RandomForestRegression algorithm for price prediction

1. INTRODUCTION

This document describes the RandomForestRegression algorithm trained to predict the prices on a kaggle dataset which focuses on property prices from the EU.

This project started with the acquiring of the data, which was downloaded from Kaggle. Fortunately it was already clean, so it didn't require any ETL to be done onto it.

After that, the information is loaded via CSV files into Pandas Dataframes in order to be able to work with them.

Then, several different RandomForestRegression algorithms are proposed, and their MAE score calculated in order to pick the best solution.

The algorithm is then trained with the training dataset, and lastly some plots are done in order to visualize the effectiveness of the algorithms predictions.

Using several categories (such as number of rooms, area, number of bathrooms, etc) the algorithm is able to predict the price of each property.

2. DATA

The data used was obtained from kaggle.com, and consist of 2 csv files :

test.csv : this is the testing set, used by the algorithm to predict the target value, which in this case is the Price for each property. It consist in a 1459 rows by 79 columns comma separated value file.

train.csv : this is the training set, used by the algorithm to learn the relationship between the categories and the target labbel, which in this case is the Price for each property. It consist in a 1460 rows by 80 columns comma separated value file.

3. LIBRARIES

To start off, the script imports the necessary libraries:

```
1  #! python 3
2  # Random Forest Algorithm for price prediction, using the 'Housing prices competition' dataset from kaggle.
3
4  #Importing libraries
5  import pandas as pd # Used to manipulate dataframes
6  import os # Used to read and write on system files
7  from sklearn.model_selection import train_test_split
8  import matplotlib.pyplot as plt # Used for plotting
9  import seaborn as sns # Used for plot themes
10 import matplotlib.ticker as mtick # Used to adjust plot ticks
11 import numpy as np # Used to create arrays
12 from sklearn.ensemble import RandomForestRegressor # Used to import the necessary model
13 from sklearn.metrics import mean_absolute_error # Used for the creation of a score system in order to determine the best model
14 sns.set_style('white')
```

Figure 1. Lines 1 to 13

Visual aid and plotting libraries: seaborn, matplotlib.pyplot and matplotlib.ticker are used to create the plots used to compare the models efficiency as well as to compare the chosen model's result with the target samples.
Algebraic and dataframing librares: numpy and pandas are used to create and analyse the dataframes objects used by the models.

Machine learning libraries: sklearn.model_selection is used to import the train_test_split used to obtain the validation set out of the training set. The sklearn.ensemble is used to import the RandomForestRegression model and the sklearn.metrics is used to compare the mae score between each model, in order to chose the most accurate model.

4. DATA MANIPULATION

```
15  # Read the data
16  # In order for the code to find the file, please be sure to have both train.csv and test.csv files in the same directory as this python script
17  cwd = os.getcwd()
18  X_full = pd.read_csv( cwd + '\\train.csv', index_col='Id')
19  X_test_full = pd.read_csv(cwd + '\\test.csv', index_col='Id')
20
21  # Obtain target and predictors
22  y = X_full.SalePrice
23  features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
24  X=X_full[features].copy()
25  X_test=X_test_full[features].copy()
26
27  # Break off validation set from training data
28  X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size = 0.8, test_size = 0.2, random_state = 0 )
```

Figure 2. Lines 15 to 28

After having imported the necessary libraries, the script then turns into the data files, and using the os libraries, opens them and assigns them into two different objects, X_full and X_train_full, which are used by the algorithm as the test and training set, respectably, on lines 19 and 20.

Not every category is used to train this algorithm. Some data is missing values, or might be irrelevant to the calculations that the algorithm will do, and will only slow down the process. Even though adding every single column would produce a more accurate learning algorithm (theoretically), the slowing of the process time isn't worth the slight increase in prediction value.

And so the features object is created in order to select only some of these predictors, which in this case are 7:

```
23  features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
```

after that, in lines 25 and 26 the script will remove any category other than the ones labelled in the feature object.

On line 29 the script is going to split some of the training set to be used as the validation set, this will enable some fine tuning on the models hyper-parameters ((this set is used to give an unbiased estimate of the skill of the final tuned model). The percentage of the split is the standard 20%, that is, 20% of the training set will be separated and used as the validation set. The lost data will slightly reduce the models available training set, but having a data validation set to which compare to if mandatory if we are to create an accurate model.

```
29  X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size = 0.8, test_size = 0.2, random_state = 0 )
```

5. MODEL SELECTION

Once we have the data sets ready to be implemented into our model, it is time to select the hyperparameters of the RandomForestRegression machine learning algorithm. To do this, the script will compare between 5 different models with different hyperparameters, and we will end up with a combination of the best parameters (the ones which fared best using our mae score)

```
31  # Defining the models, testing for the lowest MAE score
32
33  model_1 = RandomForestRegressor(n_estimators=50, random_state=0)
34  model_2 = RandomForestRegressor(n_estimators=100, random_state=0)
35  model_3 = RandomForestRegressor(n_estimators=100, criterion='mae', random_state=0)
36  model_4 = RandomForestRegressor(n_estimators=200, min_samples_split=20, random_state=0)
37  model_5 = RandomForestRegressor(n_estimators=100, max_depth=7, random_state=0)
38
39  #Model N° 6 is a combination of the models that scored the lowest mae, that is model N°3 and N°5, adding some more depth
40  model_6 = RandomForestRegressor(n_estimators=100, criterion='mae', max_depth=20, random_state=0)
41
42  models = [model_1,model_2,model_3,model_4,model_5, model_6]
```

Figure 3. Model creation, lines 31 to 42

The RandomForestRegressor has a couple of hyperparemeters we can toy with:

n_estimators: Controls the number of trees that are going to be used in the forest. Even though this parameter will not generate an overfitting of the model, having over a couple of hundreds of trees is unresourceful, since we will see no real improvement on the prediction value, and yet have the algorithm take much more time to compute. Generally, the recommended value is between 64 an 128 trees.

1. *random_state:* This parameter is simply used as seed on the random numbers generated, as to be able to compare between the models.

2. *max_depth:* This hyperparameter is used to tell the algorithm how big are the trees going to grow. If max_depth is too high, then the model will overfit , causing a discrepancy between the predictions that the model is going to compute and the actual value. Even though the official documentation for the RandomForest models say that the model will never overfit, some studies have shown that there are a couple of datasets that are more susceptible to the model overfiting.

3. *min_samples_split*: The min_samples_split parameter specifies the minimum number of samples required to split an internal leaf node. The default value for this parameter is 2, which means that an internal node must have at least two samples before it can be split to have a more specific classification.

4. *criterion*: The function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion, and “mae” for the mean absolute error.

In order to select the best model, we are using the ‘mae’ score, this is the ‘mean absolute error’ of each model, comparing them between them, and choosing the one with the lowest ‘mae’.
To do this, we need to create a function that will show each models mae value, and then print it.

```
44  #Function for comparing different models
45
46  def score_model(model, X_t=X_train, X_v=X_valid, y_t=y_train, y_v=y_valid):
47      model.fit(X_t, y_t)
48      preds = model.predict(X_v)
49      return mean_absolute_error(y_v, preds)
50
51  print('Model scores (lowest is best): ')
```

Figure 4. definition of score_model function

This is the result of running lines 44 to 51:

```
Model scores (lowest is best):


Model 1 MAE: 24015
Model 2 MAE: 23740
Model 3 MAE: 23528
Model 4 MAE: 23996
Model 5 MAE: 23706
Model 6 MAE: 23515


Model 6 has the lowest score, this model will be trained with the dataset:
```

Figure 5. model scoring.

Finally, we can see in the plot generated by the lines 54 to 64 that model 6 is the one to chose.

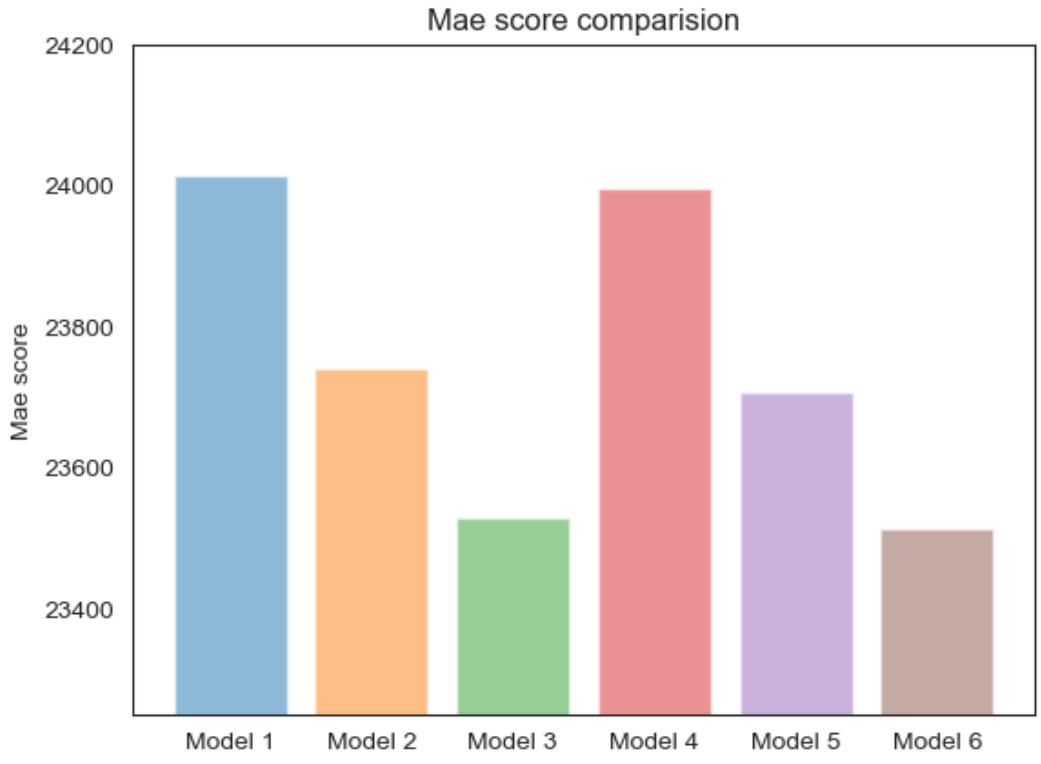

Figure 6. Plotting each models score

6. MODEL CREATION AND FITTING

```
66  #The lowest model was N3, followed by number 5. So We are going to use a combination of both that scores an even better MAE.
67  my_model = RandomForestRegressor(n_estimators=100, criterion='mae', max_depth=20, random_state=0)
68
69  # Fitting the model to the training data
70  my_model.fit(X, y)
71
72  #Generating test predictions
73  preds_test = my_model.predict(X_test)
74
75  # Fitting it into a pandas dataframe and printing the output into an CSV file
76  output = pd.DataFrame({'Id' : X_test.index, 'SalePrice': preds_test})
77  output.to_csv('prediction.csv', index=False)
```

Figure 7. Model creation and fitting of the data

The script will create the my_model object, which will be assigned the RandomForestRegressor algorithm with the shown hyperparameters on line 67.

We then fit the model with the X and y dataframes, X being the training dataset except the target value (SalePrice), defined on line 25, and y being the training set containing only the Id and SalePrice values, defined on line 23.

we then use the .predict() function with the X_test as variable in order to test out the algorithm. The output (prediction) is then stored in the 'output' object, and exported to an csv file named 'prediction.csv'

7. RESULTS

Lines 80 to 84 will return some information on the output, as well as a small statistical analysis:

```
Algorithm output and SalePrice statistical information:


        Id  SalePrice
0     1461  120146.58
1     1462  159202.50
2     1463  184623.91
3     1464  178387.52
4     1465  194842.29
...    ...        ...
1454  2915   85670.00
1455  2916   88815.00
1456  2917  155504.80
1457  2918  131659.00
1458  2919  231440.60

[1459 rows x 2 columns]


count      1459.000000
mean     178162.993012
std       70779.266029
min       55218.640000
25%      128735.260000
50%      158622.000000
75%      210027.817500
max      512020.690000
Name: SalePrice, dtype: float64
```

Figure 8. Model prediction and statistical analysis

8. GRAPHICS

Lines 86 to 125 will generate two graphics, used to compare the effectiveness (visually) of the model.

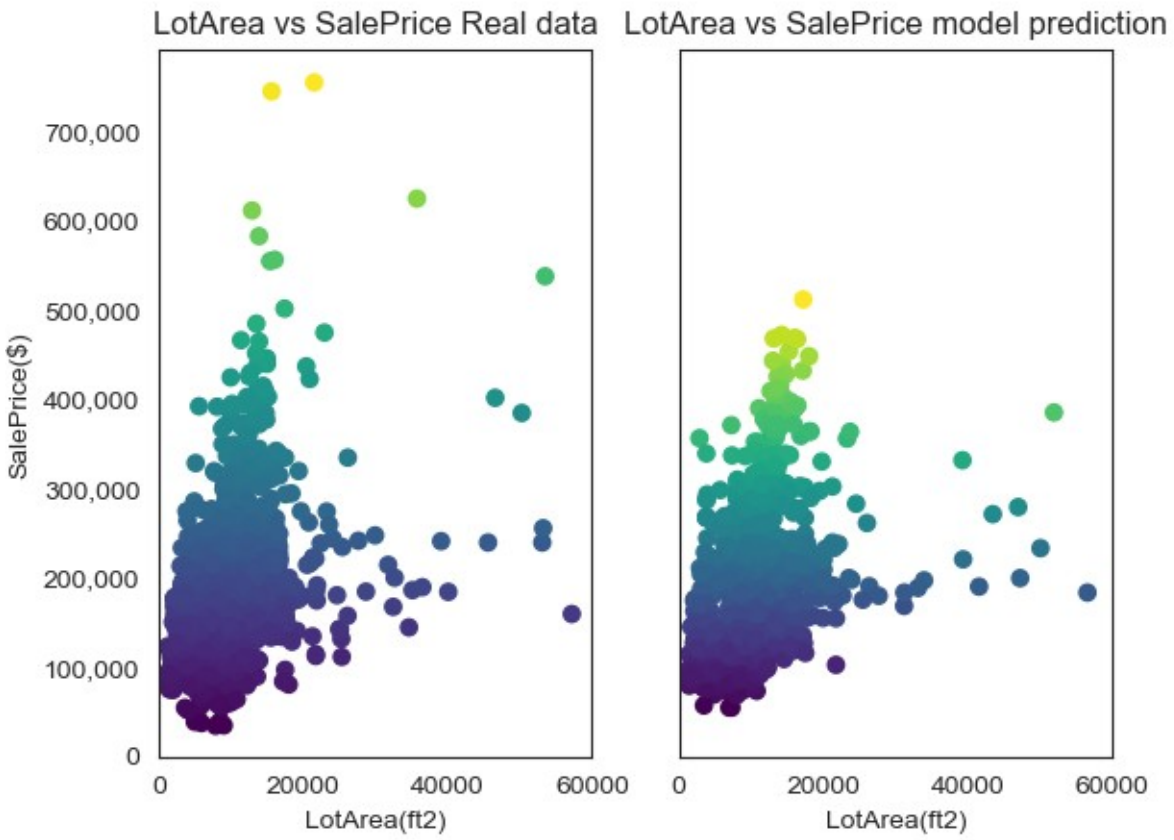

Figure 9. LotArea vs SalePrice, data vs prediction

The first subplot is showing a scatter plot on the LotArea(ft2) vs the SalePrice of the properties. This data is the factual data we have, and it's the information stored in the original datasets. We then compare it to the second subplot, which is the LotArea(ft2) vs SalePrice predicted by our model. As we can see, the model is able to sucesfully predict most of the values, shown in the similarity of the cluster obtained.

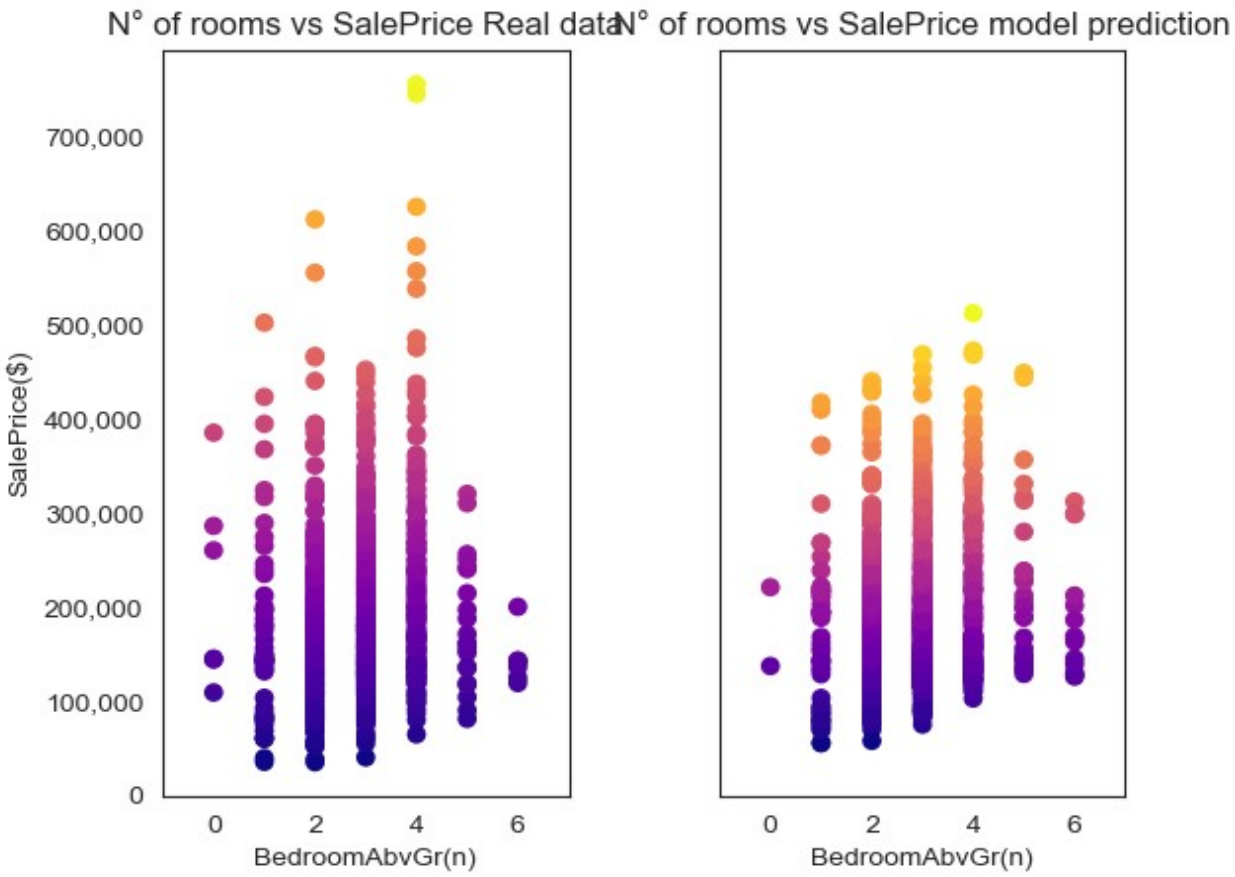

Figure 10. BedroomAbvGr(n) vs SalePrice data, vs prediction

The second graph is showing a similar result to the previous one, showing a scatter plot of the real data, against the predictions that the model did. This time we are comparing the SalePrice to the number of bedrooms, instead of the surface area of the property. Even when comparing with a completely different label, the model will successfully predict the property SalePrice.

9. CONCLUSION

In conclusion, the RandomForestRegression model successfully predicted the SalePrice for the dataset that was used to train it. Other hyperparameters can be used to further tune the model, but the score of the predictions won't increase dramatically, as there is a limit to the tuning you can actually do on each model (because of it's limitations).