Supervised Learning - Analysis and prediction of house prices

Poddaturi, Dinesh R

1/30/2022

Introduction

In this project I use multiple supervised learning methods to carefully analyze and predict house prices. Originally this project was a part of Kaggle data science competition. Although I didn't win the competition, I learned a lot during and after the competition. I believe in the mantra "learning by doing" (coined by a famous American philosopher *John Dewey*). So I have been working on this project by making multiple changes, trying different algotithms, eventually improving the prediction power.

Data and Methodology

The data are free to download from Kaggle website. I downloaded the data a while ago and has been working with the same data. There are two different files of data; (1) Training data containing the price of the house and house characteristics (e.g., the year it was constructed, number of bed rooms, number of bathrooms, latitude and so on) and (2) Testing data containing only the characteristics of the house. Our goal simply is to predict the house prices in the testing data. In order to do that, first we analyse the training data using multiple supervised learning methods and use the models to predict the house price in the testing data.

One could immediately say this is not a classification problem. Note that the house prices are not boolean i.e., 0 or 1, instead they are discrete variable (one could argue it is a continuous variable, but I stay away from that argument in this analysis). So this could be considered as a regression problem. From my rigorus training in Economics, I can immediately identify a problem with the data. Endogeneity; People choose to live in a particular area. It could depend on the school district (people with kids), location (near to a metro or a mall), clean air, less noise pollution and so on. We cannot observe all these in the data. Although we have latitude and zip code, we cannot observe individual decision process and on what basis an individual makes choices. Hence, there could be endogeneity in the data. In this work, I do not focus on that issue. The primary objective of this work is to use supervised learning methods to predict the house prices. Furthermore, this is not a causal study (I am not claiming any causality and not claiming a certain variable causes the house price to increase or decrease). So, I stay away from endogeneity and self selection issues. This is a pure supervised learning exercise.

Data description:

Variable description * id - (in the test set only) number of the test case * property - a unique identifier for a house * date - date house was sold (YYYYMMDD) * price - house selling price (prediction target) * bedrooms - number of bedrooms in house * bathrooms - number of bathrooms in the house * sqft_living - square footage of house * sqft_lot - lot area * floors - total floors/levels in the house * waterfront - indicator that the house has a view to a waterfront * view - has been viewed (??by whom for what and when?? * condition - an overall condition rating * grade - an overall grade given to the housing unit per a King County grading system * sqft_above - square footage apart from basement * sqft_basement - square footage of basement * yr_built - year of initial construction * yr_renovated - year when house was renovated * zipcode - US Postal Service zipcode * lat - latitude * long - longitude * sqft_living15 - square footage of house in 2015 * sqft_lot15 - lot size in 2015

```r
# install.packages("librarian")

librarian::shelf(stringr, Matrix, glmnet, xgboost, randomForest, Metrics, caret, scales, e1071, corrplo
                 psych, tidyverse, lubridate, pls, gdata, graphics, rpart, gbm, earth, Boruta, ggcorrpl
```

```r
# Reading training and testing data
train_data <- read.csv("./Data/train.csv")
test_data <- read.csv("./Data/test.csv")

### Converting date to ymd format
train_data$date <- ymd(train_data$date)
train_data <- train_data %>% select(price:sqft_lot15)
```

Preliminary analysis:

In this section, I perfom some preliminary analysis. Preliminary analysis include carefully studying each variable (house characteristic), it's data type (boolean, string, integer, character etc.), each variables correlation with outcome, importance of each variable, and whether the data type of a variable needs to be modified.

Simple correlation analysis:

```r
housePrice <- train_data$price
correlationMatrix <- cor(cbind(housePrice,train_data %>% select(-price)))
correlationMatrix_sorted <- as.matrix(sort(correlationMatrix[,'housePrice'], decreasing = TRUE))
correlationMatrix_sorted
```

From the correlation matrix the predictors sqft_living, grade, sqft_above, sqft_living15, and bathrooms are highly correlated and Zipcode, longitude, condition, and yr_built are less correlated with the price.

Multiple Linear Regression (MLR) to find the important variables:

```r
mreg <- lm(price ~ . , data=train_data)

an <- anova(mreg)

impVar_lm <- varImp(mreg)
```

The above code snippet simply performs MLR using training data, and using the fitted model gives variable importance and their significance. From anova results, I conclude that almost all the explanatory variables are significant explaining the house price. However, the variable importance method concludes that bedrooms, sqft_living, waterfront, grade, yr_built, lat are the most important variables. By close analysis I find other variables important as well.

Using eXtreme Gradient Boosting (XG Boost) to further analyze variable importance:

```r
designX <- train_data %>% select(-price)
xgBoost_fit <- xgboost(data.matrix(designX), train_data$price, nrounds=800)
xgBoost_Imp <- xgb.importance(model = xgBoost_fit, feature_names = names(designX))
xgBoost_Imp
```

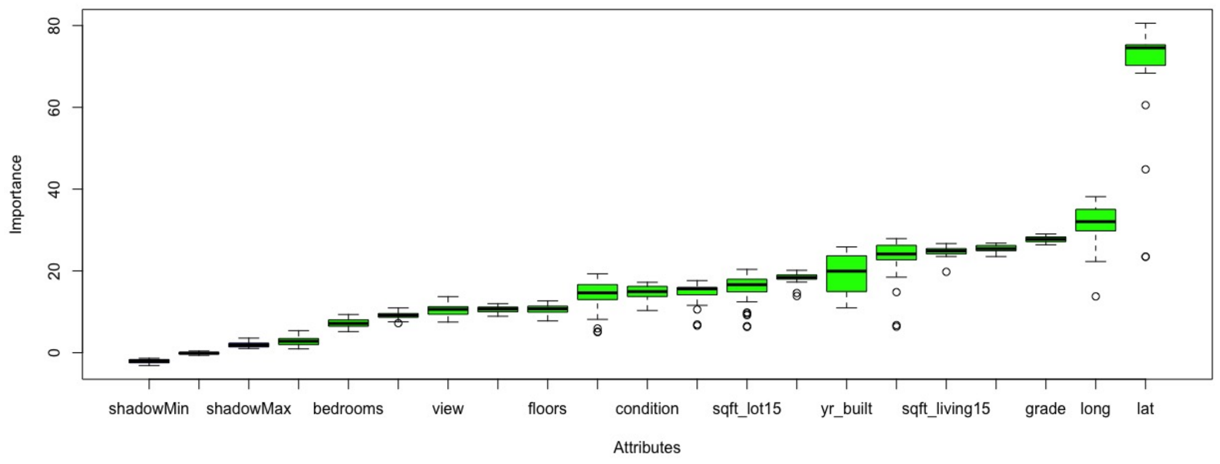
Recently XGB has become a go-to algorithm in machine learning predictions and also in data science competitions. Execution speed and model performance are primary feature of this algorithm. I use this algorithm later to train the data. The above code snippet fits the training data and gives the important variables to consider (not remove). The XG boost selectd sqft__living, grade, lat, long, sqft__living15, yr__built, and waterfront as the most important variables. Similar to the variable importance from MLR, XG boost selected similar variables as important.

Boruta feature selection algorithm to determine the variable importance:

```r
boruta_fit <- Boruta(price~. , data = train_data , doTrace = 2)
print(boruta_fit)
plot(boruta_fit)
colMeans(boruta_fit$ImpHistory)
```

Boruta feature selection algorithm is also widely used to find the variable importance. The above Boruta performed 68 iterations in 15.8 mins, and 18 attributes confirmed important including bathrooms, bedrooms, condition, floors, grade and 13 more. No attributes deemed unimportant from the fit. Boruta also provides a simple visualization of the variable importance.

Figure 1: Boruta fit plot



From all the tests for variable importance above, I conclude that almost all the variables are important. Now, I read description of each variable and use my judgement to drop, modify, or change the data type.

1. I remove *date* from the data since the houses sold are in the same year.
2. I change *zipcode* to *factor* variable since it is not an integer/numeric.
3. I change *waterfront* to *factor* variable since it's data type is binary (0 or 1).
4. I change *condition* to *factor* variable since it is a rating.
5. I drop *sqft__living*, *sqft__lot*, *sqft__above*, and *sqft__basement*. This is because *sqft__living15* and *sqft__lot15* contain the most recent information about the house. Although more information is better. Including all the variables might make our design matrix a singular matrix. Although the algorithms we are going to use would drop those variables, I drop them to make the data as clean and informative as possible.

The above changes are made in both training and testing data.

Implementing Supervised learning algorithms:

For every algorithm, I provide a tune grid and also ask the algorithm to perform cross validation. In particular, I choose cross validation with repeats. Of course repeated cross validation will increase the execution time. But I prefer good results over time. With increased computing power we have now, we can use HPC or even use parallel computing to execute the code. So a reapeated cross validation is used with 2 repeats. Since I am doing this in my machine I chose 2 repeats. If it were to be executed on HPC I would choose 20 repeats.

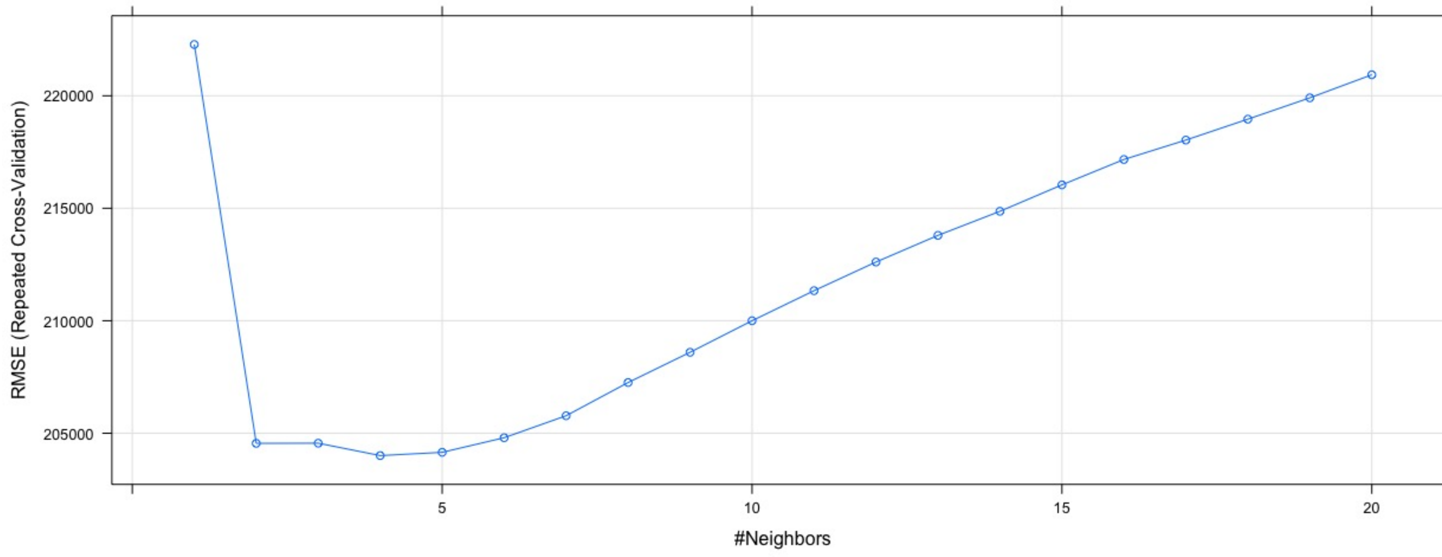
k-Nearest Neighbour:

```
knnTune_housePrice <- train(y = house_train[,1], x = house_train[,2:13],
                            method = "knn",
                            preProcess = c("center","scale"),
                            tuneGrid = data.frame(.k=1:20),
                            trControl = trainControl(method = "repeatedcv",
                                                     repeats = 2, number = 10))

##### I standardize the data and perform repeated cross validation. Basically I perform 10-fold cross v
##### with repetition (Note: As the number of repetitions increase, the execution time also increases).
##### I am also asking the algorithm to try values of k from 1 to 20.

plot(knnTune_housePrice)
knnBestTune <- knnTune_housePrice$bestTune
#### From the plot and bestTune above, the model selected k = 4
# knnTune_housePrice$results
```

Figure 2: kNN Tuned Plot



From the plot above, the algorithm picked $k = 4$ as the best tune. From the plot, we can observe that for $k = 4$ the Root Mean Squared Error (RMSE) is the lowest.

Neural Network:

First we give the algorithm a grid of size and decay to look over. This is to give soime direction the algorithm and to make sure to get the global optimum. The values chosen for the size and decay are widely practiced in the literature. *Size* is the nuber of units hidden layers and *Decay* is the regularization parameter to avoid over-fitting.

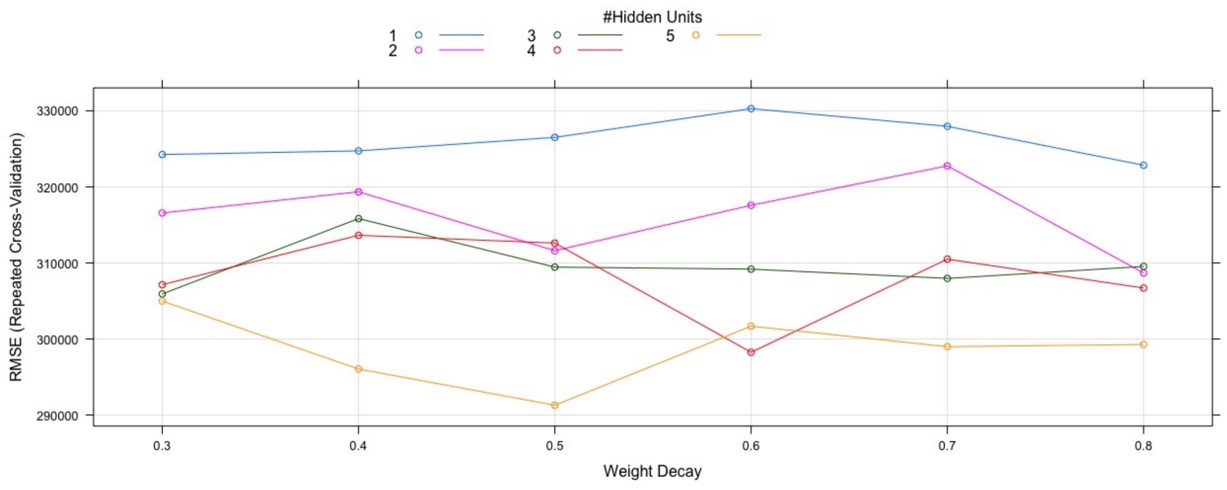
```r
nnet.grid <- expand.grid(size = seq(from = 1, to = 5, length.out = 5),
                         decay = seq(from = .3, to = .8, length.out = 6))

nnetTune_housePrice <- train(y = house_train[,1], x = house_train[,2:13],
                           method = "nnet", trace = FALSE,
                           preProc = c("center","scale"),
                           linout = TRUE, tuneGrid = nnet.grid,
                           maxit = 50,
                           trControl = trainControl(method = "repeatedcv",
                                                    repeats = 2, number = 10) )

plot(nnetTune_housePrice)

nnetBestTune <- nnetTune_housePrice$bestTune
# size decay
# 5   0.5
nnetResults <- nnetTune_housePrice$results
```

Figure 3: Neural Net Tuned Plot


#Hidden Units
RMSE (Repeated Cross-Validation)
Weight Decay