統計學習與深度學習房價預測專案

B11705051 資管三 陳奕廷

專案背景

本專案以 Kaggle 提供的資料集為基礎，進行房價預測，並採用 RMSE（Root Mean Squared Error）作為模型評估指標。選擇 XGBoost 作為主要模型，是基於其高效性及優越性能，特別是在面對大規模資料時，相較於 Random Forest 的訓練速度更快。以下為專案的執行流程與優化策略。

機器學習流程

資料探索與分析

- - 讀取 Kaggle 資料集，對資料進行基本清理與初步觀察。
- - 分析各特徵的分佈與關聯性，探索影響房價的可能因子。

初步模型測試

- - 套用簡單的線性回歸模型，快速了解資料的複雜度與特徵表現。
- - 使用基礎的 XGBoost 模型（無需調整參數），進行初步訓練與預測。

新增特徵工程

- - 基於對資料的理解，新增重要特徵，例如：屋齡、特徵的 One-Hot Encoding。
- - 通過對特徵的轉換與交互（例如平方或組合），提升模型表現。

模型訓練與初步結果

- - 訓練 XGBoost 模型，預測結果 RMSE 約為 40,000。

後續優化與迭代流程

特徵重要性分析

- - 觀察模型的特徵重要性排序，找出對預測結果影響較大的特徵。

問題發現與特徵創造

- - 根據特徵表現，設計新的特徵，例如：
- - 高重要性特徵之間的交互特徵。
- - 特徵的平方或對數轉換。
- - 提升模型對資料的擬合能力。

超參數調整

- - 使用 RandomizedSearchCV，對超參數進行優化，例如：
- - 樹的深度（max_depth）。
- - 學習率（learning_rate）。

- - 子樣本比例（subsample）。
- - 正則化參數等。

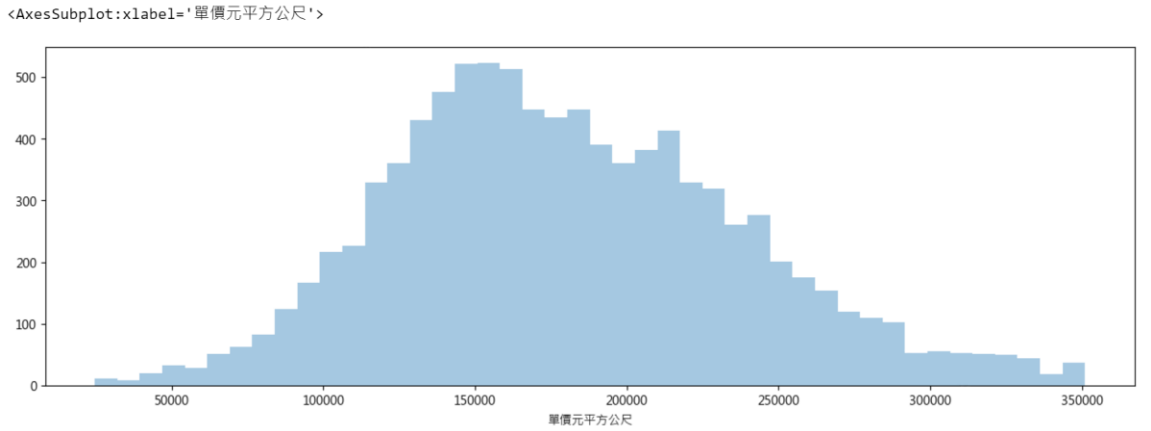
模型評估與提交

- - 訓練優化後的 XGBoost 模型，並進行最終預測。
- - 觀察 RMSE 的變化，並上傳結果至 Kaggle 平台。

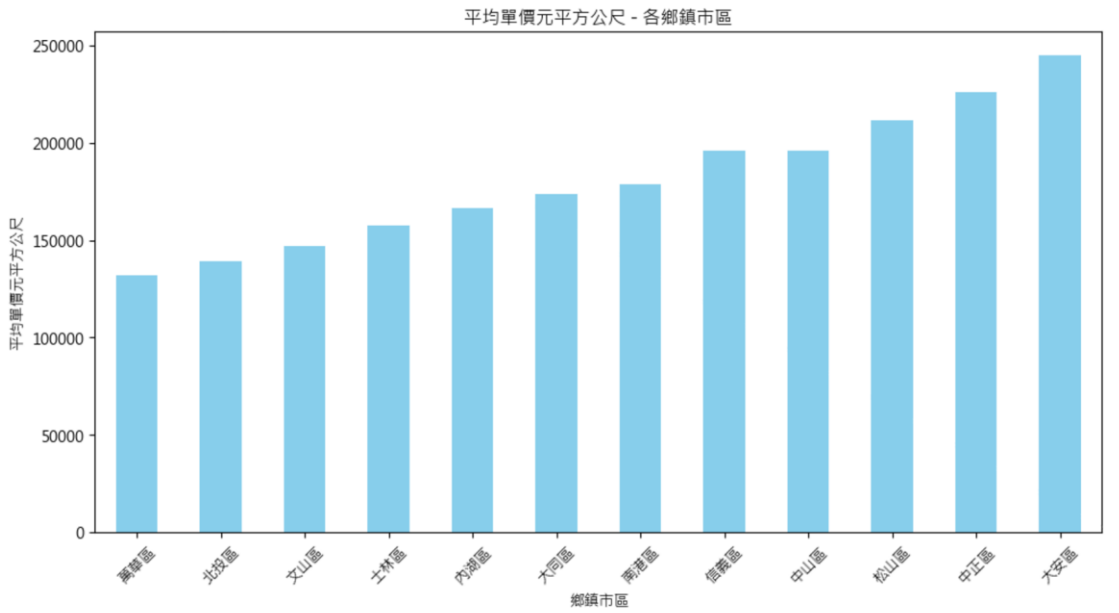
總結

本專案以 XGBoost 模型為核心，通過特徵工程與超參數調整，循環優化模型表現。未來可根據特徵的重要性進一步設計創新參數，提升模型的準確，進一步提高 RMSE 表現。

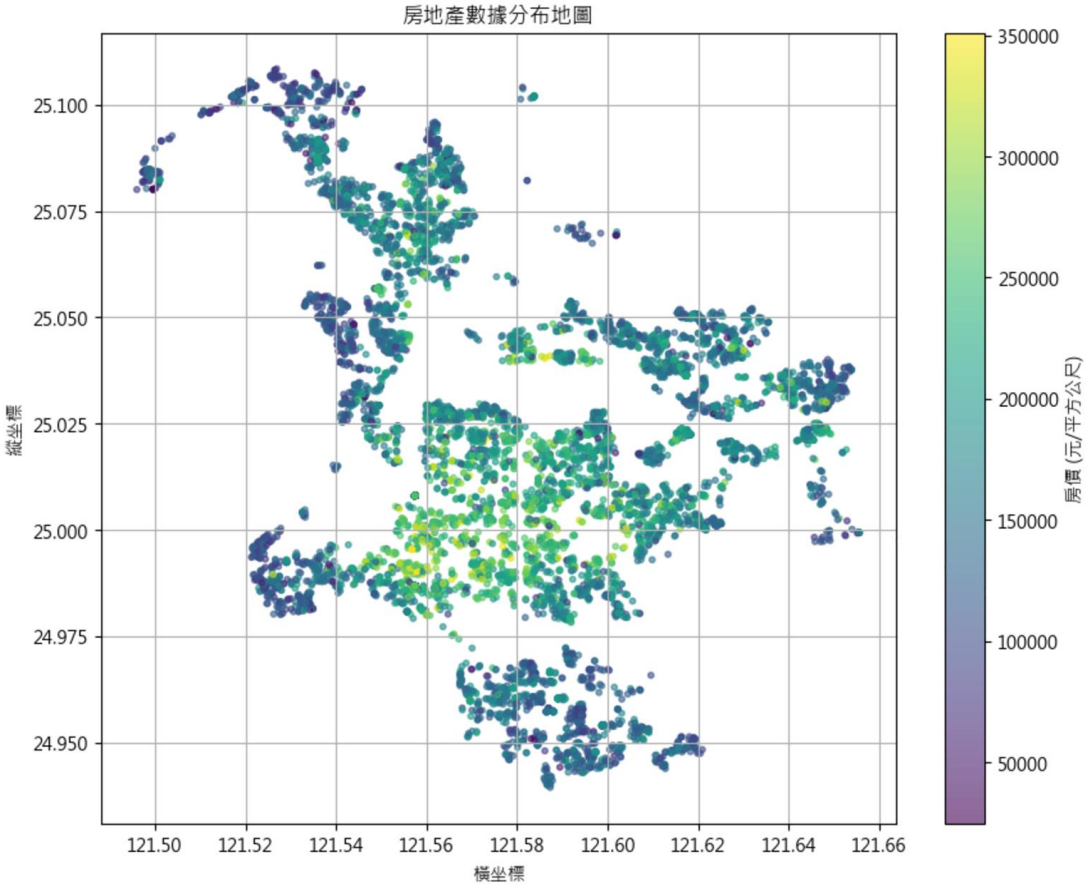
資料分析



上圖可以看到，單價元平方公尺主要呈現 normal distribution 並沒有資料偏移的問題。



上圖，可以看到台北市的房價的確與你所在的地區有所差異，大安區房價在現實中概念也是大於萬華區的。因此想知道路段是否也呈現這種樣態。



因此我將台北市房價依照座標以及顏色，可以發現顏色的確有集中的現象，也就是相鄰房價並不會差到太多。

	距離_南港區	距離_士林區	距離_大同區	
0	0.066559	0.109100	0.079421	0.068570
1	0.071102	0.064854	0.030626	0.027300
2	0.062643	0.116556	0.089334	0.076504
3	0.070732	0.053943	0.020193	0.017945
4	0.074520	0.075164	0.040348	0.037373

使用 anchor 的方式來做特徵工程，也就是拿某區的所有點平均，當作錨點，所有的資料點都必須計算他與所有區域的距離。

其他特徵包括：屋齡、區域環境評估(附近的機能 超商等)、鄰里便捷性指數(地鐵站等)，這些對模型有歸納的作用的特徵。

以上特徵也都有做圖，並都發現有呈現相關性，屋齡越高價格越低等等。

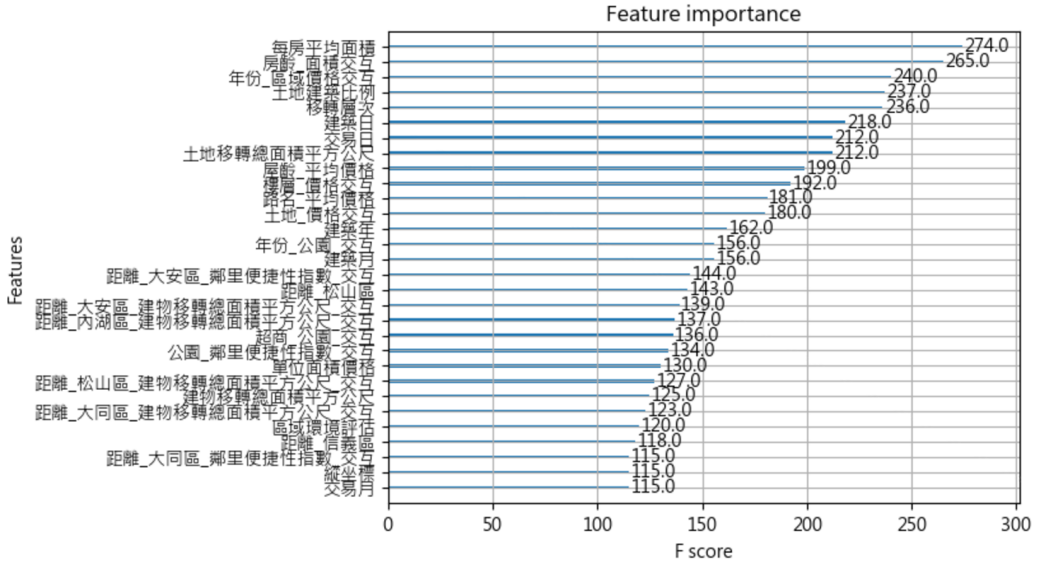
處理 One Hot 極多的情形

從資料中可以發現，光是路名就有上百種，因此使用 One Hot Encoded 會發生的問題也顯而易見地會很多，模型複雜度以及特徵代表性等等。為了不讓路名失去他的代表性，我必須採用其他的手段。

我使用 Target Encoded 這項作法來解決上述的問題，而大致上的作法是拿某條路上面所有的資料點，平均其房價的值 (y_train)成為一個字典的形式，把值賦予在資料上，例如：內湖路一段的平均價格是 a，某資料如果是內湖路一段的，我就賦予在「路名_平均價格」這個特徵上 a 這個值。在測試資料上也是拿 preprocess 的字典分配給他們，假使沒有遇到那條路，也可以給他總平均。

此作法我也有應用在特徵內容非常多元的特徵上，避免 One Hot 過多的問題。

在後續優化迭代



當模型訓練完後，我會拿到上圖這種特徵重要圖，我會針對上述類似的特徵做交互，產生新的特徵，例如：第三個特徵 年份_區域價格交互，告訴模型年份跟區域價格並不是無關，甚至不是線性關係，經過反覆的過程，上圖的重要性也會一直變動，而我也就可以持續發掘新特徵，並依照以上模型的表現來做特徵的篩選，避免 Overfitting。

```python
# 每房平均面積
x_test_df_concat["每房平均面積"] = x_test_df_concat["建物移轉總面積平方公尺"] / (x_test_df_concat["建物現況格局-房"] + 1)

# 土地建築比例
x_test_df_concat["土地建築比例"] = x_test_df_concat["土地移轉總面積平方公尺"] / (x_test_df_concat["建物移轉總面積平方公尺"] + 1)

# 樓層價格交互
x_test_df_concat["樓層_價格交互"] = x_test_df_concat["總樓層數"] * x_test_df_concat["屋齡_平均價格"]

# 單位面積價格
x_test_df_concat["單位面積價格"] = x_test_df_concat["屋齡_平均價格"] / (x_test_df_concat["建物移轉總面積平方公尺"] + 1)

# 土地與房價交互
x_test_df_concat["土地_價格交互"] = x_test_df_concat["土地移轉總面積平方公尺"] * x_test_df_concat["屋齡_平均價格"]

# 房齡與面積交互
x_test_df_concat["房齡_面積交互"] = (x_test_df_concat["屋齡"]) * x_test_df_concat["建物移轉總面積平方公尺"]

# 年份和地區價格交互
x_test_df_concat["年份_區域價格交互"] = x_test_df_concat["建築年"] * x_test_df_concat["路名_平均價格"]
```

大致上用上述方法產生新特徵，而我也發現若沒有使用 domain knowledge 來做交互，會產生出非常多 Overfitted 的特徵。