assignment-code

March 21, 2024

[1]:
```python
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import math

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from plotly.subplots import make_subplots
from plotnine import *

prng = np.random.RandomState(20240321)
```

[2]:
```python
real_estate_data = pd.read_csv("https://raw.githubusercontent.com/divenyijanos/
 ↪ceu-ml/2023/data/real_estate/real_estate.csv")
real_estate_sample = real_estate_data.sample(frac=0.2, random_state=prng)
```

[3]:
```python
print(real_estate_data.shape,real_estate_sample.shape)
```

```
(414, 8) (83, 8)
```

[4]:
```python
real_estate_sample.head()
```

[4]:
```
      id  transaction_date  house_age  distance_to_the_nearest_MRT_station  \
397  398          2013.417       13.1                           1164.83800
96    97          2013.417        6.4                             90.45606
77    78          2012.833       20.5                           2185.12800
86    87          2012.833        1.8                           1455.79800
241  242          2013.500       13.7                            250.63100

     number_of_convenience_stores  latitude  longitude  \
397                             4  24.99156  121.53406
96                              9  24.97433  121.54310
77                              3  24.96322  121.51237
86                              1  24.95120  121.54900
```

```
241                              7  24.96606  121.54297

     house_price_of_unit_area
397                      32.2
96                       59.5
77                       25.6
86                       27.0
241                      41.4
```

0.0.1 1) Think about an appropriate loss function you can use to evaluate your predictive models. What is the risk (from a business perspective) that you would have to take by making a wrong prediction?

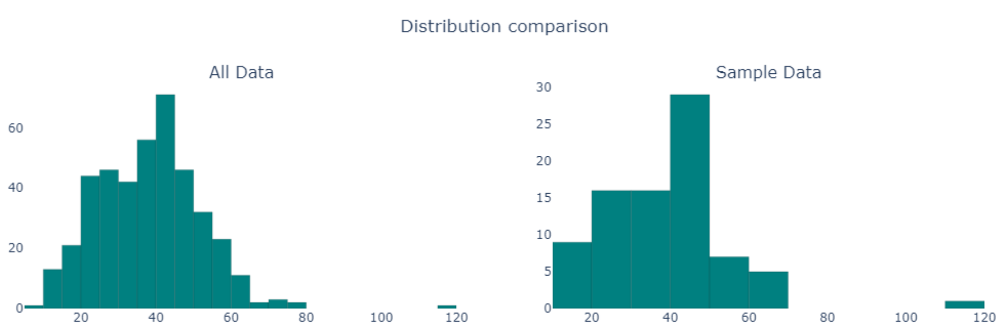
[5]:
```python
fig = make_subplots(rows=1, cols=2, subplot_titles=('All Data', 'Sample Data'))

fig.add_trace(go.Histogram(x=real_estate_data['house_price_of_unit_area'],
                           marker=dict(color='teal',
                           line=dict(color='black',
                           width=0.15))),
                           row=1, col=1)


fig.add_trace(go.Histogram(x=real_estate_sample['house_price_of_unit_area'],
                           marker=dict(color='teal',
                           line=dict(color='black',
                           width=0.15))),
                           row=1, col=2)


# Update layout
fig.update_layout(title_text='Distribution comparison', title_x=0.5,␣
 ↪plot_bgcolor='white', showlegend=False,
                  width = 1100, height=400)

fig.show()
```



If you look at the distributions graph above, we can see that there is only one extreme value in both **All Data** while the **Sample Data**. I will be dropping this value from my training sets to get an accurate prediction model. But otherwise from the distributions, the dataset is pretty normal

Root Mean Squared Error (RMSE) here would be the best choice for a loss function, RMSE measures the square root of the average squared difference between the predicted values and the actual values in the dataset. Using RMSE as the loss function is appropriate because it penalizes large errors more heavily than smaller errors, which aligns with the goal of minimizing prediction errors here.

The risk associated with making wrong predictions in this include

1. Financial Reason: Wrong predictions can lead to financial losses for both buyers and sellers. If the predicted price is higher than the actual price, sellers may struggle to sell their properties, resulting in lost opportunities. Conversely, if the predicted price is lower than the actual price, buyers may overpay for properties, leading to dissatisfaction.

2. Brand Reputation: Inaccurate predictions could damage the reputation of the web app and our business. Users may lose trust in the platform if they consistently receive inaccurate price estimates.

3. Competition: If our competitionoffer more accurate predictions, the web app may lose its competitive edge and struggle to attract users.

[6]:
```python
# Removing the extreme value
print(f'Before removing extreme value, Overall Data:{real_estate_data.
 ↪shape},Sample Data:{real_estate_sample.shape}')

real_estate_data = real_estate_data.
 ↪loc[real_estate_data['house_price_of_unit_area'] <100]
real_estate_sample = real_estate_sample.
 ↪loc[real_estate_sample['house_price_of_unit_area'] <100]

print(f'After removing extreme value, Overall Data:{real_estate_data.
 ↪shape},Sample Data:{real_estate_sample.shape}')
```

```
Before removing extreme value, Overall Data:(414, 8),Sample Data:(83, 8)
After removing extreme value, Overall Data:(413, 8),Sample Data:(82, 8)
```

0.0.2 2) Build a simple benchmark model and evaluate its performance on the hold-out set (using your chosen loss function).

[7]:
```python
target = real_estate_sample["house_price_of_unit_area"]
features = real_estate_sample.drop(columns = ["house_price_of_unit_area","id"])
X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪target,test_size=0.3, random_state=prng)
```

[8]:
```python
# Making our loss function
def calculateRMSE(prediction, y_obs):
    return np.sqrt(np.mean((prediction - y_obs)**2))
```

[9]:
```python
# estimate benchmark model
benchmark  = np.mean(y_train)
benchmark_result = ["Benchmark", calculateRMSE(benchmark, y_train),␣
 ↪calculateRMSE(benchmark, y_test)]
```

[10]:
```python
# collect results into a DataFrame
result_columns = ["Model", "Train", "Test"]
pd.DataFrame([benchmark_result], columns=result_columns)
```

[10]:

	Model	Train	Test
0	Benchmark	12.545831	13.329949

0.0.3 3) Build a simple linear regression model using a chosen feature and evaluate its performance. Would you launch your evaluator web app using this model?

[11]:
```python
target = real_estate_sample["house_price_of_unit_area"]
features = real_estate_sample[["house_age"]]

X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪target,test_size=0.3, random_state=prng)


# Create and fit the model
simple_ols_reg = LinearRegression().fit(X_train, y_train)

train_error = calculateRMSE(simple_ols_reg.predict(X_train), y_train)
test_error = calculateRMSE(simple_ols_reg.predict(X_test), y_test)

simple_ols_result = ["Simple OLS", train_error, test_error]
```

[12]:
```python
pd.DataFrame([benchmark_result,simple_ols_result], columns=result_columns)
```

[12]:

	Model	Train	Test
0	Benchmark	12.545831	13.329949
1	Simple OLS	11.616364	14.963487

No, I would not use this model to launch my app. The training rmse only improves a bit and test rmse is worse.

0.0.4 4) Build a multivariate linear model with all the meaningful variables available. Did it improve the predictive power?

[13]:
```
target = real_estate_sample[["house_price_of_unit_area"]]
features =␣
 ↪real_estate_sample[["house_age","distance_to_the_nearest_MRT_station","number_of_convenienc

X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪target,test_size=0.3, random_state=prng)

# Create and fit the model
multi_var_ols_reg = LinearRegression().fit(X_train, y_train)

train_error = calculateRMSE(multi_var_ols_reg.predict(X_train), y_train)
test_error = calculateRMSE(multi_var_ols_reg.predict(X_test), y_test)

multi_var_ols_reg = ["Multi Variate", train_error, test_error]
```

[14]:
```
pd.DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg],␣
 ↪columns=result_columns)
```

[14]:

	Model	Train	Test
0	Benchmark	12.545831	13.329949
1	Simple OLS	11.616364	14.963487
2	Multi Variate	8.975257	8.368991

It did increase the predictive power, the rmse decreased in both training and test

0.0.5 5) Try to make your model (even) better. Document your process and its success while taking two approaches:

1. Feature engineering - e.g. including squares and interactions or making sense of latitude&longitude by calculating the distance from the city center, etc. For calculating distance from city centre, I will be using the presidential palace in Tapei as the city center which has the coordinates: 25.041155828937818, 121.51189705508669

[15]:
```
# Calculating distance from city centre

def km_from_center(lat1, lon1, lat2, lon2):

    lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/
 ↪2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    r = 6371
    distance = r * c
```

```
    return distance
```

[16]:
```
city_center_lat = 25.041155828937818
city_center_lon = 121.51189705508669

# Calculate distance for each row and add it as a new column
real_estate_sample['distance_from_city_center'] = real_estate_sample.
 ↪apply(lambda row: km_from_center(city_center_lat, city_center_lon,␣
 ↪row['latitude'], row['longitude']), axis=1)
```

[17]:
```
real_estate_sample[['distance_from_city_center']].describe().T
```

[17]:
```
                           count      mean       std       min       25%  \
distance_from_city_center   82.0  8.447657  1.474591  4.487499  7.463757

                                50%       75%        max
distance_from_city_center  8.176487  9.373676  12.034171
```

2. Training more flexible models Multi Variate Feature Engineered Model

[18]:
```
target = real_estate_sample[["house_price_of_unit_area"]]
features = real_estate_sample[["transaction_date","house_age",␣
 ↪"distance_to_the_nearest_MRT_station",
                                     ␣
 ↪"number_of_convenience_stores","distance_from_city_center"]]


# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.3, random_state=prng)

# Create and fit the model
multi_var_fe_reg = LinearRegression().fit(X_train, y_train)

# Calculating training and testing errors
train_error = calculateRMSE(multi_var_fe_reg.predict(X_train), y_train)
test_error = calculateRMSE(multi_var_fe_reg.predict(X_test), y_test)

# Storing results
multi_var_fe_reg_results = ["Multi Variate FE", train_error, test_error]
```

[19]:
```
pd.
 ↪DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg,multi_var_fe_reg_results],␣
 ↪columns=result_columns)
```

```
[19]:               Model      Train       Test
      0         Benchmark  12.545831  13.329949
      1        Simple OLS  11.616364  14.963487
      2     Multi Variate   8.975257   8.368991
      3  Multi Variate FE   7.930315   8.026051
```

Random Forest Model

```
[20]: target = real_estate_sample["house_price_of_unit_area"]
      features = real_estate_sample[["transaction_date","house_age",␣
       ↪"distance_to_the_nearest_MRT_station",
                                     ␣
       ↪"number_of_convenience_stores","distance_from_city_center"]]

      # Splitting data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(features, target,␣
       ↪test_size=0.3, random_state=prng)

      # Create and fit the model
      rf_reg = RandomForestRegressor(random_state=prng)
      rf_reg.fit(X_train, y_train)

      # Calculating training and testing errors
      train_error = calculateRMSE(rf_reg.predict(X_train), y_train)
      test_error = calculateRMSE(rf_reg.predict(X_test), y_test)

      # Storing results
      rf_reg_results = ["Random Forest", train_error, test_error]
```

```
[21]: pd.
       ↪DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg,multi_var_fe_reg_results,rf
       ↪columns=result_columns)
```

```
[21]:               Model      Train       Test
      0         Benchmark  12.545831  13.329949
      1        Simple OLS  11.616364  14.963487
      2     Multi Variate   8.975257   8.368991
      3  Multi Variate FE   7.930315   8.026051
      4     Random Forest   2.447217  10.151938
```

Gradient Boosting Model

```
[22]: target = real_estate_sample["house_price_of_unit_area"]
      features = real_estate_sample[["transaction_date","house_age",␣
       ↪"distance_to_the_nearest_MRT_station",
                                     ␣
       ↪"number_of_convenience_stores","distance_from_city_center"]]
```

```
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
↪test_size=0.3, random_state=prng)

# Create and fit the model
gb_reg = GradientBoostingRegressor(random_state=prng)
gb_reg.fit(X_train, y_train)

# Calculating training and testing errors
train_error = calculateRMSE(gb_reg.predict(X_train), y_train)
test_error = calculateRMSE(gb_reg.predict(X_test), y_test)

# Storing results
gb_reg_results = ["Gradient Boosting", train_error, test_error]
```

[23]:

```
pd.
↪DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg,multi_var_fe_reg_results,rf
↪columns=result_columns)
```

[23]:

	Model	Train	Test
0	Benchmark	12.545831	13.329949
1	Simple OLS	11.616364	14.963487
2	Multi Variate	8.975257	8.368991
3	Multi Variate FE	7.930315	8.026051
4	Random Forest	2.447217	10.151938
5	Gradient Boosting	0.504276	7.265992

6a) Rerun three of your previous models (including both flexible and less flexible ones) on the full train set. Ensure that your test result remains comparable by keeping that dataset intact. (Hint: extend the code snippet below.) Calculating distance from city centre for overall data

[24]:

```
city_center_lat = 25.041155828937818
city_center_lon = 121.51189705508669

# Calculate distance for each row and add it as a new column
real_estate_data['distance_from_city_center'] = real_estate_data.apply(lambda␣
↪row: km_from_center(city_center_lat,
                                                                              ␣
↪                     city_center_lon, row['latitude'],
                                                                              ␣
↪                     row['longitude']), axis=1)
```

Making our full training dataset

[25]:
```
real_estate_full = real_estate_data.loc[~real_estate_data.index.isin(X_test.
 ↪index)]

print(f"Size of the full training set: {real_estate_full.shape}")
```

```
Size of the full training set: (388, 9)
```

Simple OLS Model

[26]:
```
target = real_estate_full["house_price_of_unit_area"]
features = real_estate_full[["house_age"]]

X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪target,test_size=0.3, random_state=prng)


# Create and fit the model
simple_ols_reg = LinearRegression().fit(X_train, y_train)

train_error = calculateRMSE(simple_ols_reg.predict(X_train), y_train)
test_error = calculateRMSE(simple_ols_reg.predict(X_test), y_test)

simple_ols_result_full = ["Simple OLS", train_error, test_error]
```

Multi Variate Feature Engineered Model

[27]:
```
# Extracting target and features
target = real_estate_full[["house_price_of_unit_area"]]
features = real_estate_full[["transaction_date","house_age",␣
 ↪"distance_to_the_nearest_MRT_station",
                              ␣
 ↪"number_of_convenience_stores","distance_from_city_center"]]


# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.3, random_state=prng)

# Create and fit the model
multi_var_fe_reg = LinearRegression().fit(X_train, y_train)

# Calculating training and testing errors
train_error = calculateRMSE(multi_var_fe_reg.predict(X_train), y_train)
test_error = calculateRMSE(multi_var_fe_reg.predict(X_test), y_test)

# Storing results
multi_var_fe_reg_results_full = ["Multi Variate FE", train_error, test_error]
```

Random Forest Model

[28]:
```
target = real_estate_full["house_price_of_unit_area"]
features = real_estate_full[["transaction_date","house_age",
 ↪"distance_to_the_nearest_MRT_station",
                                ␣
 ↪"number_of_convenience_stores","distance_from_city_center"]]

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.3, random_state=prng)

# Create and fit the model
rf_reg = RandomForestRegressor(random_state=prng)
rf_reg.fit(X_train, y_train)

# Calculating training and testing errors
train_error = calculateRMSE(rf_reg.predict(X_train), y_train)
test_error = calculateRMSE(rf_reg.predict(X_test), y_test)

# Storing results
rf_reg_results_full = ["Random Forest", train_error, test_error]
```

Gradient Boosting Model

[29]:
```
target = real_estate_full["house_price_of_unit_area"]
features = real_estate_full[["transaction_date","house_age",
 ↪"distance_to_the_nearest_MRT_station",
                                ␣
 ↪"number_of_convenience_stores","distance_from_city_center"]]

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.3, random_state=prng)

# Create and fit the model
gb_reg = GradientBoostingRegressor(random_state=prng)
gb_reg.fit(X_train, y_train)

# Calculating training and testing errors
train_error = calculateRMSE(gb_reg.predict(X_train), y_train)
test_error = calculateRMSE(gb_reg.predict(X_test), y_test)

# Storing results
gb_reg_results_full = ["Gradient Boosting", train_error, test_error]
```

[30]:
```
pd.
↪DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg,multi_var_fe_reg_results,rf
                ␣
↪simple_ols_result_full,multi_var_fe_reg_results_full,rf_reg_results_full,gb_reg_results_ful
↪columns=result_columns)
```

[30]:

	Model	Train	Test
0	Benchmark	12.545831	13.329949
1	Simple OLS	11.616364	14.963487
2	Multi Variate	8.975257	8.368991
3	Multi Variate FE	7.930315	8.026051
4	Random Forest	2.447217	10.151938
5	Gradient Boosting	0.504276	7.265992
6	Simple OLS	13.107783	11.712509
7	Multi Variate FE	7.664576	9.054595
8	Random Forest	2.577721	6.332639
9	Gradient Boosting	3.103552	5.923144

6b) Did it improve the predictive power of your models? Where do you observe the biggest improvement? Would you launch your web app now?

[31]:
```
results_df = pd.
↪DataFrame([benchmark_result,simple_ols_result,multi_var_ols_reg,multi_var_fe_reg_results,rf
                ␣
↪simple_ols_result_full,multi_var_fe_reg_results_full,rf_reg_results_full,gb_reg_results_ful
↪columns=result_columns)

results_df.insert(1, 'Dataset', '')
results_df.loc[:5, 'Dataset'] = 'sample train set'
results_df.loc[6:, 'Dataset'] = 'full train set'

results_df.round(3)
```

[31]:

	Model	Dataset	Train	Test
0	Benchmark	sample train set	12.546	13.330
1	Simple OLS	sample train set	11.616	14.963
2	Multi Variate	sample train set	8.975	8.369
3	Multi Variate FE	sample train set	7.930	8.026
4	Random Forest	sample train set	2.447	10.152
5	Gradient Boosting	sample train set	0.504	7.266
6	Simple OLS	full train set	13.108	11.713
7	Multi Variate FE	full train set	7.665	9.055
8	Random Forest	full train set	2.578	6.333
9	Gradient Boosting	full train set	3.104	5.923

For our flexibe models, training on the full data set did improve our rmse, meaning the models got better. However the rmse is still huge to launch our app.

For unseen data, the base of ~6 rmse means that there is a prediction error of 6 in price per unit area. This may not seem big, but if you check how the variable is described: 10000 New Taiwan Dollar/Ping, where Ping is a local unit, 1 Ping = 3.3 meter squared

The 6 would mean, 60,000 New Taiwan Dollar / Ping, and multiplying that with the average house size in pings ~ 30, this comes out to be 1,800,000 New Taiwan Dollars. If our prediction undervalues or overvalues any property by this huge amount, no customer would use it to buy/sell.