House Price Prediction- Documentation

Kosinepalli Arjun Sai

March 15, 2025

1 Introduction

The accurate prediction of house prices is a crucial task with significant implications for real estate, finance, and urban planning. This project aims to develop and deploy a high-performance machine learning model capable of providing precise house price estimates based on a variety of housing attributes. The model is deployed as a REST API using FastAPI, allowing for easy integration with other applications and services. A Streamlit web application provides a user-friendly front-end for interacting with the API and obtaining real-time predictions.

2 Data Preprocessing & Feature Engineering

2.1 Dataset

The project utilizes the well-known California Housing Dataset, obtained from the `sklearn.datasets` module in Python. This dataset contains 20,640 instances, each representing a block group in California, and includes the following features:

- **MedInc:** Median income in the block group.
- **HouseAge:** Median house age in the block group.
- **AveRooms:** Average number of rooms per household.
- **AveBedrms:** Average number of bedrooms per household.
- **Population:** Total population in the block group.
- **AveOccup:** Average household occupancy.
- **Latitude:** Latitude of the block group.
- **Longitude:** Longitude of the block group.
- **MedHouseVal:** Median house value (the target variable).

2.2 Data Preprocessing Steps

1. **Exploratory Data Analysis (EDA).** A thorough EDA was conducted to gain insights into the data distribution, identify potential outliers, and understand relationships between features. This included:
   - Calculating summary statistics (mean, median, standard deviation, etc.) using `df.describe()`.
   - Visualizing feature distributions using histograms and box plots.
   - Computing and visualizing the correlation matrix using a heatmap, revealing strong correlations between `MedHouseVal` and `MedInc`, for instance.
2. **Handling Missing Values.** The dataset was carefully inspected for missing values using `df.isnull().sum()`. While the California Housing Dataset typically does not have missing values, the code was designed to handle them robustly. Any missing values would be imputed using the median of the respective feature to avoid introducing bias, a strategy appropriate for skewed distributions commonly found in real estate data.
3. **Feature Scaling.** To ensure that all features contribute equally to the model and to improve the performance of distance-based algorithms, feature scaling was applied using `StandardScaler` from `sklearn.preprocessing`. This transforms the data to have zero mean and unit variance. The scaled data was saved to `processed_data.csv` for reproducibility.

3 Model Selection & Training

3.1 Train-Test Split

The dataset was divided into training and testing sets using an 80/20 split via `train_test_split` from `sklearn.model_selection`. This ensures that the model is evaluated on unseen data, providing a reliable estimate of its generalization performance. The ‘random$_{s}tate‘parameterwassetforrep$

3.2 Model Training and Evaluation

Four different regression models were trained and evaluated to identify the best-performing algorithm for this specific task:

The models were evaluated using the following metrics:

Root Mean Squared Error (RMSE): The square root of the average squared difference between predicted and actual values. Lower values indicate better performance.

Mean Absolute Error (MAE): The average absolute difference between predicted and actual values. Lower values indicate better performance.

R^2 Score (Coefficient of Determination): A measure of how well the model explains the variance in the target variable. Values closer to 1 indicate a better fit.

As shown in Table 2, XGBoost outperformed the other models across all metrics, achieving the lowest RMSE and MAE, and the highest R^2 score. This indicates that XGBoost is the most suitable model for this particular dataset and prediction task.

Model	**Description**
Linear Regression	A fundamental model that assumes a linear relationship between the features and the target variable. Serves as a baseline for comparison.
Decision Tree	A non-parametric model that partitions the feature space into regions, making predictions based on the average target value within each region.
Random Forest	An ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting.
XGBoost	An optimized gradient boosting algorithm known for its high performance and efficiency, particularly on structured datasets.

Table 1: Models Trained and Evaluated

Model	**RMSE**	**MAE**	**R^2 Score**
Linear Regression	0.75	0.48	0.62
Decision Tree	0.68	0.41	0.70
Random Forest	0.55	0.33	0.82
XGBoost	**0.52**	**0.30**	**0.85**

Table 2: Model Evaluation Results

4 Hyperparameter Optimization

To further improve the performance of the Random Forest and XGBoost models, hyperparameter optimization was performed.

4.1 Random Forest Tuning

- **Parameters Tuned:**
  - `n_estimators`: [50, 100, 200] (The number of trees in the forest)
  - `max_depth`: [None, 10, 20] (The maximum depth of each tree)
- **Optimization Method:** `GridSearchCV` from `sklearn.model_selection` was used to perform an exhaustive search over the specified parameter grid. Cross-validation was employed to ensure robust evaluation and prevent overfitting.

4.2 XGBoost Tuning

- **Parameters Tuned:**
  - `n_estimators`: [50, 100, 200] (The number of boosting rounds)
  - `learning_rate`: [0.01, 0.1, 0.2] (The step size shrinkage used to prevent overfitting)

- **Optimization Method:** Similar to Random Forest, `GridSearchCV` with cross-validation was used to determine the optimal combination of hyperparameters.

Final Model: After hyperparameter optimization, XGBoost remained the best-performing model, further solidifying its selection as the final model.

4.3 Model Persistence

The trained and optimized XGBoost model was saved to disk as `house_price_model.pkl` using the `pickle` library. This allows for easy loading and reuse of the model without retraining, crucial for deployment and real-time prediction.

5 Deployment

The deployment strategy involved creating a RESTful API using FastAPI, optimizing the model for inference with ONNX, and deploying the API to Render. A user-friendly web application was also developed using Streamlit.

5.1 API Development (FastAPI)

FastAPI was chosen for its speed, ease of use, and automatic data validation capabilities. The trained XGBoost model was integrated into a FastAPI application, exposing a `/predict` endpoint. This endpoint accepts a JSON payload containing the house features and returns a JSON response with the predicted house price. FastAPI's automatic documentation generation (via Swagger UI) significantly simplifies API usage and testing.

5.1.1 Example API Request:

```
{
  "MedInc": 3.5,
  "HouseAge": 15,
  "AveRooms": 5.4,
  "AveBedrms": 1.2,
  "Population": 1500,
  "AveOccup": 3.0,
  "Latitude": 37.5,
  "Longitude": -122.0
}
```

Listing 1: Example API Request

5.1.2 API Response:

```
{
  "predicted_price": 356800.00
}
```

Listing 2: Example API Response

5.2 Model Optimization (ONNX)

To further improve the efficiency and speed of predictions, the trained XGBoost model was converted to the ONNX (Open Neural Network Exchange) format. ONNX provides a standardized representation of machine learning models, enabling optimized execution across different platforms and frameworks. `onnxruntime` was used to load and run the ONNX model, resulting in significantly reduced inference latency compared to the original `pickle` format.

5.3 Deployment on Render

The FastAPI backend was deployed to Render, a cloud platform that provides a simple and scalable solution for hosting web applications and APIs. Render handles the infrastructure and scaling, allowing for easy deployment and maintenance.

Live API Endpoint: https://house-price-prediction-oo91.onrender.com

6 Web Application (Streamlit UI)

A Streamlit web application was created to provide a user-friendly interface for interacting with the deployed API. Streamlit simplifies the development of data-driven web applications, requiring minimal code to create interactive components.

Interactive Input: The app features sliders for each house feature, allowing users to easily adjust the input values.

Real-time Predictions: The app makes asynchronous calls to the FastAPI `/predict` endpoint to fetch predictions in real-time.

Dynamic Display: The predicted house price is displayed dynamically, updating instantly as the user changes the input values.

Live Streamlit App: https://housepricepredictionbyarjun.streamlit.app

6.1 Running the Web App Locally

To run the Streamlit application locally, execute the following command in the terminal:

```
streamlit run streamlit_app.py
```

7 API Usage Guide

7.1 Local FastAPI Setup

To run the FastAPI application locally, follow these steps:

1. Install the required dependencies:

```
pip install -r requirements.txt
```

2. Start the Uvicorn server:

```
uvicorn app:app --host 0.0.0.0 --port 8000
```

The API will be accessible at **http://localhost:8000**.
Swagger UI (interactive API documentation) will be available at **http://localhost:8000/docs**.

7.2 Testing with cURL

The API can be tested using cURL or any other HTTP client. Here's an example cURL command to send a prediction request:

```
curl -X 'POST' \
  'https://house-price-prediction-oo91.onrender.com/predict' \
  -H 'Content-Type: application/json' \
  -d '{
  "MedInc": 3.5,
  "HouseAge": 15,
  "AveRooms": 5.4,
  "AveBedrms": 1.2,
  "Population": 1500,
  "AveOccup": 3.0,
  "Latitude": 37.5,
  "Longitude": -122.0
}'
```

8 Future Improvements

Several potential improvements could be implemented to further enhance the project:

Advanced Model Optimization: Explore more advanced ONNX optimization techniques and hardware acceleration (e.g., GPU utilization) to achieve even lower latency and higher throughput for the API.

Streamlit Cloud Deployment: Deploy the Streamlit application to Streamlit Cloud for improved scalability, reliability, and accessibility.

Model Explainability: Integrate SHAP (SHapley Additive exPlanations) or similar techniques to provide insights into feature importance and explain individual predictions, increasing model transparency and trust.

API Security: Implement robust authentication (e.g., API keys, JWT tokens) and rate-limiting to secure the API and prevent abuse.

Advanced Modeling Techniques: Investigate more sophisticated models, such as deep learning models (e.g., neural networks) or ensemble methods that combine multiple models, to potentially improve prediction accuracy.

Data Enrichment: Incorporate additional data sources, such as crime rates, school ratings, proximity to amenities, and economic indicators, to provide a more comprehensive and accurate representation of the housing market.

Continuous Integration and Continuous Deployment (CI/CD): Implement a CI/CD pipeline to automate the testing, building, and deployment process, ensuring faster iteration cycles and improved code quality.

Monitoring and Alerting: Set up monitoring and alerting systems to track API performance, resource utilization, and potential errors, allowing for proactive issue resolution.

9 Conclusion

This project successfully demonstrates the end-to-end development and deployment of a high-performance machine learning model for predicting house prices. The key accomplishments include:

Comprehensive data preprocessing, handling missing values, and feature scaling.

Training, evaluation, and comparison of multiple regression models, with XGBoost emerging as the best performer.

Hyperparameter optimization of the XGBoost model using GridSearchCV, further improving its accuracy.

Conversion of the trained model to ONNX format for optimized inference.

Development of a RESTful API using FastAPI and deployment to Render, providing a scalable and accessible prediction service.

Creation of a user-friendly Streamlit web application for interactive predictions.

This project provides a robust, scalable, and maintainable solution for house price prediction, serving as a valuable tool for various applications in real estate, finance, and urban planning. The implemented solution is designed for future expansion and improvement, incorporating best practices in machine learning and software engineering.

10 License

This project is licensed under the Apache 2.0 License. See the LICENSE file for more details.