Machine Learning Techniques 2020 (Fall) Final Report

B06703094 馬愷若 B06902125 黃柏瑋

Task exploration

Revenue

我們的目標是要預測 daily revenue label，而題目說 daily revenue 是由同天中所有 fulfilled requests (沒被取消) 的 adr 乘上 total_nights 後加總而來，另外題目也提及 daily revenue is quantized to 10 scales ，因此我們想要看計算出來的 daily revenue 該如何轉為 label。這是我們在 train data 上跑出 不同 label 與 daily revenue 對應的結果：

```
Label = 0: min revenue = 183.35630398510574, max revenue = 9887.119494847002
Label = 1: min revenue = 10136.212971949979, max revenue = 19909.56713576649
Label = 2: min revenue = 20007.362398111112, max revenue = 29760.288189155846
Label = 3: min revenue = 30033.421662054432, max revenue = 39984.69387868113
Label = 4: min revenue = 40675.17297414791, max revenue = 49934.0087966217
Label = 5: min revenue = 50009.79459197348, max revenue = 59790.77136037836
Label = 6: min revenue = 60518.61371995729, max revenue = 69668.52278220604
Label = 7: min revenue = 78092.05230779444, max revenue = 78092.05230779444
Label = 8: min revenue = 83757.1373748766, max revenue = 88521.30041624818
Label = 9: min revenue = 95364.35669256675, max revenue = 95364.35669256675
Correlation coefficient = 0.9808141787630956
```

以 row 為單位，label 代表 train revenue label ，min revenue 為屬於該 revenue label 所有 daily revenues 中的最小值，max revenue 則為 daily revenues 中的最大值。從結果可知，train data 中的 daily revenue 大概是以 10000 為單位量化為 revenue label，也就是說 daily revenue 是 0 ~ 9999 的 revenue label 為 0， daily revenue 是 10000~ 19999 的 revenue label 為 1，以此類推。

我們將該結果作為將 daily revenue 轉成 label 的重要依據，而 daily revenue 是由 adr、is_canceled 和 total_nights 組合成的，由於 test data 沒有包含 adr 與 is_canceled 這兩項 features ，因此我們想要建立兩個模型分別預測每筆訂單的 adr (Regression) 與 is_canceled (Binary Classification)，最後的 daily revenue 則由該日期所有訂單的 adr、(1 - is_canceled)、total_nights 相乘後加總而得。

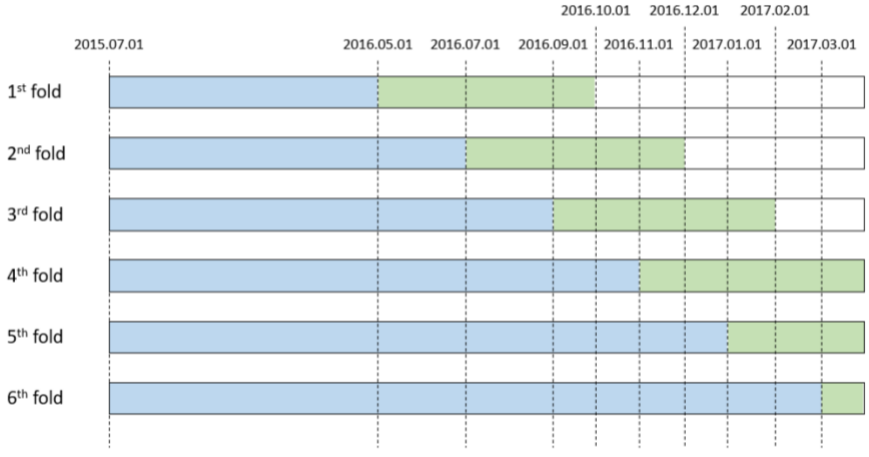
Validation

我們觀察到 train data 的發生時間為 2015 年 7 月至 2017 年 3 月，而 test data 發生的時間為 2017 年 4 月到同年 8 月。

首先，我們認為一般的 K-fold cross validation 不太適合我們的題目，雖然考慮多個 fold 可以有效降低單一 dataset 在分布上 bias 的影響 ，但某些 fold 的 validation data 發生時間會比 train data 還早 (即 data leakage 問題)，與 train data 和 test data 之間的關係不吻合，這樣的切法可能會使 validation score 很高的模型在 test score 上表現很糟。

接著，我們考慮了 hold-out validation，然而要切出與 test data 分布相近的 validation dataset 並不容易。對於 label，我們為每個月的 adr (連結) 和 is_canceled (連結) 作了簡易的分布統計，由連結中的圖表可以發現 adr 和 is_canceled 在不同月的 label 分布不盡相同，其中相鄰月的分佈會比較接近。如果我們將 2017 年 1 月至 3 月的資料切作 validation data，可能最後挑出的模型在 2017 年 1 至 3 月有不錯的表現，但由於 test data 為 2017 年 4 到 8 月，我們將很難保證該模型在那幾個月也能有優良的表現，尤其是距離 validation data 較遠的 6 至 8 月。

最後，我們保留了 K-fold cross validation 中利用多個 fold 的特色，以消弭單一 dataset 在分布上的 bias，並結合 hold-out validation 維持時間序列的正確性，發展出以下 time-series cross validation：



上圖中，藍色的部分為 train data 的時間區間，而綠色為 validation data 的時間區間。為了盡量模擬 test data，我們盡可能讓 validation set 橫跨五個月，而最後兩個 fold 的設計主要是為了關注模型在 2017 年的表現，可惜因為資料不足， validation set 只能橫跨三個月和一個月。最後我們將每個 fold 的表現平均，當作該模型的 validation score。

Feature Engineering

Feature Preprocessing

首先，我們發現 test data 裡面有的 features 與 train data 不盡相同，因此我們取兩者中 features 的交集。另外，我們將 ID 以及 company 這兩個 features 也拿掉，原因是我們認為 ID 對於預測結果的影響不大，而 company 的 missing value 太多，大約佔了全部的 94%。接著，我們將 categorical features （例如：meal、country、agent⋯⋯）轉成 one-hot 形式。值得注意的是，我們認為 arrival_date_month 的不同種類之間有距離上的關係 (如八月與七月的距離小於與一月的距離)，但與實際數值沒有太大關聯 (如我們無法斷言三月和八月誰比較大)；面對這樣的 feature，我們針對不同的模型採取不同的處理方式：

1. 對於 Linear Models，我們直接轉成 one-hot 形式，讓模型自行學習不同月份之間的關聯性。綜合前述其他的 preprocessing 技巧，最後共有 543 個 features。
2. 對於 tree-based ensemble models (即 Random Forest 和 XGBoost)，由於他們的 branching 或其他特殊設計 (詳見下方 Xgboost histogram-based method) 某程度上會將相鄰且影響力差不多的數值分成 discrete bins，因此我們保留月份間的時序性，將 arrival_date_month 轉成數字後直接餵入模型。綜合前述其他的 preprocessing 技巧，最後共有 532 個 features。

Feature Selection for Agent and Country

在比賽結束前，我們僅用上述做完 feature preprocessing 的 feature matrix 進行模型訓練與預測。然而在比賽結束後，我們又想到了一個能提升 feature 質量的方法，在某些成績上有一定提升，我們將這個結果與發現放在 Appendix 裡。

Models

Linear Models

第一個模型，我們使用 Linear Models。

剛開始資料會先經過用來進行 feature selection 的 L1-regularized model，由於 L1-regularization 會將不重要的 feature 權重限制在 0，因此我們會根據這項特性將權重為 0 的 feature 濾掉；之後資料才會進入用來預測的 L2-regularized model 輸出預測結果。而針對 adr 的預測我們皆使用 Linear Regression 搭配 Mean Square Error，針對 is_canceled 的預測則用 Logistic Regression，並使用 Scikit-Learn (0.24.0) 建模 (參考連結: Lasso、Ridge、LogisticRegression)。

在參數選擇上，我們嘗試調整不同強度的 regularization，並將 max_iter 固定為 1e+8 以確保能夠收斂，其他參數則維持預設。在這裡 Linear Models 的總參數組合並不多，因此我們選擇以預測結果對於 revenue label 的 Mean Absolute Error 作為 validation score，並利用 random search 從 200 多組參數中抽出 50 組，挑選其中 validation score 最高者作為最佳參數。

以下為參數選擇的範圍以及最後獲選的參數 (紅色部分)：

	L1 regularization	L2 regularization
adr model	alpha = [1, 1e-1, 1e-2, 1e-3]	alpha = [1, 1e-1, 1e-2, 1e-3]
is_canceled model	C = [1, 1e+1, 1e+2, 1e+3]	C = [1, 1e+1, 1e+2, 1e+3]

以下為模型的表現。值得注意的是在搜出參數之後，我們會將所有資料搭配該參數組合重新訓練模型，再預測 test data 上繳評分系統。

Valid Score	Public Test Score	Private Test Score
0.56	0.407895	0.441558

Random Forest

第二個模型，我們使用 Random Forest。這項演算法運用 Bagging 技巧來滿足 tree diversity，其中不同 trees 之間彼此獨立，並且每個 trees 都需要擁有較高的準確率，最後再透過 Uniform Blending 消弭模型預測時可能產生的 variance。和 Linear Models 不同的是，這裡只使用預測模型，並沒有額外的 feature selection model。而針對 adr 的預測我們使用 Random Forest Regressor，並依 Mean Square Error 評估 branching 的品質；針對 is_canceled 的預測則用 Random F

orest Classifier，並依 Entropy 評估 branching 的品質。這裡一樣使用 Scikit-Learn (0.24.0) 建模 (參考連結: Regressor、Classifier)。

在參數選擇上，我們決定兩個模型皆使用 300 棵 decision trees，並嘗試調整不同 feature sampling 的個數以及 regularization 的強度，而 random state 固定為 0，其他參數則維持預設。不選擇更多 trees 的原因是因為我們的資料量夠大，在使用較少 trees 的情況下也能訓練出表現不錯的模型，我們也發現當使用更多 decision trees 時會徒增訓練時間，成績卻沒有明顯進步。

由於這裡的總參數組合較多，因此我們決定對兩個模型分別調參；預測結果對於 adr label 的 Mean Absolute Error 將成為 adr model 的 validation score，而預測結果對於 is_canceled label 的 Accuracy 則是 is_canceled model 的 validation score。對於兩者，我們皆利用 random search 從 100 多組參數中抽出 50 組，挑選其中 validation score 最高者作為最佳參數。

以下為參數選擇的範圍以及最後獲選的參數 (紅色部分)：

	Feature Sampling	Regularization
adr model	max_feature = ["Null", "sqrt"]	max_depth = [60, 70, 80, 90, 100, None] min_samples_leaf = [2, 3, 5, 8, 10, 1e-4, 2e-4, 5e-4, 1e-3]
is_canceled model	max_feature = ["Null", "sqrt"]	max_depth = [8, 10, 20, 40, 60, 80, 100, None] min_samples_leaf = [2, 3, 5, 8, 1e-4, 2e-4, 5e-4, 1e-3, 2e-3, 5e-3]

以下為模型的表現。值得注意的是在搜出參數之後，我們會將所有資料搭配該參數組合重新訓練模型，再預測 test data 上繳評分系統。

Valid Score	Public Test Score	Private Test Score
0.45	0.381579	0.428571

XGBoost

第三個模型，我們使用 XGBoost。這項演算法主要延續了傳統 Gradient Boosting Machine (GBM) 的框架，其中較新 tree 會根據較舊 tree 的表現加以改進，藉此降低最終模型預測時的 bias，同時也要求每個 tree 都不要太強，最後會根據每個 tree 的準確率計算出 Linear Blending 的權重，以減少模型在預測時可能產生的 variance。除此之外 ，XGBoost 還利用了許多技巧精進預測表現，例如多考慮了 error function 的二次微分、新增 L1 和 L2-regularization、參考 Random Forest 的 bootstrapping 和 feature subsampling 等等。

在這裡，我們只使用預測模型，並沒有額外的 feature selection model。而針對 adr 的預測我們使用 Regression 搭配 Squared Error；針對 is_canceled 的預測則用 Binary Logistic Regression。在這裡我們使用 XGBoost (1.31.0) 建模，為了和其他兩個模型配合，選用其 Scikit-Learn API (參考連結: XGBoost python)。

在參數選擇上，我們首先將兩個模型的 tree method 皆固定為 GPU 版本的 histogram-based algorithm，該想法最初由 LightGBM 提出，會將連續型 feature 分裝成 discrete bins，在不傷及模型精確度下加速處理資料的速度，同時也減少占用的記憶體。此外，我們為兩個模型準備的 trees 並不多，原因和我們對於 Random Forest 的看法相同；其中我們用 200 棵 trees 決定 adr model，用 250 棵 trees 決定 is_canceled model，兩者的差異主要是因為我們發現 is_canceled 的任務比較簡單，較容易發生 overfitting，因此給予其較多 trees。

至於其他參數的調整，我們主要討論了不同的 learning rate、regularization 強度以及 sampling 比例，並且將 random state 設為 0，其餘參數則維持預設。

同樣地，由於這裡的總參數組合較多，因此我們決定對兩個模型分別調參，validation score 的算法與 Random Forest 相同。和其他模型不同的是，對於 XGBoost 我們想要調整的參數種類與總組合數眾多 (約 100 萬組)，單用 random search 調參的效率可能不佳，因此我們選擇將參數依特性分群後進行漸進式調參 (參考連結: xgb tuning)，過程如下：

1. 為每一個參數設定預設值
2. 固定所有參數，利用 grid search 暴搜第一組參數，找出 validation score 最高者作為最佳參數值
3. 更新並固定第一組的最佳參數值，利用 grid search 暴搜第二組參數，找出 validation score 最高者作為最佳參數值
4. 依此類推步驟 3 直到最後一組參數調整完畢

依此調參，雖然很容易卡在 local optimum，但在整個過程中可以發現 validation score 只會變好不會變差，且只需考慮大約 100 組參數。

以下為調參順序、參數選擇的範圍以及最後獲選的參數 (紅色部分)：

	1. Learning rate →	2. Regularization (Tree structure) →
adr model	learning_rate = [2e-1, 1e-1, 8e-2, 5e-2, 2e-2, 1e-1]	min_child_weight (default: 1) = [2, 5, 8, 10, 12, 15, 18, 20] max_depth (default: 6) = [2, 5, 6, 8, 10, 12]
is_canceled model	learning_rate = [2e-1, 1e-1, 8e-2, 5e-2, 2e-2, 1e-1]	min_child_weight (default: 1) = [8, 10, 12, 15, 18, 20] max_depth (default: 6) = [2, 3, 5, 6, 8, 10]

→	3. Regularization (Error reduction) →	4. Sampling →
adr model	gamma (default: 0) = [0, 1, 2, 3, 5, 8, 10, 15]	subsample (default: 0.8) = [0.6, 0.7, 0.8, 0.9, 1.0] colsample_bytree (default: 0.8) = [0.6, 0.7, 0.8, 0.9, 1.0]
is_canceled model	gamma (default: 0) = [0, 1, 2, 3, 5, 8, 10, 15]	subsample (default: 0.8) = [0.6, 0.7, 0.8, 0.9, 1.0] colsample_bytree (default: 0.8) = [0.6, 0.7, 0.8, 0.9, 1.0]

→	5. Regularization (Error terms)
adr model	reg_lambda (default: 1) = [1, 1e-1, 1e-2, 1e-3] reg_alpha (default: 0)) = [1, 1e-1, 1e-2, 1e-3, 0]
is_canceled model	reg_lambda (default: 1) = [1, 1e-1, 1e-2, 1e-3] reg_alpha (default: 0)) = [1, 1e-1, 1e-2, 1e-3, 0]

以下為模型的表現。值得注意的是在搜出參數之後，我們會將所有資料搭配該參數組合重新訓練模型，再預測 test data 上繳評分系統。

Valid Score	Public Test Score	Private Test Score
0.45	0.355263	0.402597

Comparison between models

1. Performance

   三個模型的表現統整如下：

Models	Valid Score	Public Test Score	Private Test Score
Linear	0.56	0.407895	0.441558
Random Forest	0.45	0.381579	0.428571
XGBoost	0.45	0.355263	0.402597

   首先，我們發現 Linear Models 的表現明顯比較遜色，可能是因為我們給予模型的 feature matrix 偏稀疏且資料沒有經過太多的處理，加上 Linear Models 架構簡單，對抗 overfitting 的機制又僅有 error 中的 regularization term，因此表現不如其他兩者好。接著比較 Random Forest 和 XGBoost，這兩個方法皆屬於 tree-based ensemble models，預測效果明顯比 Linear Models 還好。此外，根據先前模型簡介中的敘述，我們可以發現 XGBoost 在對抗 bias 時擁有較細膩的處理機制，但這也使得 XGBoost 的模型複雜度較高，因此我們有格外注意 XGBoost 的調參，盡量避免 overfitting 發生，而最後結果顯示，我們的 XGBoost 優於 Random Forest，可以推得我們的參數確實有讓 XGBoost 發揮不錯的作用。

2. Efficiency

   每個模型的訓練時間可能會因參數的不同而有所長短：對 Linear Models 來說，regularization 的強度越低會需要越久的收斂時程；而對 Random Forest 和 XGBoost 來說，trees 較多或較深、 pruning 較少等情況會需要較長的訓練時間。

   為了觀察不同模型之間的效率表現，我們將每個模型在不同參數所需的訓練時間加總後平均，如下表所示：

   (這裡所有的實驗皆使用台大資工系工作站，可能會因為不同的壅塞情形而有所誤差)

Models	adr model training time	is_canceled model training time	device
Linear Models	764 sec		CPU
Random Forest	759 sec	804 sec	CPU with n_jobs=4
XGBoost	120 sec	59 sec	GPU

   觀察得知，模型訓練效率的排序為 XGBoost > Linear Models > Random Forest。

其中，我們想討論一下兩個 tree-based ensemble models：在 Random Forest 中，不同 trees 之間是獨立的，因此可以在同一時間平行建立許多 trees；而在 XGBoost 中，較新的 tree 需要等候較舊的 tree，因此同一時間僅能處理一棵 tree，雖然建立的過程也可以平行化處理，但照理來說速度還是會比 Random Forest 慢上許多。然而，實驗結果並非如此，我們發現雖然 XGBoost 一次僅能建立一棵 tree，但它要求每棵 tree 的表現不能太好，反觀 Random Forest 需要每棵 tree 的準確率都達一定水準，因此 XGBoost 中單一 tree 的深度通常會比 Random Forest 淺，進而導致單一 tree 的訓練時程較短；此外，我們的 XGBoost 使用了 GPU 版本的 histogram-based tree method，更是大幅提升了模型訓練的效率。

3. Scalability

   我們將以兩個角度切入討論模型的使用彈性：

   a. 以參數的可適性來說，我們認為 Linear Models > Random Forest > XGBoost。主要是因為 Linear Models 所需調整的參數較為單純，而 tree-based models 在面對不同大小的資料集所需的參數範圍可能大不相同 (例如面對大資料集需要很深的 tree 等等)，因此需要花更多時間進行參數的探勘和調整。此外， XGBoost 的演算法設計使其較容易受到 noise 影響，因此比起 Random Forest 多了不少防止 overfitting 的機制，導致調參的難度和複雜程度增加。

   b. 以 feature preprocessing 的費力程度來說，我們認為 Random Forest 和 XGBoost 的彈性會優於 Linear Models。原因和我們列在 Appendix 的結論有關，若 Linear Models 的結果要能追上其他的模型，可能需要整理一下 feature matrix，因此在 feature preprocessing 上較費工夫。

4. Popularity

   目前這三個模型都有相當多人在用，但就以 tabular data 作為資料集的比賽來說，排行榜上前幾名的大神幾乎都有使用 ensemble，其中 Gradient Boosting 系列 (例如 XGBoost、LightGBM 等) 略勝一籌， 因此我們認為這三個模型的排名應為 XGBoost > Random Forest > Linear Models。

5. Interpretibility

   我們認為 Linear Models 的可解釋性優於其他兩者。因為 Linear Models 並非 ensemble，我們可以列出所有 features 所對應到的權重，在得知 features 的重要性之餘，也很容易理解模型是如何計算出答案的；至於其他兩個 tree-based ensemble models，我們雖然可以根據每個 features 在建立 trees 時被使用的頻率推估其重要性，但由於 trees 的數目過多，我們很難觀察模型究竟是如何做出決策，因此可解釋性不如 Linear Models。

Best Model

根據上述的比較，我們最後以 XGBoost 作為首推的模型，主要是因為其成績表現優於其他兩者，也是目前最知名的演算法之一，加上訓練速度快，讓我們可以嘗試更多的調參與實驗。然而，這樣的選擇必須犧牲更多調參的人力成本，並且 overfitting 的風險較大，我們也無法明確解釋模型是如何預測結果的，需承擔模型一定程度的不確定性。

Final selection

Single-type model

我們發現 validation score 表現好的在 test score 上不會表現太差，且 valid score 表現極差的在 test score 上也會很糟，因此可以推得 validation score 有一定的參考價值。然而，畢竟 validation data 的發生時間還是和 test data 不同，單看 validation score 還是有些不安，因此我們最後挑選 validaion score 不差且 public test score 表現不錯者作為每個模型的最佳預測，並指定 XGBoost 的預測為 final selection 之一 (public test score = 0.355263、private test score = 0.402597)。

Blending

我們想看如果拿三個模型最佳預測的 revenue labels 進行 Uniform Blending，是否能夠有更好的成績？我們將上述三個模型所預測出的 labels 平均並四捨五入作為新的 revenue labels，得到 pubilc test score = 0.315789，是我們所有 public test score 中的最高，因此被指定為另一個 final selection。而這個預測的 private test score = 0.363636，成為我們最終的比賽成績。

這個成績比個別三個模型預測都來得好，我們認為可能的原因是從成績來看，我們的模型不管是 Linear Models、Random Forest、XGBoost 的 Mean Absolute Error 都非常接近且滿小的，也就是說三個模型在預測 label 時可能僅與 true label

有 ± 1 之差，因此若我們在這三個模型分別預測的 label 上做 Uniform Blending 並四捨五入時，那些與 true label 有些微之差的 label 變得夠答對，使 Mean Absolute Error 變小。

綜上述，Uniform Blending on revenue 的優點為只要不同的模型間有 diversity ，其表現有機會比個別的模型還要好，然而其缺點是就我們的設計沒辦法利用 validation set 驗證結果的好壞，只能從 public / private test 上看結果，因此會有較多的不確定性。

Appendix

Feature Selection for Agent and Country

在比賽結束後，我們又想到一個能夠提升 feature 質量的方法。當我們做完 feature preprocessing 後，會發現整個 feature matrix 變得非常稀疏，其中一個原因是因為 agent 與 country 有非常多種類，分別有 334 與 353 種，且其中某些種類僅含少數資料。因此，我們決定觀察 agent 和 country 中每個種類跟 adr label 與跟 is_canceled label 的關連性，先抽出他們兩者的 one-hot features，並計算他們與 adr label 跟與 is_canceled label 的 mutual information，接著將小於 0.0001 的 agent 與 country one-hot features 拿掉。

在篩選完之後，用來訓練 adr 模型的 feature matrix 會減少 212 個 features，而 is_canceled 的部分則會刪去 223 個 features。我們拿這些較小的 feature matrices 重新為每個模型找出最佳參數後，會發現 Linear Models 有明顯地進步，而 tree-based models 會些微退步 (Random Forest) 或大致持平 (XGBoost)：

Linear Models	Valid Score	Public Test Score	Private Test Score
Original	0.56	0.407895	0.441558
Drop	0.49	0.368421	0.428571

Random Forest	Valid Score	Public Test Score	Private Test Score
Original	0.45	0.381579	0.428571
Drop	0.44	0.407895	0.454545

Xgboost	Valid Score	Public Test Score	Private Test Score
Original	0.45	0.355263	0.402597
Drop	0.44	0.381579	0.389610

整體而言，較精實的 feature matrix 確實可以提升模型的表現，尤其是 Linear Models，但也許是這邊提出的 feature selection 方法是只單看個別 feature 就決定其重要性的 univariate selection，那些被丟去的 features 可能在和其他 features 結合後是存在利用價值的，才會使得某些模型沒有太大的進步。

未來，我們期許自己嘗試利用每個模型自帶的 multivariate feature selection 功能來篩選像 agent 和 country 這樣的種類，方法可能為先建一個簡易的架構 (例如：較少的 trees) 篩選掉不重要的種類後，再建立較大的架構 (例如：較多的 trees) 訓練最終模型等等，希望能更加精進我們的表現。

Work Loads

學號	姓名	分工比重
B06703094	馬愷若	50%
B06902125	黃柏瑋	50%

References

Validation XGB tree method XGB v.s. RF (1) XGB v.s. RF (2)

Lasso Ridge LogisticRegression RFRegressor RFClassifier XGBoost python xgb tuning