HW5 Report

HW5 H24081163 陳詠翰 B14081027 劉恩兆 H24081341 耿皓昀

作業資料集簡介：下方表格為我們所採用的資料集以及其資訊，包含變數個數、離散型變數個數等等。另外由於 BlastChar 在某一個 column 出現了 11 個空值，因此我們在該資料及移除含有空值的資料。

表 1、資料集描述

	#Features	#Cat	#Num	Size	#Pos	%Pos	Task
Income	14	8	6	32561	7841	24.08%	Binary
Arrhythmia	274	0	274	452	66	14.60%	Binary
Bank	16	9	7	45211	5289	11.70%	Binary
HTRU2	8	0	8	17898	1639	9.16%	Binary
BlastChar	19	16	3	7032	1869	26.57%	Binary
Shoppers	17	2	15	12330	1908	15.47%	Binary
QSAR Bio	41	0	41	1055	356	33.74%	Binary
Shrutime	10	2	8	10000	2037	20.37%	Binary
Spambase	57	0	57	4601	1813	39.40%	Binary
Default	23	9	14	30000	6636	22.12%	Binary

以下將分題描述 HW5 的十個問題，且所有實驗皆以 f1-score 作為評估指標，並使用 5-fold 方式把五次結果平均作為比較。

由於這次作業的程式碼比較多，且為了平均分配運算資源，我們將程式碼分成以下三個檔案，以下分別為檔名及其對應的題目：

- hw5_p_2_5_6 : Problem 2、5、6
- hw5_p3-4 : Problem 3、4
- hw5_p_1_7-10 : Problem 1、7~10

Problem 1

How does feature scaling affect the performance?

這題我們採用上面的 10 個資料集，使用了 logistic regression & lightgbm 兩種模型針對是否有使用 StandardScaler 影響表現進行實驗，並採用 f1-score 當作評估指標，而結果列在下方的表格。

表 2、是否有使用 StandardScaler 的結果

	LR	LR + scale	Lightgbm	Lightgbm+scale
Income	0.3875	**0.6623**	0.7144	**0.7161**
Arrhythmia	**0.5114**	0.4981	0.5990	**0.6200**
Bank	0.3129	**0.4508**	0.5643	**0.5651**
HTRU2	0.8746	**0.8790**	**0.8847**	0.8809
BlastChar	0.5941	**0.5970**	**0.5819**	0.5720
Shoppers	0.4815	**0.5042**	0.6491	**0.6522**
QSAR Bio	0.8036	**0.8130**	**0.8016**	0.8010
Shrutime	0.0992	**0.3138**	0.5875	**0.5965**
Spambase	**0.9029**	0.9027	**0.9481**	0.9450
Default	0.0006	**0.4670**	**0.4782**	**0.4782**

從上方的表格中中我們可以看到對於 Logistic Regression 來說，有經過 scale 的表現是明顯比沒有經過 scale 來的好，雖然有兩個資料集經過 scale 之後效果沒有提升，但也沒有下降太多。至於 LightGBM 的部分，10 個資料集中有 5 個資資料集經過 scale 之後的 f1-score 比較好，有 4 個資料集沒有經過 scale 之後的 f1-score 比較好，因此對於 LightGBM 模型來說是否有 scale 的表現差異不大。

結論：我們覺得不論是什麼模型，經過 scale 之後再套用模型是相對比較好的選擇，因為沒有 scale 可能效果會變很差，但有 scale 之後效果即使沒有提升很多，表現也跟沒有 scale 的表現相差不遠。

Problem 2

When using tree-based algorithms, will using one-hot encoding for categorical features generate worse performance than using label encoding? Why?

由於這題是觀察類別型變數，我們只觀察有類別型變數的 income、bank、blastchar、shoppers、shrutime、default 這六個資料集。針對連續型的變數，我們統一使用 Standard Scaler 進行標準化。模型的部分由於要使用 tree-based 的演算法，因此我們選用 Random Forest 當作模型，參數設定為預設值。以下表格為各個資料集使用 One-Hot Encoding 和 Label Encoding 的表現差異。

表 3、使用 One-Hot Encoding 和 Label Encoding 的差異

Dataset	One-Hot Encoding	Label Encoding
Income	0.6737	**0.6781**
Bank	0.4955	**0.5118**
Blastchar	**0.5487**	0.5485
Shoppers	**0.6493**	0.6458
shrutime	0.5666	**0.5721**
default	0.4697	**0.4718**

從表一可以看出當使用 tree based 演算法時，使用 Label Encoding 的效果在大部分的資料上是比使用 One-Hot Encoding 還好的。老師上課時有提過，當使用 tree based 演算法時，建議不要使用 One-Hot Encoding，因為決策樹的分類方式就是看怎麼樣分類會使 Information Gain 最高，然而當類別型變數都只有 0/1 這兩種數值時，反而會讓決策樹的 Information Gain 變小，進而影響模型效果。加上分類樹會傾向往 0 的方向去生長，因為 1 的值通常比較少，導致分類樹都只會往同一個方向長，且分類出的 region 裡的值也很少，因為 region 裡面通常都為 1。

結論：綜合上述的缺點，Label Encoding 在 tree based 方法的效果就會相對比較好，因為它可以避免資料中出現太多只有 0/1 的變數。而表二的結果也符合此結論。

Problem 3：

Will feature binning provide performance improvement? When does binning be useful (which models or which kinds of datasets)? Which binning methods work better?

在這題中，我們使用了 Equal Width Binning 將連續型變數轉換為離散型變數，並探討了 bins 的個數對不同模型表現的影響，以及未處理連續型變數及使用 StandardScaler() 標準化處理方式的比較。

表 4、 Equal Width Binning 於不同 bins 數量與模型的實驗結果

# of bins	# of bins = 5	# of bins = 10	# of bins = 15	# of bins = 20	# of bins = 50
model / dataset	LightGBM：LGBMClassifier()				
income	0.6592	0.6737	0.6893	**0.6965**	0.6925
arrhy	**0.4669**	0.4019	0.4064	0.4514	0.3371
bank	0.4276	0.4757	0.5024	0.5191	**0.5363**
htru	0.8246	0.8735	**0.8759**	0.8596	0.8610
blastchar	0.5816	**0.5841**	0.5715	0.5779	0.5610
shopper	0.2062	0.4250	0.5131	0.5512	**0.6163**
bio	**0.7625**	0.7538	0.7473	0.7110	0.6072
shrutime	**0.4240**	0.4008	0.4115	0.4013	0.3925
spam	0.6338	0.7799	**0.8009**	0.7847	0.5560
default	0.5464	**0.5843**	0.4723	0.4747	0.4681
average	0.5464	0.5843	0.5991	**0.6027**	0.5628
model / dataset	Logistic Regression：LogisticRegression()				
income	0.6289	0.6455	0.6631	**0.6640**	0.6608
arrhy	**0.4730**	0.4609	0.3946	0.3900	0.4018
bank	0.3572	0.4074	0.4283	0.4429	**0.4486**
htru	0.8140	0.8554	**0.8591**	0.8403	0.8463
blastchar	0.5918	0.5951	0.5928	0.5968	**0.5978**
shopper	0.1738	0.3681	0.4076	0.4419	**0.4843**
bio	0.6885	**0.7408**	0.7393	0.6825	0.6201
shrutime	**0.3104**	0.1944	0.1867	0.1881	0.1909
spam	0.6138	0.7352	**0.7714**	0.7671	0.4169
default	0.4677	**0.4684**	0.4668	0.4676	0.4658
average	0.5119	0.5471	**0.5510**	0.5481	0.5133
Total average	0.5292	0.5657	0.5751	**0.5754**	0.5381

由表 4 可見，使用 Equal Width Binning 後選擇不同的 bins 有不同的表現，整體來說，bins 個數不能過少，若是 bins 個數過少，則可能會漏掉資料中的某些特徵，導致模型表現下滑；反之，若是 bins 個數過多，則會造成每個 bins 中的樣本數量過少，導致資料損失，使模型無法得到足夠學習。此外，bins 數量

過多同時也會造成資料中某些潛在特徵遭到忽略，使模型表現下降。

就實驗結果來說，bins 設置在 15-20 在兩個模型中可以獲得較佳的表現，若是 bins=5，則無法顧慮到特徵的多樣性，使得表現最差；若是將 bins 個數調高至 50，則又會因為 bins 數量過大導致類別中樣本數量不足訓練的情形發生，表現與 bins=50 相差無幾。

而在模型選擇上面，LightGBM 的表現較 Logistic Regression 表現突出，推測是因為使用 Equal Width Binning 後幫助 LightGBM 有效地捕捉到一些重要特徵，從而導致模型表現較突出；另外，也可以推測 Logistic Regression 基於其邏輯斯回歸方程式，模型本身在處理連續型變數表現就會較好，但若使用了 Equal Width Binning 將連續型變數轉換為離散型變數，反而會導致資料的損失，從而使 Logistic Regression 表現下降。

表 5、 Equal Width Binning 於不同 bins 數量與模型的實驗結果

Numerical Data Process Method	**None**	**# of bins = 15**	**StandardScaler()**
model / dataset	**LightGBM：LGBMClassifier()**		
income	**0.7150**	0.6893	0.7147
arrhy	**0.5962**	0.4064	0.5743
bank	0.5610	0.5024	**0.5616**
htru	**0.8840**	0.8759	0.8821
blastchar	**0.5822**	0.5715	0.5726
shopper	0.6542	0.5131	**0.6564**
bio	**0.7953**	0.7473	0.7914
shrutime	**0.5946**	0.4115	0.5929
spam	**0.9444**	0.8009	0.9443
default	0.4800	0.4723	**0.4803**
average	**0.6807**	0.5991	0.6771
model / dataset	**Logistic Regression：LogisticRegression()**		
income	0.3831	**0.6631**	0.6623
arrhy	0.5074	0.3946	**0.4981**
bank	0.3172	0.4283	**0.4508**
Htru	0.8746	0.8591	**0.8749**
blastchar	0.5921	0.5928	**0.5970**

shopper	0.4982	0.4076	**0.5042**
bio	0.8003	0.7393	**0.8130**
shrutime	0.0992	0.1867	**0.3138**
spam	**0.9308**	0.7714	0.9027
default	0.0006	0.4668	**0.4670**
average	0.5004	0.5510	**0.6084**
Total average	0.5906	0.5751	**0.6428**

由表 5 可見，對連續型變數使用 StandardScaler()，可以獲得最佳表現，至於位對連續型變數處理，在 LightGBM 也有不錯的表現，甚至勝過使用 StandardScaler()處理，但在 Logistic Regression 中表現就有大幅滑落。而使用 Equal Width Binning 與其他兩種方法相比，表現皆較為遜色，推測是因為 binning 後每個取值範圍相等，無法有效地捕捉到重要特徵，使表現不盡理想。而我們也針對資料集特性推測 Equal Width Binning 較適合用於存在較強的非線性趨勢的資料中，或是希望捕捉資料中的某些重要特徵，且不需要考慮資料中的較細微的結構，並搭配較簡單的模型，比較可能達到較好表現。

結論：

1. 若是 bins 個數過少，則可能會漏掉資料中的某些特徵，導致模型表現下滑；反之，若是 bins 個數過多，則會造成每個 bins 中的樣本數量過少，導致資料損失，使模型無法得到足夠學習
2. Binning 後較適合 LightGBM，較不適合 Logistic Regression
3. 對連續型變數處理，普遍還是建議使用 StandardScaler 而非 Equal Width Binning

Problem 4：

Compare the performance of 6 different categorical feature encoding methods based on Random Forest, XGBoost, LightGBM, MLP, SVM. Which of the 6 encoding methods work better?

在這題中，我們分別對類別資料嘗試了 6 種不同的 encoding 方法，分別為 One-Hot、Label、Frequency、Target、Beta Target、Leave-One_out encoding，以及分別嘗試了 Random Forest、XGBoost、LightGBM、MLP、Logistic Regression 共五個模型(皆使用預設參數)。其中，對於連續型數值資

料處理，我們統一使用 StandardScaler() 將其標準化。在實驗中，因 Target、Beta Target、Leave-One_out 於 encoding 都需與 y 相關，因此將其擺在 K-fold 迴圈中，分別於 training set 與 testing set 中各自編碼；而 One-Hot、Label、Frequency 此三種方法在編碼與預測值無關的情況下，則是先將資料集編碼後再切分為 training set 與 testing set。

【註：因資料集眾多且資料量龐大，使用 SVM 計算所需要耗費的時長過高，因此使用 Logistic Regression 代替】

表 6、categorical feature encoding methods 於不同模型的實驗結果

encoding method	One-Hot	Label	Frequency	Target	Beta Target	Leave-One-Out
model / dataset	**Random Forest：RandomForestClassifier()**					
income	0.6727	0.6776	0.6796	0.6894	**0.6899**	0.3721
arrhy	**0.6452**	0.6103	0.6331	0.5743	0.5743	0.5743
bank	0.4942	0.5132	**0.5231**	0.4988	0.4988	0.0913
htru	0.8824	0.8847	**0.8865**	0.8853	0.8853	0.8853
blastchar	0.5400	0.5560	0.5509	**0.5851**	**0.5851**	0.2837
shopper	0.6411	0.6429	0.6421	**0.6475**	**0.6475**	0.2279
bio	0.7897	0.7853	0.7878	**0.7907**	**0.7907**	**0.7907**
shrutime	0.5781	0.5769	0.5659	**0.5812**	**0.5812**	0.2966
spam	0.9396	0.9391	**0.9401**	0.9267	0. 9267	0. 9267
default	0.4710	0.4757	**0.4763**	0.4682	0.4733	0.3050
average	0.6654	0.6662	**0.6685**	0.6647	0.6362	0.4252
model / dataset	**XGBoost：XGBClassifier()**					
income	0.6778	0.6692	0.6846	0.6894	**0.6899**	0.3721
arrhy	**0.5743**	**0.5743**	**0.5743**	**0.5743**	**0.5743**	**0.5743**
bank	0.4790	0.4774	0.4908	**0.4988**	**0.4988**	0.0913
htru	**0.8853**	**0.8853**	**0.8853**	**0.8853**	**0.8853**	**0.8853**
blastchar	0.5838	0.5803	0.5841	**0.5851**	**0.5851**	0.2837
shopper	**0.6668**	0.6596	0.6575	0.6475	0.6475	0.2279
bio	**0.7907**	**0.7907**	**0.7907**	**0.7907**	**0.7907**	**0.7907**
shrutime	0.5799	**0.5820**	0.5687	0.5812	0.5812	0.2966
spam	**0.9267**	**0.9267**	**0.9267**	**0.9267**	**0.9267**	**0.9267**
default	0.4654	0.4685	**0.4738**	0.4682	0.4733	0.3050

average	0.6630	0.6614	0.6637	0.6647	**0.6653**	0.4754
model / dataset	**LightGBM：LGBMClassifier()**					
income	**0.7147**	0.7124	0.7138	0.7106	0.7101	0.3545
arrhy	**0.5743**	**0.5743**	**0.5743**	**0.5743**	**0.5743**	**0.5743**
bank	0.5616	0.5598	**0.5631**	0.5539	0.5539	0.0913
htru	**0.8821**	**0.8821**	**0.8821**	**0.8821**	**0.8821**	**0.8821**
blastchar	0.5726	**0.5775**	0.5771	0.5764	0.5764	0.1117
shopper	**0.6564**	0.6481	0.6507	0.6409	0.6409	0.2279
bio	**0.7914**	**0.7914**	**0.7914**	**0.7914**	**0.7914**	**0.7914**
shrutime	0.5929	0.5890	**0.5941**	0.5887	0.5887	0.3069
spam	**0.9443**	**0.9443**	**0.9443**	**0.9443**	**0.9443**	**0.9443**
default	0.4803	**0.4805**	0.4790	0.4726	0.4771	0.3135
average	**0.6771**	0.6759	0.6770	0.6735	0.6739	0.4598
model / dataset	**MLP：MLPClassifier()**					
income	0.6550	0.6592	0.5161	0.6743	**0.6779**	0.6674
arrhy	0.5348	0.5497	0.5386	0.5089	0.5581	**0.5664**
bank	**0.5268**	0.5019	0.2792	0.5210	0.5079	0.5157
htru	0.8869	0.8842	0.8839	**0.8875**	0.8825	0.8868
blastchar	0.5668	**0.5975**	0.4366	0.5963	0.5828	0.5827
shopper	0.6327	0.6184	0.3458	**0.6414**	0.6262	0.6258
bio	0.8187	0.8286	0.8251	0.8276	**0.8349**	**0.8349**
shrutime	**0.5900**	0.5794	0.4364	0.5734	0.5722	0.5742
spam	0.9364	**0.9367**	0.9342	0.9333	0.9348	0.9335
default	0.4543	0.4568	0.3652	0.4678	**0.4797**	0.4655
average	0.6602	0.6612	0.5561	0.6632	**0.6657**	0.6653
model / dataset	**Logistic Regression：LogisticRegression()**					
income	**0.6623**	0.5521	0.4853	0.6525	0.6522	0.6487
arrhy	**0.4981**	**0.4981**	**0.4981**	**0.4981**	**0.4981**	**0.4981**
bank	**0.4508**	0.4125	0.1881	0.4441	0.4441	0.4380
htru	**0.8749**	**0.8749**	**0.8749**	**0.8749**	**0.8749**	**0.8749**
blastchar	0.5970	0.5949	0.5678	**0.5997**	**0.5997**	0.5943
shopper	0.5042	0.4869	**0.5223**	0.5011	0.5011	0.4994
Bio	**0.8130**	**0.8130**	**0.8130**	**0.8130**	**0.8130**	**0.8130**
shrutime	**0.3138**	0.3093	0.2519	0.3087	0.3087	0.3027

spam	**0.9027**	**0.9027**	**0.9027**	**0.9027**	**0.9027**	**0.9027**
default	**0.4670**	0.2869	0.1483	0.4635	0.4621	0.4602
average	**0.6084**	0.5731	0.5252	0.6058	0.6057	0.6032
Total average	**0.6548**	0.6476	0.6181	0.6544	0.6494	0.5258

由表 6 可見，綜合五種模型的表現之下，one-hot enconding 與 target encoding 有最好的表現，並且在五種模型中的表現都相對良好，無明顯缺點。若此二種編碼方式相互比較，可以發現 one-hot enconding 於 Logistic Regression 有明顯較好的表現，其他 tree-based 模型如隨機森林，使用 one-hot enconding 表現則較不理想，特別是在像是 Blastchar 這類別欄數很多的資料集，表現會低於平均，推測原因為 tree-based 模型本身就是依各特徵對樣本進行分類，所以若是使用 one-hot enconding 將類別型特徵轉化為多個二元特徵對模型表現也不會有很大影響。至於表現最差的編碼方式則為 Leave-One-Out encoding，藉由觀察資料及特性，我們可以推測，若是資料集偏差大，或是 imbalance 狀況較嚴重，皆會使 Leave-One-Out encoding 後模型表現較差，而再更深入的觀察資料欄位中與其他表現較好的編碼方式比較後，發現若是欄位中某些類別的樣本數量非常少，也會使 Leave-One-Out encoding 表現不佳，此原因應與本身的編碼方式有關，因其編碼方式是將每個類別分別跟其他類別比較，並建立新特徵來表示差異，表示只要某些類別的樣本數量非常少的話，新特徵就可能建立的不盡理想。

結論：

1. 綜合表現最佳的類別型資料編碼方式為：one-hot & target encoding
2. one-hot enconding 較不適用於 tree-based model
3. Leave-One-Out encoding 不適用於偏差大，或是 imbalance 狀況較嚴重的資料集

Problem 5

Which combinations of numerical and categorical feature transformation methods generally lead to better results?

這一題針對 income、bank、blastchar、shoppers、shrutime、default 這六個資料集進行分析。連續型數值的處理方式我採用 Standardization、Normalization、Non-linear transformation、Equal-width binning、Equal-Frequency binning 和 Rank conversion 六種方法。

以下為各個連續型變數轉換模型的參數。

- Standardization: StandardScaler()
- Normalization: MinMaxScaler()
- Non-linear transformation: PowerTransformer()
- Equal-width binning: KBinsDiscretizer(n_bins =10, encode='ordinal', strategy='uniform')
- Equal-Frequency binning: KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='quantile')
- Rank conversion: QuantileTransformer(n_quantiles=100, random_state=0, output_distribution='normal')

類別型變數的部分，則挑選 One-Hot encoder、Label encoder 還有 Leave-one-out encoder 三種方法。訓練模型的部分這一題使用的是 LGBM Classifier。以下表格為針對各個資料集，6 種連續型變數和 3 種類別型變數搭配出的模型的結果。

表 7、One-Hot encoder 搭配六種連續型變數處理方法

	One-Hot encoder					
Dataset	**A**	**B**	**C**	**D**	**E**	**F**
Income	0.7145	0.7134	0.7158	0.6746	0.6485	**0.7165**
Bank	0.5651	0.5615	**0.5657**	0.4808	0.5136	0.5615
Blastchar	0.5720	0.5835	0.5751	0.5871	0.5692	0.575
Shoppers	**0.6561**	0.6491	0.6496	0.4325	0.6546	0.6542
Shrutime	0.5965	0.5936	0.5913	0.5806	0.4566	**0.5996**
default	0.4822	0.4806	0.4772	0.4735	0.4774	0.4780

表 8、Label encoder 搭配六種連續型變數處理方法

	Label encoder					
Dataset	**A**	**B**	**C**	**D**	**E**	**F**
Income	0.7159	0.7149	0.7163	0.6717	0.6460	0.7151
Bank	0.5540	0.5492	0.5498	0.4781	0.5033	0.5510
Blastchar	0.5696	0.5730	0.5677	0.5778	0.5726	0.5759
Shoppers	0.6538	0.6507	0.6529	0.4270	0.6521	0.6516
Shrutime	0.5907	0.5899	0.5980	0.5831	0.4396	0.5847
default	0.4808	0.4795	0.4800	0.4741	0.4780	0.4783

表 9、Leave-one-out encoder 搭配六種連續型變數處理方法

	Leave-one-out encoder					
Dataset	**A**	**B**	**C**	**D**	**E**	**F**
Income	0.7085	0.7084	0.7044	0.6582	0.6330	0.7061
Bank	0.5273	0.5343	0.5280	0.4414	0.4480	0.5286
Blastchar	0.5990	0.5843	0.5954	**0.6054**	0.5877	0.6042
Shoppers	0.6451	0.6511	0.6393	0.4249	0.6510	0.6452
Shrutime	0.5954	0.5862	0.5941	0.5759	0.4554	0.5864
default	0.4918	0.4846	0.4869	**0.4923**	0.4873	0.4862

[註]由於縮小版面的關係，將連續型變數處理方法以英文字母代表

A:Standardization
B:Normalization
C:Non-linear transformation
D:Equal-width binning
E:Equal-Frequency binning
F:Rank conversion

表 10、表現最佳模型比較

Dataset	**表現最佳模型**
Income	One-Hot encoder + Rank conversion
Bank	One-Hot encoder + Non-linear transformation
Blastchar	Leave-one-out + Equal-width binning
Shoppers	One-Hot encoder + Standardization
shrutime	One-Hot encoder + Rank conversion
default	Leave-one-out + Equal-width binning

從表五的結果分析，連續型變數處理似乎沒有一個最好的方法，根據不同資料特性應有相對應的處理方法。類別型變數處理的部分，多數的資料仍以 One-Hot encoder 處理表現較佳。最後，雖然沒有一個類別型和連續型變數處理方法的組合是最好的，但是 One-Hot encoder + Rank conversion 的組合同時是 income 和 shrutime 的最佳模型，這兩種方法的組合為這次分析中表現相對較好的組合。

Problem 6

If the number of possible categorical values of a feature is high, which encoding methods among target encoding, one-hot encoding, and label encoding will have better performance? Why?

這一題會針對 income、bank、blastchar、shoppers、shrutime、default 這六個資料集進行分析。由於這些資料中的類別型變數中並不只是單純的二元分類，有些變數裡包含的類別有很多種。以下列出各個資料中類別型變數類別大於 5 種的變數以及這些變數的類別個數。

- Income : workclass(8 個)、education(16 個)、marital_status(7 個)、occupation(14 個)、native_country(40 個)
- Bank: job(12 個)、month(12 個)、
- Shoppers: Month(12 個)
- Default: Education(7 個)、PAY_0(11 個)、PAY_2(10 個)、PAY_3(10 個)、PAY_4(10 個)、PAY_5(9 個)、PAY_6(9 個)

Income 和 default 兩個資料集的類別型變數都有很多種類別，bank 和 shoppers 則有一些變數是多種類的，而 blastchar 裡的類別變數都只有 2~3 種類別，但是 blastchar 共有 17 個類別型變數，因此加起來類別型變數個數也不少於 income 和 default。

接下來針對每個資料集，類別型變數以 Target encoder、One-Hot encoder、Label encoder 進行處理。連續型變數統一以 Standardscaler 進行標準化。訓練模型用 LGBM Classifier、Logistic Regression，選用 Logistic Regression 的原因是怕 One -Hot encoder 會影響 LGBM 的表現，進而影響到後續的比較。

表 11、One-Hot、Label、Target encoder 表現比較

	One-Hot		Label encoder		Target encoder	
Dataset	**LGBM**	**LR**	**LGBM**	**LR**	**LGBM**	**LR**
Income	0.7145	0.6594	**0.7159**	0.557	0.7133	0.3847
Bank	**0.5651**	0.4508	0.554	0.3162	0.5592	0.2445
Blastchar	0.5720	**0.5970**	0.5696	0.5945	0.5779	0.5811
Shoppers	**0.6561**	0.5042	0.6538	0.4993	0.6518	0.4996
shrutime	**0.5965**	0.3138	0.5907	0.2708	0.5881	0.0984
default	**0.4822**	0.4682	0.4808	0.3657	0.4783	0.0006

從表 11 可以看出除了 income 資料集外，其他資料集表現最好的模型都是使用 One-Hot encoder，在此次分析中，One-Hot encoder 整體來說仍是最佳的 encoder。

照理來說當類別型變數內包含很多種類時，One-Hot encoder 仍會是最佳的模型，因為 Label encoder 會賦予每個類別數值，但是當種類太多時反而會讓模型造成誤解。例如:當一個類別型變數內有 20 個分類時，每個分類會被分別賦予 1~20 的數值，這樣 20 跟 1 差異很大，容易會被誤解成 20 的類別比 1 的類別重要，但實際上可能並沒有。但此一情況有一個例外，當訓練模型是 tree-based method 時，Label encoder 就會表現較好。此現象可以從 income 資料集中發現，在使用 tree-based 的 LGBM 時，Label encoder 的表現會比 One-Hot 來的好。但是當訓練模型換成 Logistic regression 後，Label encoder 表現明顯比 One-Hot encoder 差上許多。

我們也可以觀察 One-Hot encoder 的部分，原先以為 LGBM 的表現應該會比 Logistic regression 差，但是結果看來除了 blastchar 資料集外，LGBM 表現仍是最好的。推測是因為 blastchar 資料集的類別型變數比較多(17 個)才比較有影響到 LGBM 的結果。

Target encoder 的部分則和 One-Hot encoder 表現差異不大，但是 One-Hot 仍略勝一籌。若使用 Leave-one-out 可以減少 over-fitting 的機會，或許會有更好的結果

總結來說，當類別型變數內的種類是極端的多時(例如 income 資料的 native_country 變數)，使用其他 encoder 確實可能效果會較好。若類別型變數內的種類沒有到極端的高的話，One-Hot encoder 仍足以應付這些變數。

Problem 7

Compare the classification performance of doing nothing, 7 undersampling, 4 oversampling, 2 ensemble-based methods in the presence of class imbalance. Which method works generally the best and the worst? Why?

這題我們採用上面的 10 個資料集，使用了 lightgbm 模型比較各式各樣的 resampling strategy，由於 Cluster Centroids 和 Condensed NN 兩種方法在實驗的過程中跑太久(光是 income 資料集 20 分鐘都沒有結果)，因此並沒有把上述兩種方法列入下方表格中。另外，BalanceCascade 在 0.6 版本之後就被 sklearn deprecate，因此無法使用該套件。

為了表格的版面配置，**NCR 代表 Neighborhood Cleaning Rule，TL 代**

表 Tomkek Link，OSS 代表 One-Sided Selection，我們一樣採用 f1-score 當作評估指標，而結果列在下方的表格。

表 12、不同 resampling 方法的結果(1)

	nothing	**NearMiss**	**EditedNN**	**NCR**	**TL**	**OSS**
Income	0.7161	0.611	0.7037	0.71	**0.7296**	0.7293
Arrhythmia	0.62	0.4715	0.5841	0.5735	0.5935	0.5887
Bank	0.5651	0.3483	0.6292	**0.6344**	0.5895	0.5895
HTRU2	0.8809	0.5735	0.882	0.8808	**0.8858**	0.8855
BlastChar	0.572	0.5474	0.6209	0.623	0.6033	0.6018
Shoppers	0.6522	0.4037	0.6651	0.6749	0.6591	0.6696
QSAR Bio	**0.801**	0.7578	0.7822	0.7948	0.7983	0.7913
Shrutime	0.5965	0.4316	0.6195	**0.6209**	0.6157	0.6155
Spambase	0.945	0.8885	0.9394	0.9339	0.9459	0.9456
Default	0.4782	0.399	0.5431	**0.5425**	0.504	0.5068

表 13、不同 resampling 方法的結果(2)

	SMOTE	Borderline SMOTE	SVM SMOTE	ADASYN	Easy
Income	0.7183	0.7112	0.7174	0.7164	0.6966
Arrhythmia	**0.6525**	0.6239	0.6508	0.6445	0.5767
Bank	0.6071	0.6053	0.6197	0.6077	0.5563
HTRU2	0.854	0.8413	0.849	0.7079	0.8177
BlastChar	0.6011	0.5991	0.6148	0.6012	**0.6296**
Shoppers	0.6717	0.6733	0.6765	**0.6866**	0.6452
QSAR Bio	0.7862	0.7891	0.7999	0.7933	0.7738
Shrutime	0.6192	0.6136	0.6159	0.6178	0.5848
Spambase	0.9424	**0.9464**	0.9462	0.9417	0.9282
Default	0.5116	0.5102	0.5202	0.4995	0.5332

從上方的兩個表格中，我們可以發現 Neighborhood Cleaning Rule 和 Tomkek Link 的表現相對比較好，而在這兩個方法中表現比較好的資料集通常都有一些離散型變數且為相對大的資料。而 oversampling 的方法中，表現比較的資料集通常資料集的資料量較小。另外，NearMiss()是裡面表現最差的方法，可能跟我們沒有調整其參數或者 NearMiss()較容易移除一些重要資訊導致 f1 score 偏低。

Problem 8

Can you find SMOTE-based oversampling works better on which kinds of datasets? Why?

這題我們採用上面的 10 個資料集，使用了 lightgbm 模型加上三種 smote-based oversampling(SMOTE、BorderlineSMOTE、SVMSMOTE)進行實驗，並採用 f1-score 當作評估指標，而結果列在下方的表格。

表 14、SMOTE-based 測試在十個資料集的結果

	SMOTE	BorderlineSMOTE	SVMSMOTE
Income	0.7183	0.7112	0.7174
Arrhythmia	0.6525	0.6239	0.6508
Bank	0.6071	0.6053	0.6197
HTRU2	0.854	0.8413	0.849
BlastChar	0.6011	0.5991	0.6148
Shoppers	0.6717	0.6733	0.6765
QSAR Bio	0.7862	0.7891	0.7999
Shrutime	0.6192	0.6136	0.6159
Spambase	0.9424	0.9464	0.9462
Default	0.5116	0.5102	0.5202

從上方的表格中我們可以看到表現最好的三個資料集分別是 Spambase、HTRU2、QSAR Bio，表現最差的資料集分別是 Default、BlastChar、Bank。從上述的發現我們可以推測如果資料集的變數大多都是連續型變數的話，那使用 smote-based 的方法效果會比較好。反之，有較多離散型變數的資料集使用 smote-based 的方法的效果會比較差一些。

Problem 9

Is a dataset's imbalance ratio(e.g., %Pos) related to choosing which resampling strategy for better performance? Any insights?

這一題我們會沿用第七題的結果進行分析，下方兩個表格就是第七題的結果並在資料集的欄位加上 %Pos。

表 15、不同 resampling 方法的結果(1)

	nothing	NearMiss()	EditedNN	NCR	TL	OSS
Income 24.08%	0.7161	0.611	0.7037	0.71	**0.7296**	0.7293
Arrhythmia14.60%	0.62	0.4715	0.5841	0.5735	0.5935	0.5887
Bank 11.70%	0.5651	0.3483	0.6292	**0.6344**	0.5895	0.5895
HTRU2 9.16%	0.8809	0.5735	0.882	0.8808	**0.8858**	0.8855
BlastChar 26.57%	0.572	0.5474	0.6209	0.623	0.6033	0.6018
Shoppers 15.47%	0.6522	0.4037	0.6651	0.6749	0.6591	0.6696
QSAR Bio 33.74%	**0.801**	0.7578	0.7822	0.7948	0.7983	0.7913
Shrutime 20.37%	0.5965	0.4316	0.6195	**0.6209**	0.6157	0.6155
Spambase 39.40%	0.945	0.8885	0.9394	0.9339	0.9459	0.9456
Default 22.12%	0.4782	0.399	0.5431	**0.5425**	0.504	0.5068

表 16、不同 resampling 方法的結果(2)

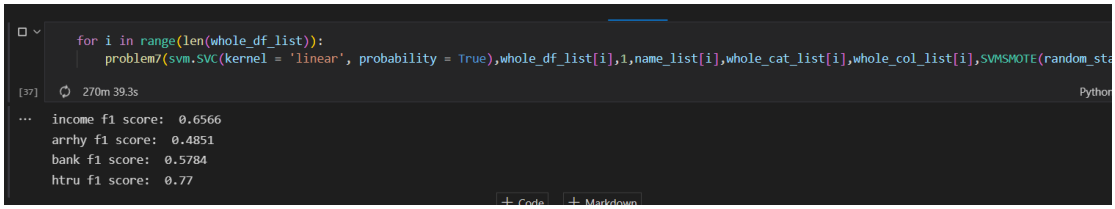
	SMOTE	Borderline SMOTE	SVM SMOTE	ADASYN	Easy
Income 24.08%	0.7183	0.7112	0.7174	0.7164	0.6966
Arrhythmia14.60%	0.6525	0.6239	0.6508	0.6445	0.5767
Bank 11.70%	0.6071	0.6053	0.6197	0.6077	0.5563
HTRU2 9.16%	0.854	0.8413	0.849	0.7079	0.8177
BlastChar 26.57%	0.6011	0.5991	0.6148	0.6012	**0.6296**
Shoppers 15.47%	0.6717	0.6733	0.6765	**0.6866**	0.6452
QSAR Bio 33.74%	0.7862	0.7891	0.7999	0.7933	0.7738
Shrutime 20.37%	0.6192	0.6136	0.6159	0.6178	0.5848
Spambase 39.40%	0.9424	**0.9464**	0.9462	0.9417	0.9282
Default 22.12%	0.5116	0.5102	0.5202	0.4995	0.5332

從上方的兩個表格中，可以看到不平衡比例和使用哪些特定 resampling strategy 有較好的表現沒有特別的關係。值得一提的是，QSAR Bio 與 Spambase 資料集在所有的 resampling 方法和沒有 resampling 的表現都差不多甚至輸給什麼都沒做的方法，因此只要不平衡比例過懸殊，使用 resampling strategy 肯定對於提升表現有幫助。另一個有趣的發現是 HTRU 資料集使用 oversampling 的效果不升反降，除此之外，oversampling 的方法基本上對於套用在其他資料集的表現都有些微的提升。

Problem 10

How do different ML algorithms (RF, XGBoost, LightGBM, MLP, SVM) prefer different resampling strategies for better performance of imbalance classification?

這一題我們會採用九種不同的 resampling strategy 分別是 NearMiss()、EditedNN、NCR、TL、OSS、SMOTE、BorderlineSMOTE、SVMSMOTE 和 ADASYN。並使用四個模型 RF、XGBoost、LightGBM、MLP 套用在十個資料集上做比較，沒有使用 SVM 的原因是使用 SVM+SVMSMOTE 套用在四個資料集就耗費了接近五小時(如下圖)。



我們會將每個資料集的結果各自畫一個表格，共十個表格。

表 17、不同 resampling 方法和模型在 Income 資料集的結果

Income 24.08%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.5394	0.5605	0.611	0.4639
EditedNN	0.6861	0.7073	0.7037	0.6743
NCR	0.6893	0.7115	0.71	0.6748
TL	0.69	**0.722**	**0.7296**	0.6727
OSS	0.6952	**0.7233**	**0.7293**	0.6638
SMOTE	0.6857	0.7165	0.7183	0.6693
BorderlineSMOTE	0.6841	0.7112	0.7112	0.6604
SVMSMOTE	0.6885	0.7174	0.7174	0.6717
ADASYN	0.6838	0.7123	0.7164	0.6669

以 Income 資料集來說，Tomkek Link/One-Sided Selection 搭配 XGBoost/LightGBM 的效果最佳，最好的組合是 Tomkek Link + LightGBM。

表 18、不同 resampling 方法和模型在 Arrhythmia 資料集的結果

Arrhythmia14.60%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.4099	0.3876	0.4715	0.4151
EditedNN	0.6449	0.5825	0.5841	0.5489
NCR	**0.6748**	0.5585	0.5735	0.5346
TL	0.6325	0.599	0.5935	0.5721
OSS	0.6394	0.5968	0.5887	0.5036
SMOTE	0.6279	0.6249	0.6525	0.51
BorderlineSMOTE	0.6475	0.6056	0.6239	0.5698
SVMSMOTE	**0.6793**	0.6098	0.6508	0.5479
ADASYN	**0.6873**	0.6256	0.6445	0.5761

以 Arrhythmia 資料集來說，Neighborhood Cleaning Rule/SVMSMOTE/ADASYN 搭配 RF 的效果最佳，最好的組合是 ADASYN + RF。

表 19、不同 resampling 方法和模型在 Bank 資料集的結果

Bank 11.70%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.319	0.3549	0.3483	0.3264
EditedNN	0.6136	**0.6305**	**0.6292**	0.5965
NCR	0.6115	**0.6254**	**0.6344**	0.5915
TL	0.5427	0.587	0.5895	0.5536
OSS	0.5446	0.5851	0.5895	0.5572
SMOTE	0.5748	0.5882	0.6071	0.5446
BorderlineSMOTE	0.5682	0.5866	0.6053	0.5451
SVMSMOTE	0.5885	0.5993	0.6197	0.5612
ADASYN	0.5744	0.587	0.6077	0.5409

以 Bank 資料集來說，EditedNN/Neighborhood Cleaning Rule 搭配 XGBoost/LightGBM 的效果最佳，最好的組合是 Neighborhood Cleaning Rule + LightGBM。

表 20、不同 resampling 方法和模型在 Bank 資料集的結果

HTRU2 9.16%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.2961	0.3219	0.5735	0.3934
EditedNN	0.8841	0.8824	0.882	0.8868
NCR	0.883	0.8807	0.8808	**0.8861**
TL	0.881	0.8805	0.8858	**0.8878**
OSS	0.8845	0.8814	0.8855	**0.8869**
SMOTE	0.8722	0.8543	0.854	0.8218
BorderlineSMOTE	0.8706	0.8549	0.8413	0.7889
SVMSMOTE	0.8735	0.8567	0.849	0.814
ADASYN	0.816	0.753	0.7079	0.6404

以 HTRU2 資料集來說， Neighborhood Cleaning Rule/ Tomkek Link/One-Sided Selection + MLP 的效果最佳，最好的組合是 Tomkek Link +MLP。

表 21、不同 resampling 方法和模型在 BlastChar 資料集的結果

BlastChar 26.57%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.4352	0.4327	0.5474	0.4316
EditedNN	0.6212	0.6192	0.6209	0.606
NCR	**0.6244**	**0.6221**	**0.623**	0.6121
TL	0.5833	0.5873	0.6033	0.5876
OSS	0.5841	0.5858	0.6018	0.5841
SMOTE	0.5799	0.5829	0.6011	0.5741
BorderlineSMOTE	0.5778	0.5957	0.5991	0.5712
SVMSMOTE	0.59	0.5955	0.6148	0.5908
ADASYN	0.5807	0.5841	0.6012	0.5746

以 BlastChar 資料集來說， Neighborhood Cleaning Rule 搭配 RF/XGBoost/LightGBM 的效果最佳，最好的組合是 Neighborhood Cleaning Rule + RF。

表 22、不同 resampling 方法和模型在 Shoppers 資料集的結果

Shoppers 15.47%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.4561	0.4705	0.4037	0.3799
EditedNN	0.6747	0.6601	0.6651	0.6363
NCR	0.6787	0.6628	0.6749	0.6459
TL	0.6596	0.649	0.6591	0.6506
OSS	0.6649	0.6497	0.6696	0.6365
SMOTE	0.6794	0.6541	0.6717	0.6275
BorderlineSMOTE	0.6751	0.6476	0.6733	0.626
SVMSMOTE	**0.6843**	0.6545	0.6765	0.6365
ADASYN	**0.6817**	0.6547	**0.6866**	0.6222

以 Shoppers 資料集來說，SVMSMOTE 搭配 RF 和 ADASYN 搭配 RF/LightGBM 的效果最佳，最好的組合是 ADASYN + LightGBM。

表 23、不同 resampling 方法和模型在 QSAR Bio 資料集的結果

QSAR Bio 33.74%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.7443	0.7331	0.7578	0.7349
EditedNN	0.7807	0.7879	0.7822	0.798
NCR	0.7952	0.7933	0.7948	0.7979
TL	0.8003	0.8022	0.7983	0.8188
OSS	0.8052	0.8036	0.7913	**0.8218**
SMOTE	0.8026	0.8	0.7862	**0.8363**
BorderlineSMOTE	0.7927	0.7796	0.7891	0.8149
SVMSMOTE	0.7962	0.7981	0.7999	**0.8216**
ADASYN	0.7899	0.789	0.7933	0.8198

以 QSAR Bio 資料集來說，One-Sided Selection /SVM/SVMSMOTE 搭配 MLP 的效果最佳，最好的組合是 SMOTE + MLP。

表 24、不同 resampling 方法和模型在 Shrutime 資料集的結果

Shrutime 20.37%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.4236	0.4314	0.4316	0.3925
EditedNN	**0.614**	0.6103	**0.6195**	0.6094
NCR	**0.6185**	0.6064	**0.6209**	0.6074
TL	0.598	0.603	0.6157	0.601
OSS	0.5959	0.6027	0.6155	0.6008
SMOTE	0.6025	0.5996	0.6192	0.585
BorderlineSMOTE	0.5956	0.5943	0.6136	0.5836
SVMSMOTE	0.608	0.6025	0.6159	0.5966
ADASYN	0.5964	0.5941	0.6178	0.5836

以 Shrutime 資料集來說，EditedNN / Neighborhood Cleaning Rule 搭配 RF / LightGBM 的效果最佳，最好的組合是 Neighborhood Cleaning Rule + LightGBM。

表 25、不同 resampling 方法和模型在 Spambase 資料集的結果

Spambase39.40%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.9373	0.9326	0.8885	0.9211
EditedNN	0.9377	0.9369	0.9394	0.9261
NCR	0.9348	0.9318	0.9339	0.9194
TL	0.9436	0.9412	**0.9459**	0.9336
OSS	0.9385	0.9428	0.9456	0.9363
SMOTE	0.9434	0.9411	0.9424	0.932
BorderlineSMOTE	0.9411	0.9399	**0.9464**	0.9253
SVMSMOTE	0.9411	0.9396	**0.9462**	0.9274
ADASYN	0.9415	0.9416	0.9417	0.9295

以 Spambase 資料集來說，Tomkek Link / BorderlineSMOTE/SVMSMOTE 搭配 LightGBM 的效果最佳，最好的組合是 BorderlineSMOTE + LightGBM。

表 26、不同 resampling 方法和模型在 Default 資料集的結果

Default 22.12%	RF	XGBoost	LightGBM	MLP
NearMiss()	0.3894	0.3953	0.399	0.3896
EditedNN	**0.5381**	0.5355	**0.5431**	0.5113
NCR	0.535	0.5324	**0.5425**	0.5125
TL	0.5006	0.4951	0.504	0.4909
OSS	0.503	0.4991	0.5068	0.4829
SMOTE	0.5179	0.4871	0.5116	0.4674
BorderlineSMOTE	0.5108	0.4874	0.5102	0.4604
SVMSMOTE	0.5184	0.4965	0.5202	0.4811
ADASYN	0.5103	0.4782	0.4995	0.4541

以 Default 資料集來說，EditedNN 搭配 RF 和 EditedNN/ Neighborhood Cleaning Rule 搭配 LightGBM 的效果最佳，最好的組合是 EditedNN + LightGBM。

結論 ： 雖然不同模型搭配不同 resampling strategy 到不同的資料集上沒有一個非常明顯的規律，但我們可以發現 **LightGBM 一般來說是表現最好的模型**，且其搭配 EditedNN/ Neighborhood Cleaning/ Tomkek Link 的表現在很多資料集上都贏過其他模型與 resampling strategy 的組合。在某些時候，RF 和 MLP 的表現也會很不錯，但至於 RF 和 MLP 不會有搭配特定的 resampling strategy 表現比較好的現象。另外 NearMiss 是 resampling strategy 裡面表現最差的，而 XGBoost 無法在十個資料集中任一個資料集脫穎而出。