UNIVERSITY OF ORLÉANS

Big Data Analytics: Lassoing the Random Forest

Authors:
Mickaël WALTER
Loïc PALMA

Lecturer:
Sessi TOKPAVI
Referent:
Sessi TOKPAVI

October 24, 2019

Abstract

The purpose of this project covering two of the three Big Data courses teached in Master ESA is to put in the various concepts discussed in class, particularly those relating on random forests and penalized regressions. To be simple, it is about focusing on a regression problem, where the goal is to predict a quantitative target variable from a number of predictors. The methodology covered in this report could be qualified as "Lassoing the Random Forest". Indeed, its principle is to check if the prediction method of the Random Forest which consists in aggregating the predictions trees in the forest using a simple arithmetic mean, can be improved by considering only some of these predictions selected by a penalization method (Ridge, Lasso or Elastic-Net). More precisely, it is necessary to obtain the forecasts for each individual i of the M trees of the forest, $\hat{y}_{1,i}$,$\hat{y}_{2,i}$,...,$\hat{y}_{M,i}$ and then consider in a second step the following penalized regression (in case of Lasso):

$$(\hat{\beta}_1^{lasso}, ..., \hat{\beta}_M^{lasso}) = \underset{(\hat{\beta}_1,...,\hat{\beta}_M)}{\operatorname{argmin}} \; \|y - W\beta\|_2^2 + \lambda\|\beta\|_1^2$$

with y being the dimension vector $(n, 1)$ of the target variable, W the dimension matrix (n, M) grouping forecasts of the M trees for the n observations, and $\lambda > 0$ the regularization parameter. Hence, the final prediction is given by the mean of the predictions $\hat{y}_{1,i}$,$\hat{y}_{2,i}$,...,$\hat{y}_{M,i}$. This method will be compared with the traditional Random Forest aggregation scheme and possibly other benchmark methods (Boosting, SVM, and so on). The GitHub repository is public and can be found here.

Contents

Chapter 1

Project Presentation

1.1 Dataset

We chose a Kaggle dataset for our analysis. The dataset can be found here. This latter is a record of every building or building unit (apartment, etc.) sold in the New York City property market over a 12-month period. This dataset is a concatenated and slightly cleaned-up version of the New York City Department of Finance's Rolling Sales dataset.
The New York City Department of Finance (DOF) is the revenue service, taxation agency and recorder of deeds of the government of New York City. Its Parking Violations Bureau is an administrative court that adjudicates parking violations, while its Sheriff's Office is the city's primary civil law enforcement agency.
The Department of Finance (DOF) collects more than $33.2 billion in revenue for the City and values more than one million properties worth a total market value of $988 billion. In addition, DOF also:

- Records property-related documents.
- Administers exemption and abatement programs.
- Adjudicates and collects parking tickets.
- Maintains the city's treasury.
- Participates on and provides administrative support for the NYC Banking Commission.
- Oversees the New York City Sheriff's Office, which acts as DOF's law enforcement division and the City's chief civil law enforcement agency.

Through the Mayor's Office of Pensions and Investments, the Department of Finance also advises the Administration on the City's $160 billion pension system and $15 billion deferred compensation plan.

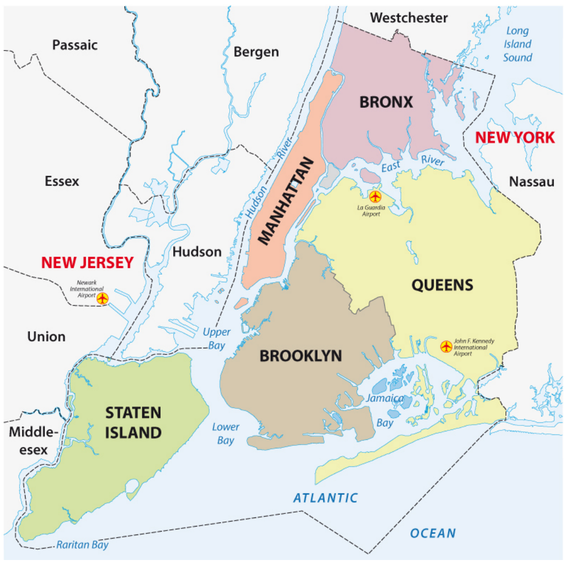


Figure 1.1: The boroughs of New York City

1.2 Problematic

What can we discover about New York City real estate by looking at a year's worth of raw transaction records? Can we spot trends in the market? Can we build a model that predicts sale value in the future?

1.3 Hypotheses

Before even thinking about a possible answer to our questions stated before, several things need to be taken cared of.

Here is a list of a few things we need to focus on:

- handle missing data;
- handle outliers;
- have a look at the Building Classifications Glossary;
- have a look at the Glossary of Terms.

Note that because this is a financial transaction dataset, there are some points that need to be kept in mind:

- 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.
- This 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 piecemeal by its residents (a condominium), a sale refers to a single apartment (or group of apartments) owned by some individual.

This dataset contains the location, address, type, sale price, and sale date of building units sold. A reference on the trickier fields:

- **Borough**: 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).
- **Block**; **Lot**: The combination of borough, block, and lot forms a unique key for property in New York City. Commonly called a **BBL**.
- **Building Class at Present** and **Building Class at Time of Sale**: The type of building at various points in time[1].

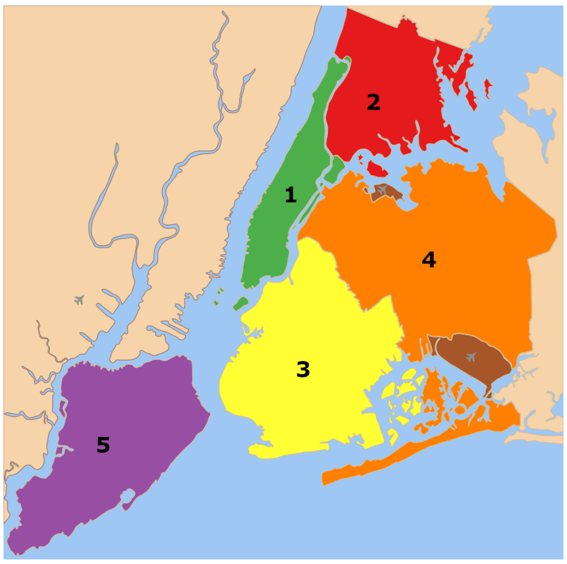


Figure 1.2: Digit code for New York City boroughs

[1]See the Glossary Of Terms linked above.

Chapter 2

Data Cleansing

Data cleansing or **data cleaning** is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

2.1 Variables

Our dataset's dimension is $(84548, 22)$. This means we do have 84548 observations that comes along with 21 predictors and our target variable.

2.1.1 Qualitative predictors

According to the Glossary of Terms:

- **Borough**: The number of the borough in which the property is located.
- **Neighborhood**: Department of Finance assessors determine the neighborhood name in the course of valuing properties. The common name of the neighborhood is generally the same as the name Finance designates. However, there may be slight differences in neighborhood boundary lines and some sub-neighborhoods may not be included.
- **Build Class Category**: This is a field that we are including so that users of the Rolling Sales Files can easily identify similar properties by broad usage (e.g. One Family Homes) without looking up individual Building Classes. Files are sorted by Borough, Neighborhood, Building Class Category, Block and Lot.
- **Tax Class at Present**: Every property in the city is assigned to one of four tax classes (Classes 1, 2, 3, and 4), based on the use of the property.
  - *Class 1*: Includes most residential property of up to three units (such as one-, two-, and three-family homes and small stores or offices with one or two attached apartments), vacant land that is zoned for residential use, and most condominiums that are not more than three stories.
  - *Class 2*: Includes all other property that is primarily residential, such as cooperatives and condominiums.
  - *Class 3*: Includes property with equipment owned by a gas, telephone or electric company.
  - *Class 4*: Includes all other properties not included in class 1,2, and 3, such as offices, factories, warehouses, garage buildings, etc.
- **Tax Class at Time of Sale**: Same as **Tax Class at Present** but at Time of Sale.
- **Block**: A Tax Block is a sub-division of the borough on which real properties are located. The Department of Finance uses a Borough-Block-Lot classification to label all real property in the City. "Whereas" addresses describe the street location of a property, the block and lot distinguishes one unit of real property from another, such as the different condominiums in a single building. Also, block and lots are not subject to name changes. based on which side of the parcel the building puts its entrance on.
- **Lot**: A Tax Lot is a subdivision of a Tax Block and represents the property unique location.
- **Easement**: An easement is a right, such as a right of way, which allows an entity to make limited use of another's real property. For example: MTA railroad tracks that run across a portion of another property.

- **Building Class at Present**: The Building Classification is used to describe a property's constructive use. The first position of the Building Class is a letter that is used to describe a general class of properties (for example "A" signifies one-family homes, "O" signifies office buildings. "R" signifies condominiums). The second position, a number, adds more specific information about the property's use or construction style (using our previous examples "A0" is a Cape Cod style one family home, "O4" is a tower type office building and "R5" is a commercial condominium unit). The term Building Class used by the Department of Finance is interchangeable with the term Building Code used by the Department of Buildings. For more information, please see the Building Classifications Glossary.
- **Building Class at Time of Sale**: Same as **Building Class at Present** but at Time of Sale.
- **Address**: The street address of the property as listed on the Sales File.
- **Apartment number**: Coop sales include the apartment number in the address field.
- **Zip Code**: The property's postal code.

2.1.2 Quantitative predictors

- **Unnamed**: The transaction number.
- **Residential Units**: The number of residential units at the listed property.
- **Commercial Units**: The number of commercial units at the listed property.
- **Total Units**: The total number of units at the listed property.
- **Land Square Feet**: The land area of the property listed in square feet.
- **Gross Square Feet**: The total area of all the floors of a building as measured from the exterior surfaces of the outside walls of the building, including the land area and space within any building or structure on the property.
- **Year Built**: Year the structure on the property was built.
- **Sale Date**: Date the property sold.

2.1.3 Target

- **Sales Price**[1]: Price paid for the property.

[1] A $0 sale indicates that there was a transfer of ownership without a cash consideration. There can be a number of reasons for a $0 sale including transfers of ownership from parents to children as discussed in 1.3.

2.2 Duplicate Data & Missing Values

Duplicate Data are entries that have been added by a system user multiple times. In statistics, **missing data**, or **missing values**, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

2.2.1 New variable

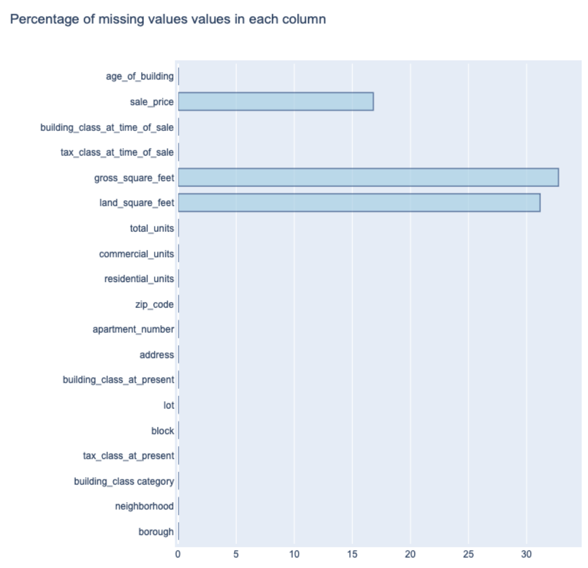
We created a new variable called **Age of Building** which is just the **Sale Date** minus the **Year Built**.
It allows us to remove those two variables and have only one variable who is probably more pertinent.

2.2.2 Duplicate Data

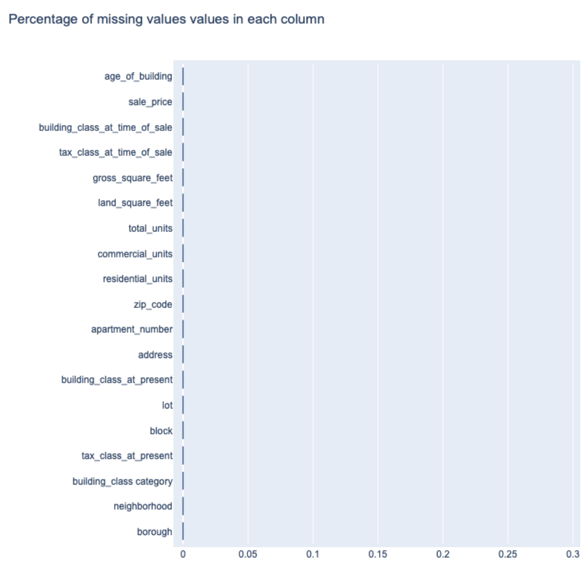
The first thing we did was to check whether there were any duplicate observations, and they were 765.
We instantly removed them from our dataset.

2.2.3 Missing Values

The particularity of our dataset is that it include a lot of missing values but only for some predictors: *Sale Price*, *Gross Square Feet* and *Land Square Feet*.
With **Sale Price** being our Target Variable, we will not try to impute any values. We will just remove them. Thereby we have 69987 observations left.
However, the correlation coefficient between **Land Square Feet** and **Gross Square Feet** is 0.6429. This means they are *highly correlated*. Unfortunately, we do not have both variables for 31.05% of our data. We could have impute predictions of these variables if we had at least one of the two for each observation. As we have only one of the two for 1.61% of our dataset (after removing Sale Price missing values and duplicates), we will neither impute them. We have 48244 observations left.



(a) Before



(b) After

Figure 2.1: Data Cleansing

2.3 Outliers

Outliers are extreme values that deviate from other observations on data , they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

First things first, we:

- removed observations with **Zip Code** having a value of 0 (mistake).
- removed observations with **Year Built** having a value of 0 (mistake).
- removed observations with **Gross Square Feet** having a value of 0 (mistake).
- removed observations with **Land Square Feet** having a value of 0 (mistake).
- removed the **Unnamed** variable as it is just the transaction number (experiment artefact).
- removed the **Easement** variable as it is filled with blanks (mistake).
- removed the **Apartment Number** variable as it is filled with blanks (mistake).
- removed the **Zip Code** variable as we already have the Borough (CPU time saving[1]).
- removed the **Address** variable as we already have the Borough (CPU time saving[1]).

We end up with 44993 observations left.

As we pointed it out earlier, there is a **lot** of observations with a Sale Price of 0$. 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.
Obviously, we are retiring those observations from our dataset. We have 29189 observations left.

	min	$Q_{5\%}$	$Q_{25\%}$	$Q_{50\%}$	$Q_{75\%}$	$Q_{95\%}$	max
Sale Price	1 $	125.000 $	420.000 $	620.000 $	950.000 $	3.500.000 $	2.210.000.000 $

Figure 2.2: Descriptive Statistics of Sale Price after removing transfer of deeds

According to this NY Times article, this is near impossible to have a place to live for 125.000$ in the Bronx. Indeed, $\$719,000$ is the price for a two-family house in this borough. We will then remove observations that have a sale price below 125.000$ ($Q_{5\%}$). We will also remove all the observations that have a sale price above 3.500.000$ ($Q_{95\%}$) because above this limit, prices are unreal ($>$ 2 Billions of dollar for the maximum sale price).
We have 26253 observations left.

	min	$Q_{5\%}$	$Q_{25\%}$	$Q_{50\%}$	$Q_{75\%}$	$Q_{95\%}$	max
Sale Price	125.500 $	250.000 $	440.000 $	620.000 $	900.000 $	1.820.252 $	3.499.000 $

Figure 2.3: Descriptive Statistics of Sale Price after removing $Q_{5\%}$ and $Q_{95\%}$ of Sale Price

2.4 Summary

After cleansing, removing duplicates and so on, we finally have:

- a clean dataset to work with.
- 26253 observations.
- 15 predictors and 1 target variable.

Our clean dataset is about 31.05% of the raw dataset.

[1]Indeed, as we create dummies for categorical variables (one-hot-encoding), we would end up with $186 - 1$ more variables just for Zip Code and $n_{obs} - 1$ for the Address variable as they are all different.

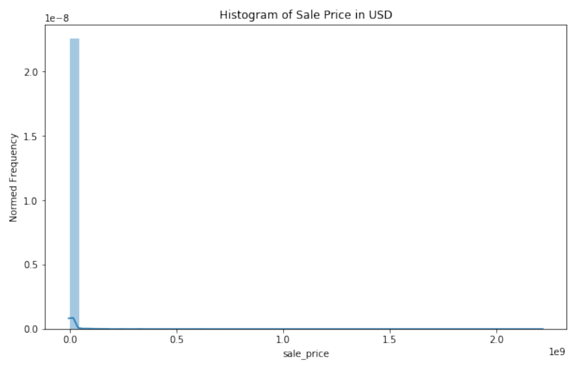
Chapter 3

Data Visualization

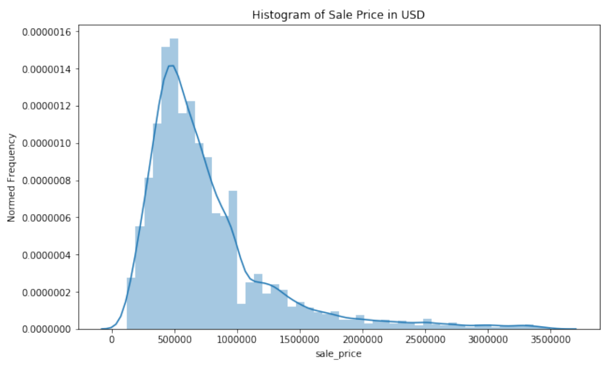
Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

3.1 Target Variable

3.1.1 Histogram of Sale Price



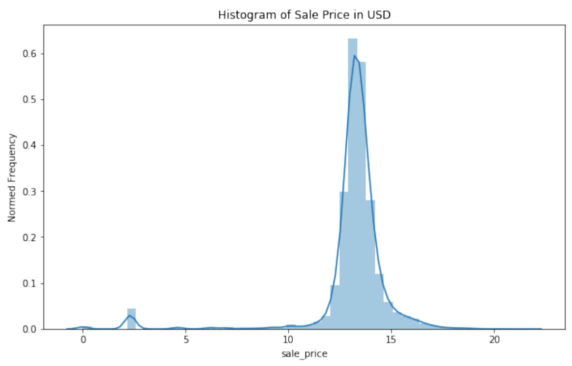
(a) Raw data



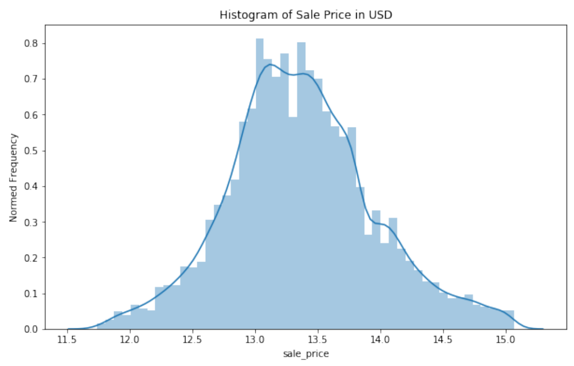
(b) Clean data

Figure 3.1: Histogram of Sale Price in USD

3.1.2 Histogram of log(Sale Price)



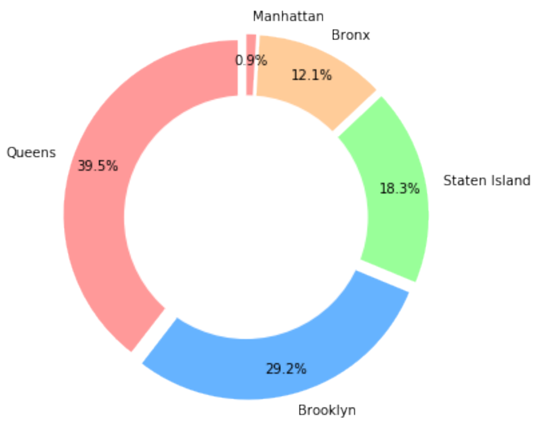
(a) Raw data



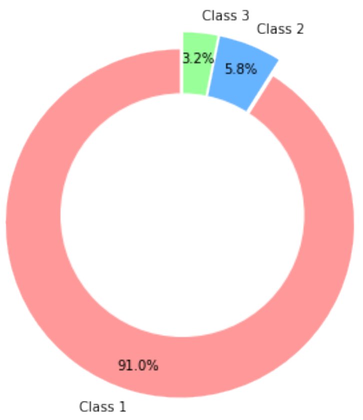
(b) Clean data

Figure 3.2: Histogram of log(Sale Price) in USD

3.2 Boroughs & Tax Classes



(a) Boroughs



(b) Tax Classes

Figure 3.3: Proportion of sales per Boroughs and per Tax Classes

We can clearly see that Manhattan is under represented here (0.9%). Moreover, there are no Class 4 Tax. This can be explained by the fact that this latter includes all other properties not included in Class 1,2 and 3. Thus if all properties are included in those three first Classes, there should not be any property in Class 4. However Class 2 and Class 3 are under represented here. Indeed, Class 2 and 3 are specific to certain types of properties whereas Class 1 includes most residential property.

3.3 Correlation Matrix

	Sale Price	Total U.	G. Sq. Feet	L. Sq. Feet	Residential U.	Commercial U.
Sale Price						
Total U.	0.02**					
G. Sq. Feet	0.03****	0.65****				
L. Sq. Feet	0.01	0.56****	0.92****			
Residential U.	0.03****	0.68****	0.93****	0.80****		
Commercial U.	0.00	0.74****	0.02***	0.03****	0.01	
Age of the Building	0.14****	0.01	-0.01	-0.02**	0.00	0.00

Table 3.1: Correlation Matrix[1] (Pearson)

We can see that some of our quantitative predictors are highly (positively) correlated with each other. For example, that is the case between Gross Square Feet and Land Square Feet (0.92%). This can be explained by the fact that Gross Square Footage is calculated from the outside of the exterior walls and is inclusive of all space within minus areas that are open to below whereas the Land Square Feet is just the Land Area of the property listed.
One interesting correlation to notice is that Age of the Building is positively correlated with Sale Price (0.14%). This is an interesting correlation as usually newer properties are sold for an higher price than older ones. However, when you have a look at CIRCA old houses, a curated historic house marketplace showcasing old homes for sale in New York, you can clearly that old houses are sold for the same (or higher) price than newer ones. That is mainly explained by the fact that old houses usually have a better emplacement than newer ones.

[1]Stars shows significance levels. ****: $p < .0001$ | *** : p < .001| ** : p < .01| * : p < .05| Blank : p > .05

Chapter 4

Methodology

4.1 Preparation

4.1.1 One-Hot-Encoding

One-hot-encoding is a common technique used to work with categorical variables.
The first thing we had to do was to create dummies for them.. We end up with 476 predictors and our Target variable.

4.1.2 Training Set & Test Set

We decided to split the dataset as follow: 75% for the Training Set and 25% for the Test Set.

	Observations	Variables
Training Features Shape	19693	476
Training Target Shape	19693	1
Testing Features Shape	6565	476
Testing Target Shape	6565	1

Figure 4.1: Training and Test sets dimension

4.2 Random Forest

Random Forest is an ensemble Machine Learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called bagging. Bagging along with boosting are two of the most popular ensemble techniques which aim to tackle high variance and high bias. The first algorithm for random decision forests was created by Tin Kam Ho. An extension of the algorithm was developed by Leo Breiman.
Random for two reasons:

- Random sampling of training observations when building trees.
- Random subsets of features for splitting nodes.

4.2.1 Random sampling of training observations

The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree.
For our analysis we will use the `Sklearn` library. In `Sklearn`, implementation of Random forest the sub-sample size of each tree is always the same as the original input sample size but the samples are drawn with replacement if `bootstrap = True`. If `bootstrap = False` each tree will use exactly the same dataset without any randomness.

4.2.2 Random subsets of features for splitting nodes

Random Forest with random features is formed by selecting at random, at each node, a small group of input variables to split on. In `Sklearn` this can be set by specifying `max features = sqrt(n_features)`.
If `max features = auto` then the algorithm will choose by itself the best max number of features.

4.2.3 Other Hyperparameters

There are several other hyperparameters for the Random Forest. Here are the ones we will focus on:

- `max_depth` = The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min samples split samples.
- `min_samples_leaf` = The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
- `min_samples_split` = The minimum number of samples required to split an internal node.
- `n_estimators` = The number of trees in the forest.

4.2.4 CPU Time saving

As said in 4.1.1 we have 476 predictors because of dummies. After trying multiple Random Forest and plotting variables importance, we found out that most of them were useless (i.e 0% importance). Only a couple of them were useful, thus we decided to only keep important variables. We end up with only 9 predictors and our Target variable.

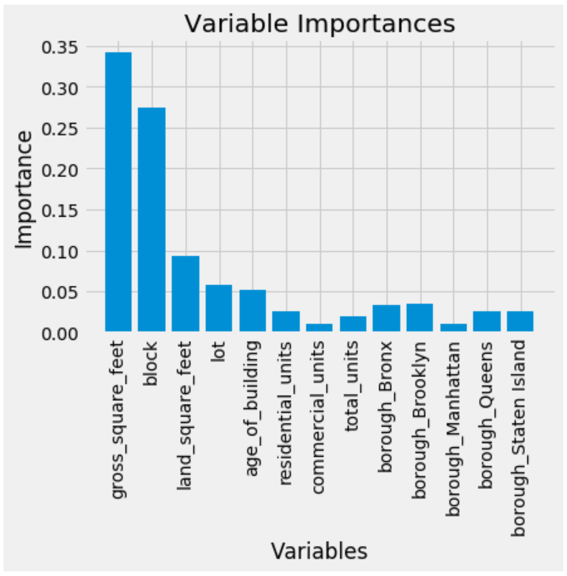


Figure 4.2: Variable Importance

4.2.5 Hyperparameter Estimation

Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions. Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.
We used **Randomized Search** which is implemented in `Sklearn` to find optimal hyperparameters that we talked about in 4.2.1, 4.2.2 & 4.2.3.
The Randomized Search and the Grid Search explore exactly the same space of parameters. The result in parameter settings is quite similar, while the run time for randomized search is drastically lower[1]. The performance is slightly worse for the randomized search, though this is most likely a noise effect and would not carry over to a held-out test set.

[1]More information about the differences between these two methods can be found here.

Here is the Grid we tried with Optimal Value being the value chosen by the Randomized:

Hyperparameter	Range	Optimal Value
Bootstrap	[True, False]	True
Deepness	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]	40
Max. Features	[Auto, Sqrt]	Auto
Min. Sample Leaf	[1, 2, 4]	2
Min. Sample Split	[2, 5, 10]	10
Number of estimators	[200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]	1200

Figure 4.3: Grid for the Random Forest Randomized Search

We fitted 3 folds (`cv = 3`) for each of 100 candidates (`n_iter = 100`), resulting in 300 fits.
Randomized Search took 38.6 minutes to execute whereas Grid Seach did not even finish after 1.5 hours (on a CPU with 8 cores at full speed).

4.3 Predictions Recovery

At that point our prediction χ is a $(6565, 1)$ array. We only have the aggregated mean of all trees for each observations.
The next step was to find a way to get the prediction for every single tree.
The following code does the job in Python:

```
predictions_every_tree = np.array([tree.predict(test_features) for tree in rf.estimators_])
```

Indeed, the `estimators_` attribute of the `RandomForest()` allows us to get every single tree. At that point we just have to compute the prediction for every single estimator[1].
At the end, here is a snippet of the prediction matrix P $(6565, 1200)$ we had :

$$P = \begin{bmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \hat{y}_{1,3} & \cdots & \hat{y}_{1,M} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & & & \vdots \\ \hat{y}_{3,1} & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ \hat{y}_{n,1} & \cdots & \cdots & \cdots & \hat{y}_{n,M} \end{bmatrix}_{n\times M} = \begin{bmatrix} 580.500 & 337.857,14 & 613.000 & \cdots & 618.631,58 \\ 548.125 & 269.666,67 & & & \vdots \\ 555.666,67 & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 581.428,57 & \cdots & \cdots & \cdots & 461.800 \end{bmatrix}_{6565\times 1200}$$

Applying the mean function along rows of the P matrix results in the χ array.

$$\frac{1}{M}\sum_{i=1}^{M} P_{n,i} = \begin{bmatrix} \frac{1}{M}\sum_{i=1}^{M} \hat{y}_{1,i} \\ \frac{1}{M}\sum_{i=1}^{M} \hat{y}_{2,i} \\ \frac{1}{M}\sum_{i=1}^{M} \hat{y}_{3,i} \\ \vdots \\ \frac{1}{M}\sum_{i=1}^{M} \hat{y}_{n,i} \end{bmatrix}_{n\times 1} = \begin{bmatrix} 639.123,92 \\ 342.773,50 \\ 669.587,35 \\ \vdots \\ 497.850,97 \end{bmatrix}_{6565\times 1} = \chi$$

[1]As far as we know, there was no options on SAS in order to get every single prediction. The solution would have been to manually code the Random Forest. On R, the predict.all parameter of the randomForest library does the job.

4.4 Training Set & Test Set on the prediction Matrix

Now that we have the prediction matrix P, we concatenated this latter with our Target Variable in order to get a $(6565, 1201)$ matrix. Indeed the trick is now to put those 1200 trees as features to predict our Target variable using different Penalization methods.
But first, we decided to split the P matrix as follow: 75% for the Training Set and 25% for the Test Set.

	Observations	Variables
Training Features Shape	4923	1200
Training Target Shape	4923	1
Testing Features Shape	1642	1200
Testing Target Shape	1642	1

Figure 4.4: Training and Test sets for the P Matrix

4.5 Penalization Methods

It was necessary to obtain the forecasts for each individual n of the M trees of the forest, $\hat{y}_{1,n}$,$\hat{y}_{2,n}$,...,$\hat{y}_{M,n}$ in order to consider in a second step Penalization Methods.

4.5.1 Ridge

Ridge regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable.

The basic form of a Ridge regression is as follow:

$$(\hat{\beta}_1^{lasso}, ..., \hat{\beta}_M^{lasso}) = \underset{(\hat{\beta}_1,...,\hat{\beta}_M)}{\text{argmin}} \; \|y - W\beta\|_2^2 + \lambda\|\beta\|_2^2$$

Now, ridge regression proceeds by adding a small value, k, to the diagonal elements of the correlation matrix. (This is where ridge regression gets its name since the diagonal of ones in the correlation matrix may be thought of as a ridge.) That is

$$\beta_{Ridge} = (X^\top \hat{X} + \lambda I_p)^{-1} X^\top y$$

Important facts:

- It can be shown that there exists a value of k for which the Mean Squared Error (the variance plus the bias squared) of the ridge estimator is less than that of the least squares estimator.
- If $\lambda = 0$ we get the OLS estimator $\hat{\beta_{OLS}}$.

Hyperparameter	Range	Optimal Value
Lambda	[0.01, 0.1, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 20, 30]	3
Normalization	[True, False]	True

Figure 4.5: Grid for the Ridge Grid Search

4.5.2 Lasso

In statistics and Machine Learning, **Lasso** (Least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.
The basic form of a Lasso regression is as follow:

$$(\hat{\beta}_1^{lasso}, ..., \hat{\beta}_M^{lasso}) = \underset{(\hat{\beta}_1,...,\hat{\beta}_M)}{\text{argmin}} \ \|y - W\beta\|_2^2 + \lambda\|\beta\|_1^2$$

Hyperparameter	Range	Optimal Value
Lambda	[0.01, 0.05, 0.1, 0.5, 0.6, 1, 5, 8, 10, 100, 199, 200, 500, 750, 1000]	199
Normalization	[True, False]	True

Figure 4.6: Grid for the Lasso Grid Search

What is the difference between Ridge and Lasso? Lasso can act as a variable selector by canceling some coefficients $\hat{\beta}_M$. If that is the case, $\hat{\beta}_M = 0$ are excluded from the predictive model.

4.5.3 ElasticNet

In 2005, Zou and Hastie introduced the **ElasticNet** to address several shortcomings of Lasso. When $p > n$ (the number of covariates is greater than the sample size) lasso can select only n covariates (even when more are associated with the outcome) and it tends to select only one covariate from any set of highly correlated covariates. Additionally, even when $n > p$, if the covariates are strongly correlated, ridge regression tends to perform better. The elastic net extends lasso by adding an additional ℓ^2 penalty term.

Hyperparameter	Range	Optimal Value
Lambda	[0.01, 0.015, 0.02, 0.05, 0.1, 0.2, 0.4, 0.5, 0.6, 1, 5, 8, 10]	0.01
ℓ^1 ratio	[0.1, 0.2, 0.4, 0.7, 0.8]	0.8
Normalization	[True, False]	True

Figure 4.7: Grid for the Elastic Net Grid Search

Important facts:

- Technically the Lasso model is optimizing the same objective function as the Elastic Net with $\ell^1 ratio = 1.0$ (no ℓ^2 penalty).
- For $\ell^1 ratio = 0$ the penalty is an ℓ^2 penalty.
- For $0 < \ell^1 ratio < 1$, the penalty is a combination of ℓ^1 and ℓ^2.

4.5.4 XGBoost

XGBoost stands for eXtreme Gradient Boosting. It has been created by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC) group. But what is boosting? The goal of boosting is to create a good model using weak classifiers. By adding models iteratively on top of each other, errors of the past models are corrected by the next predictor, until the training data is accurately predicted. We can distinguish two types of boosting:

- Adaptive Boosting (i.e adaBoost, more information here)
- Gradient Boosting (i.e XGBoost, more information here)

Both are boosting algorithms which means that they convert a set of weak learners into a single strong learner. They both initialize a strong learner (usually a decision tree) and iteratively create a weak learner that is added to the strong learner. They differ on how they create the weak learners during the iterative process.

The Evolution of Boosting Algorithms' arxiv paper suggest that “Both AdaBoost and gradient boosting follow the same fundamental idea: Both algorithms boost the performance of a simple base-learner by iteratively shifting the focus towards problematic observations that are difficult to predict. With AdaBoost, this shift is done by up-weighting observations that were misclassified before. Gradient boosting identifies difficult observations by large residuals computed in the previous iterations using a gradient descent optimization process.“

Why using XGBoost instead of adaBoost ?

Quoting Tianqi Chen, the creator of XGBoost on Quora : “Specifically, XGBoost used a more regularized model formalization [than adaBoost] to control over-fitting , which gives it better performance.“

Hyperparameter	Range	Optimal Value
Learning Rate	[0.01, 0.05, 0.075, 0.1, 0.15]	0.075
Number of estimators	[500, 1000, 2000]	500
Deepness	[3, 5, 7, 10]	7
Min. Child Weight	[1, 3, 5, 10]	5

Figure 4.8: Grid for the XGBoost Grid Search

4.6 Metrics

4.6.1 Coefficient of determination

The **coefficient of determination** R^2 is a statistical method that explains how much of the variability of a factor can be caused or explained by its relationship to another factor. The coefficient of determination is an important tool in determining the degree of linear-correlation of variables ('goodness of fit') in regression analysis. The higher the value, the better the fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

4.6.2 Root Mean Square Error

Root Mean Square Error is a quadratic scoring rule that measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} = \sqrt{MSE}$$

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

4.6.3 Mean Absolute Error

Mean Absolute Error measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}$$

If the absolute value is not taken in consideration, the average error becomes the Mean Bias Error (MBE) and is usually intended to measure average model bias.

Chapter 5

Results

Model	R^2	RMSE	Performance[1]	MAE	Performance[1]
Random Forest	0.423	$313.438,23$	100%	$187.206,597$	100%
Ridge	**0.427**	$325.304,27$	96.21%	$191.184,405$	97.88%
Lasso	0.401	$326.840,00$	$95,72\%$	$191.646,392$	97.63%
ElasticNet	0.416	$325.621,51$	96.11%	$191.088,138$	97.93%
XGBoost	0.414	**$313.301,41$**	**100.04%**	**$186.164,02$**	**100.56%**

Figure 5.1: Metrics comparison

Using Penalized Methods on the forecasts for each individual i of the M trees of the forest does not improve our accuracy to predict Sale Price. In point of fact, in terms of Root Mean Square Error (RMSE) & Mean Absolute Error (MAE), Penalized Methods performed worse than the initial Random Forest.
The cause might be that we did a penalized regression on very correlated variables (trees). Basically, each tree learns in the same way, but that's just the dataset on which they train that differs a little bit (bagging). But as they differ only a little, that will result in highly correlated variables.

Ridge and ElasticNet regressions used every coefficients (1200) whereas Lasso only used 63 of them. An important remark is that for the Ridge Regression, we do have a better R^2 than the initial Random Forest. Hence the proportion of the variance for a dependent variable that is explained by an independent variable is better for the Ridge regression than for the Random Forest. This might be explained because the Ridge takes all variables in consideration (some coefficients were negative) that we have a better understanding of the explained variance than Lasso and Elastic Net.

However, XGBoost performed better than all of them. But why does XGBoost performed better than Random Forest?

Random Forest is a bagging algorithm. It reduces variance. XGBoost is a boosting algorithm. It reduces variance through the use of multiple models (bagging) but also reduces bias by training the subsequent models with the past models' errors (boosting). Therefore, each additional tree tries to get the model closer to the target and reduce the bias of the model rather than the variance.

In other words, Random forests are built from deep decision trees (in our case, deepness is 40), which tend to overfit individually. Building multiple of them in parallel couteracts the overfitting. Gradient boosted trees on the contraty uses shallow trees (in our case, deepness is 7), *weak learners*, which tend to underfit. But building many of them in sequence (on error residuals) counteracts the underfitting.

[1]Performance are compared regarding to the Random Forest Performance.

Chapter 6

Conclusion

After cleaning our dataset and removing outliers, only few of our initial predictors were useful to predict the Sale Price of housing in New York City. Indeed, most of them had no importance in our Random Forest. Removing outliers and making sure we worked on a clean dataset was in fact a key step to a better understanding of our problem.

The goal of our project was to check whether the prediction method of the Random Forest which consists in aggregating the predictions trees in the forest using a simple arithmetic mean, can be improved by considering only some of these predictions selected by a penalization method (Ridge, Lasso or Elastic-Net).
In other words, Random Forest assume that trees make good predictions on average and that noises are the only things preventing to achieve good performance. Therefore, ensembles are used to reduce this noise. The goal of our project was to see whether it would have been possible to reduce this noise by fitting a penalized regression. However, results showed that it is not the case.
According to our results, not a single penalization method helps to get better performance. Indeed, RMSE and MAE for every single penalization method are higher than RMSE and MAE for the initial Random Forest. Only the R^2 of the Ridge regression is better than the one of the Random Forest.

The last step of our project was to benchmark our model versus XGBoost. Results showed that the XGBoost tend to outperform the initial Random Forest due to the fact that XGBoost not only reduces variance but also reduces bias.

Annexes

Annexes

Our codes are available on this GitHub repository.
The datasets needed to run this code can be found on the `DATA` folder.

1 Python

Codes can be found on the `Python` folder.

WARNINGS
If you use Jupyter Notebook please use the .ipynb scripts in the Jupyter Notebook sub-folder.
You may need to install some libraries in order to run these codes.

1.1 Descriptive Statistics

Please find the script here.

1.2 Random Forest

Please find the script here.

1.3 Grid Search

Please find scripts here and there.

1.4 Penalization Method

Please find the script here.

1.5 XGBoost

Please find the script here.

Bibliography

[1] H. Binder, O. Gefeller, M. Schmid, and A. Mayr. The evolution of boosting algorithms. *Methods of Information in Medicine*, 53(06):419–427, 2014.

[2] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. *CoRR*, abs/1603.02754, 2016.

[3] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[4] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017.

[5] Venu Gopal Lolla. Integrating Python and Base SAS. `<https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3548-2019.pdf>`, 2018. [Online; accessed 14-October-2019].

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] T. Trevor Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. `<https://web.stanford.edu/~hastie/ElemStatLearn/>`, 2009. [Online; accessed 21-October-2019].