HousePrices_Regression_2021/11/26

期中作業報告_

學號：110588014

學生：戴鴻庭

目錄

工作環境

過程介紹

因為我是使用google colab平台

來做這次做的作業

所以我需要先把我的雲端資料導入進去

然後告訴系統我的資料夾是哪個

```
[ ]  #導入google  drive
     from  google.colab  import  drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive

[ ]  DATA_HOUSE  =  '/content/gdrive/MyDrive/house'
```

數據讀取

參考網路教學與同學的分享

加上老師的上課講解

選擇這些API導入

比較重要的有

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense,Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
from keras.callbacks import *
```

pandas是為了後續讀取csv檔案

keras 是一個開發深度學習模型的工具

使用pandas讀取資料

讀取資料train-v3.csv

然後我想看一下讀到檔案的數值

跑出來東西蠻多的

```
a_datas = pd.read_csv(f'{DATA_HOUSE}/train-v3.csv') #用pandas讀取csv檔案
a_datas #看一下數值
```

	id	price	sale_yr	sale_month	sale_day	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	5615100330	200000	2015	3	27	4	2.00	1900	8160	1	0	0	3	7	1900	0	1975	0	98022	47.2114	-121.986	1280	6532
1	8835900086	350000	2014	9	2	4	3.00	3380	16133	1	0	1	3	8	2330	1050	1959	0	98118	47.5501	-122.261	2500	11100
2	9510900270	254000	2014	12	11	3	2.00	2070	9000	1	0	0	4	7	1450	620	1969	0	98023	47.3085	-122.376	1630	7885
3	2621600015	175000	2015	4	30	3	1.00	1150	8924	1	0	0	3	6	1150	0	1943	0	98030	47.3865	-122.217	1492	8924
4	8078350090	619000	2015	3	31	3	2.50	2040	7503	2	0	0	3	8	2040	0	1987	0	98029	47.5718	-122.021	2170	7503
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12962	9253900354	580000	2014	7	1	3	2.50	2200	11000	2	0	2	3	9	2200	0	1978	0	98008	47.5916	-122.112	2200	12851
12963	9510300130	598000	2014	6	28	4	2.50	3130	40918	2	0	0	3	9	3130	0	1994	0	98045	47.4761	-121.723	2760	35440
12964	1105000373	252500	2015	5	6	2	1.50	1110	986	2	0	0	3	7	950	160	2009	0	98118	47.5427	-122.272	1110	3515
12965	3629990280	497000	2014	6	23	3	2.25	1630	3817	2	0	0	3	7	1630	0	2005	0	98029	47.5485	-121.999	1630	3348
12966	9521100586	479000	2014	5	24	3	1.00	1370	3000	1	0	0	3	7	1370	0	1924	0	98103	47.6619	-122.351	1510	2151

12967 rows × 23 columns

去空值

我想看看資料裡面是否有缺失值

測試並沒有空值

```
[ ]  a_datas.dropna()  #去空值看看，測試結果沒有空值
```

	id	price	sale_yr	sale_month	sale_day	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	5615100330	200000	2015	3	27	4	2.00	1900	8160	1	0	0	3	7	1900	0	1975	0	98022	47.2114	-121.986	1280	6532
1	8835900086	350000	2014	9	2	4	3.00	3380	16133	1	0	1	3	8	2330	1050	1959	0	98118	47.5501	-122.261	2500	11100
2	9510900270	254000	2014	12	11	3	2.00	2070	9000	1	0	0	4	7	1450	620	1969	0	98023	47.3085	-122.376	1630	7885
3	2621600015	175000	2015	4	30	3	1.00	1150	8924	1	0	0	3	6	1150	0	1943	0	98030	47.3865	-122.217	1492	8924
4	8078350090	619000	2015	3	31	3	2.50	2040	7503	2	0	0	3	8	2040	0	1987	0	98029	47.5718	-122.021	2170	7503
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12962	9253900354	580000	2014	7	1	3	2.50	2200	11000	2	0	2	3	9	2200	0	1978	0	98008	47.5916	-122.112	2200	12851
12963	9510300130	598000	2014	6	28	4	2.50	3130	40918	2	0	0	3	9	3130	0	1994	0	98045	47.4761	-121.723	2760	35440
12964	1105000373	252500	2015	5	6	2	1.50	1110	986	2	0	0	3	7	950	160	2009	0	98118	47.5427	-122.272	1110	3515
12965	3629990280	497000	2014	6	23	3	2.25	1630	3817	2	0	0	3	7	1630	0	2005	0	98029	47.5485	-121.999	1630	3348
12966	9521100586	479000	2014	5	24	3	1.00	1370	3000	1	0	0	3	7	1370	0	1924	0	98103	47.6619	-122.351	1510	2151

12967 rows × 23 columns

擬定預測的部分

後面我就先把price、id先拿掉
一個需要預測一個似似乎不太重要

再來就是我們需要預測房價
所以我把price做一個變數以便後續

```
[ ]  X_train  =  a_datas.drop(['price','id'],axis=1).values
     Y_train  =  a_datas['price'].values

[ ]  b_datas  =  pd.read_csv(f'{DATA_HOUSE}/valid-v3.csv')
     X_valid  =  b_datas.drop(['price','id'],axis=1).values
     Y_valid  =  b_datas['price'].values

[ ]  c_datas  =  pd.read_csv(f'{DATA_HOUSE}/test-v3.csv')
     X_test   =  c_datas.drop(['id'],axis=1).values
```

正規化

其實我還不太了解正規化的意思

目前大略知道是為了

避免overfitting的問題

就是避免訓練過程中表現很好

但是在測試的結果卻表現很差

```
[ ]  transfer = StandardScaler().fit(X_train)  #正規化  還不是很理解意思
     X_train = transfer.fit_transform(X_train)
     X_valid_1 = transfer.fit_transform(X_valid)
     X_test = transfer.fit_transform(X_test)
```

模型的部分

這部分也是參考同學分享的作業

我先用Sequential 順序模型

因為它是多個網絡層的線性堆疊

所以我可以自行定義

要幾層結果才會比較好

而我記得activation是為了讓模型

變成能解決非線性問題

最後就是讓模型開始跑

```
model = Sequential()
model.add(Dense(units=90, input_dim=X_train.shape[1], kernel_initializer='normal',activation='relu'))
model.add(Dense(units=150, kernel_initializer='normal',activation='relu')) #activation是輸出
model.add(Dense(units=300, kernel_initializer='normal',activation='relu'))
model.add(Dense(units=300, kernel_initializer='normal',activation='relu'))
model.add(Dense(units=200, kernel_initializer='normal',activation='relu'))
model.add(Dense(units=90, kernel_initializer='normal',activation='relu'))
model.add(Dense(units=1, kernel_initializer='normal', activation='relu'))


model.compile(loss='MAE', optimizer='adam',) #規則要loss=MAE
epochs = 110 #epochs調高batch_size就要調低_反之也是
batch_size = 39
```

```
[ ] file_name=str(epochs)+'_'+str(batch_size) #開始讓模型跑
    TB=TensorBoard(log_dir='logs/'+file_name, histogram_freq=0)
    model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs,verbose=1,validation_data=(X_valid_1, Y_valid),callbacks=[TB])
```

運算結果

這是試了幾次之後

覺得結果還不錯而定的訓練次數

像是epochs 次數調高

而batch_size就要調低反之也是

可以看到loss值都在下降

下一步就是儲存檔案

```
[ ]  file_name=str(epochs)+'_'+str(batch_size)  #開始讓模型跑
     TB=TensorBoard(log_dir='logs/'+file_name,  histogram_freq=0)
     model.fit(X_train,  Y_train,  batch_size=batch_size,  epochs=epochs,verbose=1,validation_data=(X_valid_1,  Y_valid),callbacks=[TB])

     333/333 [==============================] - 2s 6ms/step - loss: 39519.9805 - val_loss: 72709.0391
     Epoch 83/110
     333/333 [==============================] - 2s 6ms/step - loss: 38795.9492 - val_loss: 73838.4531
     Epoch 84/110
     333/333 [==============================] - 2s 6ms/step - loss: 38685.6094 - val_loss: 73153.3516
     Epoch 85/110
     333/333 [==============================] - 2s 6ms/step - loss: 38943.3320 - val_loss: 72842.7109
     Epoch 86/110
     333/333 [==============================] - 2s 6ms/step - loss: 38724.7734 - val_loss: 74588.4922
     Epoch 87/110
     333/333 [==============================] - 2s 6ms/step - loss: 38567.9062 - val_loss: 74834.8906
     Epoch 88/110
     333/333 [==============================] - 2s 6ms/step - loss: 39248.6367 - val_loss: 72772.5000
     Epoch 89/110
     333/333 [==============================] - 2s 6ms/step - loss: 38727.1406 - val_loss: 73460.5000
     Epoch 90/110
     333/333 [==============================] - 2s 7ms/step - loss: 37765.2969 - val_loss: 72377.0078
     Epoch 91/110
     333/333 [==============================] - 2s 7ms/step - loss: 38181.0117 - val_loss: 73903.4219
     Epoch 92/110
     333/333 [==============================] - 2s 6ms/step - loss: 38794.0625 - val_loss: 72675.8047
     Epoch 93/110
     333/333 [==============================] - 2s 6ms/step - loss: 38279.8281 - val_loss: 72474.5859
     Epoch 94/110
     333/333 [==============================] - 2s 6ms/step - loss: 38235.9883 - val_loss: 72139.4844
     Epoch 95/110
     333/333 [==============================] - 2s 6ms/step - loss: 37891.8945 - val_loss: 72634.5391
     Epoch 96/110
     333/333 [==============================] - 2s 6ms/step - loss: 38154.2695 - val_loss: 73851.1484
     Epoch 97/110
     333/333 [==============================] - 2s 6ms/step - loss: 37560.1172 - val_loss: 72698.5391
     Epoch 98/110
     333/333 [==============================] - 2s 6ms/step - loss: 37285.6211 - val_loss: 74454.4531
     Epoch 99/110
     333/333 [==============================] - 2s 6ms/step - loss: 37662.6211 - val_loss: 73115.2422
     Epoch 100/110
     333/333 [==============================] - 2s 6ms/step - loss: 37877.4219 - val_loss: 73809.4766
     Epoch 101/110
     333/333 [==============================] - 2s 6ms/step - loss: 37773.6094 - val_loss: 73798.3359
     Epoch 102/110
     333/333 [==============================] - 2s 6ms/step - loss: 37120.9375 - val_loss: 74011.8203
     Epoch 103/110
     333/333 [==============================] - 2s 6ms/step - loss: 36927.0898 - val_loss: 73214.3438
     Epoch 104/110
     333/333 [==============================] - 2s 6ms/step - loss: 38087.6602 - val_loss: 76600.4453
     Epoch 105/110
     333/333 [==============================] - 2s 6ms/step - loss: 37299.2852 - val_loss: 74294.1562
     Epoch 106/110
     333/333 [==============================] - 2s 6ms/step - loss: 36448.0820 - val_loss: 73824.6406
     Epoch 107/110
     333/333 [==============================] - 2s 6ms/step - loss: 37019.5117 - val_loss: 74167.7031
     Epoch 108/110
     333/333 [==============================] - 2s 7ms/step - loss: 37027.6562 - val_loss: 72281.5938
     Epoch 109/110
     333/333 [==============================] - 2s 6ms/step - loss: 36371.7734 - val_loss: 76929.5859
     Epoch 110/110
     333/333 [==============================] - 2s 6ms/step - loss: 36576.3867 - val_loss: 74508.3750
     <keras.callbacks.History at 0x7f19b0e547d0>
```

儲存csv

我也是先把模型存檔

之後再用model預測test

然後設定成一行id一行price

再把結果存成csv檔案

以符合上傳kaggle比賽的格式

```
[ ]  model.save('h5/'+file_name+'.h5')

[ ]  test_predict  =  model.predict(X_test)

[ ]  with  open("test.csv","w")  as  f  :
         f.write("id,price\n")
         for  i  in  range(len(test_predict)):
             f.write(str(i+1)+","+str(float(test_predict[i]))+  "\n")
```

心得

因為我是工業設計跨領域選修這堂課
很多地方都不太清楚
走一步學一步的感覺

但是老師講解很細
加上會有同學分享他們的程式報告
看了各式各樣同學的分析、模型方式
真的學到很多
希望能更熟悉以致未來我也能像他們一樣