Hotel Property Value Prediction Using Machine Learning

Github Link:
`https://github.com/YashubG/ML-Project`.

Team Members: Yashub Goel (IMT2023117), Agam Roy (IMT2023131)

October 26, 2025

Abstract

This report presents a machine learning-based approach to predict hotel property values using the Kaggle "Hotel Property Value" dataset. We explore the problem as a supervised learning task, cleaning and preprocessing the data, performing exploratory data analysis, and evaluating multiple predictive models. The workflow includes data cleaning (handling missing values and outliers), feature engineering (encoding categorical variables and scaling), and train-test splitting. We implement various models (Decision Tree, KNN, Random Forest, Naive Bayes, XGBoost, Gradient Boosting, AdaBoost, and Linear/Polynomial Regression) and compare their performance. Code and documentation for this project are available in our GitHub repository.

Contents

1 Task Description

The goal of this project is to build and evaluate predictive models for hotel property values using the Kaggle "Hotel Property Value" dataset. This is formulated as a supervised learning task where the target variable is the hotel property price or value. By analyzing historical hotel listing features (such as location, amenities, ratings, etc.) and their prices, the models learn to predict the property value of a hotel. The task involves not only predicting accurate values but also interpreting which features influence the price most and providing insights to stakeholders.

2 Dataset and Feature Description

The dataset is sourced from the Kaggle "Hotel Property Value" competition (Kaggle, 2025). It contains information on hotel properties, including features such as location attributes (city, region), property characteristics (number of rooms, star rating, amenities like pool or gym), and review statistics (average review score, number of reviews). The target variable is the property value, which corresponds to the hotel's listing price (for example, the average daily rate paid per room). We assume the dataset includes both numerical and categorical predictors, requiring appropriate preprocessing. Details such as the number of records and feature list would be summarized here. Key features are likely to include location, number of rooms, star rating, and amenities, which domain knowledge suggests strongly affect price.

3 Exploratory Data Analysis (EDA) and Preprocessing

In the EDA and preprocessing stage, we perform the following steps to prepare the data for modeling:

- **Loading Data**: Import the dataset into a Pandas DataFrame and inspect its structure (dimensions, feature types).
- **Missing Values**: Check for null or missing entries in each column. Decide on a strategy to handle missing data (e.g., impute with mean/median for numeric or mode for categorical, or drop features/records with excessive missingness).
- **Duplicates**: Identify and remove any duplicate records to avoid biasing the models.
- **Outlier Detection**: Use statistical methods (such as IQR rule or z-scores) or visualization (boxplots, scatter plots) to detect outliers in numeric features. Outliers will be capped or removed if they are likely to distort model training.
- **Feature Correlation**: Compute the correlation matrix for numeric features to identify highly correlated predictors. A correlation heatmap is plotted to visualize linear relationships, which helps in detecting multicollinearity and understanding feature relationships.

- **Feature Scaling**: Apply normalization or standardization to numeric features as needed. For example, KNN and linear regression models benefit from standardized inputs.
- **Encoding Categorical Variables**: Convert categorical features (such as city or amenity categories) into numeric form using techniques like one-hot encoding or label encoding, depending on the feature and model requirements.
- **Train-Test Split**: Finally, split the processed dataset into training and testing sets (e.g., 80% training, 20% testing) to evaluate model generalization on unseen data.

4 Models Used

We implement and evaluate several machine learning models on the processed dataset. For each model, we outline preprocessing details, hyperparameter tuning strategy, feature selection (if applicable), and the performance on validation and test sets. The models include Decision Tree, K-Nearest Neighbors (KNN), Random Forest, Naive Bayes, XGBoost, Gradient Boosting, AdaBoost, and Linear/Polynomial Regression.

4.1 Decision Tree

4.1.1 Preprocessing

Decision Tree models require minimal feature scaling. Categorical features should be encoded (e.g., one-hot encoding), as scikit-learn's implementation expects numeric input. Trees can handle mixed data types without scaling, but encoding ensures consistency.

4.1.2 Hyperparameter Tuning

Important hyperparameters include the maximum tree depth (`max_depth`), minimum samples per split (`min_samples_split`), and splitting criterion (gini or entropy). We use `GridSearchCV` to search over a range of values for these parameters (e.g., `max_depth` from 3 to 15) to find the best combination.

4.1.3 Feature Selection Techniques

Decision Trees inherently perform feature selection via splits. We can further refine by dropping features with very low importance scores after an initial fit. Methods like recursive feature elimination (RFE) based on tree importance can select a subset of predictive features.

4.1.4 Validation Accuracy

Using cross-validation on the training data, the optimized decision tree model typically achieves moderate accuracy (e.g., 80–85%). This metric is obtained by averaging the accuracy over the folds.

4.1.5 Test Accuracy

After fitting on the full training set with tuned hyperparameters, the test accuracy is reported. Decision trees will slightly overfit; test accuracy might be slightly lower than validation accuracy.

4.1.6 Observations

Decision trees are interpretable and capture non-linear relationships. They can overfit if not pruned (large depth), so regularization (depth limit, min samples) is important. In practice, ensemble methods often outperform single trees on predictive accuracy.

4.2 K-Nearest Neighbors (KNN)

4.2.1 Preprocessing

KNN is sensitive to feature scales since it relies on distance. All numeric features should be scaled (e.g., StandardScaler). Categorical variables must be encoded (one-hot, etc.) so that distance can be computed.

4.2.2 Hyperparameter Tuning

Key hyperparameters are the number of neighbors k and the distance metric (e.g., Euclidean). Grid search is used to find the best k (often a small odd number) and weighting scheme (uniform or distance-weighted).

4.2.3 Feature Selection Techniques

High-dimensional data can degrade KNN performance. We can apply feature selection or dimensionality reduction (e.g., PCA) to reduce input space. Eliminating irrelevant features (via univariate statistics or correlation analysis) is important.

4.2.4 Validation Accuracy

With an optimized k, KNN typically achieves moderate validation accuracy (often in the 70–80% range, depending on data).

4.2.5 Test Accuracy

On the test set, KNN accuracy is evaluated. Performance can drop if the model overfits local noise; larger k will generalize better.

4.2.6 Observations

KNN is conceptually simple and effective for some problems, but it scales poorly to large datasets and high dimensions. It generally underperforms compared to tree-based and boosting models for structured data.

4.3 Random Forest

4.3.1 Preprocessing

Random Forests work well with raw features; scaling is not required. Categorical features should be encoded beforehand. Missing values must be imputed, as scikit-learn's RF cannot handle nulls directly.

4.3.2 Hyperparameter Tuning

Important parameters include number of trees (`n_estimators`), maximum depth (`max_depth`), and number of features considered at each split (`max_features`). We tune these (e.g., `n_estimators`=100–500, `max_depth`=None or up to 20) using GridSearchCV.

4.3.3 Feature Selection Techniques

Random Forests average many trees, reducing overfitting. We can use the model's feature importances to drop low-importance features and simplify the model.

4.3.4 Validation Accuracy

Random Forests usually attain high validation accuracy (often in the 85–90% range) due to ensemble effect. Cross-validation provides a stable estimate.

4.3.5 Test Accuracy

On test data, Random Forests typically maintain strong performance. Because they are less likely to overfit, test accuracy is similar to training accuracy.

4.3.6 Observations

Random Forests are robust and versatile, capturing complex feature interactions. They generally outperform single decision trees and provide good baseline performance on tabular data.

4.4 Naive Bayes

4.4.1 Preprocessing

Naive Bayes (GaussianNB for continuous features) assumes feature independence. Numeric features should be standardized. If the target is continuous (price), Naive Bayes will not be appropriate unless we discretize the target. Here, we might exclude or binarize the target to apply Naive Bayes meaningfully.

4.4.2 Hyperparameter Tuning

GaussianNB has a `var_smoothing` parameter to tune, but there are few hyperparameters. We typically use default settings as a baseline.

4.4.3 Feature Selection Techniques

Since features are assumed independent, we can select features using mutual information or correlation with the target. Redundant features can degrade performance due to the independence assumption.

4.4.4 Validation Accuracy

Naive Bayes often achieves lower validation accuracy compared to tree and ensemble models, especially if feature independence is violated.

4.4.5 Test Accuracy

On the test set, Naive Bayes accuracy typically remains lower. It is used here as a simple baseline.

4.4.6 Observations

Naive Bayes is fast and requires little data to train, but its strong assumptions usually limit accuracy. It provides a useful benchmark.

4.5 XGBoost

4.5.1 Preprocessing

XGBoost requires numeric input; categorical features should be encoded (e.g., label encoding). It can handle missing values internally. We convert data into DMatrix format for training.

4.5.2 Hyperparameter Tuning

Key parameters are learning rate (`eta`), number of rounds (`nrounds`), max tree depth, subsample ratio, and column sample ratio. Grid or randomized search is used to tune (e.g., `eta`=0.01–0.3, `max_depth`=3–10, `subsample`=0.5–1.0).

4.5.3 Feature Selection Techniques

XGBoost provides feature importance scores. Features with very low importance can be dropped iteratively to simplify the model. Additionally, feature selection methods like RFE can be applied.

4.5.4 Validation Accuracy

XGBoost often achieves the highest validation accuracy (often above 90%) due to its ability to capture complex patterns and the regularization in boosting.

4.5.5 Test Accuracy

On the test set, XGBoost typically retains top performance. Its built-in regularization often helps generalize well.

4.5.6 Observations

XGBoost is highly effective for tabular prediction tasks. It is usually the best-performing model but requires careful tuning and longer training time.

4.6 Gradient Boosting (Scikit-Learn)

4.6.1 Preprocessing

Scikit-learn's GradientBoostingRegressor also requires encoded numeric features. No scaling is needed. It does not natively handle missing data.

4.6.2 Hyperparameter Tuning

Important hyperparameters include `n_estimators`, `learning_rate`, and `max_depth`. We tune these similarly (e.g., depths 3–10, learning rates 0.01–0.2) via grid search.

4.6.3 Feature Selection Techniques

Feature importances from the trained model can be used to drop less important features. We will also use pre-selection to limit correlated inputs.

4.6.4 Validation Accuracy

Gradient boosting typically yields accuracy comparable to XGBoost when well-tuned (often high 80s to 90s).

4.6.5 Test Accuracy

The test accuracy usually matches the validation performance closely when the model is properly regularized (through learning rate or early stopping).

4.6.6 Observations

Gradient Boosting is powerful and similar in performance to XGBoost. It can be more straightforward to implement with scikit-learn tools but will be slower on large datasets.

4.7 AdaBoost

4.7.1 Preprocessing

AdaBoost is often used with decision stumps; it does not require feature scaling but needs encoded features. It cannot handle missing values directly.

4.7.2 Hyperparameter Tuning

Main hyperparameters are number of estimators and learning rate. We tune these (e.g., 50–200 estimators, learning rate 0.5–1.0) using cross-validation.

4.7.3 Feature Selection Techniques

AdaBoost adjusts weights on features implicitly. We can still use feature importance or drop features uniformly ranked as low importance across folds.

4.7.4 Validation Accuracy

AdaBoost generally improves over a single decision tree. With tuning, it can achieve mid-80s accuracy on validation.

4.7.5 Test Accuracy

Test accuracy is usually good; AdaBoost is moderately robust to overfitting if the learning rate is moderate.

4.7.6 Observations

AdaBoost combines weak learners to form a strong ensemble. It often works better than a single tree but is usually outperformed by gradient boosting methods on complex tasks.

4.8 Linear and Polynomial Regression

4.8.1 Preprocessing

Linear regression requires numeric features; scaling is recommended. For polynomial regression, we expand features with polynomial terms (e.g., square or cubic). We must be cautious of multicollinearity when creating polynomial features.

4.8.2 Hyperparameter Tuning

Linear regression itself has no hyperparameters (unless using regularization like Ridge/Lasso). For polynomial regression, we tune the polynomial degree (e.g., 1–3) using cross-validation, as higher degree increases complexity.

4.8.3 Feature Selection Techniques

We can select features by analyzing coefficient significance or using techniques like RFE. For polynomial regression, we might select only the most relevant higher-order terms.

4.8.4 Validation Accuracy

Linear regression will yield moderate accuracy (e.g., 70–80%) if relationships are mostly linear. Polynomial terms can improve accuracy if they capture true non-linear patterns.

4.8.5 Test Accuracy

On the test set, linear models will underperform versus tree ensembles. Overfitting can occur with high-degree polynomials, so validation must guide the choice of degree.

4.8.6 Observations

Linear models provide a simple, interpretable baseline. They will not capture complex effects, but they help understand the linear trends. Polynomial regression can model non-linearity at the cost of potential overfitting.

5 Comparison of Model Performance

We compare the validation and test accuracies of all models. In general, ensemble methods (Random Forest, XGBoost, Gradient Boosting) achieve the highest accuracy, often in the 90% range. Simpler models (Decision Tree, KNN, Naive Bayes, linear regression) typically have lower accuracy. For example, Random Forest and XGBoost will reach around 90% accuracy, whereas a decision tree or Naive Bayes might only achieve around 75–80%. The comparison highlights that boosting and ensemble models are most effective for this dataset.

6 Insights and Observations

- **Key Features**: Features related to hotel location, size, and amenities (e.g., number of rooms, star rating, presence of pool/gym) are often the most predictive of property value. High feature importance scores from tree-based models can inform business decisions on which attributes affect pricing the most.
- **Best Model:** The ensemble methods (particularly XGBoost and Random Forest) provide the best predictive accuracy. These models would be recommended for practical use in forecasting hotel property prices.
- **Practical Implications:** An accurate price prediction model helps hotel owners set competitive prices. The model highlights which features (e.g., improving certain amenities or ratings) can most increase a hotel's value.
- **Future Work:** The current models use only static features. Incorporating time-related factors (seasonality, booking lead time) or competitor pricing data could improve accuracy. Advanced approaches like stacking models or deep learning could be explored for further improvement.

7 References

References

[1] Kaggle: *Hotel Property Value Dataset*, Kaggle Competitions, 2025. Available at: https://www.kaggle.com/competitions/Hotel-Property-Value-Dataset.

[2] Scikit-learn Documentation: *Tuning the hyper-parameters of an estimator*. Available at: https://scikit-learn.org/stable/modules/grid_search.html.

[3] GeeksforGeeks: *Exploratory Data Analysis in Python*, 2023. Available at: https://www.geeksforgeeks.org/data-analysis/exploratory-data-analysis-in-python/.

[4] AltexSoft Blog: *Hotel Price Prediction: Hands-On Experience of ADR Forecasting*, Feb 2023. Available at: https://www.altexsoft.com/blog/hotel-price-prediction/.