

期末專題報告

林子翔、董皓文、謝秉翰

Section 1 - Team Information

題目：Cyber Security Attack Defender

組別：NTU_b02901080_GottaGetThat

組員：b02901069 林子翔、b02901080 董皓文、r04945025 謝秉翰

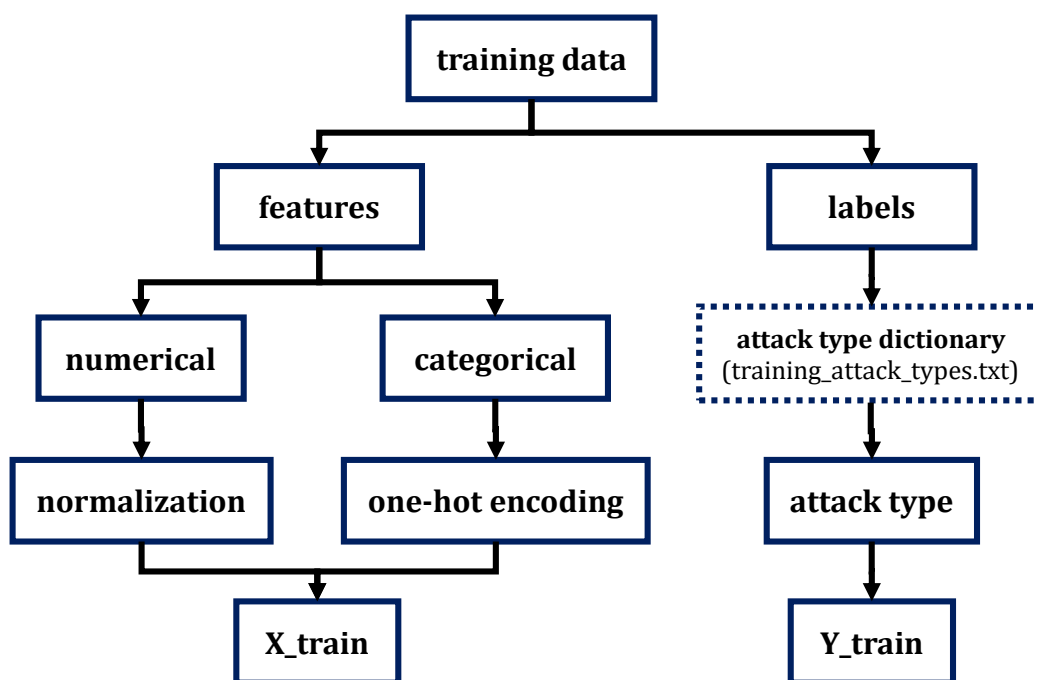
分工：

林子翔：進階 model 的設計 (second baseline)、DNN 模型層數與結點數的測試及討論。

董皓文：資料的前處理、特徵值的取出與轉換 (feature engineering)、初始 model 的設計 (first baseline)。

謝秉翰：初始 model 的改良、random forest 跟 support vector machine 的討論。

Section 2 - Preprocessing / Feature Engineering



圖一 前處理架構

Step1 - 特徵分類

首先我們按投影片所附特徵資訊表，將特徵分為數值特徵 (numerical features) 及類別特徵 (categorical features)，因為兩者所需進行的前處理不同。

由於我們使用的是 DNN 模型，故我們並沒有對 feature 進行篩選，讓 DNN 訓練時自動調整權重。各特徵的重要性在 Section 4 將進一步討論。

在此處我們發現有一個特徵 (# of outbound cmd) 在所有欄位的值皆為零，不具有用資訊，因此我們先將之移除。

Step2 - 特徵 / 標籤處理

數值特徵 (numerical features)

我們對數值特徵進行標準化，使不同特徵不因絕對數值的大小而在 DNN 有先天權重之差異 (在 weight 上面特別 dominate)。

類別特徵 (categorical features)

我們使用 one-hot encoding 將每個類別特徵轉換為多個 dummy bools，以利後續訓練使用。

標籤 (labels)

由於 training data 中的標籤是 attack subtype，我們要進行適當的轉換，才可作為 DNN 的 input。我們先利用 training_attack_type.txt，產生一個 python dictionary，將 subtype 對應到 type (例如 subtype_to_type['apache2'] = 'dos')。接著再將各個 attack type 編碼成對應的 code (根據題目規定的編碼)。

這裡我們發現 training data 中的資料大多是 DOS 攻擊，為了達到更好的準確度，我們對 DOS 的資料使用 subtype 去進行編碼 (即用 apache2, back, mailbomb 等 subtype 作為標籤的意思)，在最後產生 test data 的預測類別後，在轉換成對應的 attack type。由於其他三個 type 的攻擊 (u2r, r2l, probe) 的資料筆數沒有 DOS 多，所以我們沒有用 subtype 下去訓練。這個改變我們從 simple baseline 到 strong baseline 的重要改進。

Step 3 - 完成 X_train 及 Y_train

最後我們將處理過的數值特徵及類別特徵結合成 X_train，而編碼過的標籤作為 Y_train，送到下一階段的 DNN 模型。

Section 3 - Model Description

這邊我們使用了三種不同的演算法：

Deep Learning

我們使用 keras (Theano backend) 建立 DNN 模型。我們測試了不同 layer design 的方法 (在 Experiments and Discussion 的區域會加以討論)，以下是我們最後上傳 kaggle public score 最高的模型架構：

- 四個 hidden layer，node 數量分別為 500, 800, 250, 100，前三層使用 relu 作為 activation function，最後一層使用 sigmoid 產生最終分類。
- 使用 binary cross-entropy 作為 loss function
- 使用 adam (momentum + scaling) 的方式更新 gradient descent
- 每一層 hidden layer 後設定 dropout ratio = 0.5，避免 overfitting
- nb_epoch=10、batch_size=5000

Support Vector Machine Classification (SVC)

我們使用 scikit-learn Python package 中的 SVC 物件。在參數的設定上調整不同的 kernel 進行測試，測試的 kernel 有 radial basis function (RBF)、linear、sigmoid 三種。

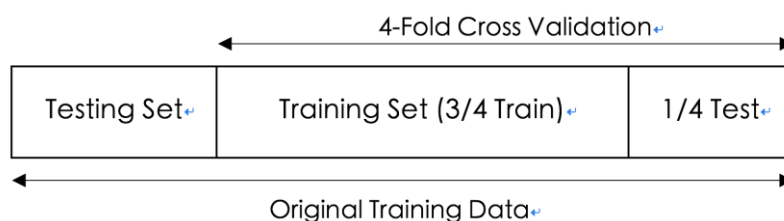
Random Forest Classification (RF)

我們使用的是 scikit-learn Python package 中的 RandomForestClassifier 物件。在參數的設定上調整不同樹的數量，測試的樹數量有 1000、5000、10000。

演算法比較

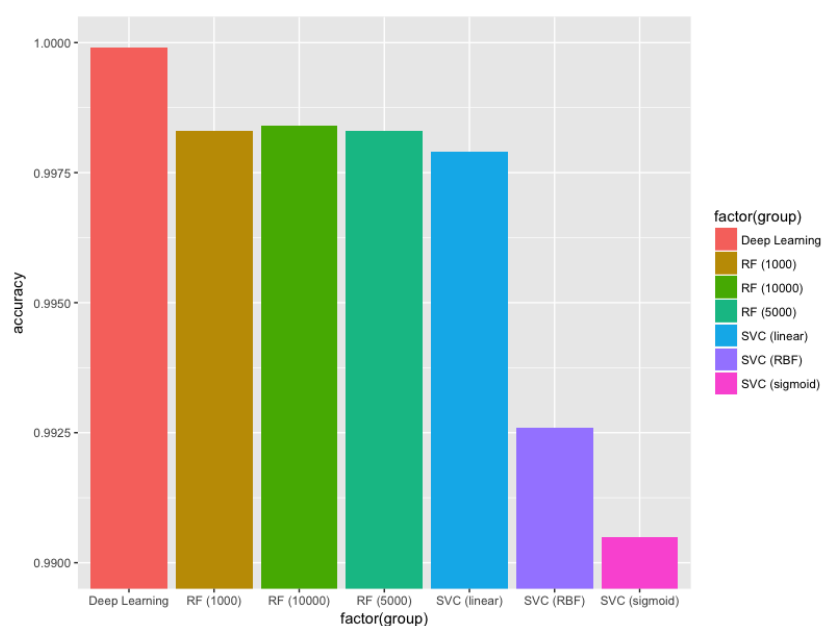
為了比較不同演算法的差異，我們這邊將原本的資料隨機取出 1000 個 sample 作為 independent testing set。（因為 kaggle 上的設定一天只能上傳 2 次，且在 deadline 以後就無法上傳了，所以不容易用 public score 有效率的比較不同演算法，因此自己設計 independent testing set。）

接著我們把剩下的資料作為 training set 進行以下分析。對於 Deep Learning 模型，我們自行切割 validation set (隨機取 1/4 的資料)，並以 keras 訓練時的 val_acc 監測器做為模型評估的標準。對於 SVM 模型及 RF 模型，我們利用 scikit-learn 進行 4-fold cross validation 來進行模型評估，示意圖如圖二所示。。

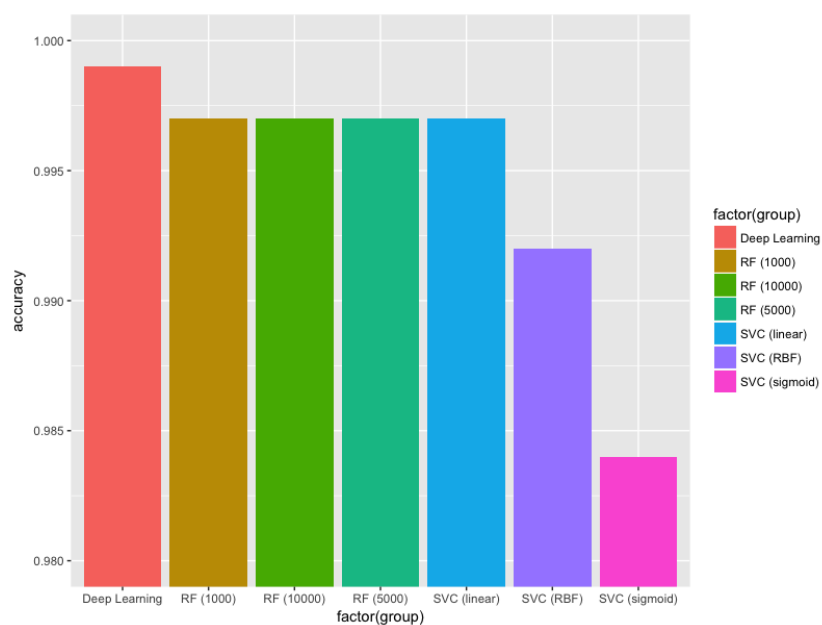


圖二 Training Set 資料分割示意圖

針對上述的三種演算法，我們檢驗了 validation 以及對 testingset 的 score (這邊使用 accuracy 作為評估的依據)，可以得到圖三、圖四結果。



圖三 Validation Set Accuracy 比較



圖四 Testing Set Accuracy 比較

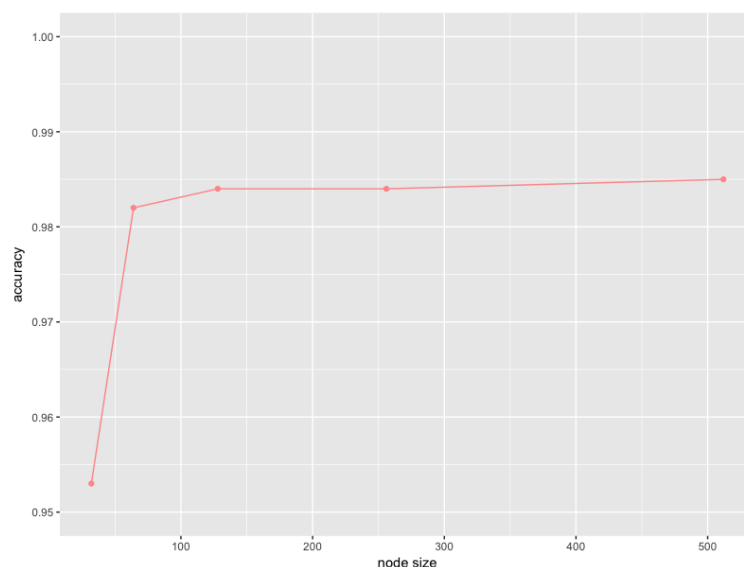
從結果看起來，我們所設計的 DNN model 對於 independent set 或 testing set 的表現都優於其他兩種演算法，因此最後選擇此模型結果上傳 kaggle，在 kaggle public set 得到 **0.96305** 的 accuracy。

Section 4 - Experiments and Discussion

此處為了節省時間以及計算複雜度，我們使用隨機挑選出的 50000 筆 sample 進行討論，並且利用 1/4 的 validation set 以及上述的 1000 筆 independent testing set 進行以下效能討論。

DNN hidden layers 內 node 數量之影響

固定使用一個 hidden layer，設定 nb_epoch=10, batch = 5000, activation function 為 relu。檢驗不同的 node 數對結果的影響，結果如圖五。從結果可以發現，node size 對於整個架構的設計具有蠻大的影響，差不多在 node 數量為 500 的時候，accuracy 可以收斂在 0.985 左右，而太小的 node（數量少於 100），則沒有辦法得到較好的 accuracy。

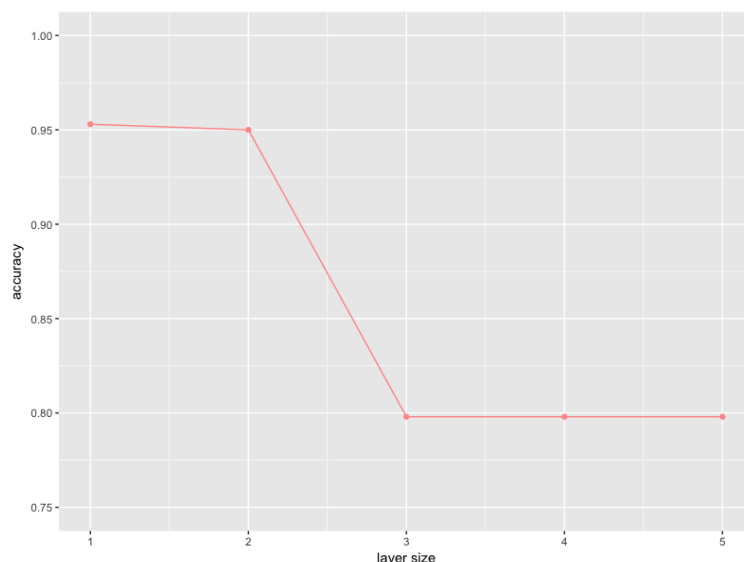


圖五 DNN node 數量對 testing set accuracy 的影響

DNN hidden layers 層數之影響

固定每個 layer 內的 node 數為 32，設定 nb_epoch=10、batch = 5000，activation function 為 relu。結果如圖六。從結果可以發現，越多的 layer，其實不代表會產

生越適合的 data。結合上面的 layer 深度與 node 數量的分析，經過不同次的嘗試後，我們最後選用的是 4 個 hidden layer，所使用的 layer 數量分別是 500、800、250、100。



圖六 DNN hidden layers 層數對 testing set accuracy 的影響

DNN dropout layer 之影響

固定一個 hidden layer，裡面的 node 數為 32，設定 nb_epoch=10, batch = 5000，activation function 為 relu。檢驗有設定 dropout = 0.5 對結果的影響。可以發現如果沒有設定 dropout，在 validation set 的 accuracy 可以達到 0.9962，但是在 testing set 的表現為 0.984。而如果設定了 dropout = 0.5，可以觀察到 validation set 的 accuracy 有一定的下降 (0.9856)，而且在 testing set 的表現為 0.957，兩者的差異如圖七、八。由此可發現，設定 dropout 能夠減少 model 的 overfitting 現象，讓我們更能夠利用 validation set 來檢驗我們的 model 在真實的應用情況的表現。

(a)

```

Train on 37500 samples, validate on 12500 samples
Epoch 1/10
37500/37500 [=====] - 0s - loss: 0.0740 - acc: 0.9854 - val_loss: 0.0605 - val_acc: 0.9884
Epoch 2/10
37500/37500 [=====] - 0s - loss: 0.0543 - acc: 0.9905 - val_loss: 0.0442 - val_acc: 0.9957
Epoch 3/10
37500/37500 [=====] - 0s - loss: 0.0401 - acc: 0.9952 - val_loss: 0.0331 - val_acc: 0.9957
Epoch 4/10
37500/37500 [=====] - 0s - loss: 0.0308 - acc: 0.9953 - val_loss: 0.0260 - val_acc: 0.9959
Epoch 5/10
37500/37500 [=====] - 0s - loss: 0.0250 - acc: 0.9955 - val_loss: 0.0215 - val_acc: 0.9960
Epoch 6/10
37500/37500 [=====] - 0s - loss: 0.0213 - acc: 0.9956 - val_loss: 0.0187 - val_acc: 0.9961
Epoch 7/10
37500/37500 [=====] - 0s - loss: 0.0189 - acc: 0.9958 - val_loss: 0.0168 - val_acc: 0.9965
Epoch 8/10
37500/37500 [=====] - 0s - loss: 0.0174 - acc: 0.9960 - val_loss: 0.0155 - val_acc: 0.9965
Epoch 9/10
37500/37500 [=====] - 0s - loss: 0.0162 - acc: 0.9960 - val_loss: 0.0145 - val_acc: 0.9963
Epoch 10/10
37500/37500 [=====] - 0s - loss: 0.0154 - acc: 0.9958 - val_loss: 0.0137 - val_acc: 0.9962

```

```

Train on 37500 samples, validate on 12500 samples
Epoch 1/10
37500/37500 [=====] - 1s - loss: 0.6819 - acc: 0.7315 - val_loss: 0.6618 - val_acc: 0.9184
Epoch 2/10
37500/37500 [=====] - 0s - loss: 0.6433 - acc: 0.8907 - val_loss: 0.6078 - val_acc: 0.9184
Epoch 3/10
37500/37500 [=====] - 0s - loss: 0.5738 - acc: 0.9079 - val_loss: 0.5089 - val_acc: 0.9184
Epoch 4/10
37500/37500 [=====] - 0s - loss: 0.4604 - acc: 0.9144 - val_loss: 0.3686 - val_acc: 0.9184
Epoch 5/10
37500/37500 [=====] - 0s - loss: 0.3305 - acc: 0.9163 - val_loss: 0.2466 - val_acc: 0.9184
Epoch 6/10
37500/37500 [=====] - 0s - loss: 0.2381 - acc: 0.9167 - val_loss: 0.1836 - val_acc: 0.9184
Epoch 7/10
37500/37500 [=====] - 0s - loss: 0.1888 - acc: 0.9175 - val_loss: 0.1452 - val_acc: 0.9184
Epoch 8/10
37500/37500 [=====] - 0s - loss: 0.1530 - acc: 0.9253 - val_loss: 0.1151 - val_acc: 0.9270
Epoch 9/10
37500/37500 [=====] - 0s - loss: 0.1263 - acc: 0.9457 - val_loss: 0.0930 - val_acc: 0.9747
Epoch 10/10
37500/37500 [=====] - 0s - loss: 0.1057 - acc: 0.9631 - val_loss: 0.0759 - val_acc: 0.9856

```

(b)

圖七 (a)有 dropout 及(b)沒有 dropout 的 validation accuracy

利用 Random Forest 找出重要 features

由於在測試 random forest 的時候發現 accuracy 其實表現得不差(而且不管測試幾棵樹，表現都相當的一致) 因為 random forest 的演算法是隨機取出 features 並且排列組合決定最能夠區分資料的 features 再透過 decision tree 的 ensemble 來進行結果的預測因此我們推斷這次的 training set 的 feature 與 label 之間的關係應該還蠻直接的，所以我們便利用 scikit-learn 中的 ExtraTreesClassifier 來檢驗 feature 的重要性。得到的結果如下圖六所示：

```

[ 8 18 64 55 19 37 27 24 120 31 38 39 29 90 28 20 116 33
 21 34 30 26 22 35 36 85 23 95 25 32 112 52 1 54 2 115
 60 9 4 6 106 0 121 113 58 84 17 59 114 12 51 44 15 101
100 117 104 74 79 57 77 80 92 48 89 40 108 105 13 72 107 61
 73 45 83 53 81 70 69 76 63 98 88 94 75 82 97 93 14 10
 46 99 47 110 66 50 78 49 56 71 42 109 87 96 119 111 118 3
 5 7 62 11 16 103 102 68 91 41 43 67 65 86]

```

圖八 所有特徵以重要性高到低排列

因為我們的 feature 有經過 encoding，對回去原本的 feature header 的話，可以發現前五名重要的 feature 分別為 number of compromised, error rate, service, srv error rate 與 dist host error rate。

Future Work