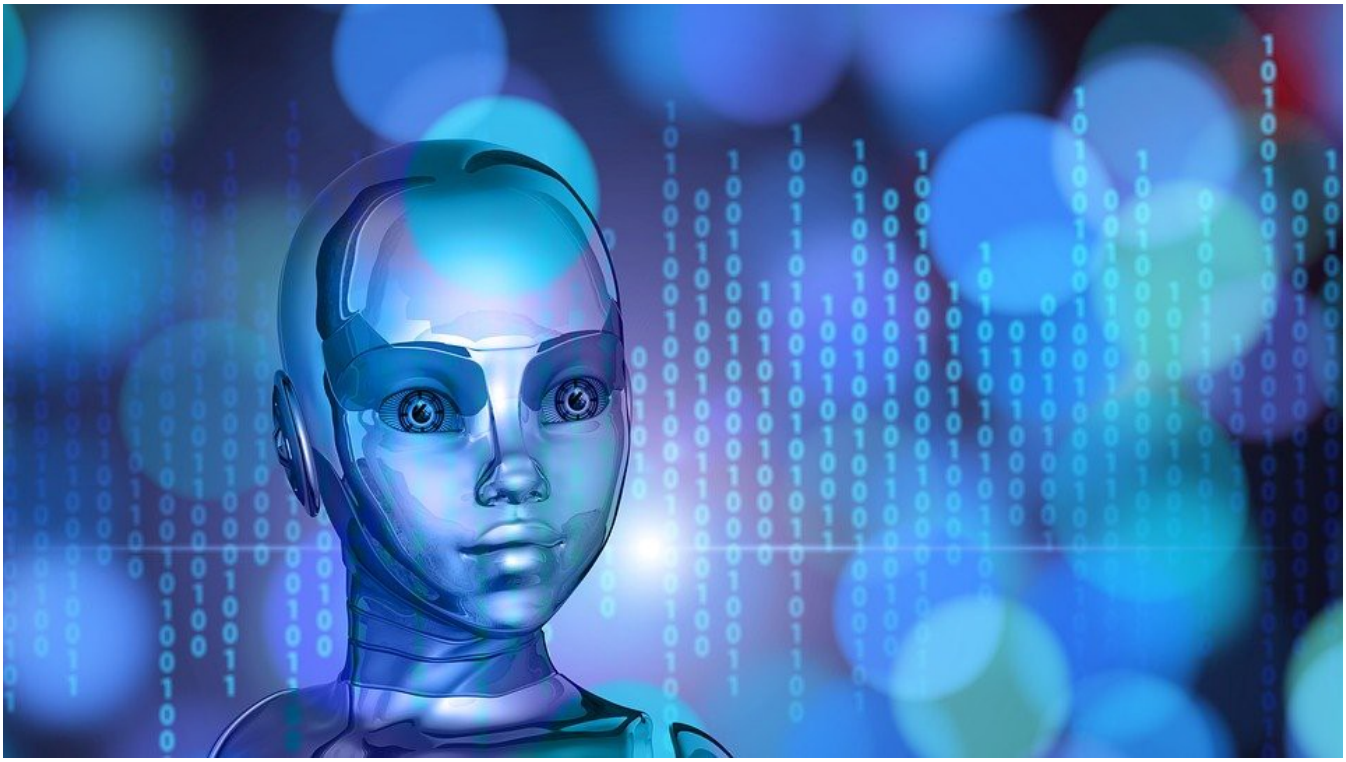


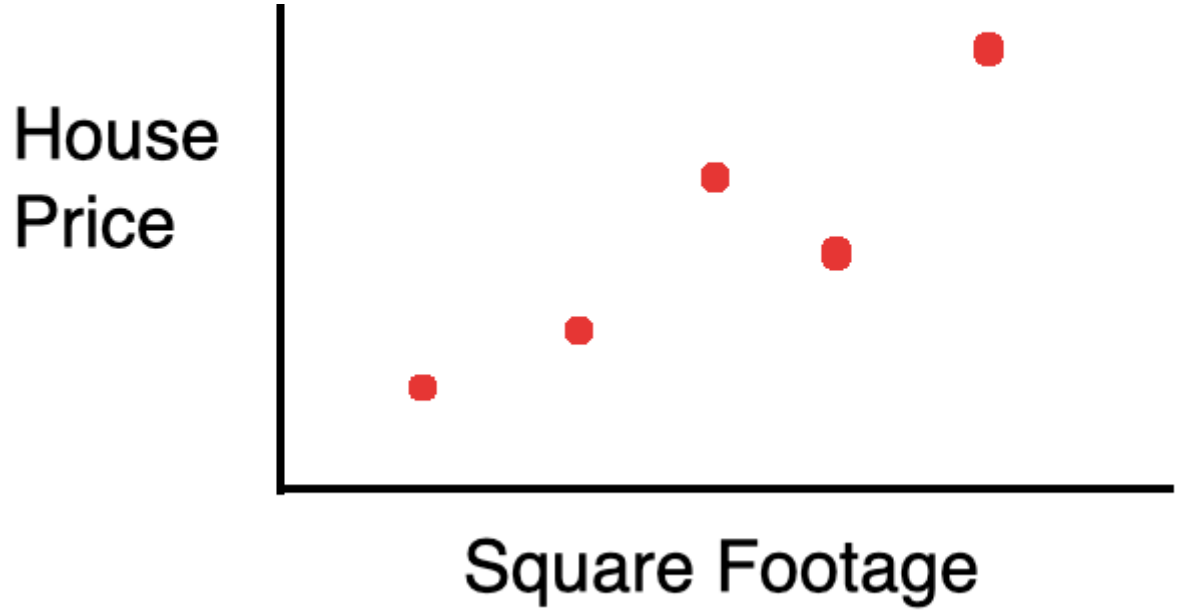
geralt pixabay

XGBoost Python Example

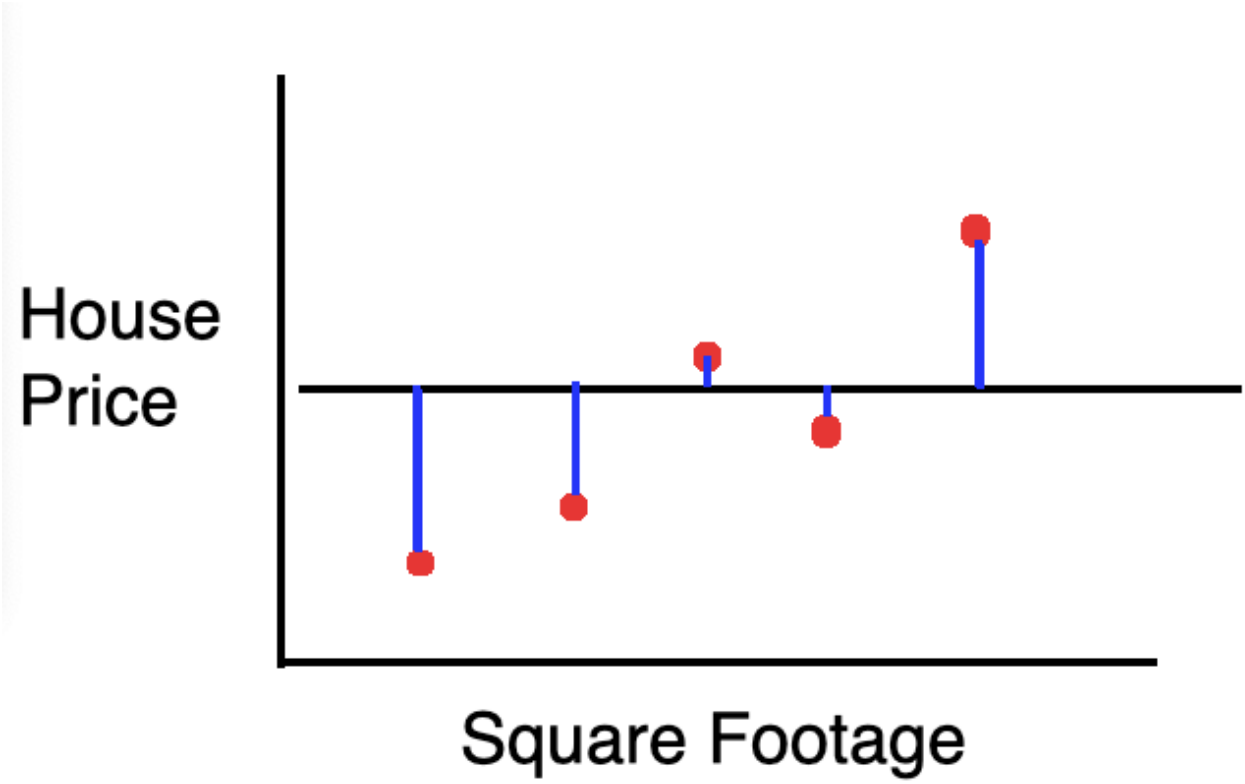
Cory Maklin
May 9 · 8 min read ★

XGBoost is short for Extreme Gradient Boost (I wrote an article that provides the gist of gradient boost here). Unlike Gradient Boost, XGBoost makes use of regularization parameters that helps against overfitting.

Suppose we wanted to construct a model to predict the price of a house given its square footage.



We start with an arbitrary initial prediction. This could be the average in the case of regression and 0.5 in the case of classification.



For every sample, we calculate the residual with the proceeding formula.

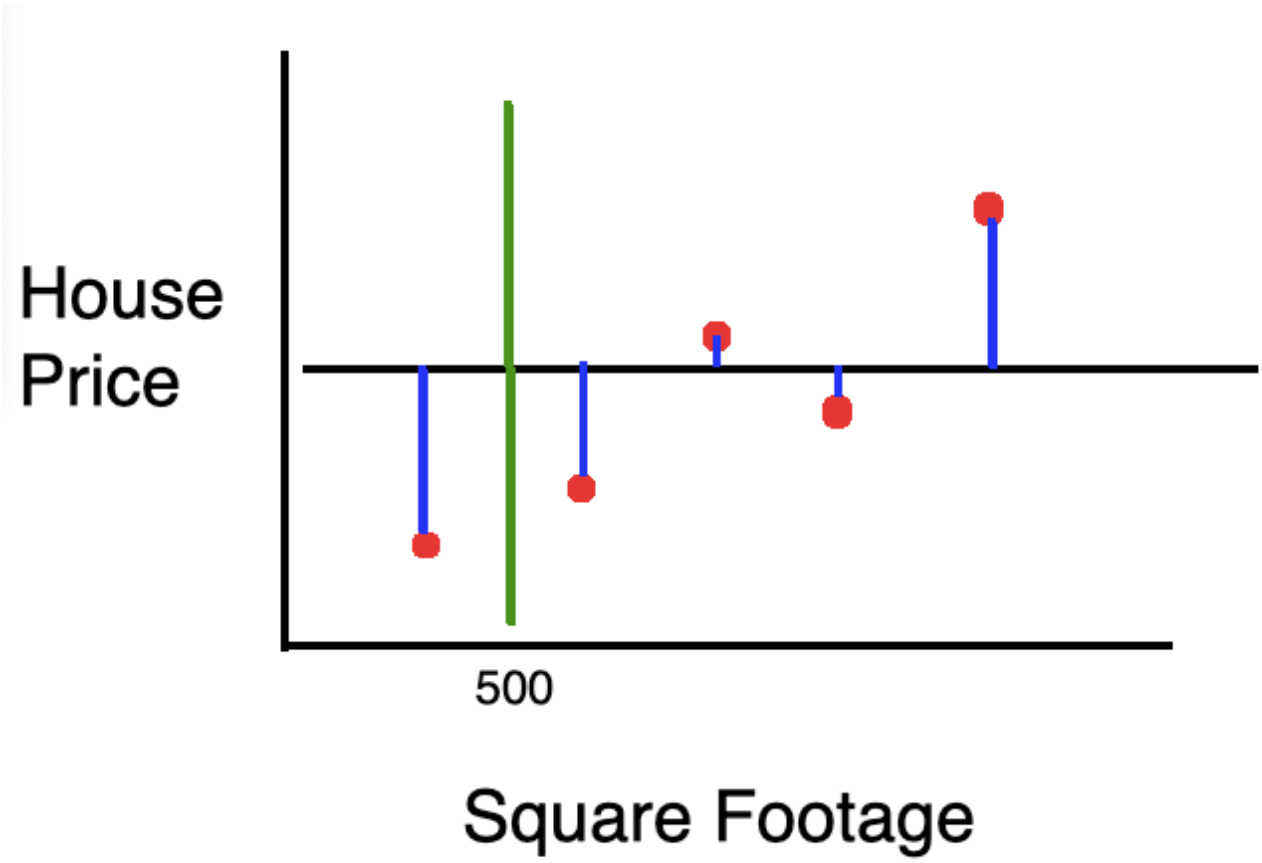
> residual = actual value — predicted value

Suppose, after applying the formula, we end up with the following residuals, starting with the samples from left to right.

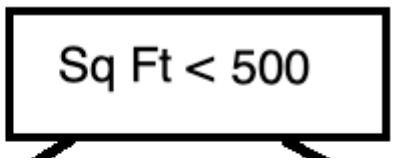
-30,000 / -24,000 / 2,500 / -3,000 / 30,000

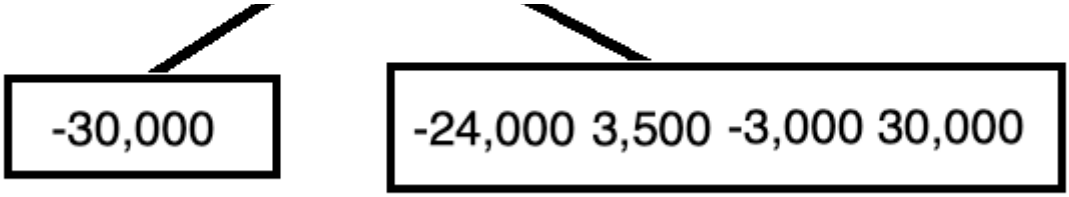
Next, we use a linear scan to decide the best split along the given feature (Square Footage). By linear scan, we mean that we select a threshold between the first pair of points (their average), then select a threshold between the next pair of points (their average) and so on until we've explored all possibilities.

In our example, we start off by selecting a threshold of 500.



The corresponding tree is:





Notice how the values in each leaf are the residuals. That is, the difference between the prediction and the actual value of the independent variable, and not the house price of a given sample.

In order to compare splits, we introduce the concept of *gain*. Gain is the improvement in accuracy brought about by the split. The gain is calculated as follows.

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

where

$$G_L = \sum residuals\ in\ left\ child$$

$$G_R = \sum residuals\ in\ right\ child$$

$$H_R = \#\ residuals\ in\ right\ child$$

$$H_L = \#\ residuals\ in\ left\ child$$

Lambda and Gamma are both hyperparameters. Lambda is a regularization parameter that reduces the prediction's sensitivity to individual observations, whereas Gamma is the minimum loss reduction required to make a further partition on a leaf node of the tree.

Say, we arbitrarily set Lambda and Gamma to the following.

$$\lambda = 0$$

$$\gamma = 100,000,000$$

We can proceed to compute the gain for the initial split.

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = -30,000$$

$$H_L = 1$$

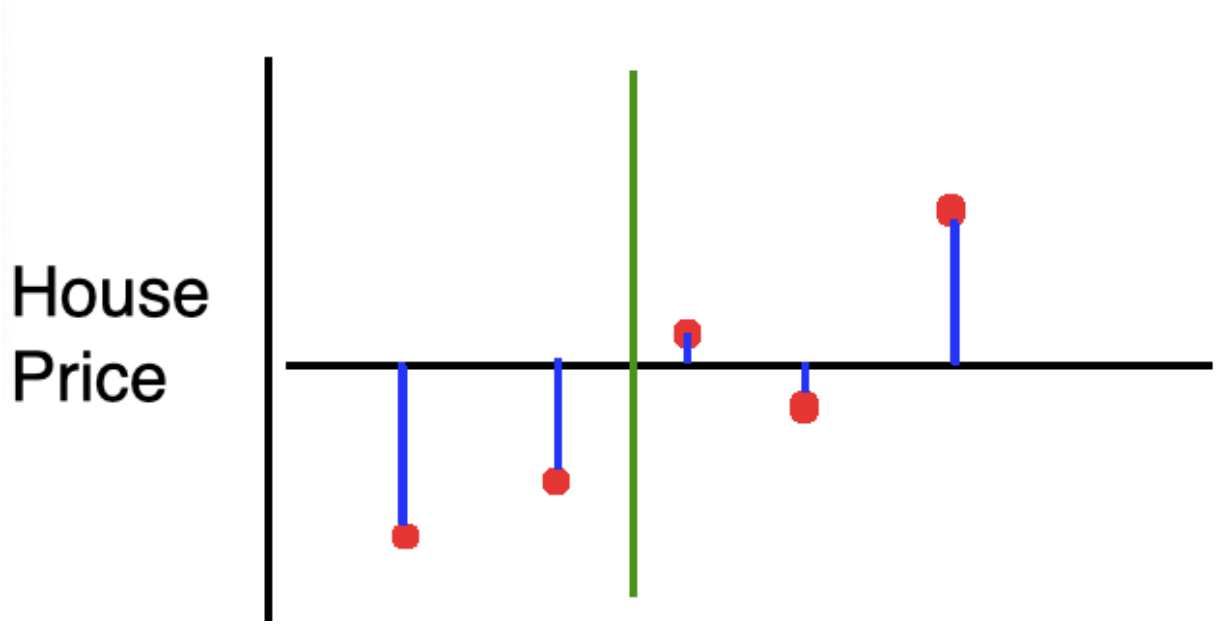
$$G_R = -24,000 + 2,500 - 3,000 + 30,000 = 5,500$$

$$H_R = 4$$

$$Gain = \frac{(-30,000)^2}{1+0} + \frac{(5,500)^2}{4+0} - \frac{(-30,000+5,500)^2}{1+4+0} - 100,000,000$$

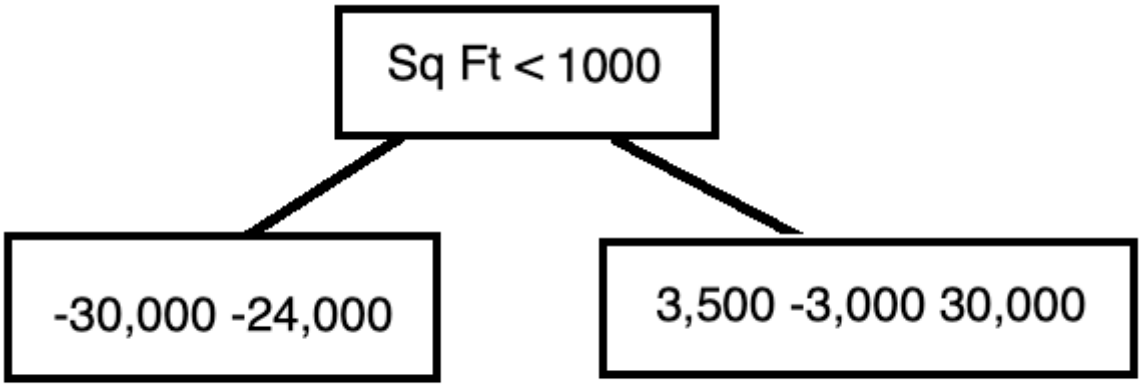
$$Gain = 687,512,500$$

We continue and compute the gains corresponding to the remaining permutations.



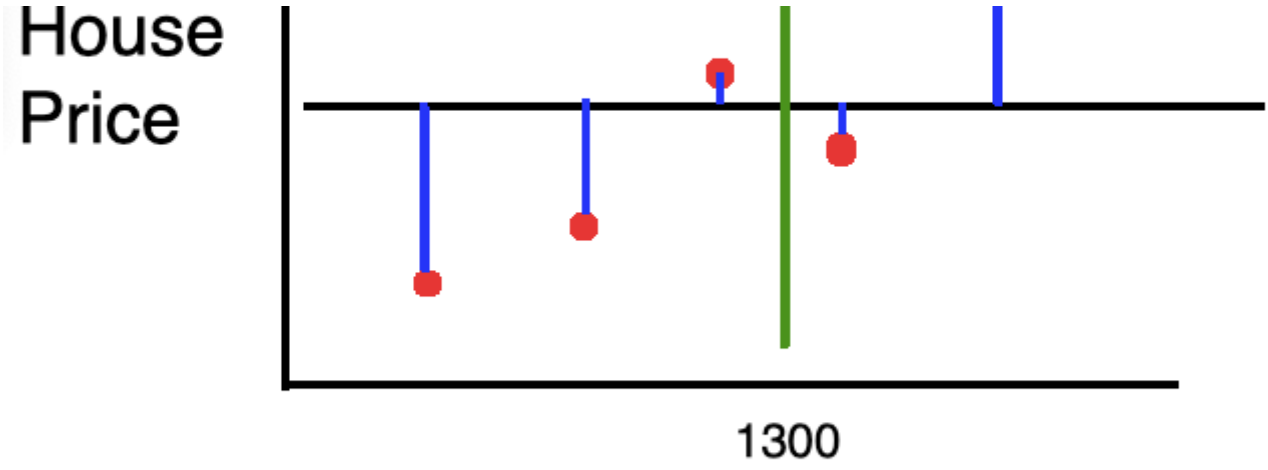
1000

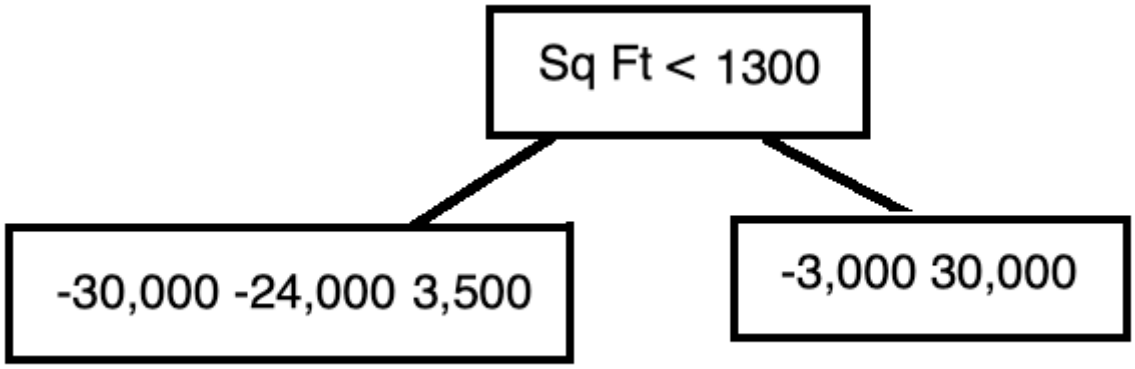
Square Footage



$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

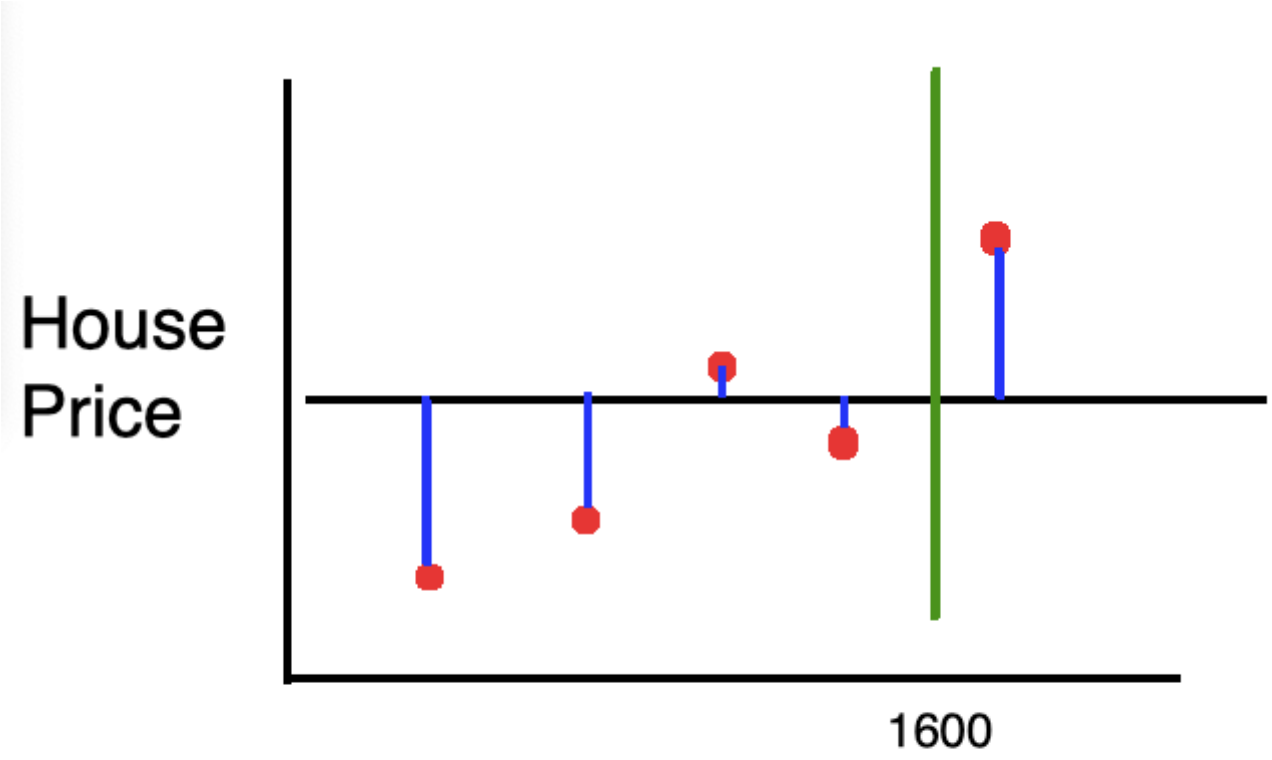
$$G_L = -30,000 + -24,000 = -54,000$$
$$H_L = 2$$
$$G_R = 2,500 + -3,000 + 30,000 = 29,500$$
$$H_R = 3$$
$$Gain = \frac{(-54,000)^2}{2+0} + \frac{(29,500)^2}{3+0} - \frac{(-54,000+29,500)^2}{2+3+0} - 100,000,000$$
$$Gain = 1,528,033,333$$

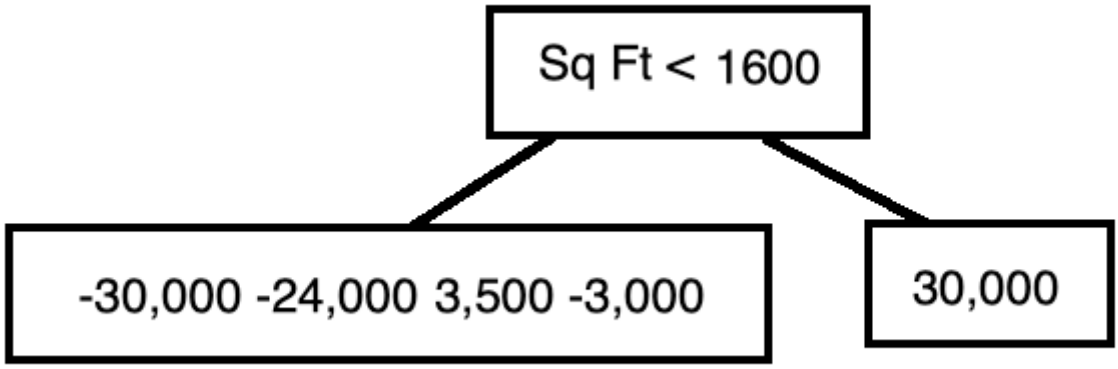




$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = -30,000 + -24,000 + 2,500 = -51,500$$
$$H_L = 3$$
$$G_R = -3,000 + 30,000 = 27,000$$
$$H_R = 2$$
$$Gain = \frac{(-51,500)^2}{3+0} + \frac{(27,000)^2}{2+0} - \frac{(-51,500+27,000)^2}{3+2+0} - 100,000,000$$
$$Gain = 1,028,833,333$$





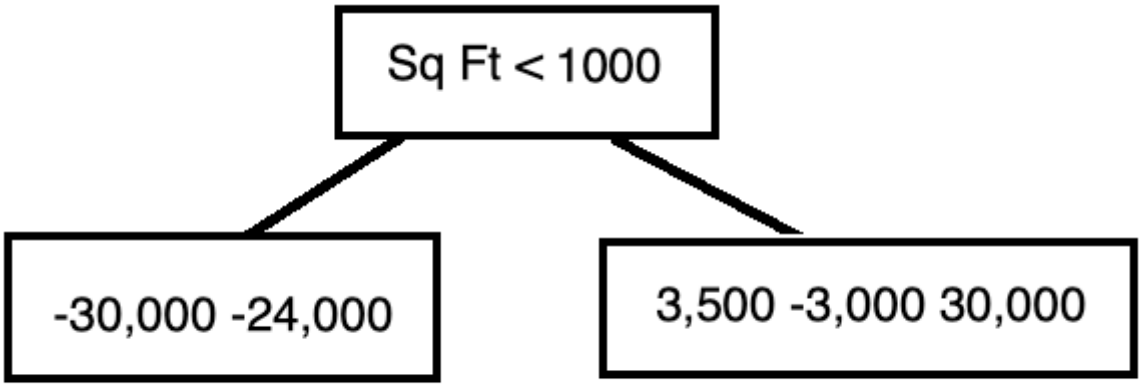
$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = -30,000 + -24,000 + 2,500 + -3,000 = -54,500$$
$$H_L = 4$$
$$G_R = 30,000$$
$$H_R = 1$$

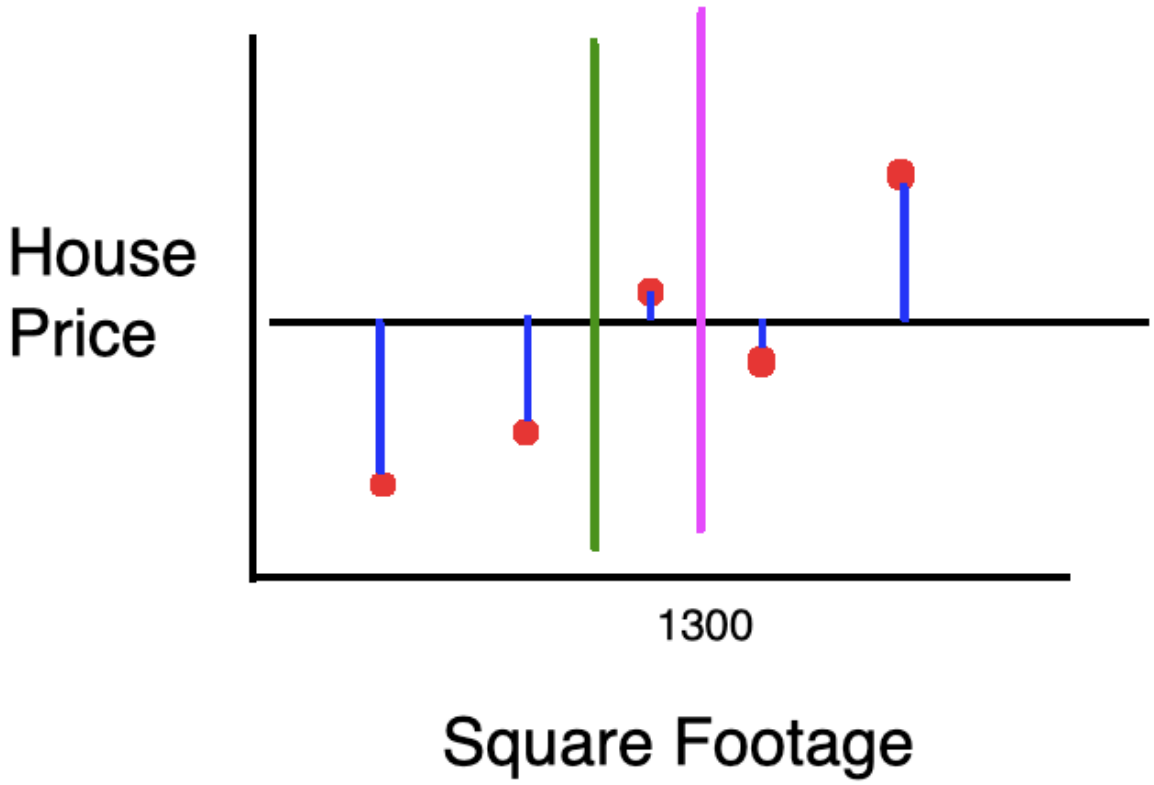
$$Gain = \frac{(-54,500)^2}{4+0} + \frac{(30,000)^2}{1+0} - \frac{(-54,500+30,000)^2}{4+1+0} - 100,000,000$$

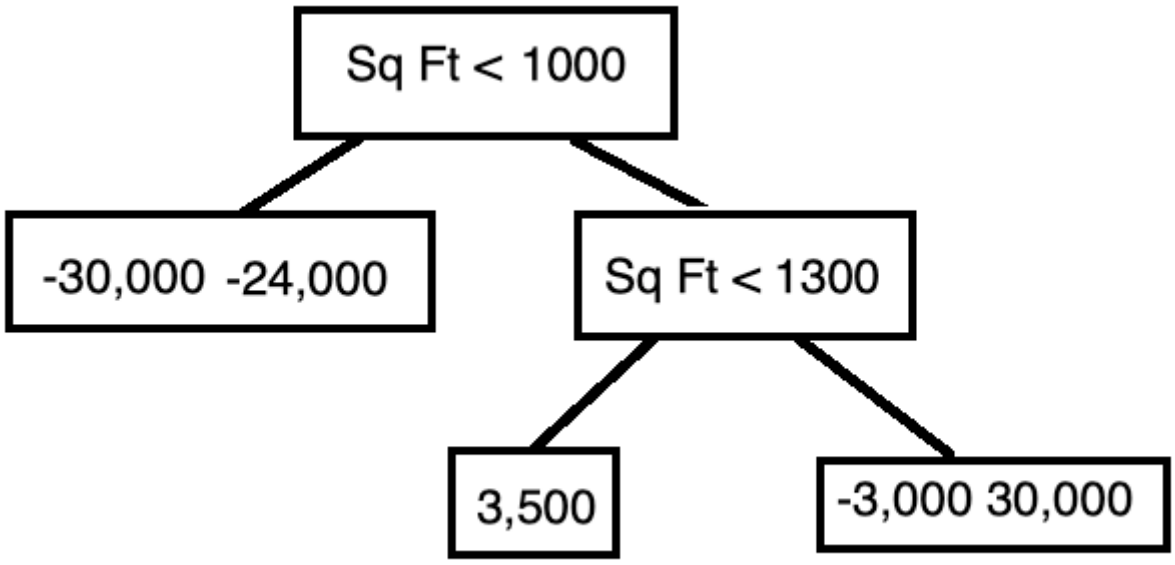
$$Gain = 1,422,512,500$$

Then, we use the threshold that resulted in the maximum gain. In this case, the optimal threshold is **Sq Ft < 1000**. Thus, we end up with the following tree.



We repeat the process for each of the leaves. That is to say, we select a threshold to



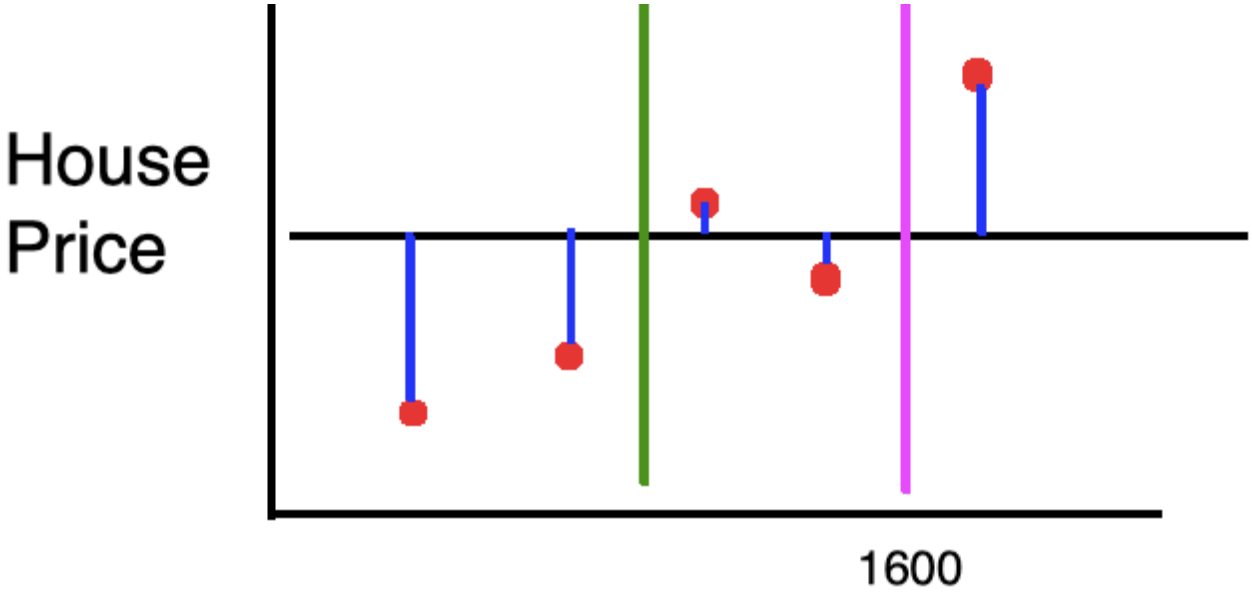


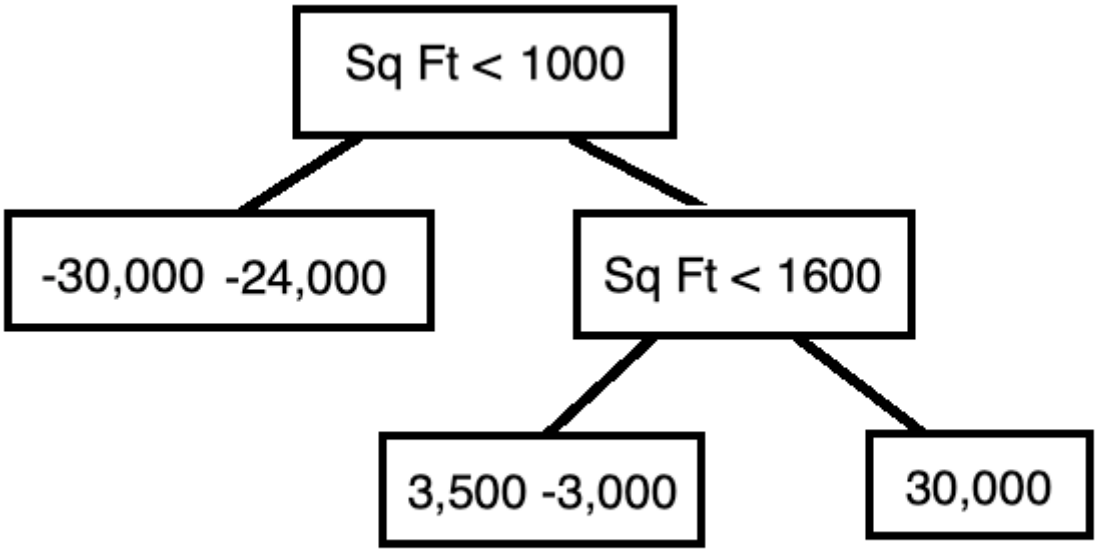
$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = 3,500$$
$$H_L = 1$$
$$G_R = -3000 + 30,000 = 27,000$$
$$H_R = 2$$
$$Gain = \frac{(3,500)^2}{1+0} + \frac{(27,000)^2}{2+0} - \frac{(3,500+27,000)^2}{1+2+0} - 100,000,000$$
$$Gain = -33,333,333$$

When the gain is negative, it implies that the split does not yield better results than would otherwise have been the case had we left the tree as it was.

We still need to check that a different threshold used in splitting the leaf doesn't improve the model's accuracy.





$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

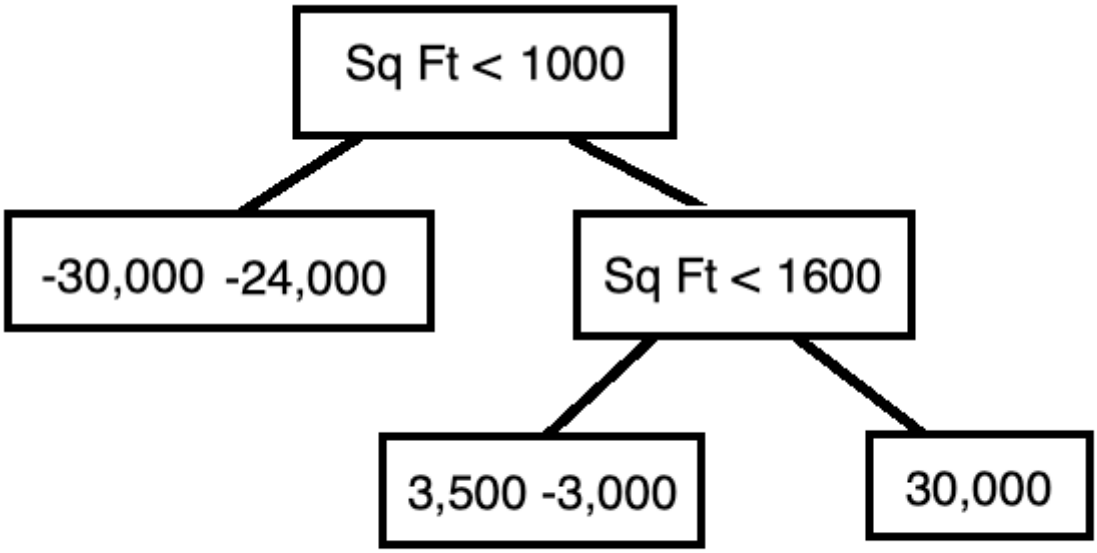
$$G_L = 3,500 + -3,000 = 500$$
$$H_L = 2$$
$$G_R = 30,000$$

$$H_R = 1$$

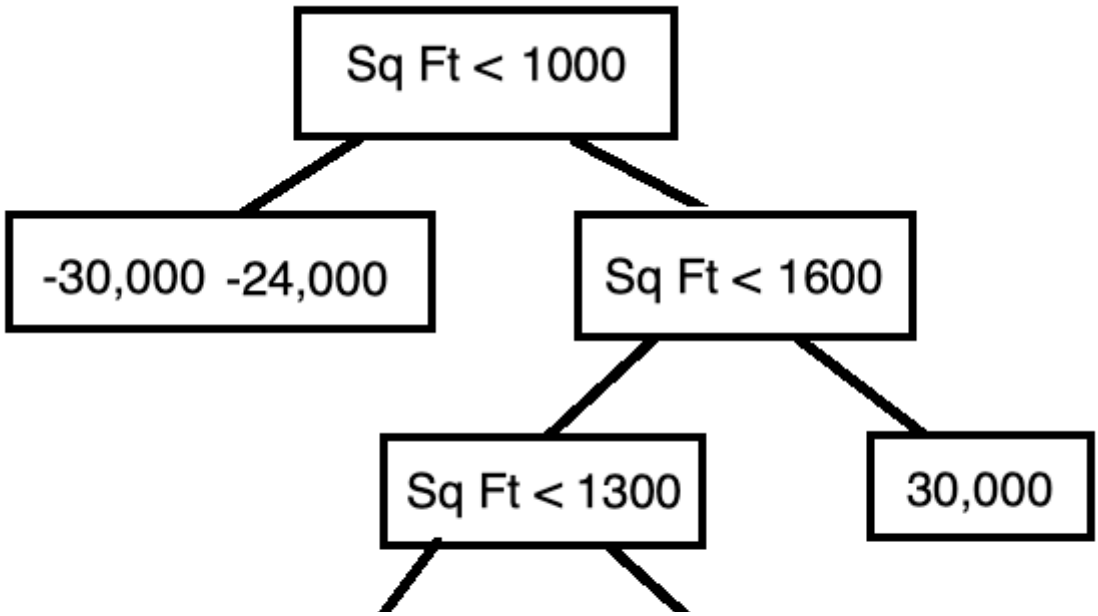
$$Gain = \frac{(500)^2}{2+0} + \frac{(30,000)^2}{1+0} - \frac{(500+30,000)^2}{2+1+0} - 100,000,000$$

$$Gain = 490,041,667$$

The gain is positive. Therefore, we still benefit from splitting the tree further. In doing so, we end up with the following tree.



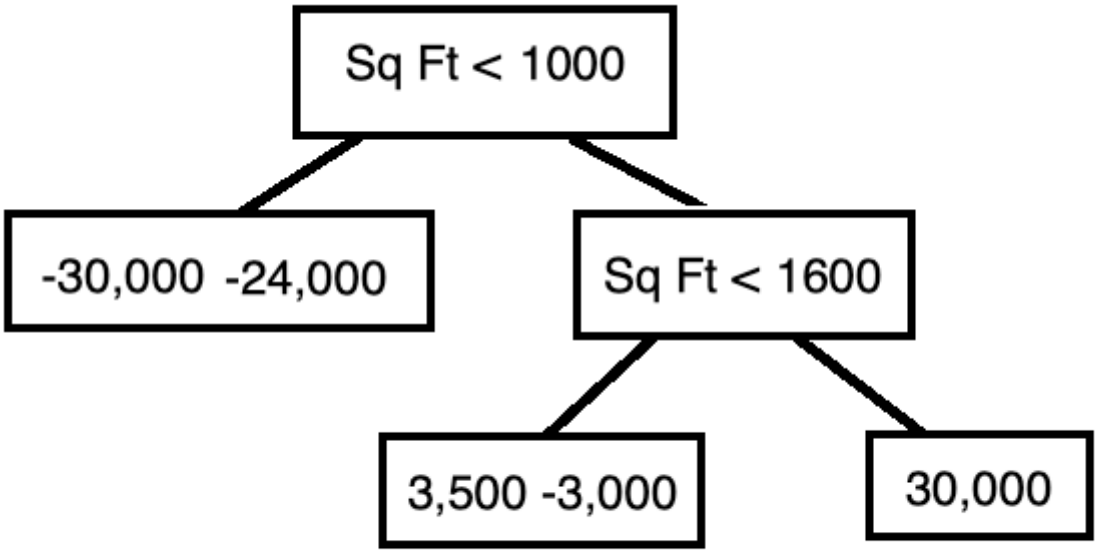
We examine whether it would beneficial to split the whose samples have a square footage between 1,000 and 1,600.



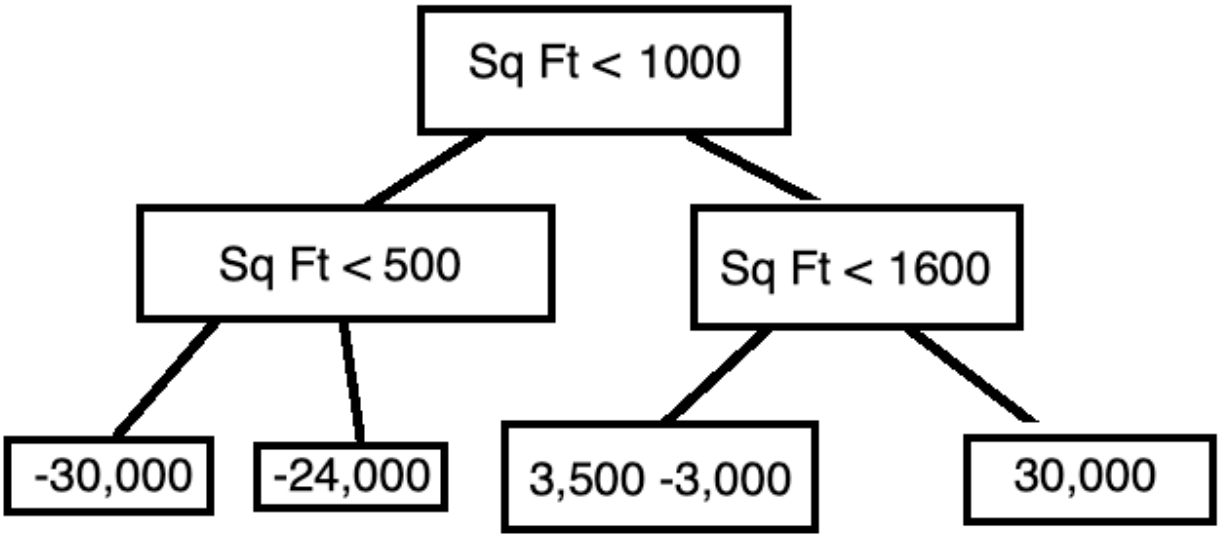
$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = 3,500$$
$$H_L = 1$$
$$G_R = -3,000$$
$$H_R = 1$$
$$Gain = \frac{(3,500)^2}{1+0} + \frac{(-3,000)^2}{1+0} - \frac{(-3,000+3,500)^2}{1+1+0} - 100,000,000$$
$$Gain = -78,875,000$$

The gain is negative. Therefore, we leave the tree as it is.



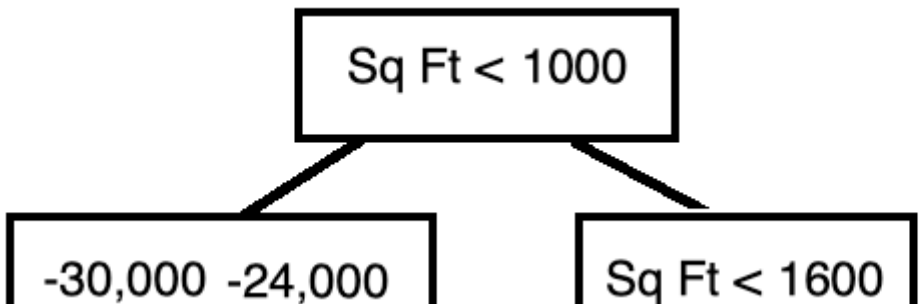
We still need to check whether we should split the leaf on the left (square footage < 1000).

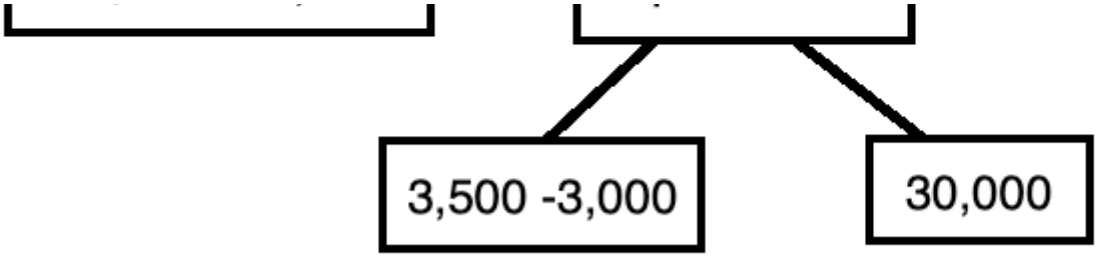


$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

$$G_L = -30,000$$
$$H_L = 1$$
$$G_R = -24,000$$
$$H_R = 1$$
$$Gain = \frac{(-30,000)^2}{1+0} + \frac{(-24,000)}{1+0} - \frac{(-30,000-24,000)^2}{1+1+0} - 100,000,000$$
$$Gain = -82,000,000$$

Again, the gain is negative. Therefore, the final decision tree is:





When presented with a sample, the decision tree must return a single scalar value. Therefore, we use to following formula that takes into account multiple residuals in a single leaf node.

$$Output = \frac{\sum residuals}{\#residuals + \lambda}$$

$$Output = \frac{-54,000}{2+0} = -27,000$$

$$Output = \frac{500}{2+0} = 250$$

$$Output = \frac{30,000}{1+0} = 30,000$$

The first prediction is the sum of the initial prediction and the prediction made by the tree multiplied by the learning rate.

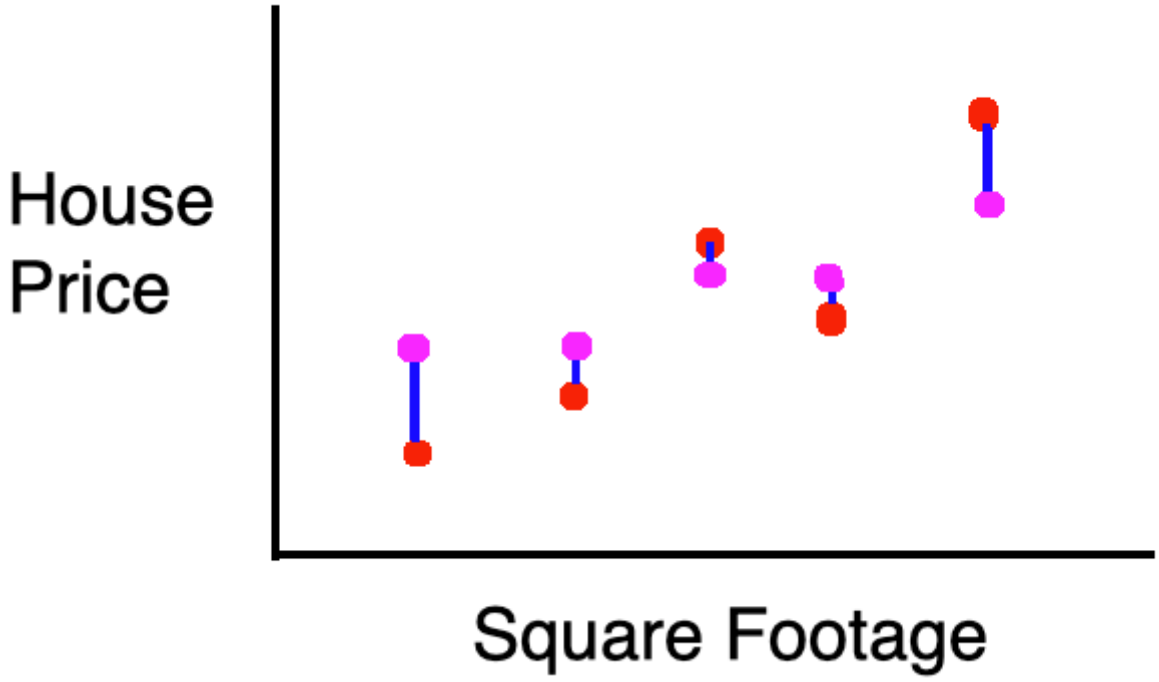
$$Prediction = Initial\ Prediction + Learning\ Rate\ * Prediction$$

Assuming a learning rate of 0.5, the model makes the following predictions.

$$Prediction = 100,000 + 0.5 * -27,000 = 86,500$$

$$Prediction = 100,000 + 0.5 * 250 = 100,125$$

$$Prediction = 100,000 + 0.5 * 30,000 = 115,000$$



The new residuals are:

-16,500 / -10,500 / -2,375 / 3,125 / 15,000

We then use these residuals to construct another decision tree, and repeat the process until we've reached the maximum number of estimators (default of 100). Once we've finished training the model, the predictions made by the XGBoost model as a whole are the sum of the initial prediction and the predictions made by each individual decision tree multiplied by the learning rate.

$$Prediction = Initial\ Prediction + Learning\ Rate * Prediction_1 + Learning\ Rate * Prediction_2 + \cdots$$

Python Code

Unlike other machine learning models, XGBoost isn't included in the Scikit-Learn package. Therefore,

The XGBoost library has a lot of dependencies that can make installing it a nightmare. Lucky for you, I went through that process so you don't have to. By far, the simplest way to install XGBoost is to install Anaconda (if you haven't already) and run the following commands.

```
conda install -c conda-forge xgboost

conda install -c anaconda py-xgboost
```

Once, we have XGBoost installed, we can proceed and import the desired libraries.

```
import pandas as pd
import xgboost as xgb
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Just like in the example from above, we'll be using a XGBoost model to predict house prices. We use the Scikit-Learn API to load the Boston house prices dataset into our notebook.

```
boston = load_boston()
X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.Series(boston.target)
```

We use the head function to examine the data.

```
X.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14

1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Here's the list of the different features and their acronyms.

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in $1000's

In order to evaluate the performance of our model, we split the data into training and test sets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Next, we initialize an instance of the XGBRegressor class. We can select the value of Lambda and Gamma, as well as the number of estimators and maximum tree depth.

```
regressor = xgb.XGBRegressor(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)
```

We fit our model to the training set.

```
regressor.fit(X_train, y_train)
```

We can examine the relative importance attributed to each feature, in determining the house price.

```
pd.DataFrame(regressor.feature_importances_.reshape(1, -1),
columns=boston.feature_names)
```


As we can see, the percentage of the lower class population is the greatest predictor of house price.

Finally, we use our model to predict the price of a house in Boston given what it has learnt.

```
y_pred = regressor.predict(X_test)
```

We use the mean squared error to evaluate the model performance. The mean squared error is the average of the differences between the predictions and the actual values squared.

```
mean_squared_error(y_test, y_pred)
```

Thanks to Anne Bonner.

Machine Learning | Data Science | Artificial Intelligence | Programming | Technology