053.0069	15393.7237	-10.92	0.0000
kitchen_typeeat in	-1033.1223	9249.0530	-0.11	0.9111
kitchen_typeefficiency	-13387.4088	8975.0735	-1.49	0.1366
num_bedrooms	35740.8667	7396.8283	4.83	0.0000
num_full_bathrooms	20349.4848	14492.5098	1.40	0.1611
num_total_rooms	6802.4141	5109.9016	1.33	0.1839
seasonspring	-12399.4676	8266.8543	-1.50	0.1345
seasonsummer	-14199.6609	8168.4213	-1.74	0.0830
seasonwinter	4328.8330	8083.6936	0.54	0.5926
Latitude	889756.8071	97643.0803	9.11	0.0000
Longitude	-108243.2589	71552.8031	-1.51	0.1312
total_maintenance	-0.7567	4.3879	-0.17	0.8632
is_missing_dining_room_type	-133.4281	7021.3286	-0.02	0.9848
is_missing_fuel_type	-6043.4737	14653.0562	-0.41	0.6803
is_missing_num_floors_in_building	-879.8882	7240.2965	-0.12	0.9033
is_missing_parking_charges	-3605.1479	6821.8142	-0.53	0.5975
is_missing_sq_footage	-7630.8312	6276.4969	-1.22	0.2248

Residual standard error: 55500 on 380 degrees of freedom
Multiple R-squared: 0.9578, Adjusted R-squared: 0.9545
F-statistic: 287.5 on 30 and 380 DF, p-value: < 2.2e-16