Machine Learning Final Project

主題：" *DengAI: Predicting Disease Spread* "

Reproduce 方法 :

bash final.sh train_feature.csv 路徑 test_feature.csv 路徑 label.csv 路徑 output.csv 路徑

1. Team name, members and your work division

Team name: NTU_b03705013_BIG

Members:

B03705013：徐逸然 - Xgboost, DNN

B03705014：陳威宇 - RNN

B03705027：鄭從德 - RNN

B03705028：陳星宇 - Xgboost, DNN

2. Preprocessing/Feature Engineering:

我們在 model 的選擇上一共實作了三種 Model：Xgboost、DNN 以及 RNN。我們最終結果最佳的 model 是 Xgboost 以及 RNN，所以此篇報告會在每一個段落分別介紹我們 Xgboost 以及 RNN 使用的方法。

Xgboost :

首先將每一個 feature 與 label 算出相關係數 r，刪去掉 | r | < 0.2 之 feature。接著將每一筆資料納入前四天與後四天之所有 features，頭尾幾筆資料這時會出現問題(即缺少前四天或後四天之資料)，我用折衷的方法 – 補上自己的 feature 來代替。與組員討論後 label 也納入考量，將一筆資料前四天之 label 納入 feature，頭尾資料處理方式如上。最後我們將 weekofyear 做了一些調整，將其值 week 用 sin(week/52) 替換，以符合一年循環中第一周與第五十二周應該相近的特性。最後將所有資料作標準化。

RNN :

我們在 feature 的選擇上跟 Xgboost 比較不同的是，我們只選擇前面標註 reanalysis 的 features(經測試效果較全選佳)。其他方面與 Xgboost 作法相似：我們將前一週的 label 加入本週已經過標準化的 features(label feedback)、並且將 weekofyear 轉換成 sin(week/52)。

而關於 preprocessing 的部分，因為 RNN 看的是一序列的資料，所以我們會取連續 seq_len 週的 data 餵到 LSTM 裡面。因此我們將我們的 data 每 seq_len 筆切一份(其中每一筆包含前一周的 label)，然後再利用隨機排序的方式讓 model 吃到 training data 的順序隨機，以此增進總體表現。

而對於 Nan 資料的處理方法我們都是使用了'ffill'，取前一筆 valid 的 data 來當作值。我們認為這在以周為單位資料上是較為合理的。

3. Model Description :

Xgboost :

Boosting classifier 屬於 Ensemble 模型的一種，將數個準確率較低的 Tree 組合起來，成為一個準確率較高的模型。每個 iteration 就生成一顆新的樹。利用 gradient descent 縮小 loss。Xgboost 是 Extreme Gradient Boosting 的縮寫。能夠利用 CPU 的 multithread 的特性進行平行運算，同時提高了精度。我們將兩個城市(sj 與 iq)分開訓練，兩個城市之 model 參數都相同，number of rounds 設為 400，每棵樹的 max depth 設為 4，subsample (每次生成樹時會隨機從所有 data 取一部分出來，以防止 overfit)設為 0.7，learning rate 設為 0.02。在 preprocess 時將每筆 train data 前四天之 label 納入 feature 當中，然而 test data 並無 label 可用，因此在預測時我們將每筆 test data 加入前面四筆 test data 預測出來之值當作 feature。例如預測第五筆 test data 時，納入第一筆到第四筆 test data 預測出來的值當 feature。

LSTM :

因為題目是時間序列的資料，所以我們嘗試用 LSTM 來做預測。LSTM 的 input data 是一個三維的矩陣，第二維是時間回溯的長度，而第三維度就是我們的 feature 數，比較特別的是我們會把前一時間資料的 label 也當成 feature。

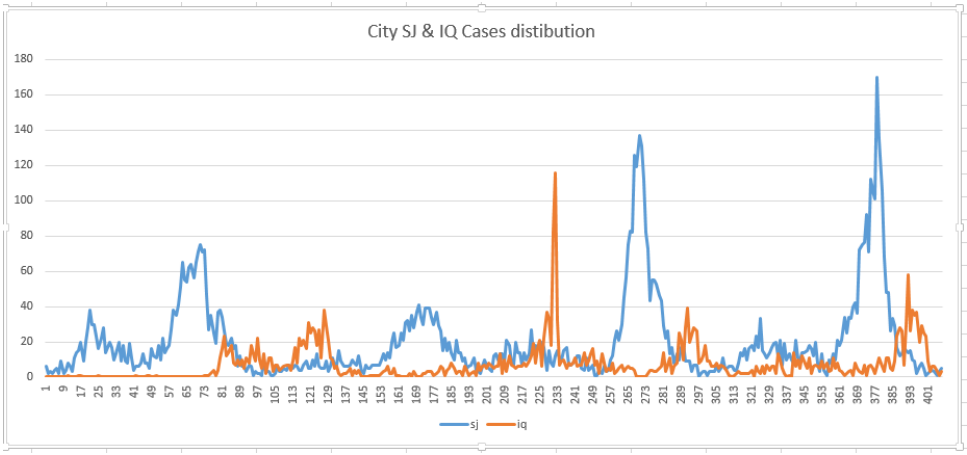
我們嘗試了不同層數 LSTM 模型，後來發現越多層的表現並沒有比較好。同時，若開太多的 Dense 則很快就有 overfit 的情況發生，所以最後我們就用了一層的 LSTM 和一層的 Dense。因為 input 一次丟了好幾週的 data 進去，所以在 dense 的時候我們用了包裝器 TimeDistributed 將其打包。

下圖為我們 LSTM model 的架構：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, None, 50)          12000
_________________________________________________________________
dropout_4 (Dropout)          (None, None, 50)          0
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 1)           51
=================================================================
Total params: 12,051
Trainable params: 12,051
Non-trainable params: 0
_________________________________________________________________
```

4. Experiments and Discussion :

首先我們將 train data 中兩個城市時間重疊的部分作圖，發現兩個城市在登革熱 case 的分布上差異極大，因此我們決定將兩個城市分別用不同模型訓練。

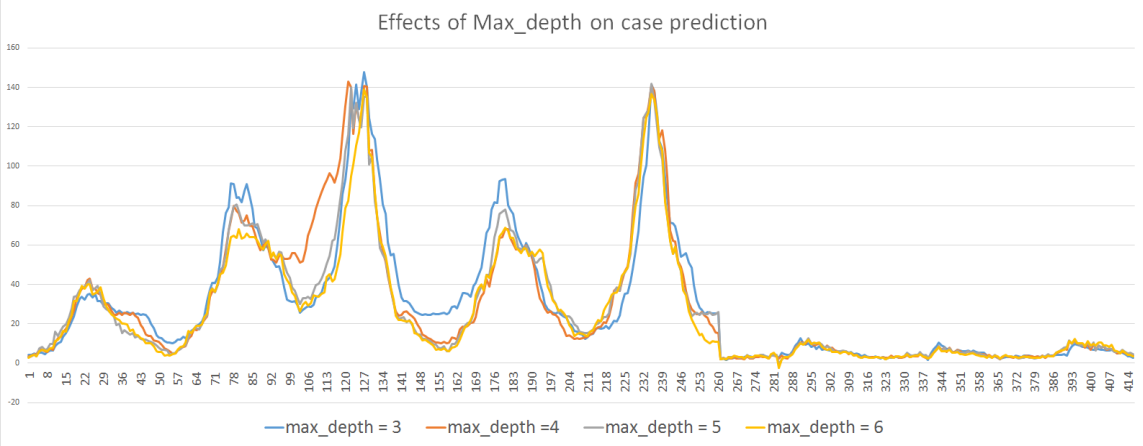


Xgboost 之實驗與討論 :

Xgboost 實作比起 DNN, LSTM 等簡單許多。參考了一些文獻之後我們決定從樹的深度開始調整。預設數值為 6，也就是每一棵樹最深能走到第六層，愈大的值代表模型越複雜，也愈有可能產生 overfit 的情況。以下是不同 max_depth 下預測之情形與 MAE 分數。我們推測 max_depth = 4 時分數較好的原因是第 90-120 筆資料預測出了較其他更高的值，剩餘資料之預測結果與其他深度沒有太大的區別。我們也發現 max_depth = 6 時已發生 overfit 的狀況，甚至預測出負結果。確定 max_depth 之後往後的實驗都以 max_depth = 4 進行。

Max_depth	MAE
3	21
4	17.1

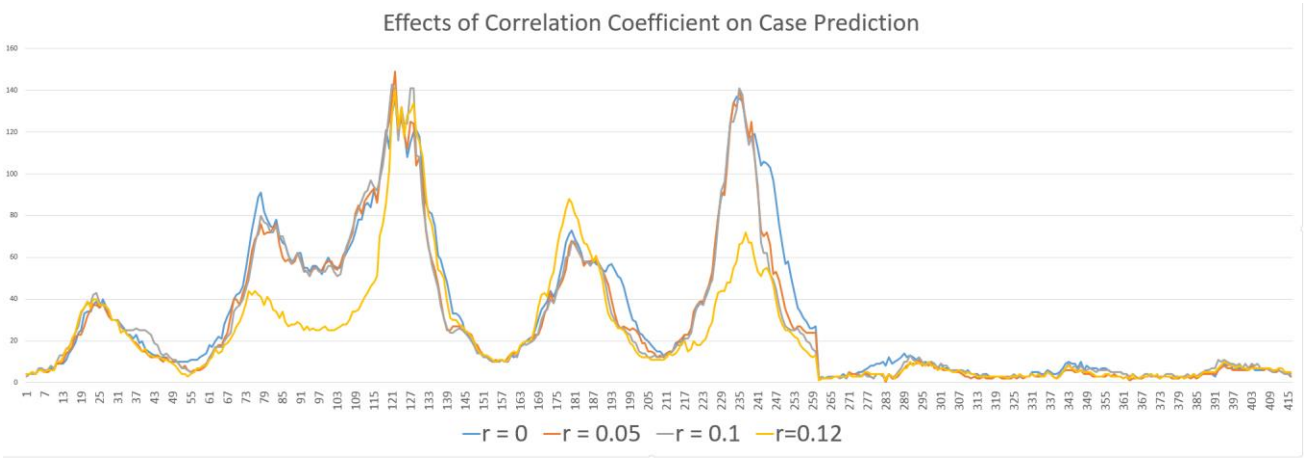
5	19.2
6	19.7



接下來針對不同 Correlation Coefficient 門檻作討論，在前一節有提到在處理資料時將相關係數小於一定值的 feature 刪除，篩選掉過多 feature 會產生 underfit 的情況，以下是不同門檻對預測結果之影響與 MAE 分數。從 MAE 來看，在門檻小於 0.1 時不同門檻對於分數影響甚小，最大差距甚至僅不到 2%。然而 0.12 開始由於刪去過多 feature，準確率下降，預測出來的結果也與門檻小於 0.1 時有相當差異，尤其是約 73 至 120 筆左右的數據沒有被預測出來。以下的測試結果與做圖皆在 Windows 環境下進行，然而由於在 Linux 環境與 Windows 環境跑出來的結果有些許差異，最後我們決定將門檻設為差異最小的 0。

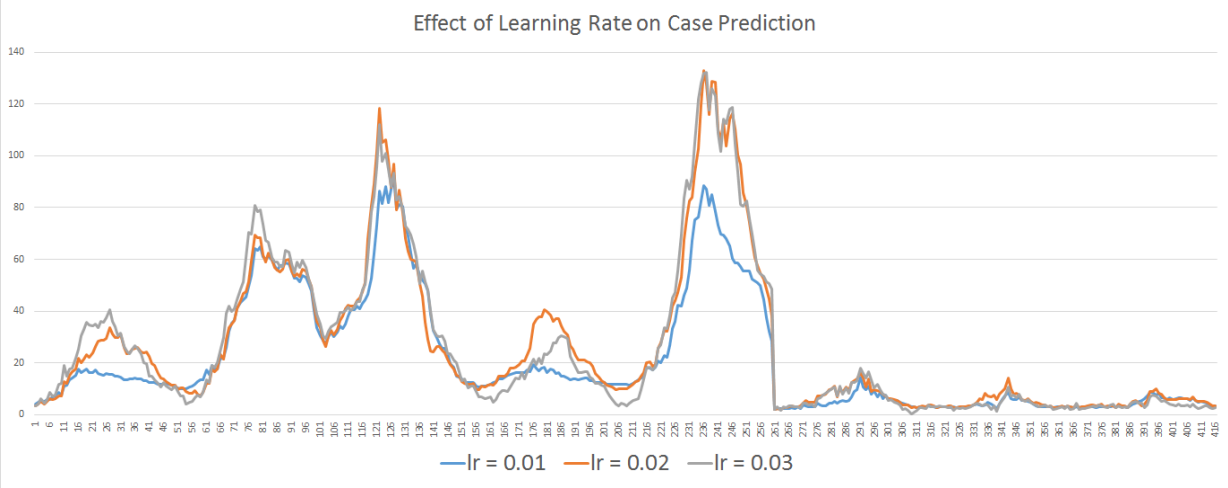
Threshold	MAE
0.0	17.15
0.05	16.939

0.1	17.12
0.12	21.08



最後是關於 learning rate 之調整。Xgboost 官方建議之值為 0.01 - 0.2，考量到資料已作標準化的處理，因此 learning rate 我們首先測試了 0.01、0.02 與 0.03，以下是實驗結果。LR = 0.01 時會產生 underfit，即高點無法正確的預測，最好的結果為 LR = 0.02，而 0.3 以上之值也沒有測試的必要，會產生 overfit 的狀況。

Learning Rate	MAE
0.01	20.9
0.02	17.14
0.03	18.44

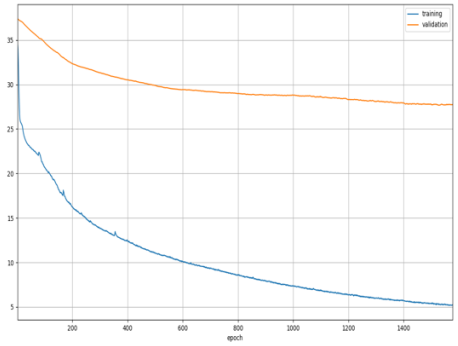


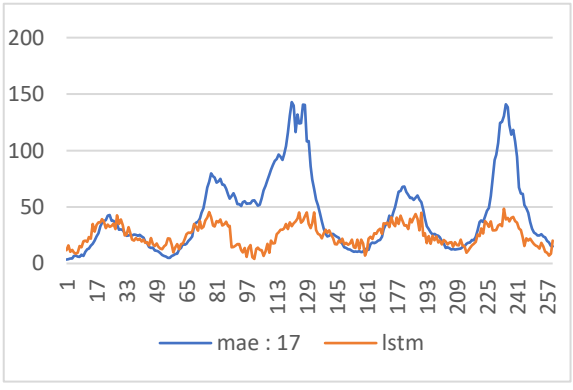
RNN 之實驗與討論：

（由於主要誤差來自於第一個城市，以下實驗及討論僅以城市 sj 來做討論）

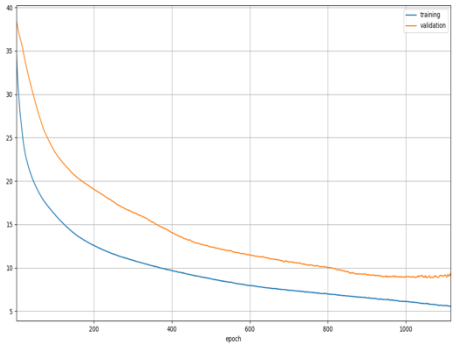
Feedback 之探討:

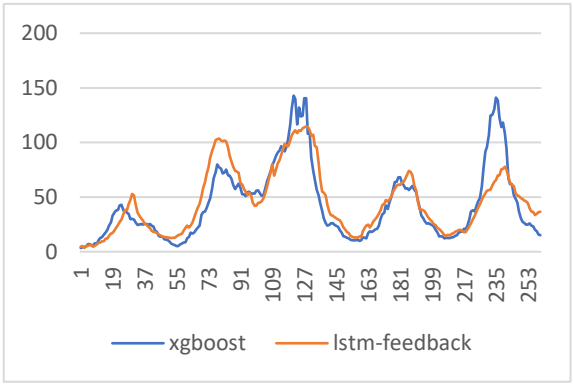
在沒有 feedback 作預測的情況下，從下左圖可以很明顯看出 validation 的預測一直很糟，沒有太大的進步。實際預測 test 的結果從右邊的圖可以發現，也僅預測出週期性的波動而已，很多峰值並沒有找出來。上傳成績 mae 大約落在 27 左右，連 simple 都沒過，所以我們決定加入 label feedback。



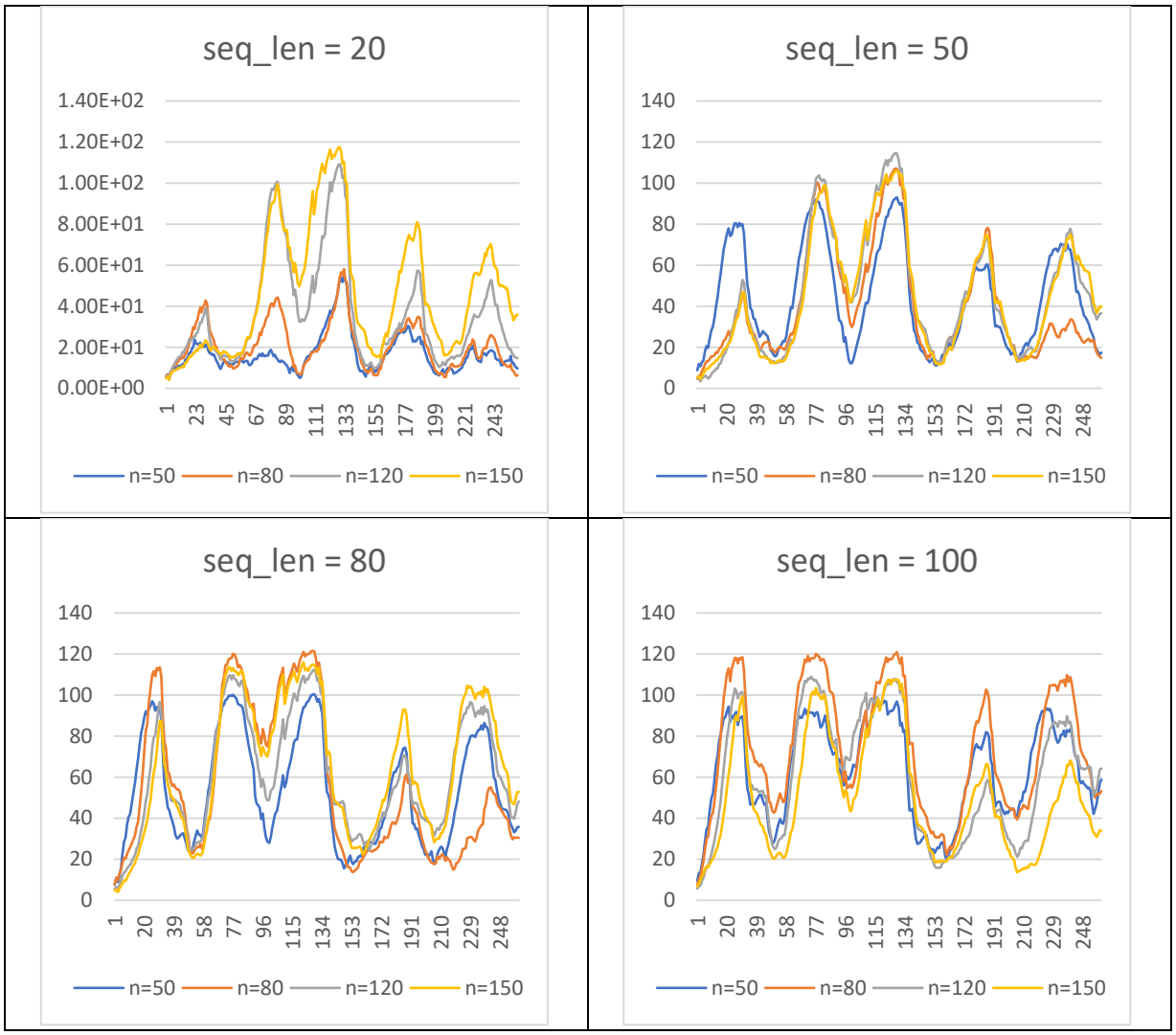


加入前一週的 label feedback 之後，從左下圖可以看出 validation 有很明顯的進步，test 的預測結果也更好，上傳後的 mae 進步至 23，順利通過 strong。





Seq_len 與 LSTM 大小 n 之探討:



我們發現當 seq_len 開太大，預我們發現當 seq_len 開太大，預測結果會普遍偏高，所以我們最後決定 seq_len 取 50。可是 seq_len 小就表示 lstm 要開大一點，因為 lstm 開太小會 train 不太起來。在 seq_len = 50 的狀況下第一個高峰會太高，在 seq_len = 20 的狀況則是明顯 underfit，試了幾種組合最後 lstm 大概可以到 mae 20 左右，然而還是不及 xgboost，在下一段討論當中將比較兩者之不同。

Xgboost 與 LSTM 之比較 (僅 SJ 城市有較大的差異) :

Xgboost 的成績大約落在 16,17 附近，相較之下 LSTM 的 20 就不是那麼的好了，比較一下兩者預測出來的折現圖。不難觀察出曲線大致相符，差異比較大的地方在於最後一個高峰，xgboost 明顯比 lstm 高出一大段，很可能是這段高峰僅 Xgboost 有預測出來。也嘗試將兩者取平均，但並沒有更好的分數，所以可能 lstm 還需要再多做一些調整，但是礙於最後我們都在研究 Xgboost，所以 lstm 並沒有做到非常好。