ML2016 Cyber Security Attack Defender

Team name: Nihon Ichiban
Members: B02901032 張桓誠
B02901039 黃馨平
B02502108 陶昇永
B02502019 朱晉德

Preprocessing/Feature Engineering:

- Discrete feature 使用 1-hot encoding
  將 Column 1:3、Column2:70、Column3:11 維的 feature 做 1-hot encoding
  原本 41 維的 feature 變成 122 維的 Feature
- Continuous Feature 有嘗試使用 Normalization
- 41 個 Attack Type 直接換成 5 個 Type (Normal、DOS、R2L、U2R、Probing)
- 有嘗試移除 Training Set 裡面重複的資料
- Stacking 時，為了將 Training set 分成 Train. Data 和 Valid. Data
  使用 Map 將 Data 按照順序排列後，奇數取為 Train. Data，偶數取為 Valid. Data，

如此可以確保 Valid. data 和 Train. data 的分布幾乎與原本的 Training data 相似，不會有資料不平均的現象 (Train.都切到某一類 Class 0，Valid.切到另一類 Class 0)。

Model Description

以下從我們一開始採用的 Model 開始介紹，隨著時間演進，我們也嘗試了更多更不一樣的 Model。

Model 1: Neural Network

直接套用 Keras 的 Neural Network。我們嘗試了不同結構的 Neural Network，詳細見實驗與討論。

Model 2: Basic Random Forest (and Decision Tree)

隨便丟 Neural Network，隨便過 Simple Baseline，但是 Hard Baseline 就哭哭了！這種情況下只好先亂試其他 Model，結果一試就過啦！直接使用 Sci-kit learn 的

Decision Tree 和 Random Forest，沒有特別調整 Feature 之類的，所以稱為 Basic Random Forest。在這個階段我們意識到不可能只靠 Model 超越其他人，因此開始針對 Training Set 數量不均的情況進行調整，做了 Resampling，包括 SMOTE 和 Down-sampling。詳細見實驗與討論。

Model 3: Decision Tree

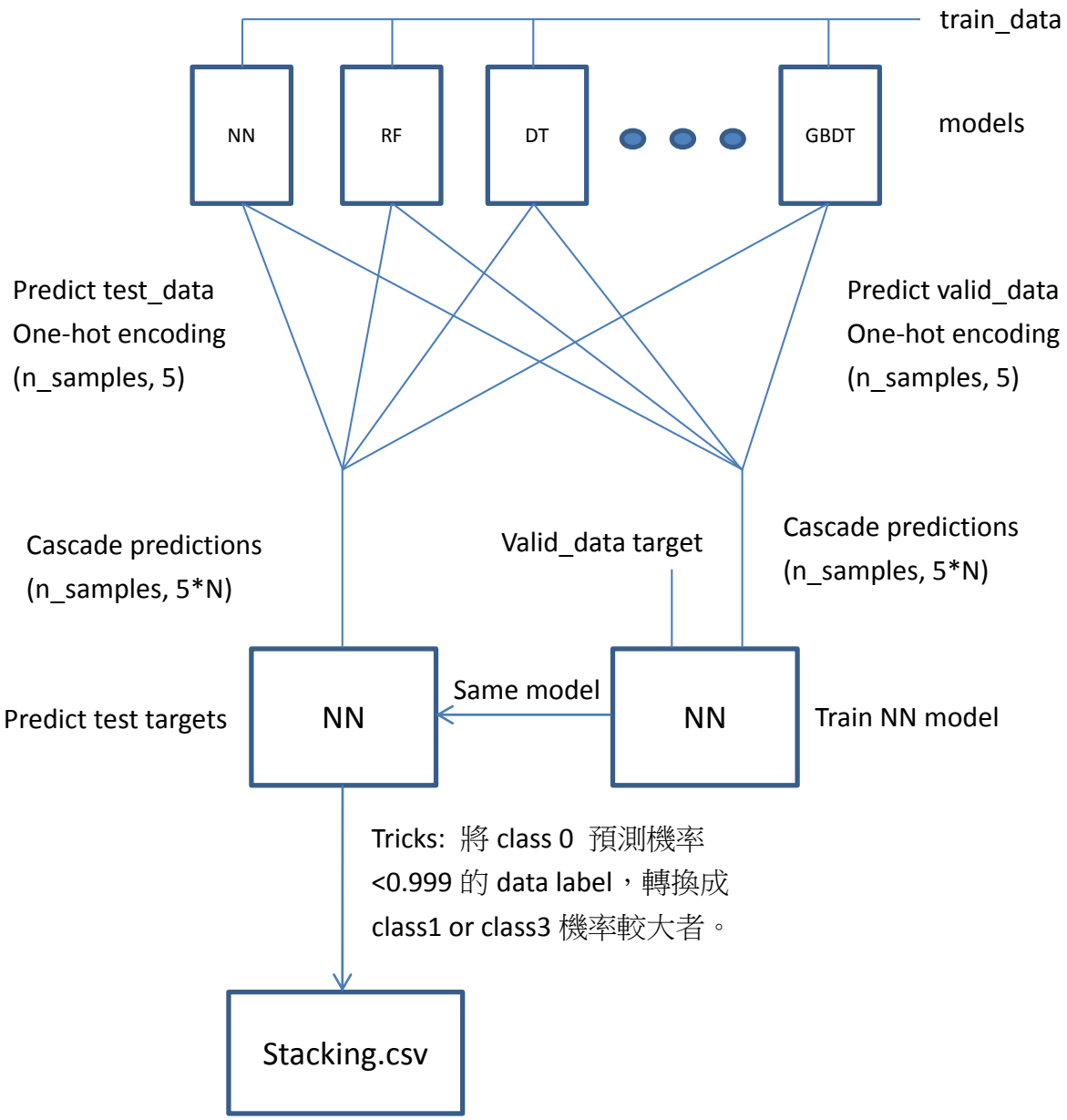
- 每次分割將目前的空間一分為二，使得每一個葉子節點都是在空間中不相交的區域，進行決策的時候，根據輸入樣本每一維 feature 的值，一步步往下使樣本落入 N 個區域中的一個。訓練時間複雜度較低，預測的過程比較快速，但容易 Over-fitting。

而我們使用了兩種與 Decision Tree 加上 Ensemble 的方法：

- Random Forest(Decision Tree + Bagging)
  對數據進行採樣，採用有放回的方式。假設輸入樣本為 N 個，那麼採樣的樣本也為 N 個。從 M 個 Feature 中，選擇 m 個(m << M)。對採樣之後的數據使用完全分裂的方式建立出決策樹(節點無法繼續分裂或者裡面的所有樣本的都是同類)。建立很多樹，讓每一棵樹都判斷後 Voting。
- Gradient Boost Decision Tree(Decision Tree + Gradient Boosting)
  每一次建立模型是為了減少上一次建立模型的 Loss Function，因此我們將上一次的模型往 Loss Function 的梯度下降方向建立新的模型。如果模型能夠使 Loss Function 不斷下降，表示模型在改進。最後將先前的所有模型和現在的模型透過加權合在一起得到新的模型。

Model 4: Stacking method (best performance)

Kaggle score: 0.96898
Model description:如下圖所示：
Preprocessing: 將 Training data 切成 2 份，分別是 train_data 跟 valid_data。



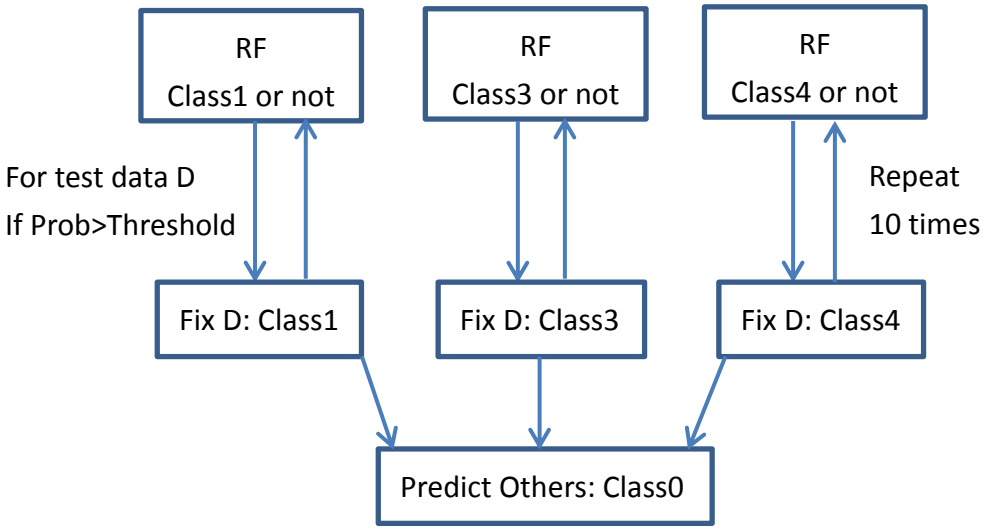
Model 5: Classifier for each class (best performance)

Kaggle score: 0.97220

Model description:

如下圖所示，分別對每個 class 都做一個 Classifier，例如用 RF 預測 data 是不是 Class 1，如果機率大於某個 Threshold，就固定這筆 Data 是 Class 1，重複建 RF 10 次，將所有機率大於 Threshold 的 Data 都固定為 Class 1。

對 Class 1、Class 3、Class 4 重複以上步驟做 Classifier 並 Predict 後，最後剩下的沒有被 Predict 為 Class 1、Class 3、Class 4 的 Data 即為 Class 0。如此可以盡量將 RF 判斷時，不足的 Class 1、Class 3 變多，過多的 Class 0、Class 4 變少。



Experiments and Discussion:

Neural Network

	structure	remove duplicate	method	score	distribution				
unique-int	32	O		0.95755	182890	416449	0	0	7440
1-hot	64	X		0.95666	182144	416424	0	63	8148
	32-16	X	Normalize	0.94796	171587	433374	0	0	1818
	32-32	O		0.95682	182766	416344	0	0	7669
		O	Normalize	0.93027	174330	403166	0	4737	24546
		O	SMOTE_FULL	0.95776	179011	416379	504	2218	8667
		X	SMOTE	0.95592	181835	416460	265	5	8214
	32-32-16	X		0.93702	165630	440325	0	0	824

- 有使用 SMOTE 時，較能夠 predict 出 class2 及 class3
- 有無使用 normalize 似乎差別不大，但似乎 class0 及 class3 的效果會好一些，class4 會差一些
- Network 越複雜效果並沒有越好，但可能是因為沒有 train 到底
- NN 幾乎無法分辨出 class2 及 class3
- NN 效果不佳的主因可能仍是 class0 過多
- NN 有時會 predict 出不合理的 distribution，如下所示:

Class0	Class1	Class2	Class3	Class4
486076	112217	778	0	7708

- 當 structure 是 32-32-16 時，可以發現 class0 的數量變準確，但 class3 及 class4 仍不足

Basic Random Forest

在這裡只探討 Resampling 的效果。固定使用 1-hot、沒有移除重複資料、Random Seed 固定。由於發現結果中 Class 0 的數量較多，所以想嘗試減少 Training Set 裡面 Class 0 的數量，因此 Down-sampling 的定義是讓 Training Set 中 Class 0 的數量降低至原來的多少比例，隨機選取資料並移除。SMOTE 則是讓其他 Class 的數量增加到數量最多的 Class 的多少比例。

Method	Score	Distribution				
-	0.96043	180480	417438	5	628	8228
SMOTE 30%	0.96052	180986	417525	26	387	7854
SMOTE 60%	0.95982	181266	417337	20	317	7838
SMOTE 100%	0.96050	180968	417632	15	367	7796
Down 90%	0.95982	181359	417433	4	279	7703
Down 60%	0.96038	181158	417611	4	404	7601
Down 30%	0.96055	180972	417698	9	315	7784

SMOTE 確實讓最容易被忽略的 Class 2 增加，但也只是微乎其微，還大幅減少了 Class 3 和 4 的數量，因此可以推測在 Testing Set 裡面，Class 2 和 3 的分布遠比 Training Set 裡面廣泛和分散，SMOTE 沒有辦法藉由增加現有資料的權重與數量更加準確的找出 Class 2 和 3，有點類似瓶頸效應。

Down-sampling 有隨機性，因此不容易從我們做的實驗中看出差異的顯著性，但如果硬要說明的話，可以推測移除少部分資料後，會影響判斷為其他類別的準確度，但移除更多資料後，則能保留 Class 0 的核心部分，減少例如邊界或 Outlier 對其他類別的影響。

Random Forest

	class weight	remove duplicate	method	vote	score	distribution				
1-hot	X	X		X	0.96043	180480	417438	5	628	8228
	O	X	class0 to 3	X	0.96047	178786	417729	11	2591	7662
		O	class23>0 class0<0.999	O	0.96450	166907	417653	437	8610	13172

- 在沒有使用任何 Trick 之前，Class 0 太多，Class 1、Class 2、Class 3 太少
- 在加入 Trick 之後，Class 0 減少但仍略多，Class 2、Class 4 增加但略多
- Class weight formula:

$$\text{class weight} = \frac{\#\text{ of class in testing data}}{\#\text{ of class in training data}}$$

➔應盡量將 class0 分到 class1 和 class3，class4 原本數量已差不多

Decision Tree

	remove duplicate	method	score	distribution				
1-hot	X	-	0.96135	176676	417903	54	1033	11113
	X	Remove class 2 and 3	0.95941	176865	418848	0	0	11066
	O	Move class0 to others	0.95950	174225	418216	346	2492	11500
	O	-	0.96088	178717	416827	90	1276	9869
	X	Random seed 32	0.96040	179218	417333	34	904	9290
	X	Random seed 5	0.96049	178025	418560	15	1297	8882
	X	Random seed 4	0.96012	176382	418422	58	1303	10614
	X	Ramdom seed 3	0.96111	176771	417890	23	891	11204
	X	min_samples_split 0.001	0.95963	177960	417533	13	629	10644
	X	min_samples_split 0.05	0.96051	180584	417489	64	399	8243
	X	Max depth 10	0.95858	182229	417132	10	0	7408
	X	Criterion: entropy	0.96050	180011	417482	151	591	8544
unique-int	O	-	0.96021	177997	417338	23	900	10521

- Class 0 太多，Class 1 太少，Class 3 太少，似乎是效果不佳的主因
- 有沒有 remove duplicate，有沒有 1-hot，似乎差別不大
- Class 4 都略多
- 想不出要如何改善的時候，為了不要浪費 Submission，不投白不投，我們也嘗試更換 Random seed，看看會不會有好手氣，從結果可見 seed 真的是可遇不可求。
- Sci-kit learn 的 Decision Tree 還沒有 Pruning 的功能，但為了減緩 Overfitting 的狀況，我們嘗試限制 Tree 生長的方式，例如分支條件與深度限制，結果沒有明顯改善。猜測在 Testing Set 裡面，有些在很淺層的 Feature 就沒辦法將 Class 0 和 1 分得很清楚，讓 Class 0 的數量大幅增加了，所以後面的修剪樹枝都沒什麼功效。

Others

method	score	distribution				
Bagged decision tree	0.96075	180614	417672	12	398	8083
K-nearest neighbor	0.95913	180941	417474	16	118	8230
Extra tree	0.96051	180821	417638	11	463	7846
Stochastic gradient descent	0.72855	81466	389209	4144	0	131960
Gradient boosting tree	0.95164	179185	423960	0	0	3634

- Stochastic gradient descent 效果不佳
- Bagged decision tree、K-nearest neighbor、Extra tree 都讓 class0 太多、class1 及 class3 太少，與其他方法結果大致差不多

Ensemble

Method	Score	Distribution				
voting (DT_Remove,GTB_Remove,DT,GTB)	0.96183	179994	417927	7	977	7874
voting, class0<0.999 to other	0.95973	174975	417496	300	2492	11516
stacking(NN,RF,GTB,DT)	0.96202	180143	418286	5	833	7512
stacking(NN,RF,GTB,DT)+trick	0.96898	172194	420009	8	7063	7505
stack	0.96034	180548	418975	0	421	6835
	0.96117	179301	417209	3	1741	8525
vote	0.96183	179057	417934	9	979	8800
	0.96131	177448	419932	14	987	8398
	0.95956	181178	418416	0	308	6877
	0.96196	179504	417958	3	982	8332

Gradient boosting:

- 優點：能 predict 出較多的 class 1 數量，可補足其他 model predict class 1 數量不夠多的問題。

Voting:

- 將各個 model predict 的結果進行 majority vote，得到各種 model 互補的結果。但要注意多數決的投票仍有可能讓少數 model 正確預測的結果被覆蓋掉，因此我們最終的方法是：對於相同的 model 做多次取 voting，而非對於不同的 model prediction 取 voting。

Stacking:

- 可以有效的讓各個 model 發揮作用，即使有表現較差的 model，也可以透過第二階段 train validation set 時，將其影響力淡化，相較於 voting 對於表現較差的 model 無法有效預測而言，stacking 是相當好的 ensemble method。
- 可以發現透過 stacking 和 voting 可以將 class4 數量變得比較穩定且準確，class1 和 class3 的數目也較為穩定，但是數量仍不足

Conclusion

- 嘗試手動調整結果以符合 Public Set 其實是很糟糕的方法，但在這次 Project 中相當有效，是許多組成功衝分數的關鍵。不過這種調整方法也讓我們對於資料在空間中的分布有些想像。
- 衝分數的方法很多，但讓我們可以更上層樓的大功臣，莫過於 Stacking 了，而且上次報告的組別都沒有提到！特別選取比較互補的結果們，投入 Neural Network 後再生成第二階段的模型，這美妙的模型讓我們的分數屢屢突進，好上加好！
- Unbalanced Dataset 在真實世界是相當容易發生的狀況，很多資料可能不太容易取得，或者事件發生機率本來就低，造成某些類別幾乎沒有資料可以訓練。這個 Project 讓我們清楚地了解到這件事，且讓我們在意識到這件事時彷彿看到一線曙光，想說這就是決勝的關鍵了，然而做完 Resampling 後發現其實沒什麼用處。不禁讓我們擔憂，如果真的遇到這種狀況該怎麼辦呢？關鍵少數的力量和影響無法忽視！生活真的很難啊。