(1) 姓名: 朱柏綸 系級: 資工 110 學號: F84064014
姓名: 高造擎 系級: 工資 111 學號: H34076021

(2) 競賽敘述與目標

透過機器學習，實作出可以準確預測某顧客未來是否會繼續使用該銀行的模型，是一個 binary classification 的問題。各組會有排行榜的 public 排名作為依據，實際排名則是由 private 決定。

(3) 資料前處理

高造擎:

刪除 RowNumber 和 CustomerId 明顯對預測沒有幫助的欄位後，試圖使用 SelectKBest 和 ExtraTreeClassifier 尋找最有助於分類的欄位，但放入各式模型結果皆不甚理想。因此嘗試其他方法，於使用 RandomForest 時，把所有欄位取組合放入模型，看是否能找到達到最高數值的欄位組合，並應用於其他模型，結果也是差強人意，所以最後決定維持原本欄位，只刪除 RowNumber 和 CustomerId。

朱柏綸:

- 連續型資料: 嘗試做 normalize
- 類別型資料: label encoding / target encoding
- 特徵篩選: Drop customerID, Surname, RowNumber
- Outlier 處理: 使用 IForest 決定 outlier 並移除
- 資料分割: 使用 stratified 的方式，訓練測試集皆維持原本的比例(8:2)
- 資料取樣: 對標記少的那類(佔 20%)做 oversampling，在分割訓練　測試資料集後，隨機複製訓練集中佔少數標記的資料點，增加訓練資料的平衡性

(4) 特徵處理與分析

- Oversampling:

未做 Oversampling

	accuracy	precision	f1_score
RandomForestClassifier()	0.86375	0.785467	0.575974
LGBMClassifier()	0.87175	0.787484	0.617043
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85850	0.884894	0.501699

Oversampling 2000 筆後

	accuracy	precision	f1_score
RandomForestClassifier()	0.86175	0.728967	0.600995
LGBMClassifier()	0.85350	0.646025	0.632992
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.86675	0.838020	0.566285

Oversampling 增加資料平衡度可以讓 f1_score 提升，但同時 precision 會因此下降

- Outlier Clipping:
  沒有去除 Outlier:

	accuracy	precision	f1_score
RandomForestClassifier()	0.86375	0.785467	0.575974
LGBMClassifier()	0.87175	0.787484	0.617043
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85850	0.884894	0.501699

有去除 Outlier(4% of data removed):

	accuracy	precision	f1_score
RandomForestClassifier()	0.86400	0.791563	0.574489
LGBMClassifier()	0.86650	0.769628	0.599718
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85675	0.865242	0.497710

去除 outlier 效果可能不顯著

- Normalization
  沒有做 normalization

	accuracy	precision	f1_score
LogisticRegressionCV()	0.79325	0.186667	0.016363
SVC()	0.79625	0.000000	0.000000
RandomForestClassifier()	0.86375	0.785467	0.575974
LGBMClassifier()	0.87175	0.787484	0.617043
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85850	0.884894	0.501699

有做 normalization

	accuracy	precision	f1_score
LogisticRegressionCV()	0.80775	0.593384	0.273105
SVC()	0.80675	0.930000	0.104108
RandomForestClassifier()	0.86300	0.761720	0.585639
LGBMClassifier()	0.87100	0.783096	0.615289
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85700	0.882179	0.494161

Normalization 對非 tree based 模型較有幫助，如 LogisticRegression 和 SVM 的效果都提升了不少，LGBM 和 RandomForest 則幾乎沒有影響

(5) 預測訓練模型

高造擎:

預測訓練模型:因缺乏經驗，為尋找最適合此問題的演算法，我將上課學過的套件逐一測試，包括:MLP, GaussianNB, KNeighbor, SVC, DecisionTree, RandomForest, Linear Regression, XGB, EasyEnsemble, BalancedRandomForest，觀察 accuracy, precision, f1-score 的數值是否夠佳，而最後只有 ensemble 類型的 RandomForest, XGB, EasyEnsemble, BalancedRandomForest 足夠好，所以針對他們調整參數。調參主要以 gridsearch 設定 max-depth 為 10, n_estimators=250, class_weight='balanced subsample'，以解決 Exited 比例不均的問題。

朱柏綸:

- 最終模型

```
clf1 = XGBClassifier(use_label_encoder=False, learning_rate = 0.025, n_estimators=200)
clf2 = RandomForestClassifier(n_estimators = 350, class_weight={0:85, 1:15}, n_jobs=-1)
clf3 = LGBMClassifier(max_bin=370, num_leaves=60, learning_rate=0.015, class_weight={0:0.75, 1:0.25}, n_estimators=200)
clf4 = LGBMClassifier(max_bin=370, num_leaves=50, learning_rate=0.015, class_weight={0:0.80, 1:0.2}, n_estimators=200)
clf5 = LGBMClassifier(max_bin=370, num_leaves=40, learning_rate=0.015, class_weight={0:0.77, 1:0.23}, n_estimators=200)
clf6 = LGBMClassifier(max_bin=370, num_leaves=70, learning_rate=0.015)

ensemble_clf = VotingClassifier(
    estimators=[('xgb', clf1), ('rfc', clf2), ('lgbm1', clf3), ('lgbm2', clf4), ('lgbm3', clf5), ('lgbm4', clf6)],
    voting='soft',
    weights=[1, 1, 1.1, 1.1, 1, 1]
)
```

主要使用 Xgboost、LGBM 和 RandomForest 組合效果最好。LGBM 可以使用 class_weight，在不平衡資料中給予 class_weight 可以讓判斷準確度提升，使用較多的 estimators 和較大的 max_bin 會提升模型準確度，小的 learning_rate 在足夠的 iteration 也應該可以得到更好的結果。最後使用 voting classifier 與權重做組合分類器，使用機率總和的 soft voting。

(6) 預測結果分析

- 不同模型原始效能

	accuracy	precision	f1_score
KNeighborsClassifier()	0.76100	0.258486	0.137422
LogisticRegressionCV()	0.78700	0.268606	0.038021
RandomForestClassifier()	0.86025	0.770761	0.566215
MLPClassifier()	0.73025	0.094893	0.111987
LGBMClassifier()	0.86725	0.765935	0.606589

LBGM 和 RandomForest 表現明顯大幅超越其他模型

- LGBM 的 class_weight 調整:

	accuracy	precision	f1_score
LGBMClassifier()	0.86350	0.739714	0.603077
LGBMClassifier(class_weight={0: 0.6, 1: 0.4})	0.86075	0.778549	0.564288
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85075	0.820512	0.482687
LGBMClassifier(class_weight={0: 0.9, 1: 0.1})	0.84475	0.830842	0.438672
LGBMClassifier(class_weight={0: 0.95, 1: 0.05})	0.83550	0.810106	0.381823

越大的 class_weight 可以提升 precision，但相反的 f1 score 會跟著下降，較接近分群比例的 class_weight

- LGBM learning rate 調整 [0.2, 0.15, 0.1, 0.05, 0.03, 0.02]

	accuracy	precision	f1_score
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.2)	0.85350	0.819657	0.500311
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.15)	0.85350	0.840864	0.489841
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85475	0.868908	0.486335
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.05)	0.85425	0.902523	0.471225
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.03)	0.84800	0.911966	0.429510
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.02)	0.84425	0.920330	0.402476

越小的 learning rate 也會有越高的 precision 與越低的 f1 score

- LBGM max_bin 調整

	accuracy	precision	f1_score
LGBMClassifier()	0.86025	0.721612	0.598304
LGBMClassifier(max_bin=50)	0.86550	0.744397	0.611167
LGBMClassifier(max_bin=150)	0.86150	0.735985	0.595491
LGBMClassifier(max_bin=250)	0.86300	0.733584	0.605170
LGBMClassifier(max_bin=500)	0.86575	0.747100	0.609768
LGBMClassifier(max_bin=1000)	0.86475	0.741612	0.608759
LGBMClassifier(max_bin=1500)	0.86250	0.728709	0.605229
LGBMClassifier(max_bin=2000)	0.86200	0.732752	0.599723
LGBMClassifier(max_bin=3000)	0.86450	0.741339	0.607429
LGBMClassifier(max_bin=5000)	0.86125	0.731995	0.596801

結果差異不大

- LBGM min_gain_to_split 調整

	accuracy	precision	f1_score
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, min_gain_to_split=15)	0.81550	1.000000	0.172182
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, min_gain_to_split=10)	0.83000	0.947203	0.295733
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, min_split_gain=5)	0.84200	0.923384	0.385094
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, min_split_gain=3)	0.84300	0.924860	0.391043
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, min_split_gain=1)	0.84825	0.912011	0.430469
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85950	0.867708	0.513887

透過控制 min_gain_to_split 可以控制得到更高的 precision

- LGBM n_estimator 調整

	accuracy	precision	f1_score
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, n_estimators=50)	0.85200	0.892937	0.460142
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85725	0.882098	0.495168
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, n_estimators=200)	0.86000	0.858095	0.520844
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, n_estimators=500)	0.86500	0.824267	0.562926
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, n_estimators=1000)	0.86225	0.778662	0.571190
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, n_estimators=2000)	0.85925	0.744302	0.576024

estimator 的數量多 precision 會下降 f1 score 會上升，f1 score 的上升趨緩的很快速，但 precision 則是繼續大幅下降，太多 estimators 對預測未必是好事

- 調整模型

	accuracy	precision	f1_score
LGBMClassifier()	0.85825	0.737541	0.578256
LGBMClassifier(learning_rate=0.03, max_bin=1000, n_estimators=200)	0.86225	0.761745	0.583789
LGBMClassifier(learning_rate=0.015, max_bin=1000, n_estimators=200)	0.86150	0.785092	0.565203
LGBMClassifier(class_weight={0: 0.8, 1: 0.2})	0.85475	0.857556	0.491662
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.02, max_bin=1000,\n n_estimators=200)	0.84875	0.893394	0.441162
LGBMClassifier(class_weight={0: 0.8, 1: 0.2}, learning_rate=0.017, max_bin=1000,\n n_estimators=200)	0.84700	0.898836	0.427971

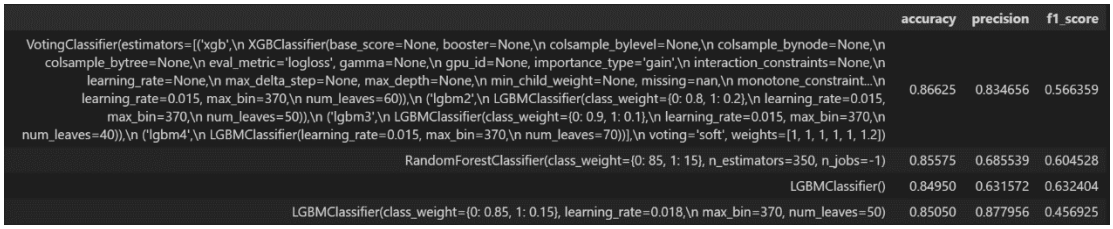
調整參數能小幅增進效果

- Ensemble 多個 classifier

	accuracy	precision	f1_score
VotingClassifier(estimators=[('xgb',\n XGBClassifier(base_score=None, booster=None,\n colsample_bylevel=None,\n colsample_bynode=None,\n colsample_bytree=None,\n eval_metric='logloss', gamma=None,\n gpu_id=None, importance_type='gain',\n interaction_constraints=None,\n learning_rate=None,\n max_delta_step=None, max_depth=None,\n min_child_weight=None, missing=nan,\n monotone_constraint...\n learning_rate=0.015, max_bin=370,\n num_leaves=60)),\n ('lgbm2',\n LGBMClassifier(class_weight={0: 0.8, 1: 0.2},\n learning_rate=0.015, max_bin=370,\n num_leaves=50)),\n ('lgbm3',\n LGBMClassifier(class_weight={0: 0.9, 1: 0.1},\n learning_rate=0.015, max_bin=370,\n num_leaves=40)),\n ('lgbm4',\n LGBMClassifier(learning_rate=0.015, max_bin=370,\n num_leaves=70))],\n voting='soft', weights=[1, 1, 1, 1, 1, 1.2])	0.85325	0.898941	0.466507
RandomForestClassifier(class_weight={0: 85, 1: 15}, n_estimators=350, n_jobs=-1)	0.86100	0.757796	0.578947
LGBMClassifier()	0.85825	0.737541	0.578256
LGBMClassifier(class_weight={0: 0.85, 1: 0.15}, learning_rate=0.018,\n max_bin=370, num_leaves=50)	0.83550	0.920004	0.344396

Ensemble 後的模型可以在 precision 和 f1score 之間做平衡並達到整體上更好的效果，也可以透過調整各個模型的權重來控制 precision 與 f1score

- 透過 oversampling 調整訓練結果



	accuracy	precision	f1_score
VotingClassifier(estimators=[('xgb',\n XGBClassifier(base_score=None, booster=None,\n colsample_bylevel=None,\n colsample_bynode=None,\n colsample_bytree=None,\n eval_metric='logloss', gamma=None,\n gpu_id=None, importance_type='gain',\n interaction_constraints=None,\n learning_rate=None,\n max_delta_step=None, max_depth=None,\n min_child_weight=None, missing=nan,\n monotone_constraint...\n learning_rate=0.015, max_bin=370,\n num_leaves=60)),\n ('lgbm2',\n LGBMClassifier(class_weight={0: 0.8, 1: 0.2},\n learning_rate=0.015, max_bin=370,\n num_leaves=50)),\n ('lgbm3',\n LGBMClassifier(class_weight={0: 0.9, 1: 0.1},\n learning_rate=0.015, max_bin=370,\n num_leaves=40)),\n ('lgbm4',\n LGBMClassifier(learning_rate=0.015, max_bin=370,\n num_leaves=70))],\n voting='soft', weights=[1, 1, 1, 1, 1, 1.2])	0.86625	0.834656	0.566359
RandomForestClassifier(class_weight={0: 85, 1: 15}, n_estimators=350, n_jobs=-1)	0.85575	0.685539	0.604528
LGBMClassifier()	0.84950	0.631572	0.632404
LGBMClassifier(class_weight={0: 0.85, 1: 0.15}, learning_rate=0.018,\n max_bin=370, num_leaves=50)	0.85050	0.877956	0.456925

也可以透過在訓練集 oversampling 犧牲一點 precision 拿到更高的 f1 score

(7) 感想與心得

高造擎:

我從競賽學習到，上課內容與實際解決問題的差距真的很大，上課藉由二維圖形，可以很容易知道甚麼模型適用於問題，但實際問題的欄位通常都會多到無法馬上知道，所以只能透過觀察和反覆嘗試才能有所進展。我一開始就花費太多時間，盲目的測試我知道的所有模型，而沒有發現 Exited 欄位的比例差很多。另外，我花費最多時間在研究演算法的參數，很多參數都不知道其意義，即使知道也不清楚設定多少可以達到最佳效果，所以只能一直排列組合，看能不能有所進步。總結來說，我覺得機器學習真的好玄，一直不斷的嘗試，但結果都不好，即使有變化，也不太清楚原因。我個人覺得，如何從資料處理到最後模型的產出，應該要有系統化的處理方式，這也是我未來需要學習的方向。

朱柏綸:

從競賽中自學到在面對不平衡資料時會遇到的一些問題與處理方法，還有自己寫一些 class 和 function 來讓各個不同模型參數間的效果關係能更快更簡潔的看出來。花最多時間的部分在於模型參數的調整，最後當結果都差不多的時候靠著些微的參數調整調出擠牙膏般的績效差異時在花了不少時間，最困難的地方可能來自資料的不平衡，還有 X 跟 y 的相關性不夠高可能也是讓問題有一些困難。我覺得競賽的網站可以不要每次載入畫面的時候都自動下滑到自己排名所在的那個位置，因為排名太下面每次都要看他在那邊下滑有一點久，可以就停在最上面讓我自己滑下去看我的排名有多低就可以了。

Hw5 git hub

F84064014 朱柏綸

Repo: https://github.com/F84064014/DataScienceHw5

Page: https://f84064014.github.io/DataScienceHw5/hw3/main.html

H34076021 高造擎

Repo: https://github.com/Rob12312368/H34076021_DataScience

Page: https://rob12312368.github.io/H34076021_DataScience/HW3/%E7%B6%B2%E9%A0%81/Tsao-Ching.html