

Random Forest Model for California Housing Price Prediction

NAME: Ankita Satapathy

ROLL NO: 22053400

1. Introduction

This project implements a Random Forest model to predict housing prices in California using the fetch_california_housing dataset. The model is implemented using a self-defined Random Forest class instead of sklearn's predefined version. It is deployed with a FastAPI-based backend and a front-end interface for user interaction.

2. Files in the Submission

- i. RandomForest3400.ipynb – Jupyter Notebook containing the implementation of the Random Forest model, data preprocessing, training, and evaluation.
- ii. RandomForest3400.py – Python script implementing the Random Forest model with self-defined logic using Decision Trees.

```
PS C:\Users\KIIT\Desktop\22053400\7) Random Forest> jupyter nbconvert --to script RandomForest3400.ipynb
```

- iii. main.py – FastAPI server script to load the trained model and handle prediction requests.
- iv. california_housing_rf_model.pkl – Trained Random Forest model serialized using Pickle.
- v. index.html – Front-end UI for user input and displaying predicted house prices.

3. Installation and Setup

i. Prerequisites

The following Python packages are required:

- NumPy (numpy) – For numerical computations and matrix operations.
- Pandas (pandas) – For dataset manipulation.
- Scikit-learn (sklearn) – Used only for dataset loading and basic utilities.
- FastAPI (fastapi) – To create a web API for model inference.
- Pickle (pickle) – To save and load the trained model.
- Uvicorn (uvicorn) – To run the FastAPI server asynchronously.

ii. Installing Dependencies

Run the following command in the terminal to install dependencies:

```
PS C:\Users\KIIT\Desktop\22053400\7) Random Forest> pip install fastapi uvicorn numpy pandas scikit-learn
```

4. Model Implementation

The SimpleRandomForest class in RandomForest3400.py implements a Random Forest model using multiple DecisionTreeRegressor instances. It trains `n_trees` decision trees using bootstrap sampling and aggregates predictions by averaging their outputs.

```
class SimpleRandomForest:
    def __init__(self, n_trees=10, max_depth=None):
        from sklearn.tree import DecisionTreeRegressor
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.trees = [DecisionTreeRegressor(max_depth=max_depth) for _ in range(n_trees)]
    def fit(self, X, y):
        for tree in self.trees:
            indices = np.random.choice(len(X), len(X), replace=True)
            X_sample, y_sample = X[indices], y[indices]
            tree.fit(X_sample, y_sample)
    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        return np.mean(tree_preds, axis=0)
```

5. Training the Model

- The dataset used is the California Housing Dataset from `sklearn.datasets`.
- Features include median income, house age, average rooms, population, and geolocation (latitude & longitude).
- Data is split into training and testing sets (80%-20%).
- The model is trained using 10 decision trees, each with a maximum depth of 5.
- After training, the model is saved using Pickle.

6. API Implementation

The FastAPI-based backend (`main.py`) loads the trained model and provides an endpoint to make predictions:

i. API Setup

```
8 app = FastAPI()
```

ii. CORS Configuration- To allow front-end requests:

```
14 app.add_middleware(
15     CORSMiddleware,
16     allow_origins=["*"],
17     allow_credentials=True,
18     allow_methods=["*"],
19     allow_headers=["*"],
20 )
```

iii. Loading the Trained Model

```
22 with open("california_housing_rf_model.pkl", "rb") as f:
23     model = pickle.load(f)
```

iv. Defining the API Endpoint

```
28 @app.post("/predict/")
29 async def predict_price(data: HouseFeatures):
30     try:
31         features = np.array(data.features).reshape(1, -1)
32         prediction = model.predict(features)[0]
33         return {"predicted_price": round(prediction, 2)}
34     except Exception as e:
35         return {"error": str(e)}
```

7. Running the Application (FastAPI Server)

To start the server, run the following command in the terminal:

```
PS C:\Users\KIIT\Desktop\22053400\7) Random Forest> uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\KIIT\\Desktop\\22053400\\7) Random Forest']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [5320] using WatchFiles
INFO: Started server process [13828]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

8. Front-End Implementation

The index.html file provides a user-friendly UI where users can input features such as median income, house age, average rooms, etc., and get the predicted price. The UI has a modern design with a gradient background, input fields, and a prediction button. JavaScript sends user input to the FastAPI backend and displays the predicted price dynamically.

```
197 document.getElementById("predictForm").addEventListener("submit", async function (e) {
198     e.preventDefault();
199     const features = [
200         parseFloat(document.getElementById("feature1").value),
201         parseFloat(document.getElementById("feature2").value),
202         parseFloat(document.getElementById("feature3").value),
203         parseFloat(document.getElementById("feature4").value),
204         parseFloat(document.getElementById("feature5").value),
205         parseFloat(document.getElementById("feature6").value),
206         parseFloat(document.getElementById("feature7").value),
207         parseFloat(document.getElementById("feature8").value)
208     ];
209     console.log("📦 Sending data:", features);
210     try {
211         const response = await fetch("http://127.0.0.1:8000/predict/", {
212             method: "POST",
213             headers: { "Content-Type": "application/json" },
214             body: JSON.stringify({ features })
215         });
216         if (!response.ok) throw new Error(`HTTP error! Status: ${response.status}`);
217         const result = await response.json();
218         document.getElementById("result").innerHTML =
219             `<strong>🏠 Predicted House Price:</strong> $${result.predicted_price * 1000}`;
```

```


220     } catch (error) {
221         console.error("❌ Fetch Error:", error);
222         document.getElementById("result").innerHTML = "❌ Error predicting. Check console.";
223     }
224 });

```

9. Conclusion

This project successfully implements a self-defined Random Forest model to predict California housing prices based on median income, house age, population, and geolocation data.


The model is trained without sklearn library. The FastAPI backend serves predictions, and the frontend UI allows user interaction, making the model accessible for real-time house price predictions.


 **California Housing Price Predictor**


Enter Details to Predict House Price

This project utilizes advanced machine learning models to estimate house prices in California.

Median Income:	House Age:
<input type="text" value="7"/>	<input type="text" value="5"/>
Average Rooms:	Average Bedrooms:
<input type="text" value="40"/>	<input type="text" value="80"/>
Population:	AveOccup:
<input type="text" value="250"/>	<input type="text" value="200"/>
Latitude:	Longitude:
<input type="text" value="36"/>	<input type="text" value="-121"/>

 **Predict Price**

 **Predicted House Price: \$3320**

 **Developed by Ankita Satapathy | 22053400**