機器學習期末報告

一、競賽資訊介紹

題目：Airbnb New User Bookings
目標：以使用者瀏覽網站時的動作以及使用者的資料來預測訂購的目的地
資料集：
(1) age_gender_bkts.csv：關於年齡分佈的統計
(2) countries.csv：目的地的資訊（精度、緯度等等）
(3) sample_submission_NDF.csv：示範上傳檔案，預設為全部 NDF
(4) sessions.csv：使用者瀏覽網站時的 sessions，例如：增加 wishlist 等等動作
(5) test_users.csv：測試資料集
(6) train_users_2.csv：訓練資料集
由於本次測試資料集已經含有年齡，因此不採用(1)年齡資料集，而測試資料集也沒有目的地資訊因此不採用(2)目的地資料集
評估方式：該競賽採用 NDCG (Normalized discounted cumulative gain) @k where k=5 的方式進行評分，也就是一個 id 可以預測五個答案，得分依據猜對的順位有著高到低的分數。

二、資料前處理

(1) 替換低筆數資料成"other"

在這 27 萬多筆的資料中，有許多的值都只占非常少部分的筆數，導致之後取 one-hot-encoding 時產生太多欄位且有的幾乎全部都是 0，因此我們寫了兩個 function：(a) replace_by_count (b) replace_by_prct，分別代表在整體資料集中根據筆數、佔的比例來做替換，只要低於我們定義的 threshold(數量或比例)，就將其全部轉換為"other"，例：first_browser 中"weibo"只有 3 筆，低於閥值，因此全部轉換為"other"，轉換欄位如下：affiliate_channel、affiliate_provider、first_browser、language、action_detail；我們測試了非常多不同組合的 threshold 以降低欄位中的類別數，在結果測試表中會一一敘述。

(2) 合併資料

```
def replace_by_count(df, columns, threshold):
    dict_={}
    remove_col = []
    for i,j in dict(df[columns].value_counts()).items():
        if j > threshold:
            dict_[i]=i
        else:
            remove_col.append(i)
            dict_[i]='other'
    df[columns] = df[columns].map(dict_)
    print(remove_col)
    print(len(remove_col))

def replace_by_prct(df, columns, threshold):
    dict_={}
    remove_col = []
    for i,j in dict(df[columns].value_counts()/len(df)).items():
        if j > threshold:
            dict_[i]=i
        else:
            remove_col.append(i)
            dict_[i]='other'
    df[columns] = df[columns].map(dict_)
    print(remove_col)
    print(len(remove_col))
```

```
# replace low propotion features to "other"

#affiliate_channel
replace_by_prct(all_data,'affiliate_channel',0.01)

#affiliate_provider
replace_by_prct(all_data,'affiliate_provider',0.01)

#first_browser
replace_by_prct(all_data,'first_browser',0.01)

#language
replace_by_prct(all_data,'language',0.01)
```

我們首先將 train 資料集除 label 以外的欄位跟 test 資料集進行列合併，目的在於一起處理時若取 one-hot-encoding 的時候可以保證資料欄位數會一致，以及 sessions 資料集根據使用者"id"將資料 left join 至 train-test 資料當中，由於一個 id 會對應多筆 sessions 資料，因此我們會先根據 id 做 groupby 之後再做合併，而 groupby 的方法使用兩種(1) mean(2) count，在考量停留時間等等特徵可能根據想去的地方有所不同，及點擊換頁次數也與使用者特定想去的目的地或有所差異，因此我們採用兩種方法來 groupby 而後以使用者 id 合併到 train-test 資料中。

(3) 日期資料轉換

```
mean_sess = sessions.groupby(by=['user_id','action_detail']).mean().reset_index().pivot(index='user_id',columns='action_detail',values='secs_elapsed').re
all_data = pd.merge(all_data, mean_sess, left_on='id', right_on ='user_id',how='left')
del mean_sess

mean_sess1 = sessions.groupby(by=['user_id','action_detail']).count().reset_index().pivot(index='user_id',columns='action_detail',values='secs_elapsed').r
all_data = pd.merge(all_data, mean_sess1, left_on='id', right_on ='user_id',how='left')
del mean_sess1
```

由於資料中的 timestamp_first_active、date_account_created 有包含年月日且並沒有切分乾淨，因此將這兩個欄位年月日分別切開，建置六個新的欄位，並且我們希望得知創建日到活動日的時間差異與目的地的選擇會不會有關係，因此以這兩個欄位差的秒數及年份作為新的欄位 diff_sec 與 diff_year。

(4) 年齡資料轉換

```
all_data['active_year'] = all_data['timestamp_first_active'].astype(str).str.slice(0,4)
all_data['active_month'] = all_data['timestamp_first_active'].astype(str).str.slice(4,6)
all_data['active_day'] = all_data['timestamp_first_active'].astype(str).str.slice(6,8)
all_data = all_data.drop(['timestamp_first_active'], axis=1)

all_data['created_year'] = all_data['date_account_created'].astype(str).str.slice(0,4)
all_data['created_month'] = all_data['date_account_created'].astype(str).str.slice(5,7)
all_data['created_day'] = all_data['date_account_created'].astype(str).str.slice(8,10)
all_data = all_data.drop(['date_account_created'], axis=1)
```

```
def diff_days(row):
    # time.mktime(datetime.datetime.strptime(s, "%d-%m-%Y").timetuple())
    active = row.active_year + '-' + row.active_month + '-' + row.active_day
    create = row.created_year + '-' + row.created_month + '-' + row.created_day
    active = time.mktime(datetime.datetime.strptime(active, "%Y-%m-%d").timetuple())
    create = time.mktime(datetime.datetime.strptime(create, "%Y-%m-%d").timetuple())
    diff = active - create
    return diff
```

```
all_data['diff_sec'] = all_data.apply(diff_days, 1)
```

經過觀察發現有些人的年齡有異常值，例如：會出現 2014、1950、180 這種較為不正常的年齡，我們推測填寫 19XX 的使用者是填寫出生年，因此我們以這些使用者訂房的年做相減推測使用者的年齡，倘若無訂房日期則根據活動日期做推測，而如果是大於 2000 或是在 200~1900 之間，則會將年齡轉換為-1，代表異常類別；考量有些使用者填寫日期時 可能不會填寫真實

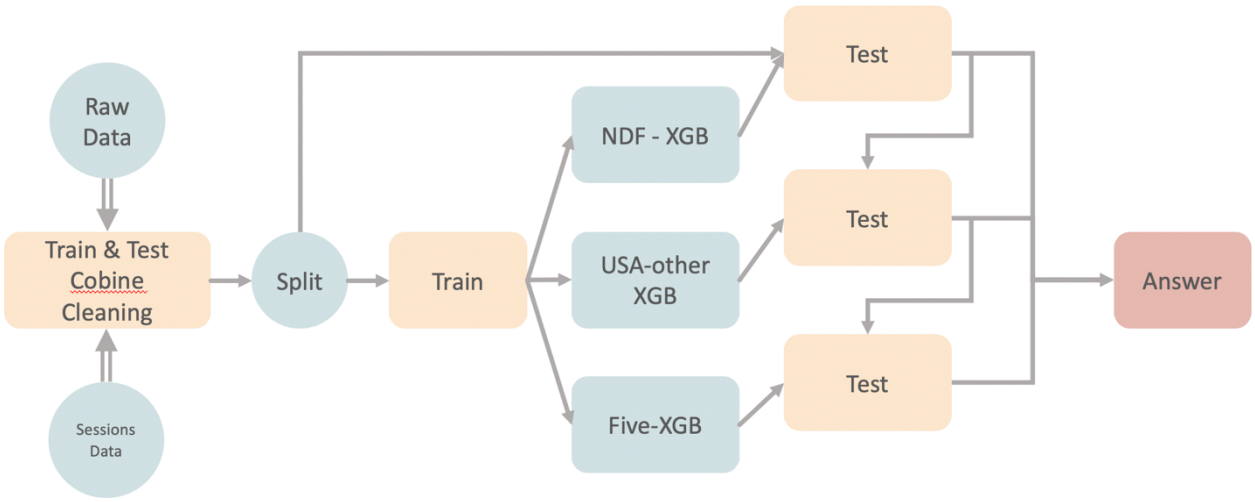
年齡，而是填寫最近整數，例如：24 歲的使用者有些會直接填寫 25 歲，因此我們將年齡做 bining 換成類別型。

```
def age_deal(row):
    if row['age'] < 200:
        return row['age']
    if row['age'] > 2000:
        return -1
    if (row['age'] > 1000) and (row['age'] < 2000):
        date = row['created_year']
        age = float(date) - row['age']
        return age
    else:
        return -1
all_data['age_trans'] = all_data.apply(age_deal, 1)

bins = [-5,0] + list(range(20,101,5)) + [200]
label = list(range(1, len(bins)))
all_data['bining'] = pd.cut(all_data['age_trans'], bins=bins, labels=label)
```

三、模型設計及訓練

(1) 流程圖



由於答案可以填五個，越高順位答對分數越高，我們只要盡量將答案佔比最高的 NDF 在第一順位就預測正確，即可拿下高分，其他答案亦是相同道理，基於這樣的計分方式我們使用的模型是以三個 XGBoost 堆疊而成的階層式模型(Hierachial XGBoosting)，這麼做的目的是由於資料分布相當不平均的關係，因此我們採用類似 SVM 多類別中 one-against-all 的概念，在前兩個模型中將除了目標類別以外其他都當作是第二類(train_label 所示)，簡單來說，我們的第一個分類器(NDF-XGB)為 NDF 與其他的二分類如下（圖一），第二個(US-other-XGB)為僅使用 US 與 NDF 之外的其他 label 做二分類訓練，如下（圖二），第三個(Five-XGB)為剩餘五個佔比最高類別的分類([FR,IT,GB,ES,CA])如下（圖三），這個方法刪除了後四個答案比例非常低的訓練類別，目的在於降低所需訓練的分類數量以及減少高順位位置答案卻去猜

比例相當低的類別的風險，並且最重要的是能在每次訓練時保持沒有那麼嚴重資料偏斜。

模型: NDF-XGB(二分類)
NDF: 47213, Not NDF: 29217
圖(一):

id	features	label	train_label
1	…	NDF	NDF
2	…	US	Not NDF
3	…	other	Not NDF
4	…	FR	Not NDF
…	…	…	…

模型: USA-other-XGB(二分類)
US: 20419, other 8798
圖(二):

id	features	label	train_label
1	…	NDF	不訓練此筆
2	…	US	USA
3	…	other	other
4	…	FR	other
…	…	….	…

模型: Five-XGB(五分類)
FR: 1461, IT: 994, GB: 741, ES: 713, CA: 450
圖(三):

id	features	label	train_label
1	…	NDF	不訓練此筆
2	…	US	不訓練此筆
3	…	other	不訓練此筆
4	…	FR	FR
5	…	IT	IT
…	…	….	…

圖中 train_label 即代表我們經過轉換後輸入模型的訓練 label，除了依此方式篩選資料外，我們也只採用 sessions 欄位非空值的資料做訓練，從實驗結果得知，只取 sessions 欄位非空值資料做訓練的準確率比全部都去訓練的準確率還要高，原因在於 sessions 為 2014 之後才開始紀錄，而測試資料則全部都是 2014 之後的，此部分在最後測試結果會有數值做說明。

(2) 輸入資料

以上述之方式處理完 label 的轉換後，我們的訓練資料以 8:2 的比例切割成 trainning set 以及 validation set，資料筆數分別如下:

NDF-XGB		US-other-XGB		Five-XGB	
train	validation	train	validation	train	validation
61144	15286	23373	5844	3487	872

(3) 模型參數

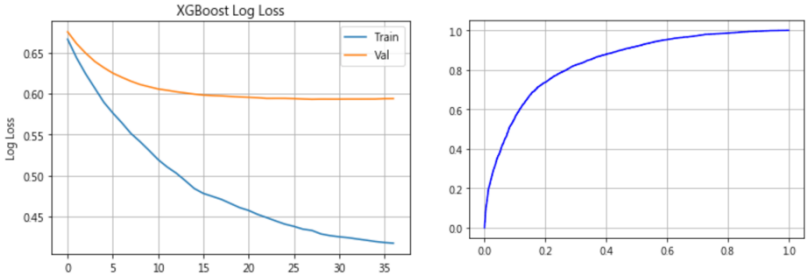
此三個模型中，在訓練過程中最大層數 12 有出現稍微過適的情況，因此皆使用層數為 9 的 XGBoost，在資料使用方面使用了全部的子集，而在建立樹的欄位數選擇了 0.8，並且在觀察 loss 變化後三個模型的 learning rate 分別為 0.1,0.1,0.05，以上是在經過多次的實驗得出來的最好結果，且也有設立 early_stops 參數預防過擬合，更多的參數調整會在結果表上顯示，而我們在預測時也都採用了機率預測以方便自行調整模型的閥值。

(4) 評估模型

```
#change threshold
result_proba = NDF_clf.predict_proba(X_test)
ndf_pred= [1 if i > 0.6 else 0 for i in result_proba[:,1]]

#change threshold
result_proba = US_clf.predict_proba(X_test)
us_pred= [1 if i > 0.63 else 0 for i in result_proba[:,1]]
```

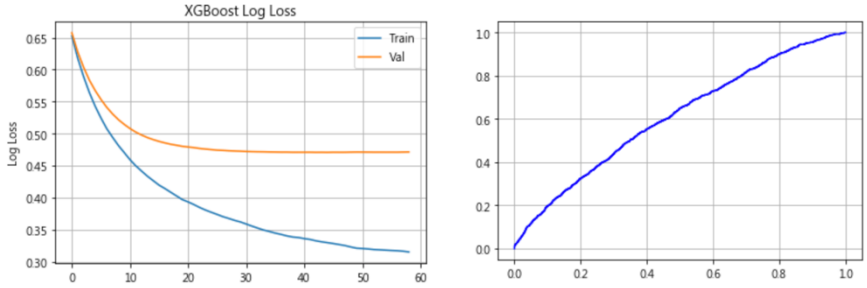
我們使用了 loss 以及 ROC curve 作為評估指標，在 NDF-XGB 中達到了相當不錯的 AUC 面積，詳細訓練結果如下：



a. NDF-XGB

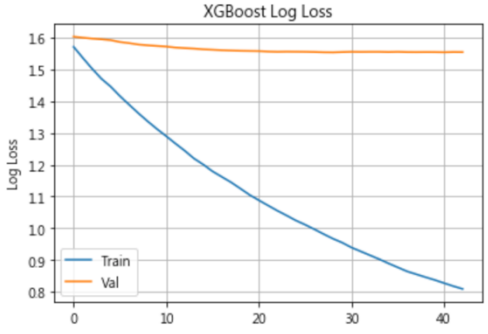
b. US-XGB

可以看到在第二次訓練的 ROC 曲線就表現得沒那麼好，雖然 AUC 依然大於 0，但由於資料量的不足以及特徵不夠明顯，訓練狀況不是那麼好。



c. Five-XGB

由於到了五分類的訓練資料非常少，儘管已經嘗試對於 max_depth 等參數進行挑整預防過擬合，但在五分類的訓練資料過於稀少的情況之下，訓練狀況並不是那麼優異。



(5) 預測結果

我們以階層式的方式即是希望能夠在預測時不以直接預測最高五個機率進行排序，這樣對於資料偏斜的誤差可能更為嚴重，導致預測出來的順位就是訓練資料 Label 的佔比排名，因此我們採用此方法為先以 NDF-XGB 預測是否為 NDF，若 Not-NDF 則將 US-other-XGB 預測出來的結果作為第一順位，但是由於資料偏斜的關係，有大概率答案可能為 NDF，因此我們在這邊設計即便第一順位為 Not-NDF，也設定在第二順位一定會將答案選為 NDF，第三順位再補上 USE-other-XGB 預測出來機率較低的另一個答案，最後以 FIVE-XGB 預測最後兩順位，預測結果與最終答案獲取方式如下：

id	NDF-XGB	US-other-XGB		Five-XGB		最終結果
		高	低	高	低	
1	NDF	US	other	FR	IT	NDF-US-other-FR-IT
2	NDF	other	US	ES	FR	NDF-other-US-ES-FR
3	Not-NDF	US	other	CA	FR	US-NDF-other-CA-FR
4	Not-NDF	other	US	IT	ES	other-NDF-US-IT-ES

四、模型結果

(1) 最佳結果

我們在經過多次調整參數以及多次訓練後，最好的成績為 Private Score 0.88525，差距第一名 0.172 個百分點，並且在 Leardboard 中排行第 130，為銅牌成績，並且高於所有 Kaggle Notebook 中的分享結果。

(2) 模型參數測試表

Model	c	diff	threshold	depth	lr	Sub sample	Col sample	Private Score	Public Score
NDF+US+other	-	-	-	-	-	-	-	0.84832	0.84418
Keras.nn(binary)	-	-	-	-	-	-	-	0.61412	0.59923
Bagging(binary)	-	-	-	-	-	-	-	0.65385	0.64271
LightGBM(binary)	-	-	-	-	0.05	-	-	0.63821	0.62953
AdaBoost(binary)	-	-	-	-	-	-	-	0.66975	0.65812
XGBoost(binary)								0.68411	0.67908
XGBoost(five)	-	-	-	-	-	-	-	0.86146	0.85573
HXGB	-	-	0.6,0.63	9,9,9	.1,.1,.1	0.8	0.6	0.86937	0.86461
HXGB+sessions	-	-	0.6,0.63	9,9,9	.1,.1,.1	0.8	0.6	0.87421	0.86816
HXGB+sessions14	-	-	0.6,0.63	9,9,9	.1,.1,.1	0.8	0.6	0.87431	0.86945
HXGB+sessions14 (with count,mean)	-	-	0.6,0.63	12,12,9	.1,.1,.1	0.8	0.6	0.87690	0.87196
HXGB+sessions14 (with count,mean)	.25	-	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.8	0.88278	0.87649

Model	c	diff	threshold	depth	lr	Sub sample	Col sample	Private Score	Public Score
HXGB+sessions14 (with count,mean)	.35	-	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.8	0.88107	0.87639
HXGB+sessions14 (with count,mean)	.15	Yes	0.6,0.63	12,12,9	.1,.1,.05	1	0.8	0.88192	0.87744
HXGB+sessions14 (with count,mean)	.0025	-	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.6	0.88278	0.87649
HXGB+sessions14 (with count,mean)	.0025	Yes	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.6	0.88385	0.87776
HXGB+sessions14 (with count,mean)	.00005	Yes	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.6	0.88332	0.87852
HXGB+sessions14 (with count,mean)	.00001	Yes	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.6	0.88363	0.87789
HXGB+sessions14 (with count,mean)	.0000025	Yes	0.6,0.63	12,12,9	.1,.1,.05	0.8	0.6	0.88402	0.87830
HXGB+sessions14 (with count,mean)	100	Yes	0.6,0.63	9,9,9	.1,.1,.05	1	0.8	0.88398	0.87878
HXGB+sessions14 (with count,mean)	180	yes	0.6,0.63	9,9,9	.1,.1,.05	1	0.8	0.88384	0.87931
HXGB+sessions14 (with count,sum)	200	yes	0.6,0.63	9,9,9	.1,.1,.05	1	0.8	0.88384	0.87931
HXGB+sessions14 (with count,mean)	100	Yes	0.6,0.63	9,9,9	.1,.1,.05	0.8	0.6	0.88436	0.87855
HXGB+sessions14 (with count,mean)	180	yes	0.6,0.63	9,9,9	.1,.1,.05	0.8	0.6	0.88525	0.87911
HXGB+sessions14 (with count,mean)	200	yes	0.6,0.63	9,9,9	.1,.1,.05	0.8	0.6	0.88461	0.87892

```python
#begging
from sklearn import ensemble, preprocessing, metrics

bag = ensemble.BaggingClassifier(n_estimators = 100)
bag_fit = bag.fit(X_train, y_train)

test_y_predicted = bag.predict(X_test)

ans = pd.DataFrame({'id':test_id,'country':lableE_NDF.inverse_transform(test_y_predicted)})

ans.to_csv('begging.csv')
```

```python
#Adaboost
boost = ensemble.AdaBoostClassifier(n_estimators = 100)
boost_fit = boost.fit(X_train, y_train)

test_y_predicted = boost.predict(X_test)

ans = pd.DataFrame({'id':test_id,'country':lableE_NDF.inverse_transform(test_y_predicted)})

ans.to_csv('Adaboost.csv')
```

```python
import lightgbm as lgb

gbm = lgb.LGBMClassifier(objective='binary',num_leaves=31,learning_rate=0.05,n_estimators=20)
gbm.fit(X_train, y_train,eval_set=[(X_val, y_val)],eval_metric='l1',early_stopping_rounds=5)

y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)

ans = pd.DataFrame({'id':test_id,'country':lableE_NDF.inverse_transform(y_pred)})

ans.to_csv('Light_gbm.csv')
```

```python
model = models.Sequential()
model.add(layers.Dense(124,activation='softmax',input_shape=(10000,)))
model.add(layers.Dense(32,activation='softmax'))
model.add(layers.Dense(16,activation='softmax'))
model.add(layers.Dense(1,activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])

history = model.fit(X_train,y_train,epochs=100,batch_size=512,validation_data=(X_val,y_val))

y_pred = model.predict(X_test)

ans = pd.DataFrame({'id':test_id,'country':lableE_NDF.inverse_transform(y_pred)})

ans.to_csv('nn.csv')
```

此四張截圖左到右上到下分別為 Begging,Adaboost,LightGBM 及 Keras.NN，我們先根據這四個模型與 XGBoost 做{NDF,US}二元分類(binary)，上傳結果後當作挑選模型的基準，由於模型並沒有做非常多的調參因此訓練出來的結果相對的也沒有很好，而 LightGBN 雖然收斂速度相較於 XGBoost 來說較快，但由於資料量沒有到很大所以還是以效能優先，最後選擇了在未調參的情況下分數最高的 XGBoost，在記錄表中的第一個紀錄沒有任何參數的原因是我們直接將每個使用者的答案設為[NDF,US,other]三個比例最高的答案排序上傳計算分數，就有 0.84832 的分數，比這個分數低乾脆直接這樣猜還比較準，因此我們以此當做 baseline 來做訓練模型的比較基準，而後我們使用了 XGBoost 直接做五類別機率預測，發現準確度大幅度上升，並且在這之後就開始創造 Hierachial XGBoost(HXGB)的模型建

構，Model 部分加入 session14 即代表僅使用 14 年之後的資料做訓練，count, mean 則為 groupby sessions 時使用何種方式，而參數部分其中 c 參數代表對 sessions 的值做篩選，以前述所提到的兩種函式分別根據最低比例以及最低筆數作為 theshold 取代為 other，diff 則為有沒有使用活動時間距離創建時間差欄位，後五項為 XGBoost 的參數，以逗點分開分別代表第幾個模型的參數，若只有一個代表皆相同。

此處列出的模型參數測試表僅為節錄，就完整上傳測試資料而言就超過 100 次，在以篩選 session 為唯一更動的變數去進行測試時，我們發現在一定的範圍內當 session 篩選掉的值越少模型就越準確，而該範圍大致位於 50 到 200 之間會呈現比較良好的結果，因此我們最後皆以該範圍去建置模型，進而在最後些微調整參數下(session 篩選 180，HXGB 的深度皆為 9)，得到了 private score 0.88525 的成績。

由於時間的因素我們無法將所有天馬行空的想法付諸行動，以下列出一項或許可行的改善方法：在每一層級的 XGBoost(NDF、US、Five)中，皆以多項曾經測試過且達到一定準確率的參數建立多個模型，再以多數決投票的方式決定每一層級中最終預測之結果，以此應得以訓練出更好的結果。