Supervised Regression Problem: Housing Prices

Théo DRUILHE, Sigurd SAUE

December 24, 2024

Contents

1 Optimization Issues

1.1 Model Selection

To predict housing prices in this regression problem, we first apply a Regression Tree and then leverage the more advanced XGBoost (Extreme Gradient Boosting) model. This two-step approach allows us to understand the effectiveness of simple decision tree models before advancing to more complex ensemble techniques.

Why a Regression Tree should perform well?

Regression Trees are well-suited for datasets with non-linear relationships between features and the target variable. In a housing dataset, the price of a property depends on multiple factors, such as location, size, age, and neighborhood. A Regression Tree captures these dependencies by splitting the data into subsets based on feature thresholds, recursively reducing the variance of the target variable. It is also unsensitive to outliers, which are very present in our database.

But a Regression tree has limitations such as sensitivity to noise and a tendency to overfit. These issues motivate the use of XGBoost as the next step.

Why XGBoost should perform even better?

XGBoost is a powerful ensemble learning algorithm that builds upon the principles of Regression Trees by combining them in a gradient boosting framework. This allows XGBoost to address the limitations of standalone Regression Trees and achieve superior predictive performance.

- **Gradient Boosting Mechanism**: XGBoost iteratively improves predictions by learning from the errors of previous trees. This process reduces bias and ensures better generalization, capturing complex interactions between features like location, size, and amenities.
- **Robustness**: XGBoost is inherently robust to missing data and outliers:
  - **Missing Data**: It learns the optimal split direction for missing values during training, handling incomplete records in the housing dataset efficiently.
  - **Outliers**: Its use of CART-based trees makes it less sensitive to extreme values, such as unusually expensive properties.
- **Regularization to Prevent Overfitting**: XGBoost incorporates both L_1-regularization (Lasso) and L_2-regularization (Ridge) to penalize overly complex models, reducing overfitting. This is particularly important for housing datasets, where correlated features can lead to overly complex trees.
- **Feature Importance Metrics**: XGBoost provides detailed insights into feature importance, helping identify key drivers of housing prices. For example, it can reveal whether square footage or proximity to schools has a more significant impact.
- **Proven Success**: XGBoost has consistently outperformed simpler models, such as Regression Trees, in structured data problems. Its demonstrated success in competitions and practical applications makes it an ideal choice for housing price prediction.

By starting with a Regression Tree, we establish a baseline model that captures the fundamental relationships in the data. Advancing to XGBoost enhances the performance further, leveraging the strengths of ensemble learning and regularization to produce more accurate and reliable predictions for housing prices.

1.2 Model Theory : Regression Tree

A Regression Tree is a decision tree algorithm used for regression tasks, which predicts continuous output values. It works by recursively partitioning the input space and fitting simple models (usually constants) to each partition. Below is a detailed explanation of the process:

1. Splitting the Data

Regression trees divide the data into subsets based on feature values to minimize a cost function. For a given feature X_j and threshold s, the dataset $\mathcal{D}$ is split into two child nodes:

$$\mathcal{D}_{\text{left}}(j, s) = \{(x_i, y_i) \mid x_{i,j} \leq s\},$$
$$\mathcal{D}_{\text{right}}(j, s) = \{(x_i, y_i) \mid x_{i,j} > s\},$$

where $x_{i,j}$ is the value of the j-th feature for the i-th data point.

2. Cost Function (Minimizing Variance)

The goal of the tree is to find the feature j and threshold s that minimize the variance of the target variable y in the child nodes. The cost function for a split is defined as:

$$C(j, s) = \frac{1}{|\mathcal{D}_{\text{left}}|} \sum_{i \in \mathcal{D}_{\text{left}}} (y_i - \bar{y}_{\text{left}})^2 + \frac{1}{|\mathcal{D}_{\text{right}}|} \sum_{i \in \mathcal{D}_{\text{right}}} (y_i - \bar{y}_{\text{right}})^2,$$

where:

- $\bar{y}_{\text{left}} = \frac{1}{|\mathcal{D}_{\text{left}}|} \sum_{i \in \mathcal{D}_{\text{left}}} y_i$ is the mean target value in the left node.
- $\bar{y}_{\text{right}} = \frac{1}{|\mathcal{D}_{\text{right}}|} \sum_{i \in \mathcal{D}_{\text{right}}} y_i$ is the mean target value in the right node.
- $|\mathcal{D}_{\text{left}}|$ and $|\mathcal{D}_{\text{right}}|$ are the number of data points in the left and right child nodes, respectively.

The algorithm selects the feature j and threshold s that minimize $C(j, s)$.

3. Stopping Criterion

The tree construction stops when a stopping criterion is met. Common criteria include:

- Minimum number of samples in a node.
- Maximum depth of the tree.
- Minimum reduction in variance achieved by a split.

4. Predictions

Once the tree is constructed, predictions for a new input x are made by traversing the tree from the root to a leaf:

$$\hat{y} = \bar{y}_{\text{leaf}},$$

where $\bar{y}_{\text{leaf}}$ is the mean target value of the samples in the leaf node reached by x.

5. Regularization Techniques

To improve the robustness of regression trees, regularization techniques are applied:

- **Pruning**: Removes splits that do not significantly improve the model's performance.
- **Minimum leaf size**: Enforces a minimum number of data points in each leaf node.
- **Maximum depth**: Limits the depth of the tree to prevent overfitting.

1.3 Model Theory : XGBoost

XGBoost is a powerful machine learning algorithm that excels in predictive performance for structured data problems. It is based on ensembles of Regression Trees, which are optimized using gradient boosting and regularization.

1. Objective Function

XGBoost optimizes a regularized objective function that balances model accuracy and complexity. The objective function is defined as:

$$\mathcal{O}(\Theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t),$$

where:

- $L(y_i, \hat{y}_i)$ is the loss function, measuring the difference between the true value y_i and the predicted value $\hat{y}_i$.
- $\Omega(f_t)$ is the regularization term for the t-th tree.
- f_t represents the t-th decision tree in the ensemble.

The regularization term $\Omega(f_t)$ is expressed as:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \|w\|^2,$$

where:

- T is the number of leaves in the tree.
- $\|w\|^2$ is the L_2-norm of the leaf weights.
- γ and λ are regularization parameters that control the complexity of the model.

2. Additive Learning Process

XGBoost builds the model iteratively by adding one tree at a time. At iteration t, the model prediction is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i),$$

where $f_t(x_i)$ is the prediction from the new tree f_t for data point x_i.

3. Taylor Approximation of the Loss Function

To efficiently optimize the objective function, XGBoost uses a second-order Taylor expansion of the loss function around the current prediction $\hat{y}_i^{(t-1)}$:

$$L(y_i, \hat{y}_i^{(t)}) \approx L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2,$$

where:

- $g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$ is the first derivative of the loss function (gradient).
- $h_i = \frac{\partial^2 L(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$ is the second derivative of the loss function (Hessian).

The objective function for iteration t becomes:

$$\mathcal{O}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t).$$

4. Structure of a Decision Tree

Each decision tree partitions the data into leaves. The predicted value for a data point x_i is the weight w_j of the leaf j where x_i falls. The objective function for a single tree can be rewritten as:

$$\mathcal{O}^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} H_j w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2,$$

where:

- $G_j = \sum_{i \in I_j} g_i$ is the sum of gradients for leaf j.
- $H_j = \sum_{i \in I_j} h_i$ is the sum of Hessians for leaf j.
- I_j represents the set of data points in leaf j.

The optimal weight for each leaf is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}.$$

The optimal value of the objective function after adding the tree is:

$$\mathcal{O}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T.$$

5. Splitting Criterion

To construct the tree, XGBoost evaluates the gain from splitting a node. The gain is defined as:

$$\text{Gain} = \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right) - \gamma,$$

where:

- G_L and H_L are the gradient and Hessian sums for the left child node.
- G_R and H_R are the gradient and Hessian sums for the right child node.

The algorithm chooses the split that maximizes the gain.

6. Regularization and Final Prediction

Regularization ensures that the model does not overfit the data. After building all T trees, the final prediction is:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i).$$

1.4 Hyperparameter Tuning

Hyperparameter tuning is essential for optimizing the performance of Regression Trees and XGBoost, especially for a housing price dataset, where capturing complex relationships between features and the target variable is critical.

Hyperparameters Tuning for Regression Trees

For Regression Trees, the key hyperparameters to tune are:

- **max_depth**: Controls the maximum depth of the tree. A larger value allows the model to capture more complex patterns but increases the risk of overfitting.
- **min_samples_split**: Specifies the minimum number of samples required to split a node. Higher values prevent overly specific splits, improving generalization.
- **min_samples_leaf**: Defines the minimum number of samples required in a leaf node. This hyperparameter ensures that leaf nodes do not represent outliers or overly small groups of data points.
- **max_features**: Determines the maximum number of features considered for a split. Limiting this value (e.g., to `sqrt` or `log2`) can reduce overfitting by introducing randomness.

Hyperparameters Tuning for XGBoost

XGBoost introduces additional hyperparameters that control the boosting process:

- **max_depth**: Similar to Regression Trees, it controls the depth of individual trees. Larger values capture more interactions but increase model complexity.

- **learning_rate** (or `eta`): Shrinks the contribution of each tree to the overall model, helping to avoid overfitting. Smaller values improve generalization but require more trees.
- **n_estimators**: Specifies the number of boosting trees. This parameter works in conjunction with the learning rate to balance bias and variance.
- **subsample**: Defines the fraction of the training data used to grow each tree. Values between 0.5 and 1.0 are common for controlling overfitting.
- **colsample_bytree**: Specifies the fraction of features considered when splitting a node. This introduces randomness, improving model robustness.
- **gamma**: Represents the minimum loss reduction required for a split. Higher values result in more conservative splits.
- **reg_alpha** (L1 regularization) and **reg_lambda** (L2 regularization): Penalize large leaf weights, improving generalization and handling feature correlations.

Grid Search method

We use the GridSearch method to optimize both models by testing a lot of values for each hyperparameter. We can evaluate each combination of hyperparameters with the cross-validation method. This means we divide our train set into 5 equal-sized random folds, and train the model on 4 folds while using the remaining fold for validation. This process is repeated 5 times, with each fold serving as the validation set once, and the results are averaged to provide a robust estimate of model performance for each hyperparameter combination.

Mean Square Error (MSE)

We decide to use the Mean Square Error (MSE) as a scoring method to evaluate our regression models. The MSE measures the average squared difference between the actual target values (y_i) and the predicted values ($\hat{y}_i$). It is defined mathematically as:

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

Where:

- n: Total number of data points.
- y_i: Actual value of the target variable for the i^{th} observation.
- $\hat{y}_i$: Predicted value of the target variable for the i^{th} observation.

The MSE penalizes larger errors more heavily due to the squaring operation, making it sensitive to outliers. A lower MSE indicates better model performance, as the predictions are closer to the actual values.