機器學習期末專題報告
Cyber Security Attack Defender

組別名稱：EE+CS
組員：孫凡耕(b09301056, 隊長)，羅啟心，林子芃，陳聖曄
工作分配：(所有人皆參與討論)
Preprocessing / Feature Engineering：
1. Preprocessing:孫凡耕
2. Feature Importance:
    randomforest feature importance：羅啟心
    violin_plot：孫凡耕
    討論：孫凡耕
3. Training set and testing set 不同分布：孫凡耕

Model Description：
1. 描述：孫凡耕、林子芃
2. 作圖：羅啟心
3. 討論：陳聖曄

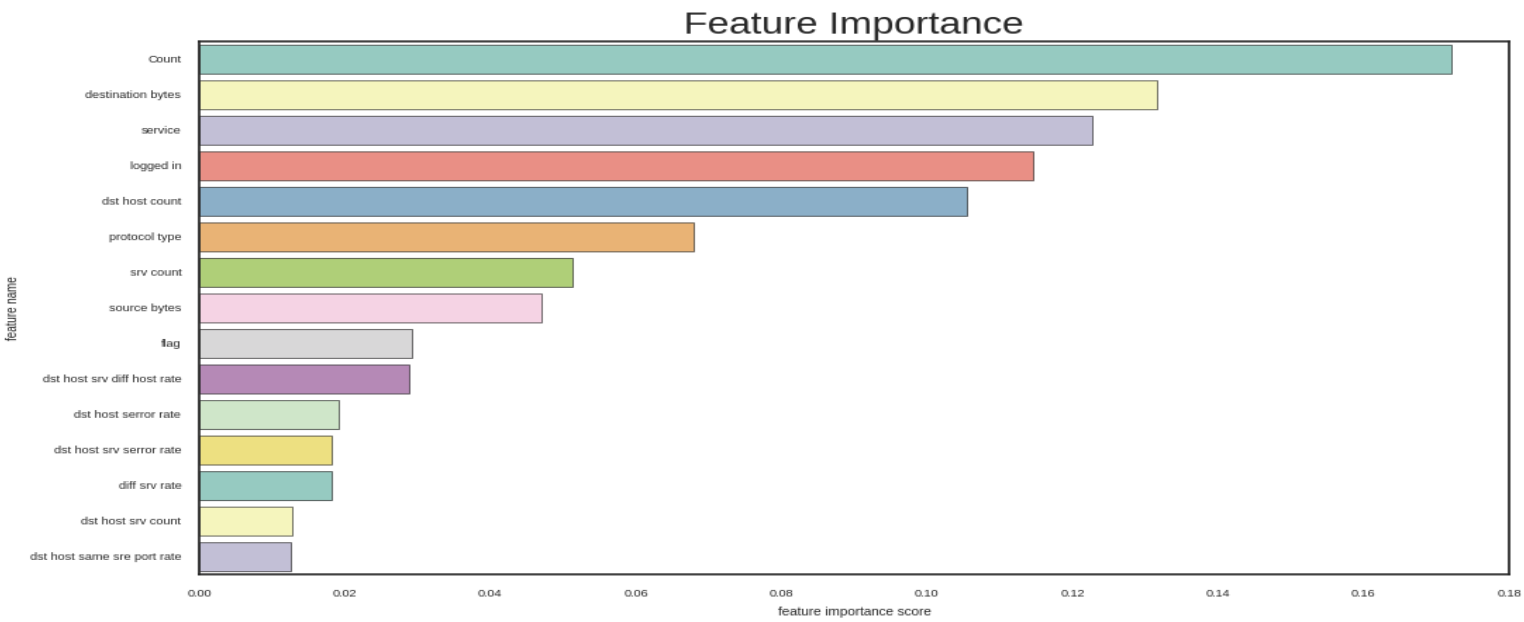
Experiments：
1. Xgboost 的參數：
    實驗與作圖：孫凡耕
    討論：羅啟心、陳聖曄
2. randomforest 的隨機性：
    實驗與作圖：羅啟心
    討論：羅啟心

● Preprocessing / Feature Engineering：

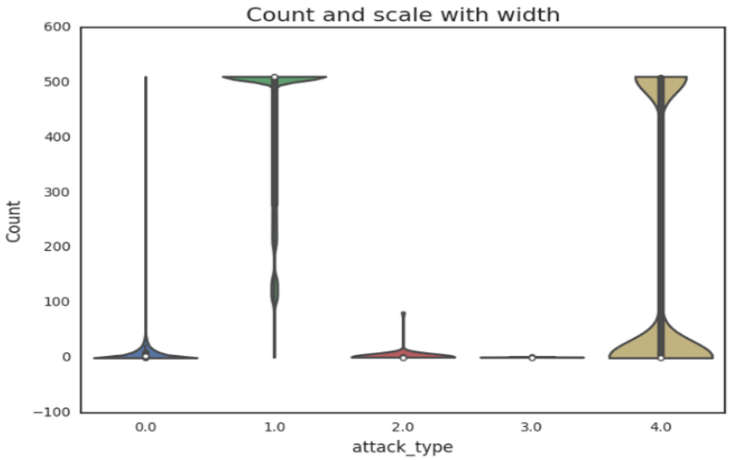
1. Preprocessing：

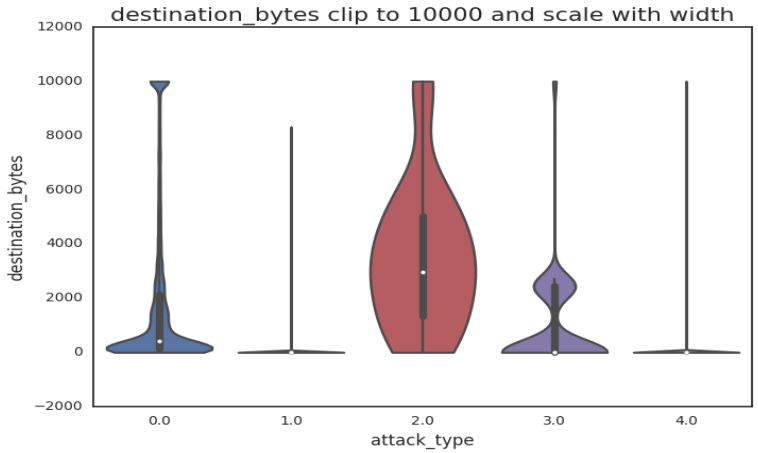
我們沒有做特別的 preprocessing，只有將原本是 string 的 data 轉成 one-hot encoding，並且把 label 一律轉換成 class 0~4 進行模型的訓練。

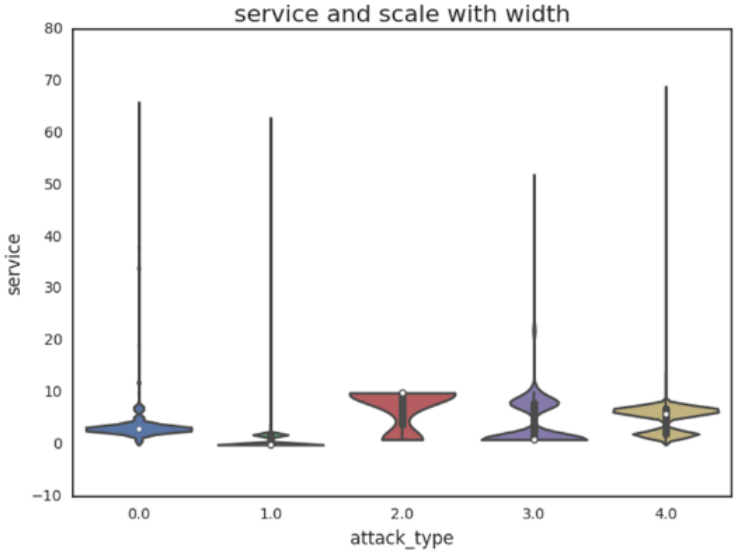
2. Feature Importance：

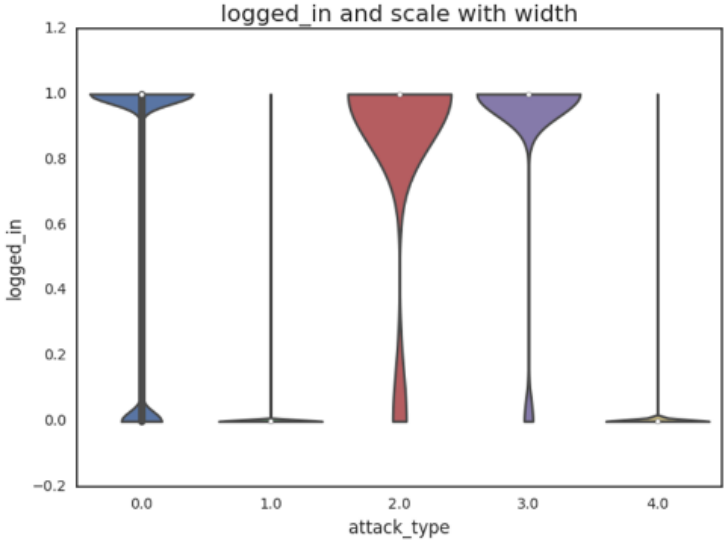


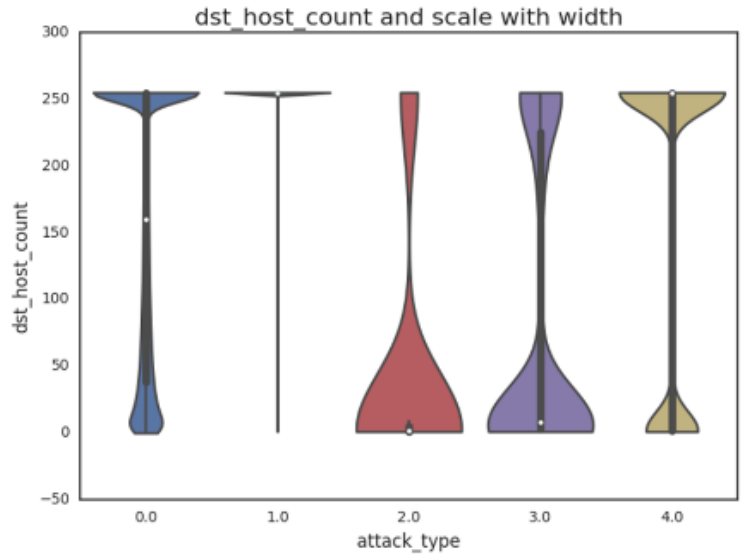
根據 randomforest 訓練的結果，輸出 randomforest 認為重要的 feature，前六名分別是：Count, destination bytes, service, logged in, dist host count, protocol type. 我們用 seaborn.violin_plot 視覺化上面六個 feature 在不同的 class 的分布如下：(橫坐標為五個 attack types, 縱坐標為數值，所有圖都是 scale with width，也就是每個圖表中五個 violin_plot 的最大寬度都一樣，因此看起來和真實數據的數量不一樣。標題若有 clip，則表示把數值最大值限定在某數字以下，避免 violion_plot 過於細長)

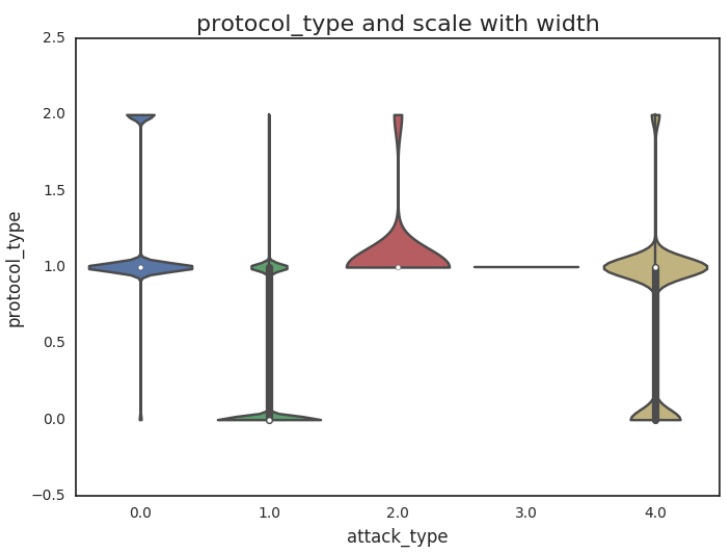












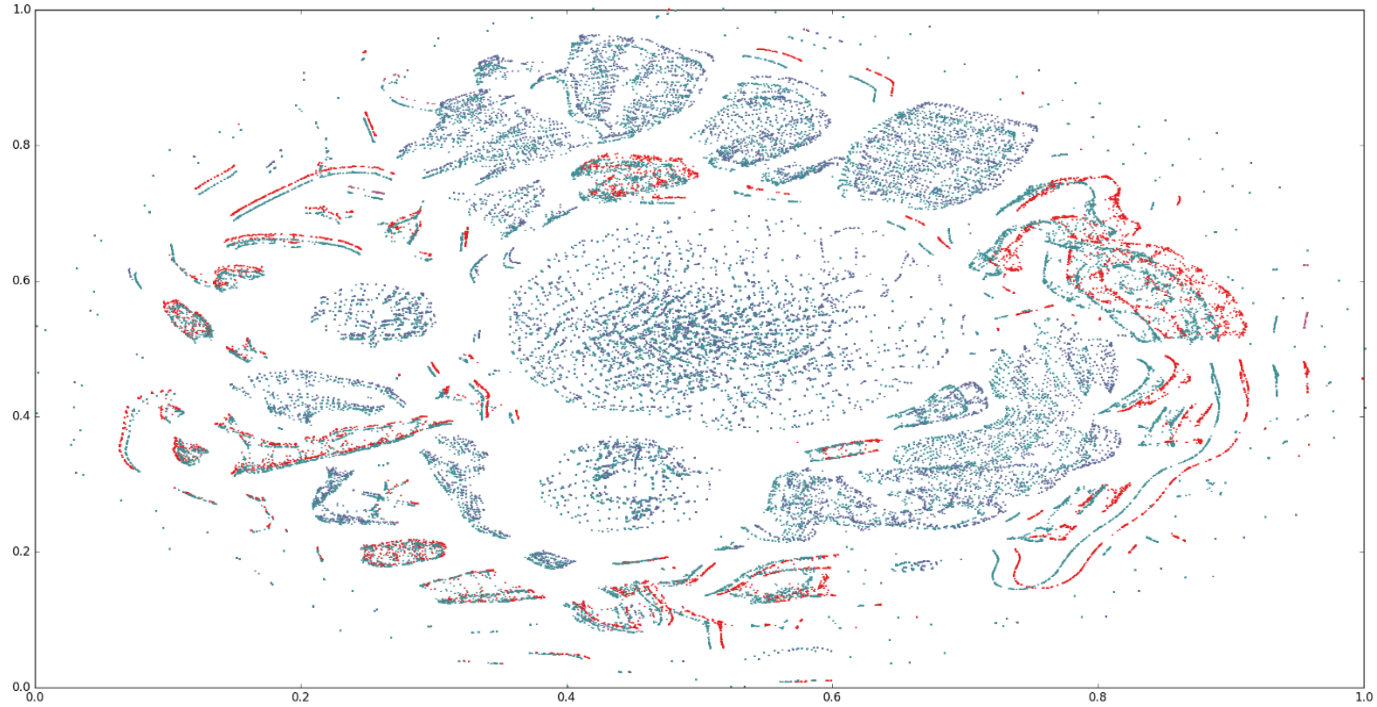
討論：

因為資料中 0 和 1 的數量最多，因此我們若只關注 0 和 1 在這五個最重要 feature 中不同的分布(白色點代表中位數)，可以發現基本上都是： 0 和 1 多數的資料存在的數值範圍(violin_plot 較胖的地方)，都不一樣，因此造成這五個 feature 重要性較高的原因。

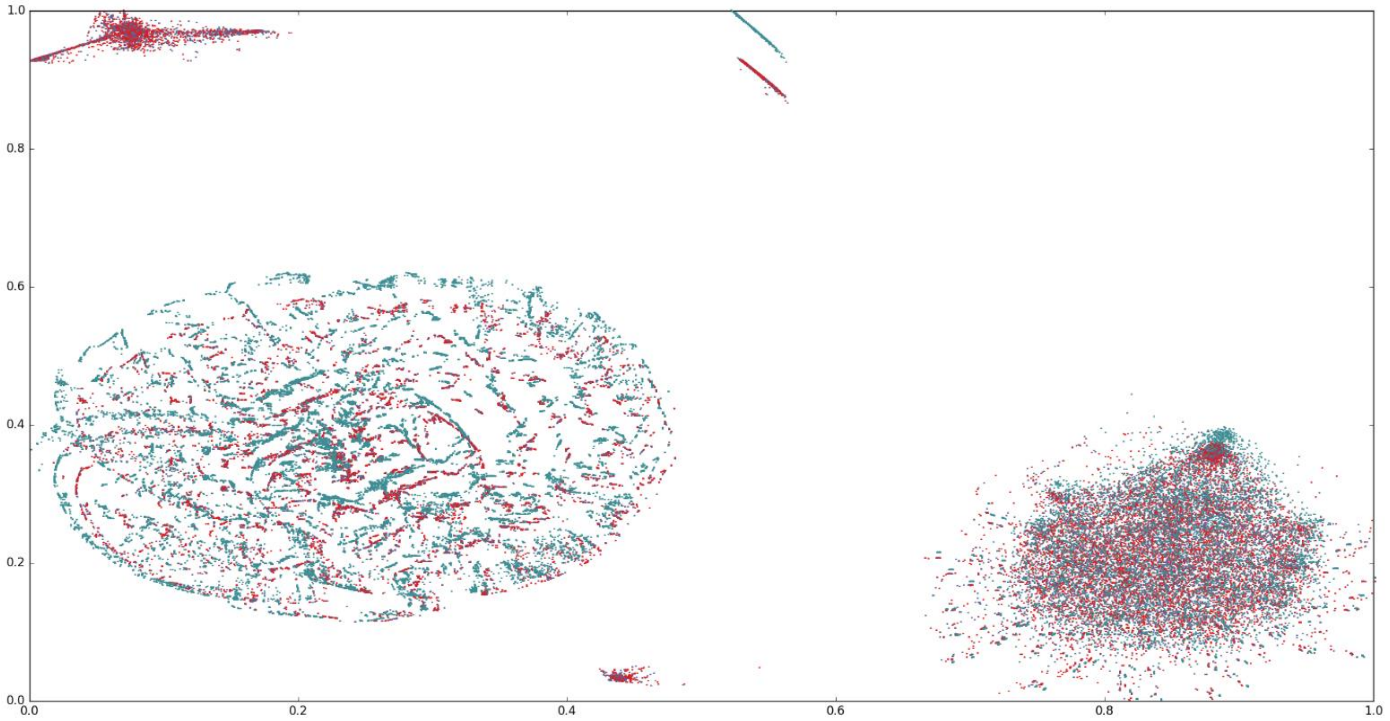
3. training set and testing set 不同的分布

我們用 MulticoreTSNE 視覺化資料的分布，針對是否對於離散的資料做 one-hot-encoding 來做區別，我們畫了兩張圖，分別有 100000 個數據點，紅點為 class 0，藍點為 class 1，綠點為 testing data(不知道 class 為何)，至於 class 2, 3, 4 因為資料相對很少，因此雖然顏色不同但圖上看不出來。

沒有做 one-hot encoding：



有做 one-hot encoding：



討論：

可以發現，training set 和 testing set 的分布雖然形狀一樣，各自的分布一樣，畫在一起時卻發現兩個分布是某種平移(也不完全是平移，不同位置有不同方向與量的平移)。同時沒有做 one-hot encoding 的似乎更能分出 class 0 和 1，我們猜測是因為做了 one-hot encoding 之後資料變得稀疏，因此 TSNE 需要跑更久或是甚至因此不適合用 TSNE 做降維視覺化。

● Model Description:

使用的 model 為 signature-based detection model，model 學習各種攻擊方式的 feature，並使用 binary/multiclass classification 分辨是否為其中一種攻擊。Signature-based detection 的好處是對於已知的攻擊方式，能在低誤判率的情況下，準確預測出攻擊方式。

1. 超越 weak baseline：**單純用 sklearn.randomforest**

   參數為 n_estimators=27 時最佳。

   最佳分數：0.96062，用時約 25 秒，記憶體用量<1GB

2. 超越 strong baseline：**用 xgboost (gradient boosting 的一種)**

   參數為{'max_depth':7, 'objective':'multi:softmax', 'num_class':5, 'eta':0.3, 'min_child_weight':2 , 'max_delta_step':2, 'subsample':0.8, 'tree_method':'exact'} num_round = 80 時最佳。

   最佳分數：0.96164，用時約 4 分鐘，記憶體用量約 4GB

3. 進步一：**五個二元分類的 xgboost**

   參數為{'max_depth':6, 'objective':'binary:logistic', 'eta':0.3, 'min_child_weight':1, 'scale_pos_weight':smn/smp , 'subsample':0.8, 'tree_method':'exact'} num_round = 50 時最佳。

   (smn 為 sum of negative, smp 為 sum of positive)

   最佳分數：0.962 用時約 20 分鐘，記憶體用量約 5GB

4. 進步二：**有權重的五個二元分類的 xgboost，並用 one-hot encoding**

   參數為 num_round = [35, 40, 35, 45, 35] fac = [0.7, 4, 1.3, 1.1, 0.9] mxdp = [14, 14, 14, 14, 14] para = { 'max_depth':mxdp[i], 'objective':'binary:logistic' , 'eta':0.26, 'min_child_weight':5, 'scale_pos_weight':smn/smp , 'subsample':0.8, 'tree_method':'exact'}時最佳。

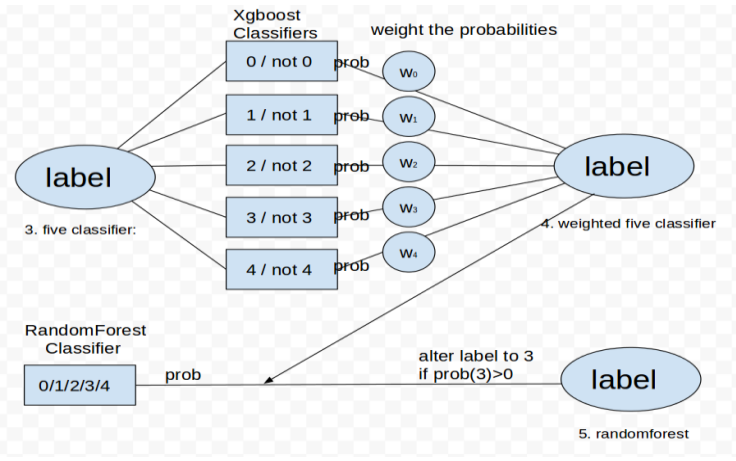
   最佳分數： 0.96357，用時約 50 分鐘，記憶體用量約 17GB

5. 最終最佳：**去除重複 training data，並用 randomforest 調整 3 的數量**

   參數為 n_estimator = 25 時最佳。

   最佳分數： 0.96941，用時約 25 秒，記憶體用量<1GB

   示意圖：



討論：

1->2：因為 xgboost 可以針對錯誤的 data 做 boosting，所以他在 training data 上可以 fit 的更好。

2->3：因為他每個 class data 數目很不平均，而有些 class 彼此之間可能沒有那麼容易分開，所以每個都做 binary 可以讓不容易分開的 class 更有機會被分開，但是 data 很少的 class fit 的可能就不太好。

3->4：調權重是因為 training data 跟 testing data 的分佈不太一樣(ex: train 的 label 3 很少，test 的 label 3 相對比較多)，這個作法可以讓 test data 出現比較多但 train 上少的可以更有機會被 predict。

4->5：因為 test 上的 label 3 比例比 train 高，再開一個 random forest，只要 predict 出是 label 3 的機率大於 0，就更改原本的 label 到 label 3。(類似跟一開始的 model 做 ensemble 的感覺)

- # Experiments and Discussions：

1. ## Xgboost 的參數：

   從 unweighted 五個二元分類的 xgboost model 開始，同時對五個 xgboost model 調整某一參數(固定其他參數)，觀察此參數對於預測出來 0, 1, 2, 3, 4 的數量的影響。

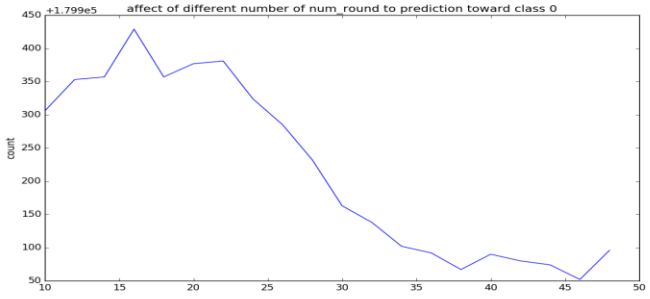
   預設參數：num_round = 30, max_depth = 5, gamma = 0, min_child_weight = 1, subsample=1

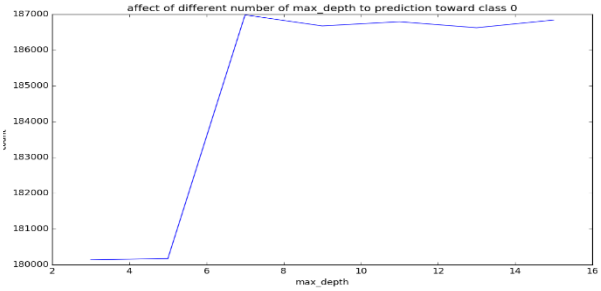
   p.s. 注意縱座標軸數值

num_round：從 10 開始，48 為止，步距 4

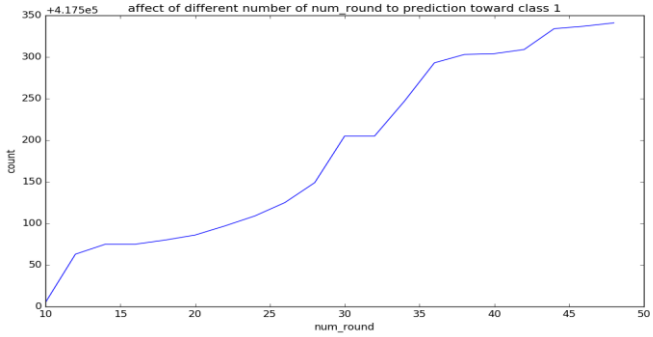
max_depth：從 3 開始，15 為止，步距 1

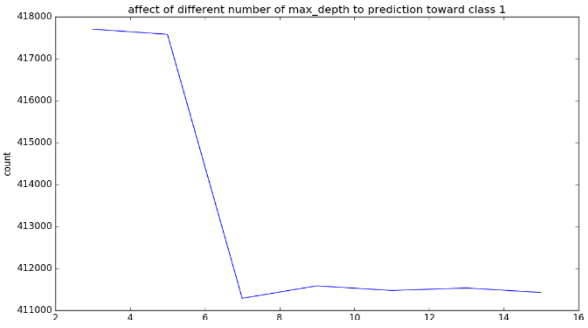
0



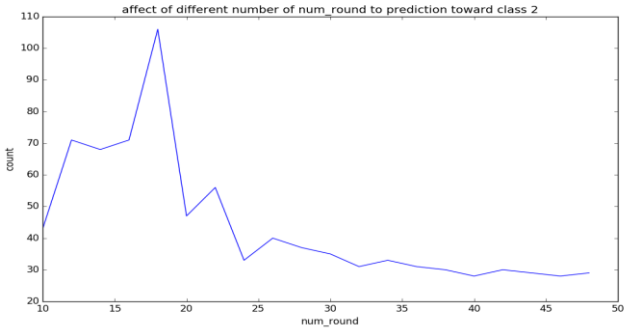


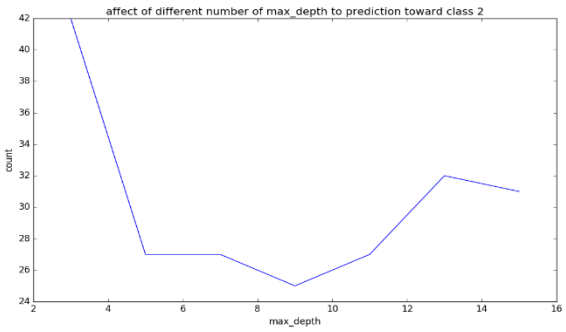
1



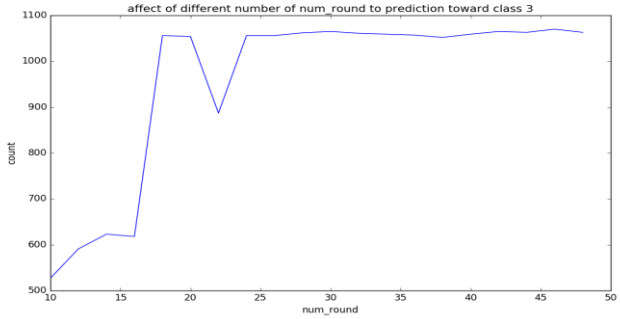


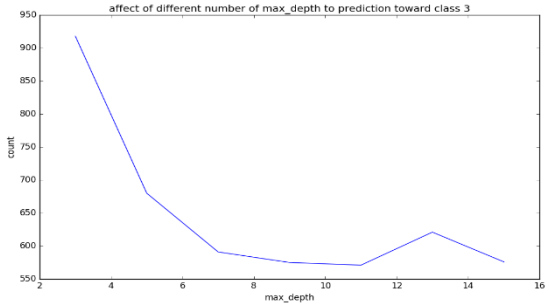
2



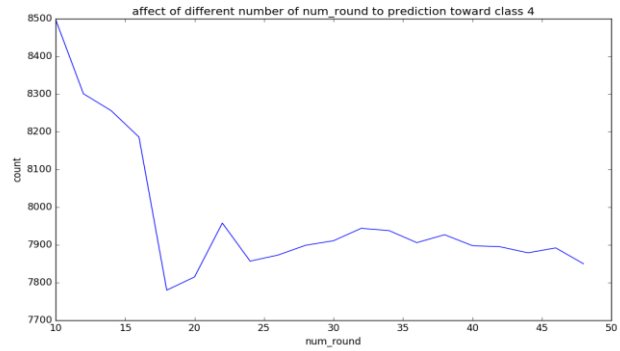


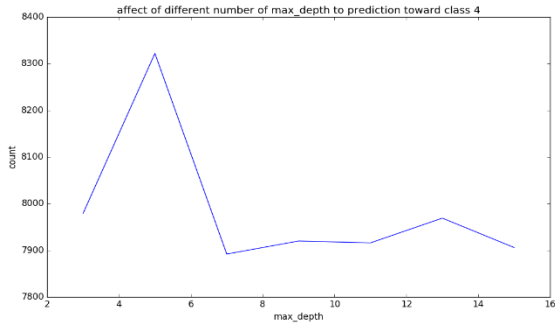
3





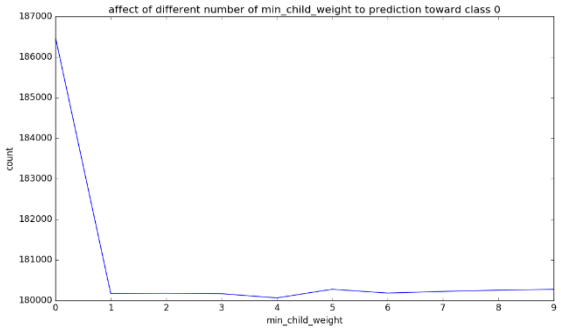
4



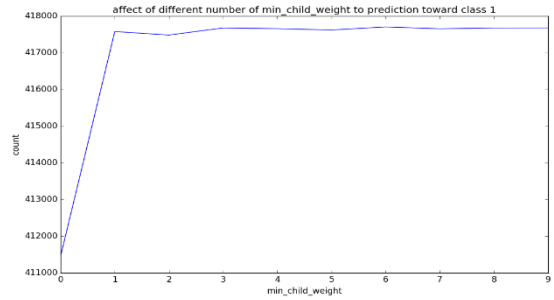


min_child_weight：從 0 開始，9 為止，步距 1

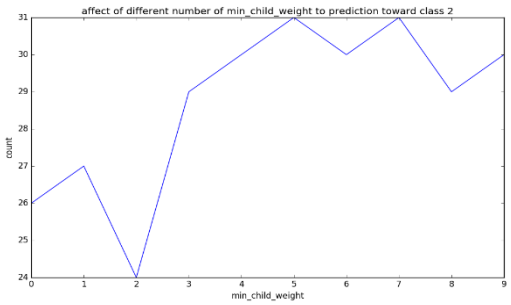
0



1

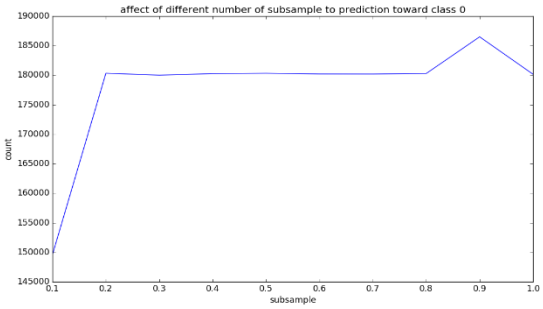


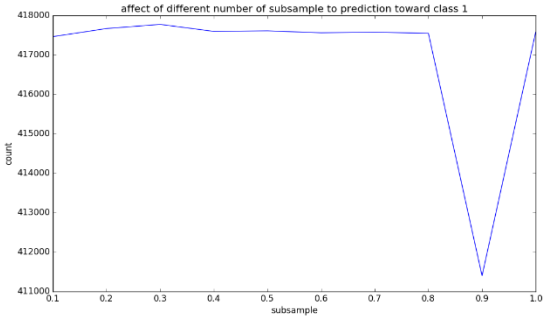
2

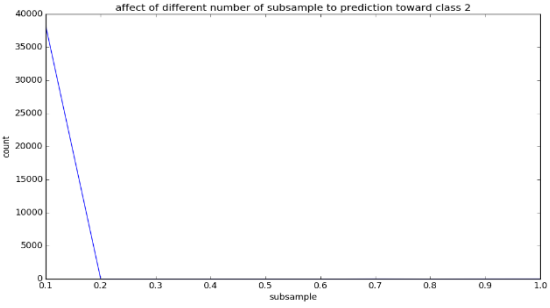


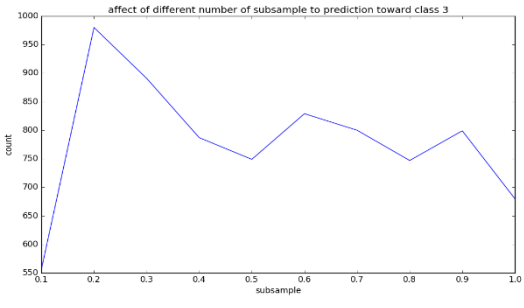
3

subsample：從 0.1 開始，1 為止，步距 0.1

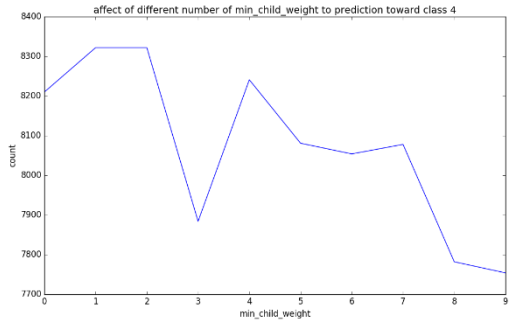


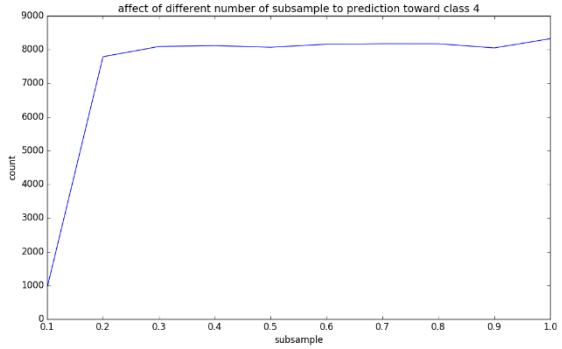






4

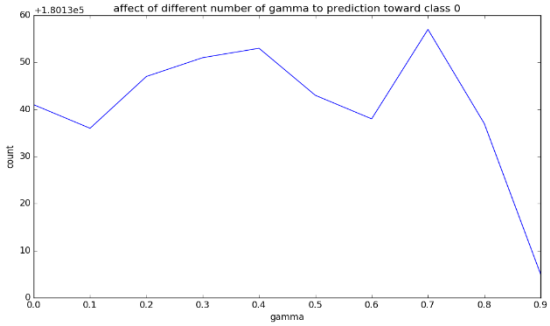


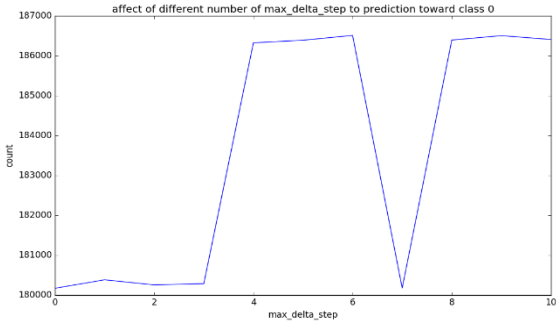


gamma：從 0 開始，0.9 為止，步距 0.1

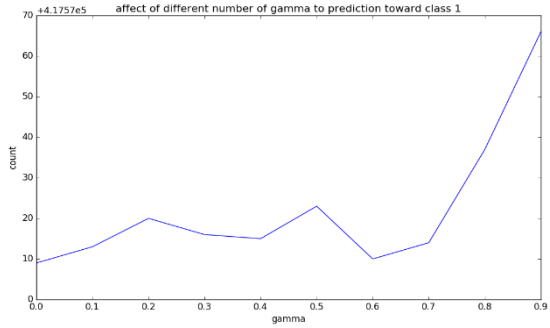
max_delta_step：從 0 開始，到 1 為止，步距 1

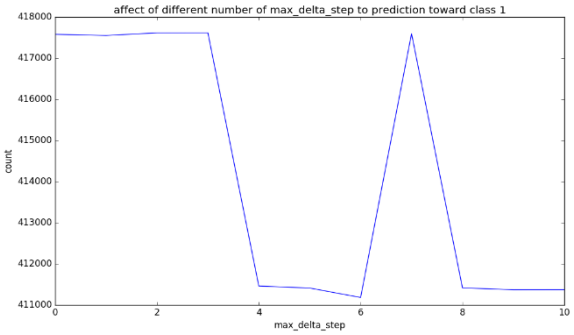
0



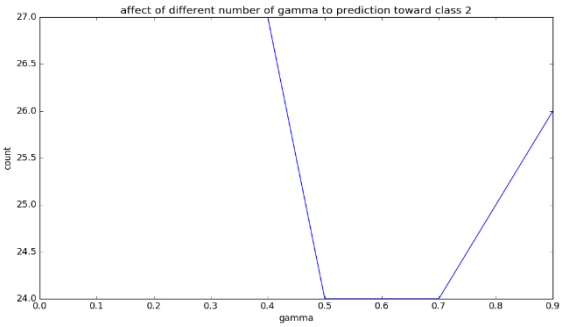


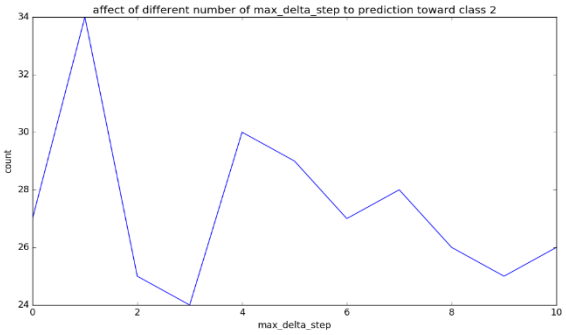
1



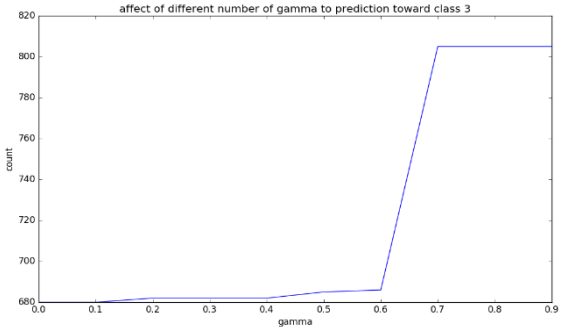


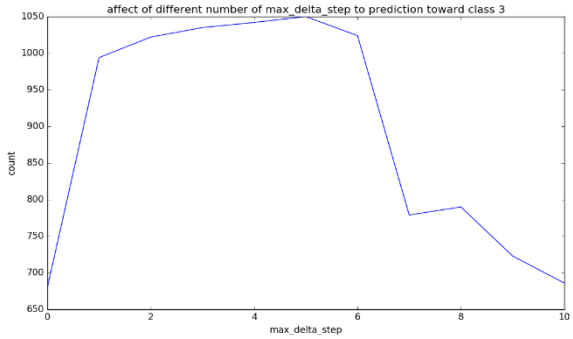
2



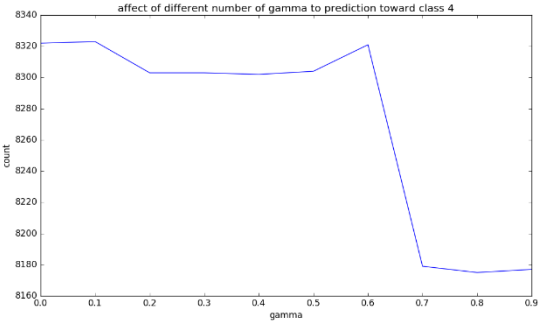


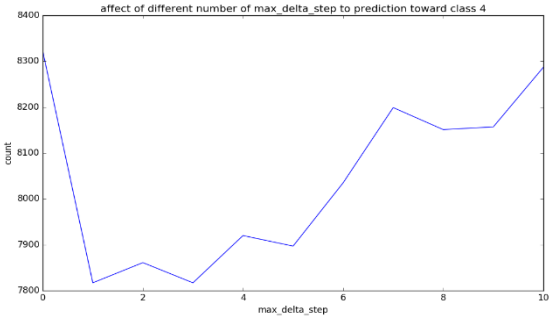
3





4





討論：

num_round：

class 0 一開始數量很高, 隨著 num_round 越多（大概 35 以上）才逐漸接近應該有的分佈（43220)

，趨勢與 class 1 正好相反，而且圖形與 class 1 接近互補，原因我們推測是因為 class 0 和 class 1 的 feature 較為接近，而與 class 3, 4, 5 較不接近，所以當沒有足夠多 tree 的時候，class 0 和 class 1 有許多 data 被誤判

class 1 與 class 0 呈現相反的走向，也是在 num_round 越多（大概 35 以上）才逐漸收斂到應有的值（157556）

class 3 和 class 4 的圖也呈現相反的走向, 形狀也大致互補（num_round = 22 處 class 3 有凹槽而 class 4 有尖角）原因可能是因為 class 3 與 class 4 的 feature 較接近, 而與其他 3 個 class 較不接近，因此可能在 num_round=22 時有多筆 testing data 被誤判

max_depth：

在 max_depth=5 之後就趨於穩定，max_depth 低時，根據 feature importance,取的 feature 分別是 Count,destination bytes, service, logged in,dst host count.推測原因應該是 class 0 的 feature 在 count,destinatioin bytes,logged in,dst host count 上和其他 class 相當接近，因此在 max-_depth 低時相當接近，必須倚賴其他 feature 才能辨認出 class 0,根據前面 violin plot 的觀察，也確實是如此，在前 5 個 feature 中，class 0 的集中處也分別都有其他的 class 也在此集中

min_child_weight：

我們預測 min child weight 的 t 影響應該對 class 2 最大，因為 class 2 的數量最少，沒想 min child weight 也對 class 3 和 class 4 影響相當大

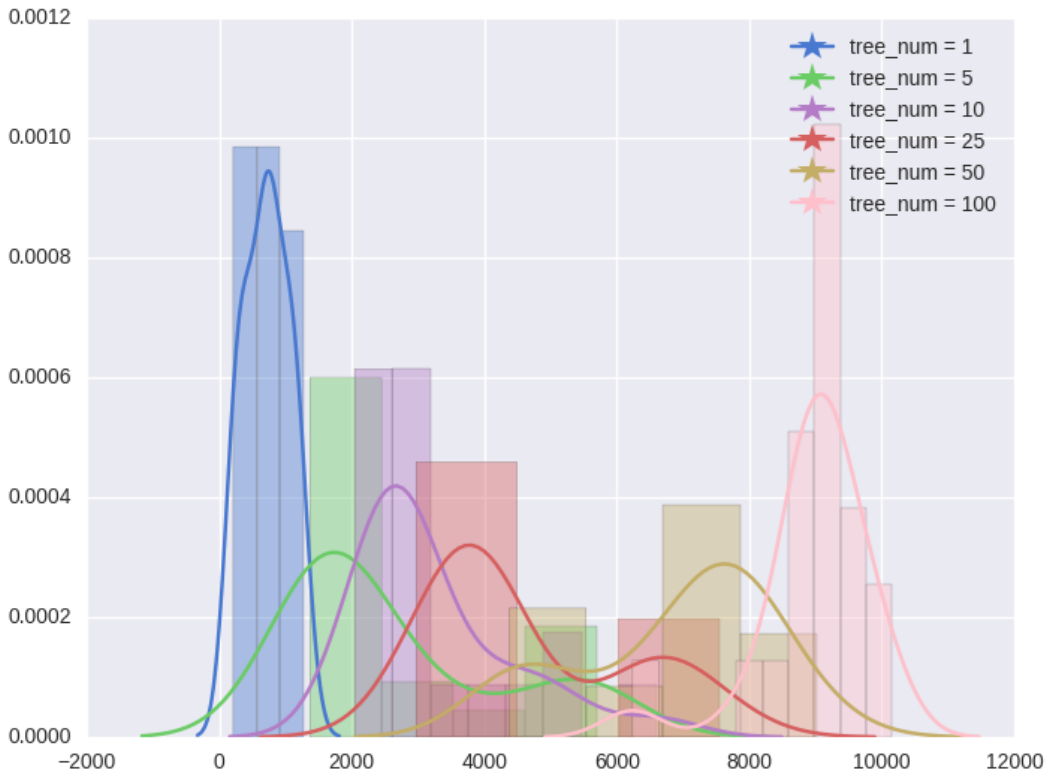
對於 class 0 和 class 1 來說，因為數量比較多，所以調整 min child weight 影響不大

sub_sample：

2. Randomforest 的隨機性：

   聽了前五名的組上台報告之後，我們決定採用其中一組的方法更改我們的 label : 用 randomforest 一個 randomforest predict label , 如果 predict 出來 3 的機率大於 0，就把原本 xgboost predict 出來的 label 改為 3。同時，為了接近 testing data 的 distribution，調整了樹的數量使 predict 出來 3 的機率大於 0 的個數越接近 8568 越好，一凱使用的是 tree number = 27，但只有極少的機率 predict 出足夠多的 3，所以做的以下實驗探討分布不同 tree number 時 predict 出來機率大於零 testing data 個數的分佈。分為 6 種 tree number,每種 tree number 紀錄 20 次。

	Tree num = 1	Tree num = 5	Tree num = 10	Tree num = 25	Tree num = 50	Tree num = 100
Index =0	527	2882	2641	3801	8114	9081
Index =1	304	5388	5022	3571	7613	9694
Index =2	1111	1798	3207	3976	7172	8782
Index =3	694	1619	2563	4138	6773	9893
Index =4	805	1668	2432	7546	6619	6239
Index =5	790	5250	5022	6680	5276	10153
Index =6	1141	1607	6603	3920	6319	7925
Index =7	768	1415	2618	3545	7814	8726
Index =8	265	1369	2045	6620	7658	9015
Index =9	1148	3962	2598	3444	8377	8824
Index =10	731	1498	2255	4028	4413	8989
Index =11	424	1592	2640	4011	7410	9221
Index =12	260	1931	2680	4342	8142	9096
Index =13	1276	1485	3172	3390	4574	9730
Index =14	390	5539	4340	3741	7849	8597
Index =15	1029	5702	2483	2972	4376	9250
Index =16	747	2743	2566	6546	7518	8528
Index =17	210	1490	4125	3724	4674	9059
Index =18	670	2363	3175	6210	9023	9393



討論：

tree num 由小到大時，data 的分布在比較大的地方，而且 standard deviation 除以 mean 較小，而且較高。這應該是因為 tree num 較大時，只要有一棵樹 predict 為 3，3 的 prabability 就會大於零，因此樹較多的話比較可能讓 predict 大零的樹較多，也比較穩定。