Statistical Learning and Deep Learning, Fall 2024 Mini Project

B11705009 An-Che, Liang

Part 1: Prepare data and feature

In [1]:
```
import pandas as pd
import numpy as np
# 載入訓練資料與標籤
X_train = pd.read_csv('./data/X_train.csv')
y_train = pd.read_csv('./data/y_train.csv')
```

In [2]:
```
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

features = [
    "鄉鎮市區",
    "交易標的",
    "路名",
    "土地移轉總面積平方公尺",
    "都市土地使用分區",
    "土地數",
    "建物數",
    "車位數",
    "移轉層次",
    "移轉層次項目",
    "總樓層數",
    "建物型態",
    "主要用途",
    "主要建材",
    "建築完成年月",
    "建物移轉總面積平方公尺",
    "建物現況格局-房",
    "建物現況格局-廳",
    "建物現況格局-衛",
    "建物現況格局-隔間",
    "有無管理組織",
    "交易年",
    "交易日",
    "交易月",
    "地鐵站",
    "超商",
    "公園",
    "托兒所",
    "國小",
    "國中",
```

```
    "高中職",
    "大學",
    "金融機構",
    "醫院",
    "大賣場",
    "超市",
    "百貨公司",
    "警察局",
    "消防局",
    "縱坐標",
    "橫坐標",
]
target = "單價元平方公尺"

numeric_features = [
    "土地移轉總面積平方公尺",
    "土地數",
    "建物數",
    "車位數",
    "移轉層次",
    "總樓層數",
    "建物移轉總面積平方公尺",
    "建物現況格局-房",
    "建物現況格局-廳",
    "建物現況格局-衛",
    "交易年",
    "交易日",
    "交易月",
    "地鐵站",
    "超商",
    "公園",
    "托兒所",
    "國小",
    "國中",
    "高中職",
    "大學",
    "金融機構",
    "醫院",
    "大賣場",
    "超市",
    "百貨公司",
    "警察局",
    "消防局",
    "縱坐標",
    "橫坐標",
]
categorical_features = [i for i in features if i not in numeric_features]
```

Part2: Add GBF feature and transaction time feature

In [3]:
```
anchor_x = X_train["橫坐標"].mean()
anchor_y = X_train["縱坐標"].mean()
std_x = X_train["橫坐標"].std()
std_y = X_train["縱坐標"].std()
```

In [4]:
```
def gaussian_basis_function(x_a, y_a, mu_xj, mu_yj, s_xj, s_yj):
    return np.exp(
        -((x_a - mu_xj) ** 2 / (2 * s_xj**2)) - ((y_a - mu_yj) ** 2 / (2 * s
    )


X_train["GBF_feature"] = gaussian_basis_function(
    X_train["橫坐標"], X_train["縱坐標"], anchor_x, anchor_y, std_x, std_y
)

import datetime

def generate_time(df):
    cols = ["交易年", "交易月", "交易日"]
    df["time"] = df[cols].apply(
        lambda row: (
            datetime.datetime(year=int(row[0]), month=int(row[1]), day=int(r
            - datetime.datetime(2012, 1, 1)
        ).total_seconds()
        // 86400,
        axis=1,
    )

    return df

X_train = generate_time(X_train)
```

```
/var/folders/p9/8zr36sx53v967c615r_8kkzh0000gn/T/ipykernel_40489/2965888669.
py:17: FutureWarning: Series.__getitem__ treating keys as positions is depre
cated. In a future version, integer keys will always be treated as labels (c
onsistent with DataFrame behavior). To access a value by position, use `ser.
iloc[pos]`
  datetime.datetime(year=int(row[0]), month=int(row[1]), day=int(row[2]))
```

In [5]:
```
def generate_anchors(df, K):
    anchors = {}

    for city in df["鄉鎮市區"].unique():
        city_df = df[df["鄉鎮市區"] == city]

        x_min, x_max = city_df["橫坐標"].min(), city_df["橫坐標"].max()
        y_min, y_max = city_df["縱坐標"].min(), city_df["縱坐標"].max()

        x_grid = np.linspace(x_min, x_max, K + 1)
        y_grid = np.linspace(y_min, y_max, K + 1)

        city_anchors = []

        for i in range(K):
            for j in range(K):
                grid_df = city_df[
                    (city_df["橫坐標"] >= x_grid[i])
                    & (city_df["橫坐標"] < x_grid[i + 1])
                    & (city_df["縱坐標"] >= y_grid[j])
                    & (city_df["縱坐標"] < y_grid[j + 1])
```

```
                ]

                if len(grid_df) > 100:
                    anchor_x = float(grid_df["橫坐標"].mean())
                    anchor_y = float(grid_df["縱坐標"].mean())
                    std_x = float(grid_df["橫坐標"].std())
                    std_y = float(grid_df["縱坐標"].std())
                    city_anchors.append((anchor_x, anchor_y, std_x, std_y))

        anchors[city] = city_anchors

    return anchors


K = 2

anchors = generate_anchors(X_train, K)
for city, city_anchors in anchors.items():
    print(f"{city}: {len(city_anchors)} anchors")
    for i, anchor in enumerate(city_anchors):
        print(f"Anchor {i+1}: ({anchor[0]:.2f}, {anchor[1]:.2f}) ")
```

```
文山區: 2 anchors
Anchor 1: (121.58, 24.96)
Anchor 2: (121.60, 24.95)
中正區: 2 anchors
Anchor 1: (121.56, 24.98)
Anchor 2: (121.56, 25.00)
內湖區: 3 anchors
Anchor 1: (121.56, 25.00)
Anchor 2: (121.60, 25.05)
Anchor 3: (121.63, 25.04)
北投區: 3 anchors
Anchor 1: (121.51, 25.10)
Anchor 2: (121.56, 25.00)
Anchor 3: (121.54, 25.09)
松山區: 2 anchors
Anchor 1: (121.57, 25.02)
Anchor 2: (121.59, 25.02)
信義區: 2 anchors
Anchor 1: (121.60, 24.99)
Anchor 2: (121.61, 25.00)
大安區: 4 anchors
Anchor 1: (121.56, 24.99)
Anchor 2: (121.57, 25.00)
Anchor 3: (121.58, 24.98)
Anchor 4: (121.58, 25.00)
萬華區: 1 anchors
Anchor 1: (121.53, 24.99)
南港區: 2 anchors
Anchor 1: (121.63, 25.00)
Anchor 2: (121.64, 25.02)
士林區: 2 anchors
Anchor 1: (121.55, 25.02)
Anchor 2: (121.56, 25.07)
大同區: 1 anchors
Anchor 1: (121.54, 25.03)
中山區: 3 anchors
Anchor 1: (121.56, 25.01)
Anchor 2: (121.57, 25.03)
Anchor 3: (121.59, 25.04)
```

In [6]:

```python
def calculate_gbf_features(df, anchors):
    for city, city_anchors in anchors.items():
        city_mask = df['鄉鎮市區'] == city

        for i, (anchor_x, anchor_y, std_x, std_y) in enumerate(city_anchors)
            gbf_feature = gaussian_basis_function(df.loc[city_mask, '橫坐標']
                                                  df.loc[city_mask, '縱坐標']
                                                  anchor_x, anchor_y, std_x,
            df.loc[city_mask, f'GBF_{city}_{i}'] = gbf_feature
    return df
```

In [7]:

```python
X_train = calculate_gbf_features(X_train, anchors)
X_train.fillna(0, inplace=True)
# numeric_features = numeric_features + [
#    col for col in X_train.columns if "GBF" in col and col != "GBF_feature"
# ]
```

```
X_train.describe()
```

Out[7]:

	Id					
count	9460.000000	9460.000000	9460.000000	9460.000000	9460.00000	9460.000(
mean	4729.500000	30.135313	1.380867	1.030655	0.45370	4.740
std	2731.011107	21.965492	1.021325	0.236957	0.71033	3.382
min	0.000000	0.090000	1.000000	1.000000	0.00000	1.000(
25%	2364.750000	18.960000	1.000000	1.000000	0.00000	2.000(
50%	4729.500000	27.095000	1.000000	1.000000	0.00000	4.000(
75%	7094.250000	35.890000	1.000000	1.000000	1.00000	6.000(
max	9459.000000	813.470000	33.000000	10.000000	14.00000	26.000(

8 rows × 60 columns

Part3: Train random forest model, then choose parameter with GridSearchCV

In [8]:

```
# set up the preprocessing steps for each type of feature

numeric_transformer = Pipeline(steps=[("scaler", StandardScaler())])

categorical_transformer = Pipeline(
    steps=[("onehot", OneHotEncoder(handle_unknown="ignore"))]
)

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)

from sklearn.ensemble import RandomForestRegressor

# set up the model
model = RandomForestRegressor()

# set up the pipeline

pipeline = Pipeline(steps=[("preprocessor", preprocessor), ("model", model)]

from sklearn.model_selection import GridSearchCV

param_grid = {
    "model__n_estimators": [100, 200],
```

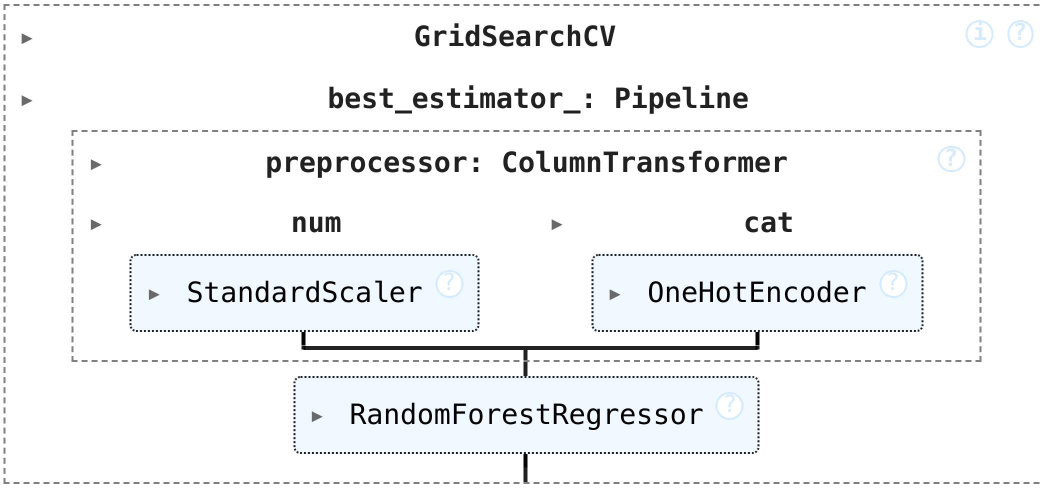
```
    "model__max_depth": [5, 10],
    "model__min_samples_split": [2, 5],
    "model__min_samples_leaf": [1, 2],
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, verbose=2)


y_train = y_train[target]
X_train = X_train[numeric_features + categorical_features]

# train the model
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.2s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.6s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.6s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.6s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=1, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.6s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
```

```
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=2, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=100; total time=   2.5s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.1s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=5, model__min_samples_leaf=2, model__min_samples_s
plit=5, model__n_estimators=200; total time=   5.0s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=100; total time=   7.6s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=100; total time=   7.5s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=100; total time=   7.5s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=100; total time=   7.5s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=100; total time=   7.5s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=200; total time=  15.3s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=200; total time=  15.0s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=200; total time=  15.3s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=200; total time=  15.4s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=2, model__n_estimators=200; total time=  15.3s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=100; total time=   6.9s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=100; total time=   6.8s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=100; total time=   6.8s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
```

```
split=5, model__n_estimators=200; total time=  13.6s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=200; total time=  13.6s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=200; total time=  13.7s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=200; total time=  13.7s
[CV] END model__max_depth=10, model__min_samples_leaf=1, model__min_samples_
split=5, model__n_estimators=200; total time=  13.7s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=100; total time=   7.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=100; total time=   7.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=100; total time=   7.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=100; total time=   7.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=100; total time=   7.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=200; total time=  14.2s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=200; total time=  14.5s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=200; total time=  14.2s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=200; total time=  14.1s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=2, model__n_estimators=200; total time=  14.4s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=100; total time=   7.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=200; total time=  14.0s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=200; total time=  13.9s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=200; total time=  13.9s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=200; total time=  13.9s
[CV] END model__max_depth=10, model__min_samples_leaf=2, model__min_samples_
split=5, model__n_estimators=200; total time=  13.9s
```

Out[8]:



In [9]:

```
best_model = grid_search.best_estimator_
```

Part4: Save the model, then generate prediction

In [10]:

```
# save the model
import joblib

joblib.dump(model, "model.pkl")

# run inference
import uuid

X_test = pd.read_csv("./data/X_test.csv")

X_test = calculate_gbf_features(X_test, anchors)
X_test.fillna(0, inplace=True)

X_test = generate_time(X_test)

y_pred = best_model.predict(X_test)

# save the prediction with ID
y_pred_df = pd.DataFrame(y_pred, columns=[target])
y_pred_df.index.name = "ID"
y_pred_df.to_csv(f"y_pred-super_rf-{uuid.uuid4()}.csv")
```

```
/var/folders/p9/8zr36sx53v967c615r_8kkzh0000gn/T/ipykernel_40489/2965888669.
py:17: FutureWarning: Series.__getitem__ treating keys as positions is depre
cated. In a future version, integer keys will always be treated as labels (c
onsistent with DataFrame behavior). To access a value by position, use `ser.
iloc[pos]`
  datetime.datetime(year=int(row[0]), month=int(row[1]), day=int(row[2]))
```

Evaluate the model with training data

In [11]:

```
# evaluate the model
from sklearn.metrics import mean_squared_error
```

```
y_train_pred = best_model.predict(X_train)
train_rmse = mean_squared_error(y_train, y_train_pred, squared=False)

print(f"Train RMSE: {train_rmse}")
```

```
Train RMSE: 30106.34194647822
```

```
/Users/namwoam/Library/Caches/pypoetry/virtualenvs/sldl-pwhQZ0rV-py3.12/lib/
python3.12/site-packages/sklearn/metrics/_regression.py:492: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calcul
ate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```