Ensemble Learning Report

Elie TRIGANO, Thomas TAYLOR, Shamir MOHAMED,
Alban SARFATY, Pierre CHAGNON

March 20, 2023

Contents

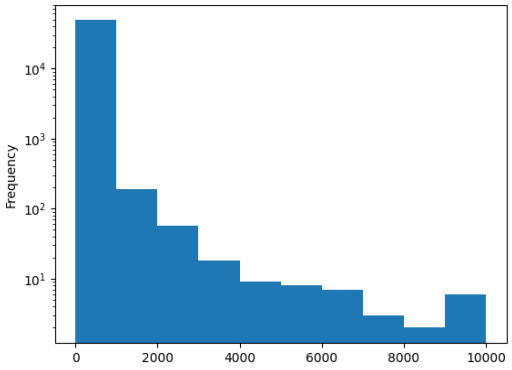
1 Airbnb Price Predictions

In this section, we will detail our methodology in predicting the price of AirBnBs in New York City from a dataset, applying the knowledge we have acquired on Decision Trees.

1.1 Dataset exploration

1.1.1 Presentation of the dataset

The dataset we use for our prediction contains the pricing and general information of the renting of AirBnB in New York in 2019. The dataset contains a total of 48895 rows with 16 different columns. The first observation we can do is that the dataset does not contain any specific information on the nature of the Airbnb being rented. In example, the feature do not allow us to differentiate a luxurious Airbnb, with many rooms and a big surface, to an Airbnb of low quality. We only have access to general information such as the location, the id of the owner and the number of location. Consequently, we do not expect to have an excellent accuracy in our predictions.



(a) Price distribution

id	0
name	16
host_id	0
host_name	21
neighbourhood_group	0
neighbourhood	0
latitude	0
longitude	0
room_type	0
price	0
minimum_nights	0
number_of_reviews	0
last_review	10052
reviews_per_month	10052
calculated_host_listings_count	0
availability_365	0

(b) Null values

Figure 1: Exploration of the dataset

1.1.2 Exploratory Data Analysis

First of all, we can see that the distribution of rental prices shows many outliers, especially around $10,000. These outliers will degrade the quality of our prediction results. We will have to perform tests to decide if they should be removed from the training set. Then, when we explore the null values, we see that `last_reviews` and `reviews_per_month` have a large (identical) number of nulls. We can assume that they correspond to locations that never had a review. We have therefore decided to replace these values by 0. Finally, a number of columns can be removed because they have similar information (`id` and `name`, or `host_id` and `host_name`).

1.2 Model Development & Evaluation

In this part we are going to cover the steps in order to predict our target variable that is the price of a night in a Airbnb in New York. This part will cover preprocessing tasks such as feature selection and engineering, model selection, model tuning and evaluation. With this information we will be able to interpret what model gives us the best score.

1.2.1 Feature Selection & Engineering

In this section we are going to focus on the feature we kept and different implementations we made such as outlier removal, encoding, feature importance and choice of our final dataset to perform modelling upon. A crucial aspect of our work was the outlier removal. Indeed, we decided to get rid of 5% of outliers in the price column, as we wanted to have a model that would be able to rightly predict a "typical" AirBnb rent in New York. Removing 5% of outliers from the price column indicates that we do not focus on luxurious stays or flats that are out of the typical range for an AirBnb. We did try several variations including runs while keeping outliers and without. For the method of outlier removal we chose `IsolationForest` removing 5% of outliers or anomaly (contamination score of 0.05). It was the outlier removal technique that gave us the best score compared to IQR or Z-score. We then decided to drop columns such as `'name'`, `'host_name'`, `'last_review'`, `'id'`, `'latitude'`, `'longitude'` based on specific domain knowledge and also Feature Selection techniques such as using a variance inference tool. Finally for encoding we decided to One Hot encode the columns `'neighbourhood_group'` ans `'room_type'` as there were not a lot of new columns that were going to be added and it was the best way to keep as much information as possible. We also decided to encode `'host_id'` and 'neighbourhood' using `TargetEncoding` as they were highly cardinal features. We had our final table ready for a first basic run with different models.

1.2.2 Model Selection

After having our final table ready we decided to perform some basic testing on several algorithms that we viewed or talked about in class. We wanted to perform a benchmark of different algorithms to have an idea of how they perform without any tuning. For the metrics we are going to focus on RMSE and MAE as they are the most interpretable as they are the measure of the error in the same terms as the target variable. MAE will be used to measure the absolute difference between the predictions and actual values, whereas RMSE will be used to interpret how the outliers impact the score. Finally, we will also use the R^2 score to see how well the model fits.

The result can be observed in the following table:

	Model	Cross-Val Score	MSE	RMSE	MAE	R2
0	XGBoostRegressor	0.597973	2187.791253	46.773831	33.052490	0.574681
1	GradientBoostingRegressor	0.597564	2244.356671	47.374642	33.575963	0.563684
2	RandomForestRegressor	0.563950	2399.456373	48.984246	34.432058	0.533532
3	ExtraTreesRegressor	0.528926	2556.434869	50.561199	35.270560	0.503015
4	Bagging Regressor	0.534064	2574.700814	50.741510	35.613579	0.499464
5	AdaBoost Regressor	0.526538	2580.081703	50.794505	35.248159	0.498418
6	Decision Tree Regressor	0.231640	4211.875195	64.898961	44.550011	0.181188

We can see that without any tuning we still have convincing results with an RMSE of 46,77 for our best model that is the XGBoost. We also have a 57% R2 score that is encouraging.

We decided to stick to 3 models in order to follow the structure of our course, the models are:

- Decision Tree
- Random Forest
- A boosted model: XGBoost

1.3 Model tuning

We have decided to focus on three models for this part of tuning as seen in the previous section. For every model we have decided to perform hyperparameter tuning, interpretation of feature importance with graphs such as feature importance, plot of expected values vs true values and the performance of the model with the same metrics.

1.3.1 Decision Tree

The first model for our tuning step is Decision Tree. We decided to tune the following parameters:

- `max_depth`: the maximum depth of the tree.
- `min_samples_split`: the minimum number of samples required to split an internal node.
- `min_samples_leaf`: the minimum number of samples required to be at a leaf node.
- `max_features`: the number of features to consider when looking for the best split

The final model for our tuned Decision Tree is the following: `'max_depth'`: 7, `'max_features'`: `'auto'`, `'min_samples_leaf'`: 4, `'min_samples_split'`: 10.

This the performance:

	Model	Cross-Val Score	MSE	RMSE	MAE	R2
0	Decision Tree Regressor	2232.608511	2419.825764	49.191725	34.649446	0.529572

The results indicate that our model is moderately accurate in predicting the price of an Airbnb price in New York with an average of 49$ in error for RMSE and 34$ for MAE. Also the model explains 52% of the variance of our target variables which is moderately accurate. We will see if we manage to have better results with our two remaining models.

1.3.2 Random Forest

Our next model is the Random Forest. We have decided to focus on the following parameters for our tuning:

- `n_estimators`: The number of trees in the forest.
- `max_depth`: The maximum depth of the tree.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node.
- `max_features`: The maximum number of features to consider when looking for the best split.

The final model for our tuned Random Forest is the following: `'max_depth'`: 9, `'max_features'`: `'auto'`, `'min_samples_leaf'`: 3, `'n_estimators'`: 1000.

	Model	Cross-Val Score	MSE	RMSE	MAE	R2
0	Random Forest Regressor	2096.776144	2249.488167	47.428769	33.366679	0.562687

So we can see that it performs better than our previous model, the Decision Tree. We have a RMSE of 47 and MAE of 33, which is a slight increase. However, the best improvement is the impact on the R^2 score where we have 56% which indicates that this model managed to fit better. We are going to see what the impact of our XGBoost will be.

1.3.3 XGboost

Finally our last model for the hyperparameter tuning phase. We have decided to tune the following parameters:

- `n_estimators`: the number of trees.
- `max_depth`: the maximum depth of each tree.
- `learning_rate`: the step size shrinkage used to prevent overfitting.
- `subsample`: the fraction of observations to be randomly sampled for each tree.

The final model for our tuned XGBoost is the following: `'subsample'`: 0.75, `'n_estimators'`: 1000, `'max_depth'`: 7, `'learning_rate'`: 0.01

	Model	Cross-Val Score	MSE	RMSE	MAE	R2
0	XGBoost	2022.330945	2154.310355	46.414549	32.65364	0.58119

So, we can see it is the tuned model that has the best overall scores. Indeed , it has approximately 46.4 and 32.6 as for RMSE and MAE. Finally, the R2 score is around 58% which is still moderately good and we see that we have room for improvements.

1.4 Performance evaluation

After tuning our models lets see how they have performed alongside the other initial models. Our final table is the following:

	Model	Cross-Val Score	MSE	RMSE	MAE	R2
0	XGBoost_tuned	-2021.451255	2152.036638	46.390049	32.628571	0.581632
1	XGBoostRegressor	0.597973	2187.791253	46.773831	33.052490	0.574681
2	GradientBoostingRegressor	0.597564	2244.356671	47.374642	33.575963	0.563684
3	Random Forest Regressor_tuned	2096.776144	2249.488167	47.428769	33.366679	0.562687
4	RandomForestRegressor	0.563950	2399.456373	48.984246	34.432058	0.533532
5	Decision Tree Regressor_tuned	2232.608511	2419.825764	49.191725	34.649446	0.529572
6	ExtraTreesRegressor	0.528926	2556.434869	50.561199	35.270560	0.503015
7	Bagging Regressor	0.534064	2574.700814	50.741510	35.613579	0.499464
8	AdaBoost Regressor	0.526538	2580.081703	50.794505	35.248159	0.498418
9	Decision Tree Regressor	0.231640	4211.875195	64.898961	44.550011	0.181188

The table represents the results of different regression models used to predict the price of an Airbnb listing in New York. The XGBoost tuned model has the best performance with the lowest RMSE and MSE, and the highest R^2 score of 0.58. This indicates that the XGBoost model is the most accurate in predicting the price of an Airbnb listing in New York. The Random Forest Regressor tuned and the Decision Tree Regressor tuned models also perform well but with slightly lower accuracy compared to the XGBoost model.

1.5 Conclusion

We see that we are on average for our best model which is 46$ away from true value if we take RMSE as a benchmark. Considering that our model was to predict the price of a regular Airbnb there is still room for improvement. As we have no real features on the characteristics of Airbnb, that would be a good start in order to improve the final score of our model.

2 Implementing a Decision Tree

Decision trees are an effective and popular tool in machine learning competitions because they are easy to understand and display visually. However, they are not as straightforward as they may seem, as they require careful consideration of their underlying mechanisms.

In the upcoming sections, we will walk through the process of building a decision tree using Python and NumPy for both classification and regression. We will also cover the concepts of entropy and information gain, which help us evaluate possible splits and guide the growth of the decision tree in a logical way.

2.1 Best split

Before going further, we need to understand how a decision tree works and what are the steps associated with this process. When implementing a decision tree, we are expected to complete two steps. The first step is to divide the input space into several distinct and disjoint regions. The second step is to predict either the most common class or the average of observations for each region. However, how should we divide the predictor space? To answer this question we have to rely on information theory. Hence, we will have to introduce two metrics, one for classification and one for regression, to evaluate the best split. For classification, we used the entropy, which measures the average level of disorder and can be defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i)$$

Regarding regression tasks, we used RSS (residual sum of squares) defined as:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

The concept of entropy can be utilized to determine the information gain of a potential split. To calculate the split's information gain (IG), the weighted entropies of the children are added together and subtracted from the parent's entropy. By understanding entropy and information gain, we can evaluate all available splits at the current stage of the tree using a greedy approach, select the most promising one, and continue recursively growing until we meet a stopping condition.

2.2 Architecture

To build our decision tree we constructed two classes:

- a helper class to store each split in a node. Each node will contain some information about the splits, information gain and will be useful to make a prediction by traversing the tree

```
    class Node:
        '''
        Helper class which implements a single tree node.
        '''
        def __init__(self, feature=None, threshold=None, data_left=None,
    data_right=None, gain=None, value=None):
            self.feature = feature
            self.threshold = threshold
            self.data_left = data_left
            self.data_right = data_right
            self.gain = gain
            self.value = value

```

Listing 1: Class Node

- a class for the whole decision tree object, containing all the parameters, the corresponding task and the root node

```
    class DecisionTree:
        '''
        Class which implements a decision tree classifier algorithm.
        '''
        def __init__(self, min_samples_split=2, max_depth=5, classifier=True):
            self.min_samples_split = min_samples_split
            self.max_depth = max_depth
            self.classifier = classifier
            self.root = None

```

Listing 2: Class Decision Tree

From there, we defined some methods to compute the information gains and hence the best splits: `_entropy`, `_rss`, `_information_gain`, `_best_split` and `_build` (used to build a decision tree from the input data). All of these helper functions are used in the fit function, used to train a decision tree (regressor or classifier).

For predicting instances, we wrote a helper recursive function, used to predict a single instance (tree traversal) and then we call the predict function to classify new instances.

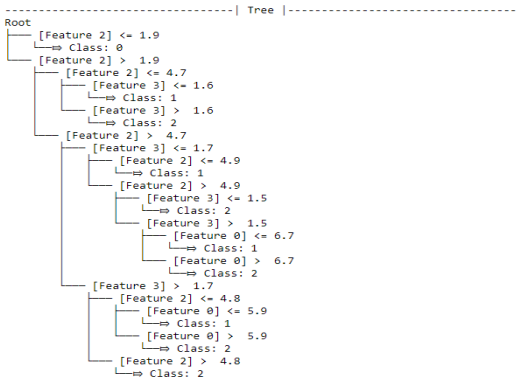
2.3 Complementary options

Here marks the final step of a decision tree. However we wanted to add a few options to increase the interpretability of our model. We added the possibility to display both text and plot representations

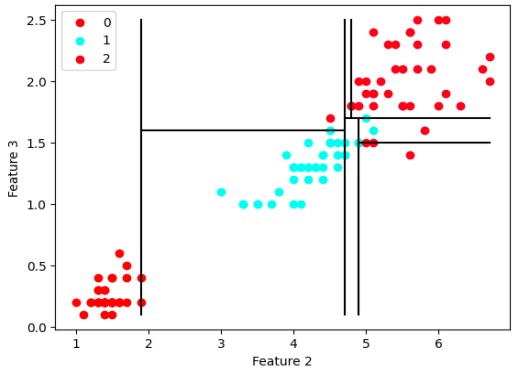
of the splits with: `text_representation` and `plot_partition` functions. We also added a function to prune the tree.

Finally, when we compare the results of our algorithm with sklearn's algorithm, we can see that our splitting points are approximately the same and we both achieve 100% accuracy on the iris dataset for classification. On the other hand, our performances for regression are slightly below sklearn's algorithm and this could come from a lack of penalization in our metric or a different definition of the splitting points. For example, we obtained the following results on the diabetes dataset:

```
MAE with our DT:  55.48314606741573
MSE with our DT:  5044.741573033708
MAE with sklearn DT:  55.651685393258425
MSE with sklearn DT:  5195.494382022472
```



(a) Tree's architecture



(b) Partition

Figure 2: Visualization of the Decision Tree