NYC property prices Project

Andreea Ariadna Szilagyi

27/5/2020

Building a machine learning model - predict sale values for real estate properties in New York

INTRODUCTION

This project describes an approach to building a machine learning model to predict sale values for real estate properties in NY.

The dataset used for analysis is a record of every building or building unit (apartment, etc.) sold in the New York City property market over a 12-month period - September 2016 to September 2017. It contains the location, type, year of building, square footage, sales price of the properties.

The dataset uses the financial definition of a building/building unit, for tax purposes. In case a single entity owns the building in question, a sale covers the value of the entire building. In case a building is owned fractional by its residents (a condominium), a sale refers to a single apartment (or group of apartments) owned by some individual.

The initial data includes a digit code for the borough the property is located in; in order these are Manhattan (1), Bronx (2), Brooklyn (3), Queens (4), and Staten Island (5). Throughout the process of data cleaning, this will be amended for better visual.

Many sales occur with a nonsensically small dollar amount: $0 most commonly. These sales are actually transfers of deeds between parties: for example, parents transferring ownership to their home to a child after moving out for retirement. Throughout the process of data cleaning, these will be eliminated for better analysis results.

The goal of this project is to try a few machine learning algorithms and find the best one to predict sale values for real estate properties in New York. The outcome to be predicted is the Sales Price of the properties, and the features that will be used to predict these values are the square footage of the property, the year of the building and the borough where the property is located.

For trying to predict a continuous numeric variable (the Sale Price), regression-based algorithms are the most appropriate, with RMSE as evaluation metric.

RMSE, the Root Mean Squared Error, can be interpreted similarly to a standard deviation: it's the average deviation of the predictions from the observations, the typical error we make when predicting the sales price. The lower the RMSE, the better the model. A good model should, on average, have better predictions than the simple estimate of the mean for all predictions. Thus, the measure of variation (RMSE) should reduce the randomness better than the standard deviation.

To this end, a few machine learning algorithms will be modeled: logistic regression, regression tree, K-nearest neighbors. The best-performing model will be the model with the lowest RMSE.

ANALYSIS

```r
# Import data
zip_url <- "https://github.com/AriadnaSzy/NYC-Property-Sales/raw/master/nyc-property-sales.zip"
download.file(zip_url, destfile = "nyc-property-sales.zip")
nyc_csv_file <- read.csv(unz("nyc-property-sales.zip", "nyc-rolling-sales.csv"),
                         stringsAsFactors = FALSE)

# Preprocessing of data:
# Keep only columns of interest, and rename
sales <- nyc_csv_file %>% select("BOROUGH", "NEIGHBORHOOD", "BUILDING.CLASS.CATEGORY",
                                 "LAND.SQUARE.FEET", "GROSS.SQUARE.FEET", "YEAR.BUILT",
                                 "SALE.PRICE", "SALE.DATE")
names <- c("Borough", "Neighborhood", "Building_class", "Land_sq_ft", "Gross_sq_ft",
           "Year_built", "Sale_price", "Sale_date")
colnames(sales) <- names

# Modify Borough values with actual borough names
codes <- data.frame(
  code = c(1, 2, 3, 4, 5),
  borough = c("Manhattan", "Bronx", "Brooklyn", "Queens", "Staten Island")
  )
sales$Borough <- as.character(codes$borough[match(sales$Borough, codes$code)])


# Cleaning of data type - from character to date and numeric,
# to reflect the variables' characteristics
sales$Sale_date <- as.Date(sales$Sale_date)
```

```r
numerics <- sales %>% select(Land_sq_ft, Gross_sq_ft, Sale_price) %>%
                mutate_if(is.character, as.numeric)
```

```r
numerics[is.na(numerics)] <- 0
sales <- sales %>% select(-Land_sq_ft, -Gross_sq_ft, -Sale_price) %>% cbind(numerics)


# Perform further data cleaning - eliminate NAs, 0 values, outliers, that would influence analysis

sales <- sales %>% filter(!is.na(sales$Sale_price) & sales$Sale_price!=0)
```

```r
sales <- sales %>% filter(sales$Sale_price < price_out)

sales <- sales %>% filter(!is.na(sales$Year_built) & sales$Year_built!=0)
```

```r
sales <- sales %>% filter(sales$Year_built > year_out)

sales <- sales %>% filter(!is.na(sales$Gross_sq_ft) & sales$Gross_sq_ft!=0)
```

```r
sales <- sales %>% filter(sales$Gross_sq_ft < square_out)


# Find out how the 'year of the building' and the 'square footage' influence the 'sales price'
## Year of Building actually has a negative relation with the Sales Price,
## while the Square Footage has a weak positive correlation.
```

```
sales %>% summarize(correlation = cor(Sale_price, Year_built)) %>% pull(correlation)
```

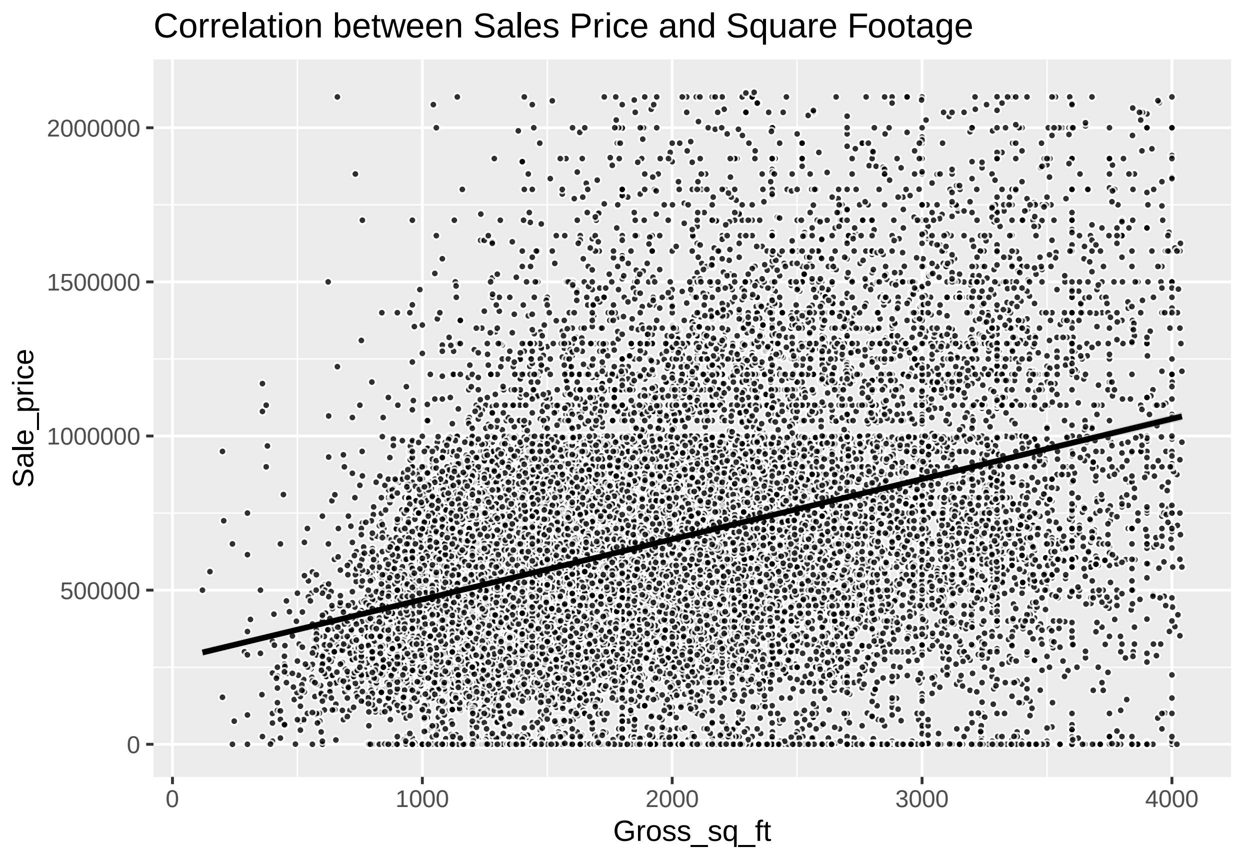
```
## [1] -0.1064797
```

```
sales %>% summarize(correlation = cor(Sale_price, Gross_sq_ft)) %>% pull(correlation)
```

```
## [1] 0.3792538
```

```
# As per the resulted correlation, a visual exploration also reveals a positive relationship
# between Sales Price and Square Footage:
```

```
sales %>%
  ggplot(aes(x = Gross_sq_ft, y = Sale_price) ) +
  geom_point(pch = 21, size = 1, color = "white", fill = "black", alpha = 0.8) +
  geom_smooth(method = "lm", colour = "black") +
  ggtitle("Correlation between Sales Price and Square Footage")
```



```
### Proceed with data analysis through machine learning techniques

# Split the data into training set for algorithms modeling,
# and test set to assess the accuracy of the implemented models
# Validation set will be 20% of NYC Sales data
set.seed(1)    # for reproducibility
test_index <- createDataPartition(y = sales$Sale_price, times = 1, p = 0.2, list = FALSE)
train_set <- sales[-test_index,]
test_set <- sales[test_index,]
```

```
# In analysing the data, linear regression  as a baseline approach
# can represent a valid method of analysis.
# This model can be fitted with the below syntax, resulting in the following:
# The intercept (271724) represents the predicted Sale Price when Square Footage is at 0
# (which doesn't make much sense in this case), whereas the slope (197) represents
# the change in Sale Price when Square Footage increases by one unit.
fit_lm <- lm(Sale_price ~ Gross_sq_ft, data = train_set)
fit_lm$coef
```

```
## (Intercept) Gross_sq_ft
##  271723.841     196.917
```

```
# An initial analysis, of average and standard deviation, provides the following:
# Average price of a property is 642132, with a standard deviation of 373826.
params <- train_set %>%
  summarize(avg = mean(Sale_price), sd = sd(Sale_price))
params
```

```
##        avg       sd
## 1 642132.1 373825.5
```

```
## For further estimating of Sales Prices, will be using several predictors
# instead of just the square footage.

# Using the RMSE evaluation metric, will write a function
# that computes this RMSE for the prices and their corresponding predictors:
RMSE <- function(true_prices, predicted_prices){
  sqrt(mean((true_prices - predicted_prices)^2))
}

# The estimate that minimizes the RMSE is the least squares estimate of the mean,
# in our case the average of all prices:
avg <- mean(train_set$Sale_price)
avg
```

```
## [1] 642132.1
```

```
# If we predict all unknown prices with this average, we obtain the following RMSE,
# that is very close to the standard deviation of our prices distribution:
avg_rmse <- RMSE(test_set$Sale_price, avg)
avg_rmse
```

```
## [1] 374993.3
```

```
# A good prediction model should, on average, have better predictions
# than the simple estimate of the mean for all predictions.
# Will therefore test a few algorithms for the purpose of getting better predictions.
```

In order to avoid overtraining when optimizing algorithm parameters, we will use cross-validation. Overtraining is the reason we have higher accuracy in the training set compared to the test set.

Cross-validation is also known as a resampling method because it involves fitting the same statistical method multiple times using different subsets of the data. The k-fold cross-validation method evaluates the model performance on different subset of the training data and then calculates the average prediction error rate. We want to choose right value of k, the value that maximizes accuracy, or minimizes the expected MSE. Lower value of k is more biased and hence undesirable; higher value of k is less biased, but can suffer from large variability. In practice, using k = 10 has been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

Throughout the project, ten-fold cross-validation will be used.

3 models will be assessed within the project: logistic regression, regression tree, K-nearest neighbors.

Logistic Regression is an alternative method to the simpler Linear Regression. Linear regression tries to predict the data by finding a linear – straight line – equation to model or predict future data points, while Logistic Regression uses the natural logarithm function to find the relationship between the variables and uses test data to find the coefficients. The function can then predict the future results using these coefficients in the logistic equation. The logistic regression model will be fit using the glm (generalized linear model) function.

A Regression Tree may be considered as a variant of decision trees, designed to approximate real-valued functions, instead of being used for classification methods. A regression tree is built through a process known as binary recursive partitioning, which is an iterative process that splits the data into partitions or branches, and then continues splitting each partition into smaller groups as the method moves up each branch. The regression tree model will be fit using the rpart function, with the use of two tuning parameters used for partition decision: the complexity parameter (cp) and the minimum number of observations required in a partition before partitioning it further (minsplit). The complexity parameter (cp) is used to control the size of the decision tree and to select the optimal tree size.

K-nearest Neighbors uses labeled input data set to predict the output of the data points, mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to. Choosing the value of k (determining the number of nearest neighbors) plays a significant role in determining the efficacy of the model. Thus, selection of k will determine how well the data can be utilized to generalize the results of the kNN algorithm.
Just as choosing the k for the k-fold cross-validation, we'll be using k = 10 to avoid excessively high bias or very high variance.

The choice of tuning parameters for the models is always going to be arbitrary to some extent. The objective is to find the value that produces the best results for the specific problem. Arriving at the best combination of parameter values is a case of trial and error.

Lots of combinations have been experimented with as part of the process, to find the ones that improved model performance. The code presents the final tuning parameters.

```r
# use ten-fold cross-validation to estimate the prediction error.
# specify the resampling scheme, that is, how cross-validation
# should be performed to find the best values of the tuning parameters:
control <- trainControl(method = "cv", number = 10)


# fit the logistic regression model:
model_glm <- train(Sale_price ~ Gross_sq_ft + Year_built + Borough,
                 data = train_set,
                 method = "glm",
                 metric = "RMSE",
                 trControl = control)
predictions_glm <- model_glm %>% predict(test_set)
rmse_glm <- RMSE(predictions_glm, test_set$Sale_price)

# fit the regression tree model:
model_rpart <- train(Sale_price ~ Gross_sq_ft + Year_built + Borough,
                   data = train_set,
                   method = "rpart",
                   metric = "RMSE",
                   tuneGrid = data.frame(cp = seq(0, 0.05, len = 10)),
                   minsplit = 2,
                   trControl = control)
predictions_rpart <- model_rpart %>% predict(test_set)
rmse_rpart <- RMSE(predictions_rpart, test_set$Sale_price)

# fit the K-nearest neighbors (kNN) model:
model_knn <- train(Sale_price ~ Gross_sq_ft + Year_built + Borough,
                 data = train_set,
                 method = "knn",
                 metric = "RMSE",
                 tuneGrid = data.frame(k = seq(5,25,2)),
                 trControl = control)
predictions_knn <- model_knn %>% predict(test_set)
rmse_knn <- RMSE(predictions_knn, test_set$Sale_price)
```

, ,

RESULTS

```
methods <- c("logistic regression", "regression tree", "K-nearest neighbors")
rmse <- c(rmse_glm, rmse_rpart, rmse_knn)
rmse_results <- tibble(method = methods, RMSE = rmse)
rmse_results %>% arrange(RMSE)
```

```
## # A tibble: 3 x 2
##   method                 RMSE
##   <chr>                 <dbl>
## 1 logistic regression 328789.
## 2 regression tree     332993.
## 3 K-nearest neighbors 351055.
```

, ,

Based on the above analysis, the best-performing model is logistic regression. This model is considered easy to implement and very efficient to train, and proved its efficacy by providing the best evaluation metric – the lowest RMSE of the modeled algorithms.

CONCLUSION

Logistic regression turned out to be the best-performing model - the model that yielded the lowest RMSE.
Reaching a lower RMSE is a challenge faced especially through the process of experimenting and choosing more adequate tuning parameters for improving the model. It could be a good idea to further try and optimize the model's code to find the best parameters.