Machine Learning Final Project

1. Description

- 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)、deep learning layer 與 node 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)、初始的 model 設計 (first baseline)。
  - 謝秉翰：初始 model 設計的改良、random forest 跟 support vector machine 的討論。

2. Preprocessing / Feature Engineering

針對資料的前處理，我們單純的把所有的 feature 納入考慮。並沒有特定地去除掉什麼 feature，並利用這樣的設定去進行一次模型的訓練，因為得到了不錯的結果，所以之後也並沒有針對 feature 再去做篩選（但 feature 的重要性有在 Experiments and Discussion 的區域進一步討論）針對不同的 data type，我們則做了以下的處理：

- 數值特徵 (numerical features)進行 normalization，使得各個 feature 之間的數值不會因為數值大小而在 weight 上面特別的 dominate。
- 類別特徵 (categorical features)我們利用 one-hot 的 encoding 把它轉成適當的 feature 形式。

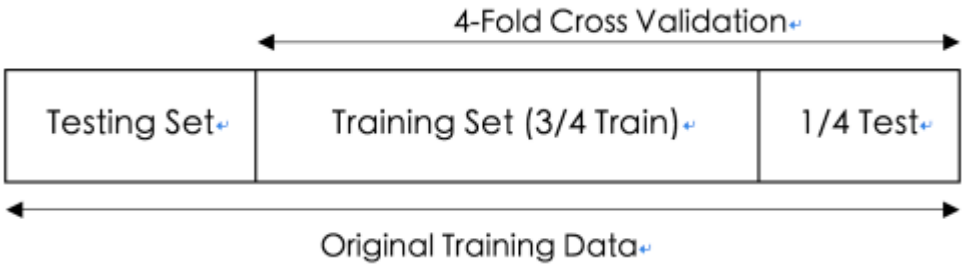
3. Model Description

這邊我們使用了三種不同的演算法：

- Support Vector Machine Classification (SVC)
  我們使用的是 scikit-learn Python package 中的 SVC 物件。在參數的設定上只有調整不同的 kernel 進行測試，測試的 kernel 有 radial basis function (RBF)、linear、sigmoid 三種。

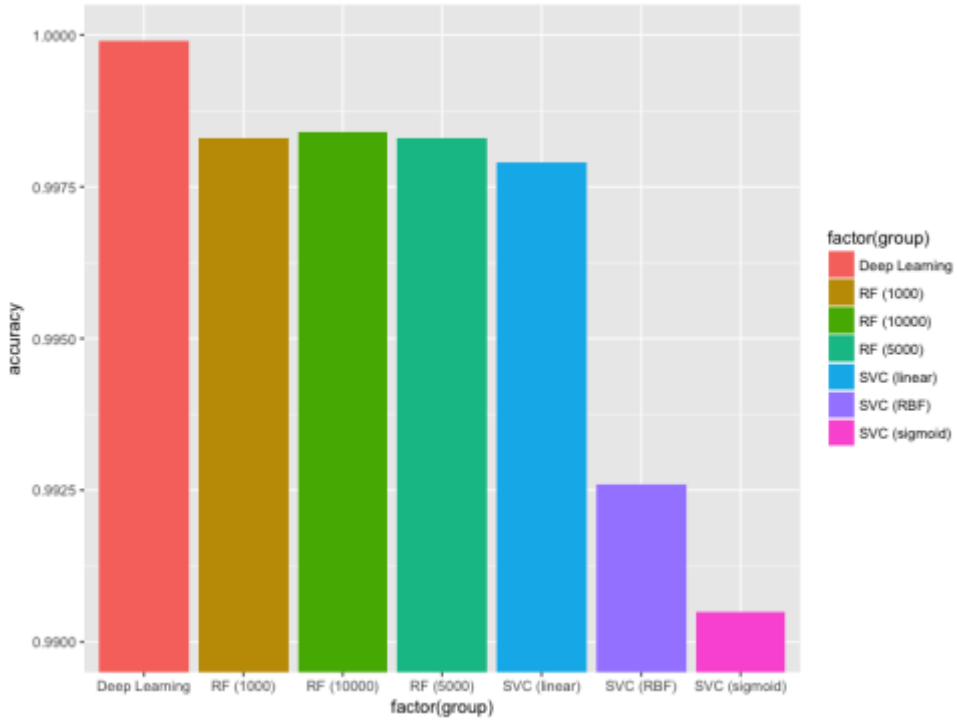
- Random Forest Classification (RF)
  我們使用的是 scikit-learn Python package 中的 RandomForestClassifier 物件。在參數的設定上只有調整不同樹的數量，測試的樹數量有 1000、5000、10000。
- Deep Learning
  我們使用的是 keras 搭配 tensorflow backend。這邊我們測試了不同 layer design 的方法（在 Experiments and Discussion 的區域會加以討論）以我們最後上傳 kaggle 分數最高的 model 來說。我們設計的架構是如下：
  - 四個 hidden layer。
  - 四個 hidden layer 的 node 數量分別為 500、800、250 跟 100。
  - 每一個 hidden layer 都使用 relu 的 activation function。
  - 最後一層的分類使用 sigmoid function。
  - 使用的 loss function 是 binary cross-entropy。
  - 利用 adam（momentum + scaling）的方式更新 gradient descent。
  - 為了避免 model 的 overfitting，會在經過每一層 layer 設定 dropout 一半的 input。
  - nb_epoch=10、batch_size=5000。

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（這邊自己設計 independent testing set 的原因是因為 kaggle 上的設定每一天都只能上傳 2 次，而且在 deadline 截止以後就無法上傳了，因此 public 的 testing set 比較不容易用來有效率的比較不同的演算法）
接著我們把剩下的資料作為 training set。在 SVM 跟 RF 兩種方法我們使用 4-fold cross validation 來作為模型的評估。Deep Learning 則是用 keras 內建的 validation set 來進行模型的評估（取 1/4 的資料）示意圖如圖一。

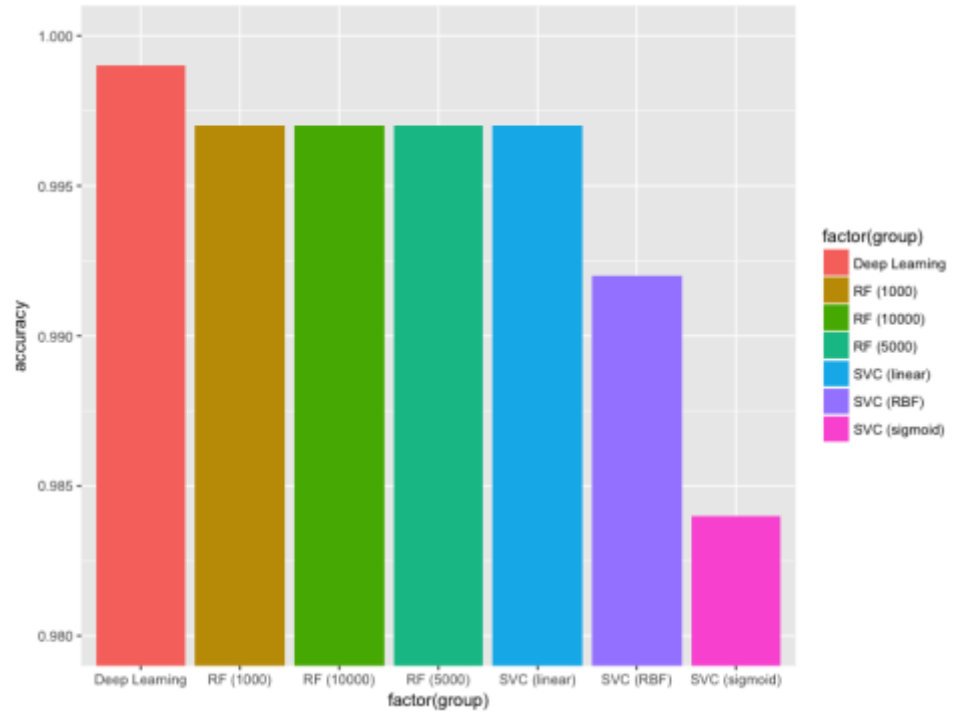


圖一、Training Set 資料分割示意圖

針對上述的三種演算法，我們檢驗了 validation 以及對 testing set 的 score (這邊使用 accuracy 作為評估的依據)，可以得到如下圖二、圖三結果。



圖二、Validation Set Accuracy 比較



圖三、Testing Set Accuracy 比較

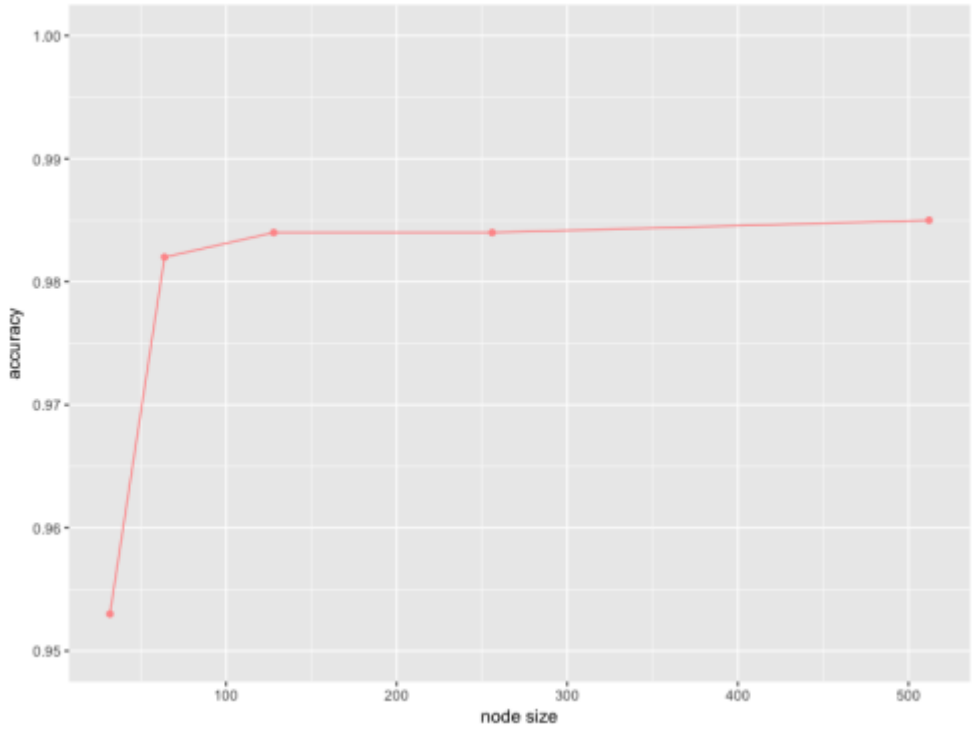
從結果看起來，我們所設計的 deep learning model 對於 independent testing set 的表現都優於其他的演算法，因此最後選擇使用此模型進行 kaggle 的上傳結果。最後在 kaggle 的 public data 可以得到 0.96305 的 accuracy。

4. Experiments and Discussion

這邊的實驗設計與討論，為了節省時間以及計算的複雜度我們只隨機使用 50000 筆 sample 進行討論，並且利用 1/4 的 validation set 以及上述的 1000 筆 testing set 來進行效能的評估。

- **Layer 內 node 數量對結果的影響**

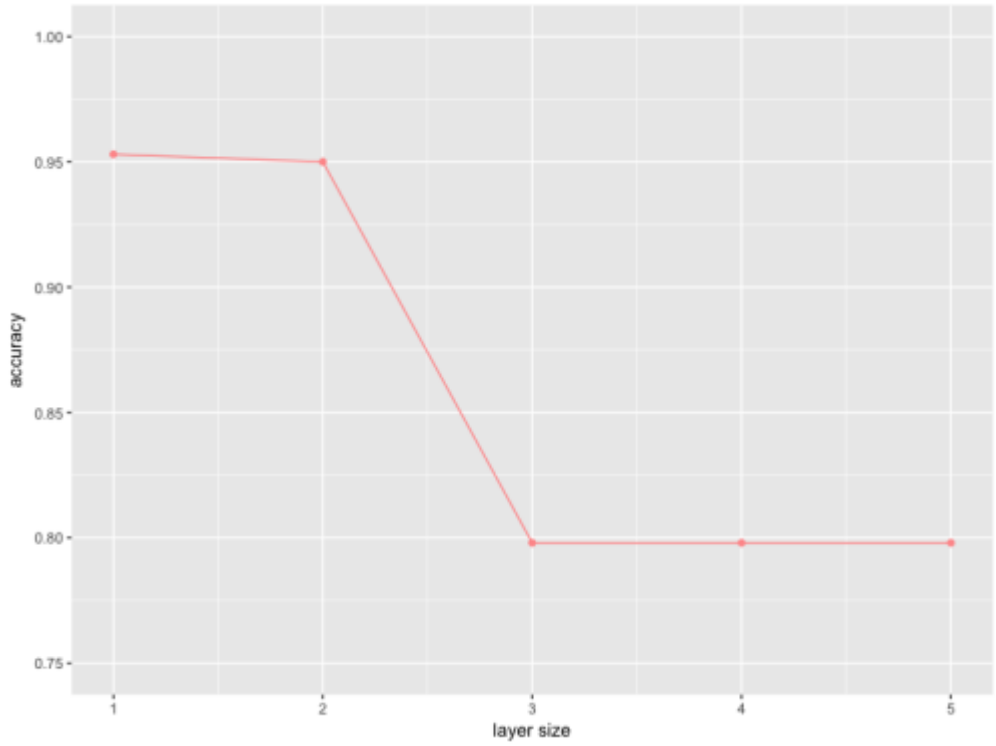
  固定使用一個 hidden layer，設定 nb_epoch=10、batch = 5000，activation function 為 relu。檢驗不同的 node 數對結果的影響，，結果如下圖四。從結果可以發現，node size 對於整個架構的設計具有蠻大的影響，差不多在 node 數量為 500 的時候，accuracy 可以收斂在 0.985 左右，而太小的 node (數量少於 100)，則沒有辦法得到較好的 accuracy。



圖四、Node Size 對 Testing Set Accuracy 的影響

- **Layer 深度對結果的影響**

  固定每個 layer 內的 node 數為 32，設定 nb_epoch=10、batch = 5000，activation function 為 relu。結果如下圖五。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。



圖五、Layer Size 對 Testing Set Accuracy 的影響

- **設定 Dropout 與否對結果的影響**

  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。

- **Random Forest 所找到重要的 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]
```

圖六、Features (Sorted by Weights)

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 comprimised、serror rate、service、srv serror rate 與 dist host rerror rate。

```
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
```

圖七、沒有使用 dropout 的 validation accuracy

```
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
```

圖八、有使用 dropout 的 validation accuracy