HTML 2023 Fall Final Project Regular Track

Group: 為什麼會變成這樣呢
Member: 官毓韋、蘇翊軒、蔡朝暐

壹、資料觀察

我們認為會影響腳踏車租借狀態的因素有：時間、天氣、附近站點的租借狀態、台大附近是否有特別活動等。以下透過圖表舉例，說明我們發現的資料特性。

一、時間

由於要預測站點均在台大附近，因此我們猜測每週會有循環的租借行為（這週一和下週一租借行為會類似），而沒有上課的假日的行為會與平日有所不同。

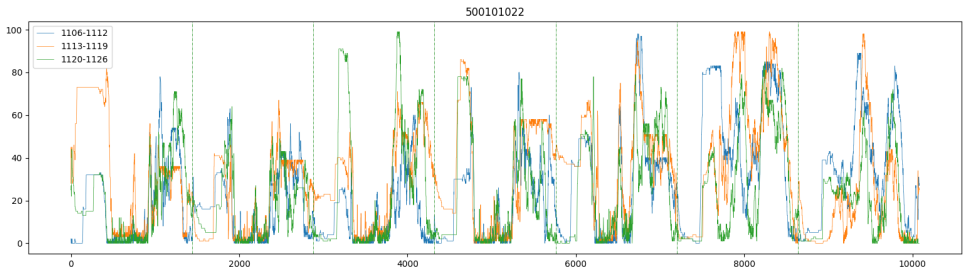

Figure 1: 捷運公館站 (2 號出口) 連三周的剩餘車數與時間關係圖

Figure 1：橫軸為每週累積分鐘數、縱軸為車數，每天以虛線分開。可發現平日白天的車偏少，到下課時間會變多（去捷運站），並且在每周同時間有類似現象。
我們發現資料大致上會每週循環，推測是上課、社團等每週固定的校內活動，導致借車會有循環現象。另外，若當天為國慶、校慶等特別節日，行為則與平時不同。

二、天氣

在預想中，會影響借車意願的有雨量（雨量大會降低騎車意願）、氣溫（太熱太冷會降低騎車意願）。但經過我們觀察，即使溫度差異大，實際租借量差異也沒有明顯變動。

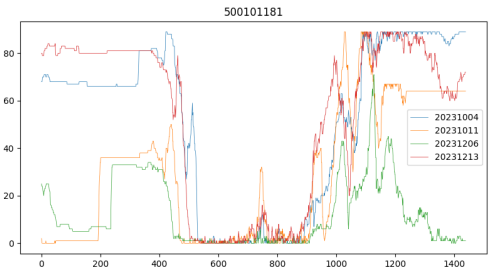

Figure 2: 捷運公館站 (3 號出口) 不同周的星期三剩餘車數與時間關係圖

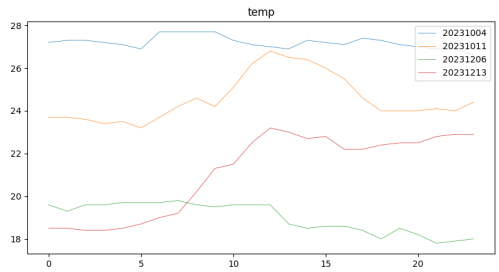

Figure 3: 不同周的星期三全日溫度變化

Figure 2：橫軸為當天累積分鐘數，縱軸是剩餘車數。

Figure 3：橫軸為累積小時數，縱軸是攝氏溫度。
可發現此站點的車數並未因為溫度而有明顯變動，反而是跟時間更有關係。故我們推測在天氣資料中溫度並沒有很強的關聯，反而是雨量與站點車數較有關係。

三、站點

以組員的個人觀察而言，鄰近區域的車站常有同時全滿或全空的情形，於是我們針對鄰近區域的車輛數進行檢視。經過檢視鄰近站點組合的資料，我們發現此鄰近關係並未有明顯影響。因此，我們決定分開處理每個站點，以減小複雜度、增加訓練效率。

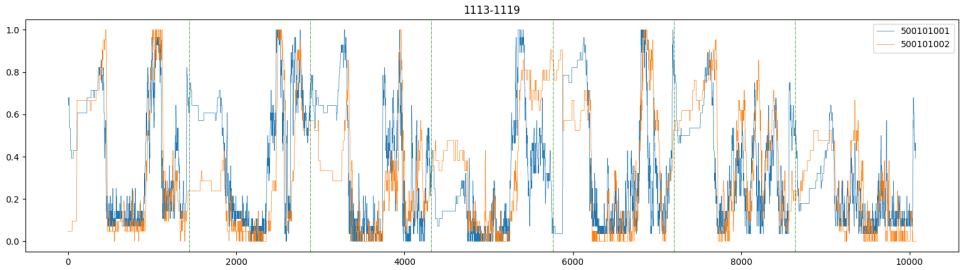

Figure 4: 捷運科技大樓 (500101001) 及復興南路二段 273 號前 (500101002) 於 11/13-11/19 剩餘腳踏車比例與時間關係圖

Figure 4：橫軸為日累積分鐘數，縱軸為剩餘車數除以車柱數，每天以虛線分開。可發現兩者並沒有預想中的明顯相似關係。

四、活動

台大時而會有特別活動（如語言檢定等）或有特別節日，導致站點的車數與平常有非常大的差別，因此新增一個參數紀錄當天是否有特別節日、活動以處理此情形。

五、總結

我們為了簡化複雜度及避免模型 overfit，我們決定每個站點使用不同的模型（分開訓練），輸入資料大致為車柱、時間、雨量及日期特性。

貳、資料處理

一、補漏遺失資料

在 github 的 Youbike 資料有所缺漏，如果有 1 小時內連續的缺漏，我們將前後資料的平均填補；若有更大的缺漏，我們會人工檢視並考慮放棄將那一天當作測試資料，以確保資料都能完整且以一天為單位。

二、氣象資料

歷史資料：
我們從中央氣象局公開資料平台 [1] 下載歷史氣象資料。由於沒有台大測站資料，我們使用永和測站的資料代替，若有資料缺漏以先後一小時資料填補。

預測資料：
我們使用 weatherbit [2] 的 API 取得未來 240 小時的氣象預測資料。

三、時間循環

為使時間能在某個周期 T 內為循環數值而非遞增，我們在時間序列模型中，將累積的分鐘數 t 轉成 $\sin(\frac{2\pi t}{T})$, $\cos(\frac{2\pi t}{T})$，以強調循環性。

參、平均

由上面的分析，我們認為車數資料具有循環性，故我們嘗試將不同周同一時間點進行平均，可得 $E_{\text{val}} = 0.31537$。優點是速度極快，即我們可以在整理資料的同時就算出；缺點是這樣無法應對每一周不同的狀況如天氣等等。

肆、時間序列相關模型

我們需要預測「連續」一個禮拜的資料，因此選用時間序列模型。我們使用了 RNN、LSTM、GRU 三個時間序列模型，並以 pytorch 在 google colab 使用 GPU 實作。
輸入包含（假設 t 是從當天累積分鐘數，時雨量為 r 毫米）：
1: $\sin(\frac{2\pi t}{180})$, **2:** $\cos(\frac{2\pi t}{180})$, **3:** $\sin(\frac{2\pi t}{1440})$, **4:** $\cos(\frac{2\pi t}{1440})$, **5-11:** ⟦星期日 $\cdots$ 六？⟧,
12: $(\frac{1}{\exp(-2.5r)} - 0.5)$, **13:** ⟦上課日？⟧, **14:** ⟦節日？⟧
輸入解釋：
1,2: 三小時循環、**3,4:** 一天循環、**5-11,13,14:** 日子特徵，**12:** 轉換雨量使其到一定的數值後接近持平，以符合現實生活中雨量到一定程度後借車意願就不會再降。
模型共有特性：
每 20 分鐘為一個時間段 (shape:(72× 天數, 20, 14))，每時間段輸出一個值。另外訓練時隱藏狀態 h_t 及 c_t (for LSTM) 在每個 batch 開始時設為隨機值，並通過每個 Layer 後作為同個 batch 下一組資料的輸入。
batch_size = 504(一周), **n_epoch**=10
Loss: $3\left|\frac{b_{i,t}-\hat{b}_{i,t}}{s_i}\right| \times \left(\left|b_{i,t} - \frac{1}{3}\right| + \left|b_{i,t} - \frac{2}{3}\right|\right)$
Optimizer: Adam, **Learning Rate**: 0.01

一、RNN

RNN (Recurrent Neural Network) 可以捕捉短期的時間依賴性。我們認為一個點當下的腳踏車數量會與前幾個小時的數量有關，因此 RNN 是一個適合的模型。

1. 模型架構

Input(14)→Linear(14, 200)→RNNCell(200, 200)×3→ReLU→Linear(200, 1)→Output(1)。

2. 分析

$E_{\text{val}} = 0.32482$
優點：可以在初期訓練資料及預測時間少時表現較佳。

缺點：記不住太久以前的資料，導致 underfit，整體表現較平均為差。

二、LSTM

LSTM (Long Short-Term Memory)[3] 是為了解決 RNN 中時間過長，以前資料會被「忘記」的問題。由於資料會有部分的循環關係，因此採用此模型。

1. 模型架構

Input(14)→Linear(14, 200)→LSTMCell(200, 200)×3→ReLU→Linear(200, 1)→Output(1)。

2. 分析

$E_{\text{val}} = 0.29316$
優點：可以在重複性高的站點時表現較佳
缺點：遇到有季節性變化或突發事件多的站點表現會較差。

三、GRU

由於 LSTM 的結構過於複雜，所耗費的運算資源較大，GRU [4](Recurrent Neural Networks) 將模型進行簡化避免長時間梯度爆炸問題。

1. 模型架構

Input(14)→Linear(14, 200)→GRUCell(200, 200)×3→ReLU→Linear(200, 1)→Output(1)。

2. 分析

$E_{\text{val}} = 0.28876$
優點：對於突發事件的處理較另外兩者好，因此 validation 也有較好的結果。
缺點：部分資料與時間相關性低的站點表現較另外兩者差。

四、整體優缺點

優點：三個模型都較能捕捉資料與時間的相關性。
缺點：三個模型訓練時間均較久（約 20 秒/站）且在資料不多時學得較差。

伍、Gradient Boosting

Gradient Boosting 利用集合各決策樹的結果以提高模型準確性，因此能夠減少極端值對整體結果的影響，並且在變化大的資料上也能迅速擬合。我們認為腳踏車租借本身帶有隨機性、且在許多時候有很不穩定的變化（像是有特別活動），因此希望能夠使用 Gradient Boosting 來處理這樣的情形。我們使用 XGBoost 和 LightGBM 兩個基於 Gradient Boosting 的模型。以下兩個模型的輸出入如下：
輸入各維度：
1: min, **2:** week_min **3:** ⟦上課？⟧, **4:** ⟦節日？⟧, **5:** 雨量
(min：日累積分鐘數、week_min：週累積分鐘數)
輸出：預測腳踏車數

處理權重：

由 evaluation metric 可知實際車數離中間三分之一越遠的時候，錯誤時的權重較重，最多為普通 mse 的三倍。
首先，我們假設某些時刻車全滿或全空。因此，若在資料上表現這些重要性，使模型特別做好這些時刻，就可能讓預測變好。
於是，我們做出第一次嘗試，使用條件式來新增加資料，在 $[0, \frac{1}{3})$ 和 $(\frac{2}{3}, 1]$ 多一倍、$[0, \frac{1}{3})$ 和 $(\frac{2}{3}, 1]$ 多一倍等等。
再來，因為條件式的判斷難以調整以找出最好的組合，我們改使用 for 迴圈。在訓練階段時，每筆原始訓練資料會重複放入訓練集 $3N(\lfloor|\frac{b_{\text{truth}}}{s} - \frac{1}{3}| + |\frac{b_{\text{truth}}}{s} - \frac{2}{3}|\rfloor) - (N-1)$ 次，並嘗試不同的最高資料倍數 N 找到最好的結果。

Table 1: 使用 lightgbm

N	**train err**	**validation err (11/23-11/29)**
1	0.26559	0.31056
10 (best N)	0.23660	0.26807

一、XGBoost

XGBoost [5] (eXtreme Gradient Boosting) 是 Gradient Boosting 的平行化、最佳化的版本。透過平行計算能大幅縮短時間。

1. 模型架構

使用 python 套件 `xgboost.XGBRegressor` 實作。
參數：在我們嘗試過各種參數組合後，以下是在 validatoin 表現最好的。
lightgbm：max_depth=6, min_child_weight=1, learning_rate=0.9, n_estimators=100
資料權重：最高倍數 $N = 10$

2. 分析

訓練結果：$E_{\text{train}} = 0.20970$, $E_{\text{validation}} = 0.27476$
優點：速度快（約 30 分鐘可訓練完），並且在 validation 有很好的表現。
缺點：因為每次的預測和前次的預測無關。因此，像是在有下雨的時候，lightGBM 無法有效的去預測腳踏車數（下大雨車數無變化時），就會在雨天表現較差。

二、LightGBM

LightGBM [6] 同樣是基於梯度提升決策樹模型的熱門機器學習架構。LightGBM 通常有比 XGBoost 快的速度，但也較容易 overfit。相較 XGBoost 樹是水平地生長，生長樹的層數（Level-wise），LightGBM 是從葉子的方向生長（Leaf-wise）。

1. 模型架構

使用 python 套件 `lightgbm` 實作。
參數：在我們嘗試過各種參數組合後，以下是在 validatoin 表現最好的。
lightgbm：num_leaves=31, learning_rate=0.05, feature_fraction=0.9, num_round=100
資料權重：最高倍數 $N = 10$

2. 分析

訓練結果：$E_{\text{train}} = 0.23660$, $E_{\text{validation}} = 0.26807$
優點：在速度方面，lightGBM 明顯快於其他模型 (約 10 分鐘可訓練完) 比起 XGBoost 還快不少，並且在 validation 的表現也很好。
缺點：同 XGBoost，不擅長處理下雨天的情況。

陸、結論

根據以上的分析，我們最後選擇的是 LigntGBM ，其理由如下：
LightGBM 在速度、準確性方面都有很好的表現，尤其在速度僅需 10 分鐘就可訓練完成（若將資料倍數 N 調整成更小的值可在 5 分鐘內跑完)。而 LightGBM 本身的特性，也使得它在這次資料少、變化大的腳踏車數預測上能夠快速地做到不錯的成績。雖然在會有無法以時間關係去預測雨天情形的問題，但一般來說仍是有用的模型。

這個 Project 是要預測現實生活中的資料，實際變因非常多而變化複雜；並且在僅兩個月的資料量預測未來一周的資料，比例上太少；而秋季、學期中的資料作為訓練也較難預測出冬季、學期末的資料。我們僅能將 Score 降到約 0.3 左右，實用意義不大。如果我們有辦法獲得更多資料諸如活動資料、使用者習慣、旅次（借還車站點)、車輛調度資料並適當的量化，或是更長時間的資料，才能使用較複雜的模型如 RNN、LSTM、GRU 等做出更好的結果。

柒、工作分配

B11902132 官毓韋：資料觀察、平均、RNN、LSTM、GRU、Report。
B11902073 蘇翊軒：XGBoost、Lightgbm、Report。
B11902040 蔡朝暐：資料處理、XGBoost、Lightgbm、Report。

捌、參考資料

[1] https://codis.cwa.gov.tw/StationData
[2] https://www.weatherbit.io/api/weather-forecast-hourly
[3] https://www.bioinf.jku.at/publications/older/2604.pdf
[4] https://arxiv.org/abs/1412.3555
[5] https://xgboost.readthedocs.io/en/stable/
[6] https://lightgbm.readthedocs.io/en/latest/