House Price Prediction - Project Report

1. Introduction

This report presents the **House Price Prediction** project, which aims to predict the house prices based on various housing features such as latitude,longitude,income,total rooms etc using **XGBoost Regressor**.

The project includes data preprocessing, feature engineering, model selection, optimization, and deployment as a web API using Flask and front-end interface with StreamLit.

Data Ingestion

For this project we downloaded a dataset related to house price prediction in .csv file format.

We loaded the dataset as a Pandas DataFrame and we found there were 20640 rows and 13 columns in the dataset.

2. Data Preprocessing & Feature Engineering

Data Cleaning

- Removed two columns namely "Unnamed: 11" and "Unnamed: 12" from the dataset as they were empty columns.
- On checking for empty cells, we found there were a total of 224 empty rows, out of which 207 empty cells were for "total_bedrooms" feature. We imputed the median value for this column in those empty cells and dropped the empty cells for the remaining rows as they were negligible in number.

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             3
total_bedrooms        207
population              0
households              0
median_income           4
median_house_value      0
ocean_proximity        10
dtype: int64
```

- There were no duplicate rows in the dataset

- Changed the datatype of “households” column to float as it contained all numeric values.
- Combined the two columns namely “total_rooms” and “total_bedrooms” into a single column named “total_rooms”.

```
df['total_rooms']=df['total_rooms']+df['total_bedrooms']
df.drop(columns=['total_bedrooms'],inplace=True)
df
```

Feature Engineering

- Converted categorical variables (e.g., `ocean_proximity`) into numerical using **One-Hot Encoding**.

```
df = pd.get_dummies(df, columns=['ocean_proximity'], drop_first=True) #ONE-HOT encoding
```

- Dropped some columns which have high multi collinearity amongst themselves and less co-relation with the target variable.

```
dff=df.drop(columns=['longitude','population','ocean_proximity_ISLAND'])
```

- Standardized numerical features using **StandardScaler** to normalize data distribution.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train= scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)
```

3.Model Selection & Optimization

Model Selection

- I tested out different regression models such as Linear Regression, RandomForest Regressor and XGBoost Regressor .
- Among all of these the XGBoost Regressor gave the highest R2score.

```
MAE: 44200.56549125545, RMSE: 62769.828153740484, R2score: 0.7095756781274993
```

- So we selected XGBoost Regressor as our base model for performing further hyperparameter tuning on it.

Hyperparameter Tuning

- Used **RandomizedSearchCV** for optimizing hyperparameters.
- Tuned parameters such as `n_estimators`, `max_depth`, and `learning_rate`.

```python
# Define Hyperparameter Grid
params = {
    'n_estimators': [30,50],  # Number of trees
    'max_depth': [3, 6, 10],  # Maximum depth of each tree
    'learning_rate': [0.01, 0.05, 0.1, 0.2],  # Step size shrinkage
    'subsample': [0.6, 0.8, 1.0],  # Fraction of samples used per tree
        }

# Randomized Search with Cross-Validation
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=params,
    n_iter=10,
    cv=5,
    scoring='r2',
    random_state=40,

)
```

- We find the best parameters for the XGBoost Regressor and get a R2 score of 72 .

```
Best Parameters: {'subsample': 0.8, 'n_estimators': 50, 'max_depth': 10, 'learning_rate': 0.1}
MAE: 42375.05
RMSE: 61573.70
R2 Score: 0.72
```

3. Deployment Strategy & API Usage

- **Flask** was used to create a REST API that accepts JSON inputs and returns predicted house prices.
- Model and scaling were stored as **Pickle (.pkl) files**
- API deployed on **Render** for public accessibility.

API Usage Guide (Postman/Streamlit)

Using Postman

- Render URL : https://flask-api-house-price-prediction.onrender.com/

Sample JSON Input:

```json
{
  "latitude": 34.14,
  "housing_median_age": 20,
  "total_rooms": 5000,
  "households": 1000,
  "median_income": 4.5,
  "ocean_proximity_INLAND": 1,
  "ocean_proximity_NEAR BAY": 0,
  "ocean_proximity_NEAR OCEAN": 0
}
```

Response Example:

```json
{
  "prediction": 250000.75
}
```

Using Streamlit Web App

- A **Streamlit** frontend was developed for easy user interaction.
- Users input house features using the web form and receive a price prediction instantly.

References And Useful Links

Github repo:

https://github.com/debdoot9804/House_price_prediction_ML/tree/main

Postman API Testing Render link (Use JSON input):

https://flask-api-house-price-prediction.onrender.com/

StreamLit link:

https://housepricepredictionml-cdg4hdgdcxcfsucmipvfwh.streamlit.app/

Author : Debdoot Sen

Email: debdutsen111@gmail.com