2023

BDA105 DATA MINING AND MACHINE LEARNING CA2

BERNADETTE O'GRADY 40030457

Contents

Introduction

This project is focused on predicting house prices in Dublin using data sourced from the Daft.ie website for residential properties on sale. To collect this data, I am using the daftlistings package developed by (Bloomer, 2023) which limits the data collection to the last 6 months of listings.

Figure [1] shows the first 5 rows after one hot encoding and its shape.

```
encoded_dublin_property.head(5)
```

	Price	Bathrooms	Bedrooms	Size_Meters_Squared	Latitude	Longitude	Sections_Apartment Duplex	Sections_Bungalow	Sections_Detached House	Sections_End of Terrace House
0	1015000.0	3.0	4.0	154	53.289637	-6.229440	0	0	1	0
1	310000.0	1.0	2.0	57	53.360486	-6.246636	0	0	0	0
2	400000.0	2.0	4.0	153	53.320455	-6.305991	0	0	1	0
3	449000.0	3.0	3.0	134	53.454265	-6.237261	0	0	1	0
4	1525000.0	4.0	4.0	231	53.444920	-6.139887	0	0	1	0

```
encoded_dublin_property.shape
```

```
(3018, 140)
```

Figure 1 Encoded Dataset

Parishes got dropped from the dataset as latitude and longitude coordinates provide the exact location of each property, and including parish information may lead to overfitting and biased predictions. Therefore, the final dataset used for building the models is presented in Figure [2].

```
dublin_property.head(5)
```

	Price	Latitude	Longitude	Bedrooms	Bathrooms	Size_Meters_Squared
0	1015000.0	53.289637	-6.229440	4.0	3.0	154
1	310000.0	53.360486	-6.246636	2.0	1.0	57
2	400000.0	53.320455	-6.305991	4.0	2.0	153
3	449000.0	53.454265	-6.237261	3.0	3.0	134
4	1525000.0	53.444920	-6.139887	4.0	4.0	231

Figure 2 Final Dataset for building the Models

Figure [3] demonstrates the shape of the dataset for building the models.

```
dublin_property.shape
```

```
(3018, 6)
```

Figure 3 Shape of dataset for building the Models

Q1. Preparing and Parameter Selection for the models

Q2. Selection of Model Generation Techniques

House price predictions is a classic example of a Supervised Machine Learning Technique. It involves training the machine learning models using labelled data. The labelled data is the independent variables which are Latitude, Longitude, Bedrooms, Bathrooms and Size Meters squared along with the target variable House Price. During the training phase, the model learns the relationship between the independent variables and the dependent variable by minimizing the difference between its predicted values and the actual prices in the labelled data. Once the model is trained, it can be used to make predictions on new, unseen data.

This project involves predicting a continuous numerical value (house price) based on a set of input features (independent variables) so it falls under the category of Regression Models.

The Regression Models selected for this project are as follows:

1. ### Multiple Linear Regression

   This model assumes that the relationship between the independent variables and the dependent variable can be represented by a linear equation. The model estimates the coefficients of this equation by minimizing the difference between its predicted values and the actual house prices in the training data. Once trained, it can make predictions on new data by plugging in the values of the independent variables.

2. ### Decision Tree Regression

   This model works by recursively splitting the data into subsets based on the values of the independent variables, until it reaches a stopping criterion. Each split creates a node in the tree, and the values at the terminal nodes are used to make predictions. Decision trees are easy to interpret and can capture non-linear relationships between the independent and dependent variables.

3. ### Random Forest Regression

   This model works by building a large number of decision trees, each trained on a random subset of the data and a random subset of the features. The final prediction is obtained by aggregating the predictions of all the trees in the forest. Random forest is known for its ability to handle non-linear relationships and interactions between variables, and it can also handle missing data and outliers well.

4. ### Gradient Boosting Regression

   This model works by creating a series of weak learners (usually decision trees) that are trained sequentially to improve prediction accuracy. Each weak learner is trained on the residual errors of the previous one to improve the overall prediction accuracy.

Q3. Generating the Test Design

The 70/30 split approach has been used for this project for splitting the dataset into 70% training and 30% testing sets. This is a common approach in machine learning for evaluating the performance of each model. It provides a good balance between having enough data to train the model effectively and having enough data to test its accuracy.

The dataset for this project has 3018 records so this split provides a sufficiently large amount of data for training the model while also having enough data for testing the model's performance on unseen data.

To split the data into training and testing sets, the train_test_split function from the scikit-learn library in Python Figure [4] was used. This function randomly shuffles the data and splits it into two sets based on the ratio specified. The training set is used to fit the model, while the testing set is used to evaluate its performance on new, unseen data.

```
# Split into train and test sets
X = dublin_property.drop(['Price'], axis=1)
y = dublin_property['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('X_train:', X_train)
print('y_train:', y_train)
```

Figure 4 Code to Split dataset into train and test sets

The following Figures [5] [6] and [7] show the shapes of X_train, y_train, X_test and y_test after split.

```
print('X_train:',X_train.shape)
print('y_train:',y_train.shape)
```

```
X_train: (2112, 5)
y_train: (2112,)
```

Figure 5 Shape of X_train and y_train

```
X_test.shape
```

```
(906, 5)
```

Figure 6 Shape of X_test

```
y_test.shape
```

```
(906,)
```

Figure 7 Shape of y_test

a. Model Building

The following is the code used for building each Model.

Figure [8] outlines the code used to build the Multiple Linear Regression Model. Standard Scaling was applied to the data prior to building the Multiple Linear Regression Model. Standardizing the features can help to improve the performance of the model by ensuring that each feature has a similar scale and variance. This is important because multiple linear regression assumes that the input features have a linear relationship with the target variable, and having features with different scales or variances can lead to biased or inaccurate predictions.

1) Multiple Linear Regression Model

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

#Standard Scale the data for Linear Regression Model
# Create an object of StandardScaler class
ss=StandardScaler()

#means apply standard scaler for X_train and X_test data
X_train_scaled=ss.fit_transform(X_train)
X_test_scaled=ss.transform(X_test)

# Create an instance of the LinearRegression class
lr = LinearRegression()

# Fit the model to the scaled training data
lr.fit(X_train_scaled, y_train)

# Use the model to make predictions on the scaled test data
y_pred = lr.predict(X_test_scaled)
```

Figure 8 Multiple Linear Regression Model Code

Figure [9] outlines the code used to build the Decision Tree Regression Model.

Build DecisionTree Regression Model

```
from sklearn.tree import DecisionTreeRegressor
# Create an instance of the DecisionTreeRegressor class
dt = DecisionTreeRegressor(max_depth=3, random_state=42)

# Fit the model to the training data
dt.fit(X_train, y_train)

# Use the model to make predictions on the test data
y_pred = dt.predict(X_test)
```

Figure 9 Decision Tree Regression Model Code

Figure [10] outlines the code used to build the Random Forest Regression Model.

Build Random Forest Regression Model

```
# Random Forest Regression
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

Figure 10 Random Forest Regression Model Code

Figure [11] outlines the code used to build the Gradient Boosting Regression Model.

Build Gradient Boosting Regression Model

```
# Gradient Boosting Regression
gb = GradientBoostingRegressor()
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)
```

Figure 11 Gradient Boosting Regression Model Code

b. Models Assessment

The following Metrics are being used to evaluate each model's performance.

- **MSE (Mean Squared Error)** is a metric that measures the average squared difference between the predicted values and the actual values.
- **R-squared** is a statistical measure that represents the proportion of variance in the dependent variable that is predictable from the independent variables. The value ranges from 0 to 1, where 1 represents a perfect fit.
- **Explained Variance Score** measures how much of the variation in the target variable is explained by the model.
- **Mean Absolute Error (MAE)** measures the average absolute difference between the predicted values and the actual values.
- **Root Mean Squared Error (RMSE)** measures the standard deviation of the residuals, which are the differences between the predicted values and the actual values.
- **Mean Absolute Percentage Error (MAPE)** measures the average percentage difference between the predicted values and the actual values.

Figure [12] outlines an example of the code used to calculate these metrics for Regression Models.

Calculate Multiple Linear Regression Evaluation Metrics

```
# Calculate evaluation metrics
lr_mse = mean_squared_error(y_test, y_pred)
lr_r2 = r2_score(y_test, y_pred)
lr_evs = explained_variance_score(y_test, y_pred)
lr_mae = mean_absolute_error(y_test, y_pred)
lr_rmse = np.sqrt(lr_mse)
lr_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
```

Figure 12 Example of Code used to Calculate Metrics for Regression Models

Q4. Evaluating Model Robustness

a. Evaluating Results

The following are the Metric Results for each Model Figure [13].

	Model	MSE	R-squared	EVS	MAE	RMSE	MAPE
0	Random Forest Regression	1.40947e+10	0.899001	0.899006	75127.8	118721	13.214
1	Gradient Boosting Regression	1.48262e+10	0.893759	0.893777	80872.8	121763	14.5139
2	Linear Regression	2.59932e+10	0.813739	0.813739	111203	161224	21.0787
3	Decision Tree Regression	3.20145e+10	0.770592	0.770593	130677	178926	25.9981

Figure 13 Evaluation Metrics for each Model

All four models meet the goal of the project, as per the evaluation metrics for each model illustrated in Figure [13]. The metrics indicate that all four models accurately predict house prices in Dublin using the given input features. Hence, the goal of the project, which is to predict house prices with the available data, is successfully met by all four models albeit that some models out perform others.

The Random Forest Regression model has the lowest mean squared error (MSE) among all models, indicating the smallest average squared difference between the predicted and actual house prices. Its R-squared value of 0.899001 and EVS of 0.899006 suggest that the model can explain about 90% of the variance in the target variable. The mean absolute error (MAE) of 75127.8 indicates that, on average, the model's predictions are off by around €75,127.80, which is the smallest among all models. The root mean squared error (RMSE) of 118721 is also the smallest among all the models, indicating that the model has the smallest standard deviation of the errors between predicted and actual house prices. Finally, the mean absolute percentage error (MAPE) of 13.214% is the smallest among all the models, indicating that the model's average percentage error in predicting house prices is the lowest.

Therefore, the Random Forest Regression model is recommended as the final solution for the project.

b. Reviewing Process

I reviewed the process and decided to apply cross validation to each model to give a more reliable estimate of the Model's performance. As one can see from the results in Figure [14] and Figure [15] the Random Forest Regression Model wins as it has the highest average score (0.8896).

Apply Cross Validation to the Multiple Linear Regression Model

```
from sklearn.model_selection import KFold, cross_val_score
k_folds = KFold(n_splits = 10)

scores = cross_val_score(lr, X_train_scaled, y_train, cv = k_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Standard Deviation CV Score: ", scores.std())
print("Number of CV Scores used in Average: ", len(scores))
```

```
Cross Validation Scores:  [0.77566417 0.80605895 0.815142   0.82887964 0.85443147 0.84298817
 0.78423121 0.82998356 0.78399062 0.84390113]
Average CV Score:  0.8165270912644225
Standard Deviation CV Score:  0.02665118275434792
Number of CV Scores used in Average:  10
```

Figure 14 Cross Validation Score for Multiple Linear Regression Model

Apply Cross Validation on Decision Tree Regression Model, Random Forest Regression Model and Gradient Boosting Regression Model

```
# Perform 10-fold cross-validation on each model and print the mean score
dt_scores = cross_val_score(dt, X, y, cv=10)
print("Decision Tree Regressor average score: ", dt_scores.mean())

rf_scores = cross_val_score(rf, X, y, cv=10)
print("Random Forest Regressor average score: ", rf_scores.mean())

gb_scores = cross_val_score(gb, X, y, cv=10)
print("Gradient Boosting Regressor average score: ", gb_scores.mean())
```

```
Decision Tree Regressor average score:  0.7500604047953237
Random Forest Regressor average score:  0.8896031826575943
Gradient Boosting Regressor average score:  0.8811958971482632
```

Figure 15 Cross Validation Score for Decision Tree, Random Forest and Gradient Boosting Regression Models

Overall, I believe that the findings from this analysis provides a solid foundation for predicting house prices based on the independent variable's latitude, longitude, number of bedrooms, number of bathrooms, and size in square meters. I have thoroughly reviewed the work accomplished so far, and I am satisfied that all the stages were properly executed. No major oversights or mistakes were found, and I am confident in the accuracy of the results.

c. Decide the next steps

The Random Forest Regression model appears to have performed well based on its metrics. However, I would recommend to conduct further testing and evaluation to ensure that the model's performance is consistent and reliable before deployment to production.

Based on the metrics **on average**, the model's predictions are off by around €75,127.80. Therefore, I would like to minimise this gap before deploying it to production which may involve looking at more advanced techniques such as hyperparameter tuning on the Random Forest Regression model which would require further testing and also using new data over a 6-month period.

I believe this work provides a solid foundation for future research and it can be built on using the suggestions mentioned above and/or including researching the potential of other models etc. There is always room for improvement and continued exploration in this area.

Q5. Deploying the data mining model

The following are the steps outlined to deploy the Random Forest Regression Model to production for end users.

1. Save the training model to a pickle file using the pickle module in Python as per Figure [16].

Save Model to Disk

```
pickle.dump(rf, open('model.pkl','wb'))
```

```
model=pickle.load(open('model.pkl','rb'))
```

Figure 16 Save Model to a Pickle file

2. Create index.html which builds the user interface for end users to enter in values to predict house prices.
3. Create app.py using Flask web application framework.
4. Upload model.pkl, app.py and index.html to hosting platform https://www.pythonanywhere.com to deploy the Flask application (app.py) to a web server, where it can be accessed by users.

5. Enter in test values (exact values preferred) to the user interface as follows. This is from the X_test dataset.

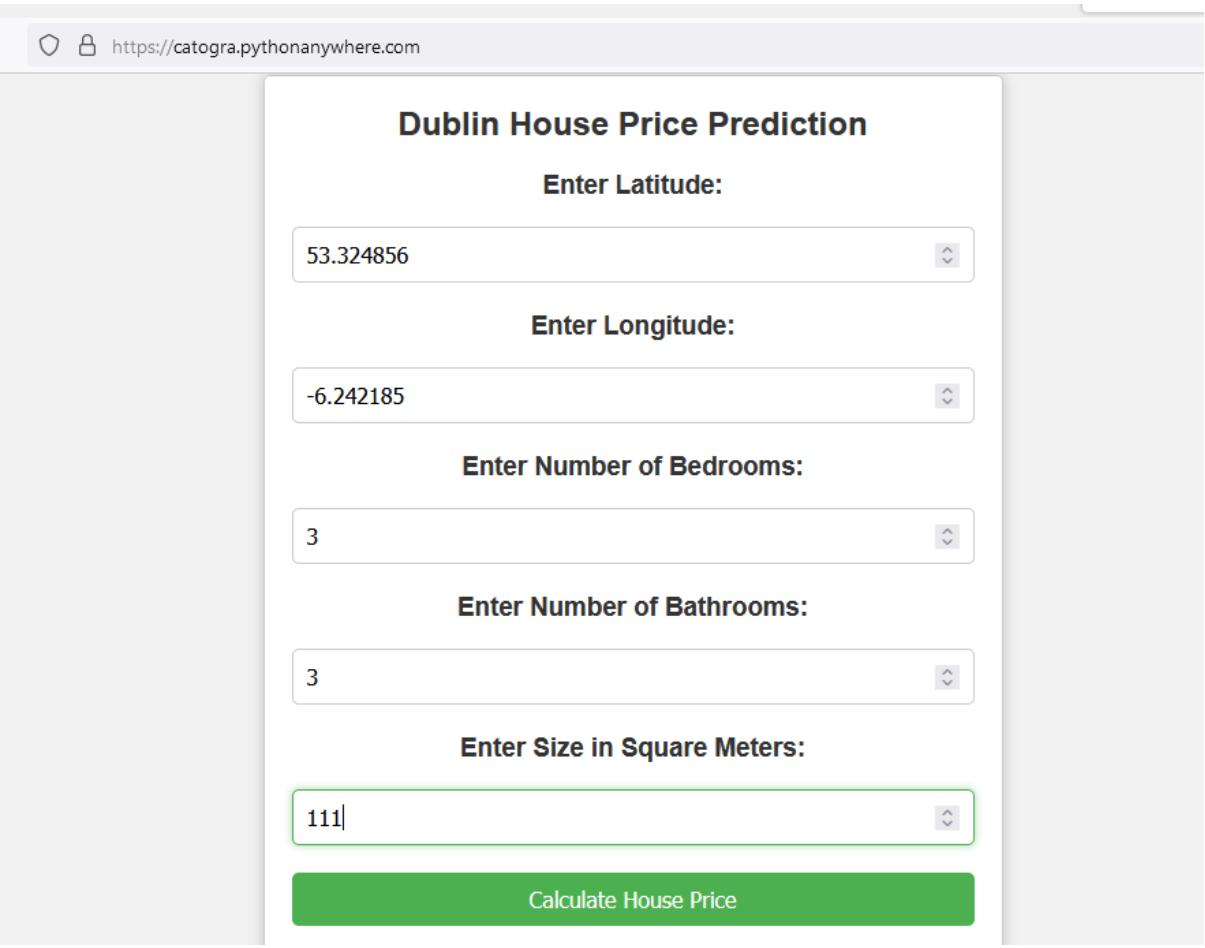

Figure 17 Dublin House Price Prediction User Interface

6. Click on Calculate House Price Button and the following result is printed on screen.

Figure 18 Predicted House Price

The actual house price on daft.ie is as follows so the prediction is quite accurate for this house.

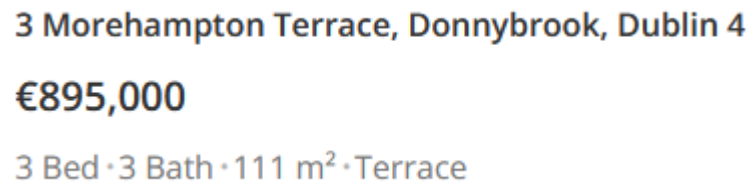

Figure 19 Actual House Price on daft.ie

Bibliography

Anaconda, 2023. *pythonanywhere by Anaconda.* [Online]
Available at: https://www.pythonanywhere.com
[Accessed 29 04 2023].

Bloomer, A., 2023. *Anthony Bloomer Daft Listings.* [Online]
Available at: https://github.com/AnthonyBloomer/daftlistings
[Accessed 27 February 2023].

limited, d. m., 2023. *www.daft.ie.* [Online]
Available at: https://www.daft.ie/
[Accessed 29 04 2023].

LYU, G., 2021. *Dublin Rental: EDA.* [Online]
Available at: https://www.kaggle.com/code/d17129765/dublin-rental-eda#3.Maps
[Accessed 27 February 2023].

OGrady, B., 2023. *Dublin House Price Prediction.* [Online]
Available at: https://catogra.pythonanywhere.com/
[Accessed 29 04 2023].