National Taiwan University

Machine Learning: Final Project

EE3 b03901016

NTU_b03901016_HeavyPig

January 19, 2017

題目：ML2016 Cyber Security Attack Defender

Team name: NTU_b03901016_HeavyPig
Members: b03901016-陳昊 b03901018-楊程皓 b03901034-吳泓霖 b03901175-張顥霆
Work Division:

- 陳昊: 負責第 1，2 題的研究，Coding 以及 Report 的撰寫。
- 楊程皓: 負責第 2 題的研究以及 Report 的撰寫。
- 吳泓霖: 負責第 3 題的研究以及 Report 的撰寫。
- 張顥霆: 負責第 1 題方法研究，Coding 及 Report 的撰寫。

1 Preprocessing & Feature Engineering

首先，由於獲得的 training data 每一次攻擊的數據中，都有描述這次的攻擊的連線 protocol type(tcp，udp)，透過何種 service(telnet, ftp...)，以及該物攻擊連線的 status flag。而因為這幾個 feature 是以英文字串的形式表示，所以我們得將這些 feature 進行轉換編碼。

而要進行編碼轉換，首先得先統計該 feature 共有幾個不同的值，舉 service 的例子來說，它除了 telnet 和 ftp 之外，仍有很多其他的 service 像是 http、private 等等，統計完全部不同的 value 數後，將其從 0 開始一一對應編碼。完成之後我們便可以得到較好利用的純數字 feature。實際的作法是以一個 list 去儲存，只要在讀 data 時發現了不在該 list 裡的 service 類型，就將其加入該 list 中，待整份 data 讀完之後，再以此 list 中的 index 去取代該 service 的值。此外，training data 的最後一項代表了該次攻擊所屬的分類，也是以英文字串的方式表示，而這部份不能單純地統計相異的 value 個數然後進行編碼，因為這裡包含了許多攻擊的 manifold 變形，

因此必須直接根據檔案的對照表將其分類。

特別的是，在我們以同樣的編碼去處理 testing data 的時候，發現 testing data 中有攻擊的 service 不在前述 training data 統計的資料中。由於發現含有未見過 service 的 testing data 數量只有 2 個，因此我們便直接將這兩個未知編碼的 service 設定為 0。之後各個 model 所使用的資料便是經過此處理的資料，對於其他 feature 的值，我們沒有做其他的更動。

2 Model Description

以下我們前後使用了六種 Model 嘗試，分別為:

i. DNN

ii. Normal Random Forest

iii. AdaBoost

iv. Gradient Boost

v. SVM-SVC

vi. One vs Rest Random Forest

注意：
我們將每個 Model 的實作 python code 分別存於一個檔案，使用 Model 需要執行的指令會在各個 model 說明後。其中 repo 代表含有所需要資料的資料夾，output 代表想要輸出的名稱。

Model-1 (DNN):

這裡我們使用簡單的 fully-connected DNN 來進行 data 的 training，此 DNN 共有四層 hidden layer 以及一層 output layer，而 hidden layer 的 feature 數量為 (32,32,32,16) 且每層都含有 0.25 的 Dropout 以及 elu activation function，而 optimizer 使用的是 adam，並以 cross entropy 的方式計算 loss function。且由於資料量龐大，這裡所使用的 batch size 為 4096(經實驗發現 batch size 分別為 8192 和 4096 時速度幾乎相同，而 4096 相較 2048 時相差較大，因此選擇 4096 作為 batch size 的大小)，而 Epoch 的部份使用了 20 個。

使用說明：`THEANO_FLAGS='device=gpu' python3 ver1.py repo output`

Model-2 (Normal Random Forest):

Random Forest 是一個包含有多個 Decision Tree 的分類器，並且最後透過每個樹投票得出的眾數所決定該 data 是屬於哪一個分類。且 Random Forest 中的 Decision Tree 也具有以下的特性：

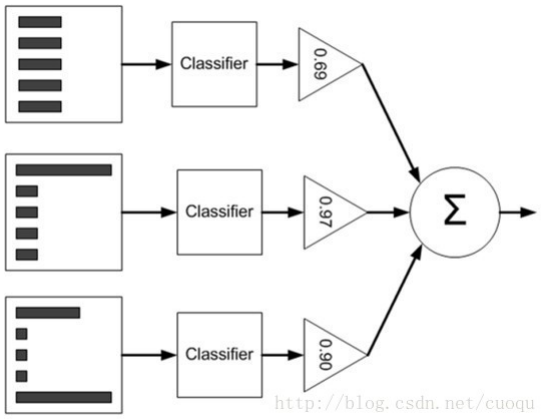

Figure 1: AdaBoost [1]

- 每個節點 (Node) 上的輸入 feature 數量 m，用於決定 Decision Tree 上一個節點的決策結果，其數量應遠小於總 feature 的數量 M。
- 根據隨機取得的 m 個 feature 在每一個節點 Node 上面計算最佳的分裂方式，通常有兩種分數作為分裂好壞的標準，分別是 cross entropy 和 gini impurity 的計算方式。這裡我們用的是 gini impurity 的計算方式。
- 每顆樹都是未經過 Pruning 的。

而實作上，我們嘗試了各種不同設定的 Random Forest 參數，主要的不同點在於 Decision Tree 的數量，小至 1 大至 512 都有測試過。而只要節點分裂後的 impurity 值大於 10^{-7}，我們就會讓其分裂，要是小於 10^{-7}，則他將不再分裂且該節點就成為 leaf。

使用說明：`python3 ver3_RF.py repo output`

Model-3 (AdaBoost):

我們直接使用對 preprocess 後的資料進行 AdaBoost 算法，基底的 Classifier 使用 Decision Tree ，而其中也試過許多不同數量的 estimator，不過通常在不同 estimator 數量下也差異不大，只要 estimator 的數量不要過小就可以。而 AdaBoost 中的 learning rate 我們都令他等於 1。

使用說明：`python3 adaboost.py repo output`

Model-4 (Gradient Boost):

Gradient Boost 也是一個有效的 boosting 方法，通常也是使用 Decision Tree 作為 ensemble 的方法，並同時有一個 loss function 來 estimate 並使 estimator 調整進步。而我們試過許多不同數量的 estimator(1 700)，leanring rate 設定為 0.01, loss function 設定為 deviance。

此外，這裡有個地方和先前的 Random Forest 有設計上的不同，我們設定了 max_depth ，也就是每個 estimator 的最大深度只能為 3。

使用說明：`python3 gradboost.py repo output`

Model-5 (SVM-SVC):

這裡用的是 SVM classifier(rbf kernel) 的方式來做，首先我們得先將 preprocess 好的 training data 降維才能使用 SVM 的方法對 data 做分析。而我們是使用 PCA 來做 data 的降維，先以 PCA 將 training data 降成 2 維，再透過 SVM 做出分類。

使用說明：(python version)`python3 SVC repo output`

Model-6 (OvR Random Forest):

這裡使用的技巧是將欲分類的 5 個 class 都各自創建一個專屬的 classifier 來判斷，當輸入一個 testing data 時，classifier 給的結果是 yes or no，只判斷該 data 是不是屬於該 class，並回傳一個數值代表該 data 屬於該 class 的機率，而這裡的每個 classifier 都是一個獨立的 Random Forest。同 Random Forest 一樣，這裡的每個 classifier 都能各自調整其中的參數，不過我們使用的方法是每個 Random Forest 架構是相同的，但分別判斷不同的目標 class，最後再透過調整每個 classifier 回傳的 result 做 weight tuning，調整各個 classifier 的 weight。
使用說明：`python3 ver4.py repo output`

3 Experiment and Discussion

我們先將 training data labels 數量做計算，並上傳分別是全 0-全 4，共五筆的上傳結果，以所得分數推得 testing data labels 中 0-4 分別的數量，分別是:

$$\text{training}\begin{cases}0:875363\\1:3495181\\2:43\\3:1011\\4:36989\end{cases}\text{, testing}\begin{cases}0:157566\\1:432209\\2:231\\3:8566\\4:8216\end{cases}$$

將兩分佈 normalize 後再取 natural log:

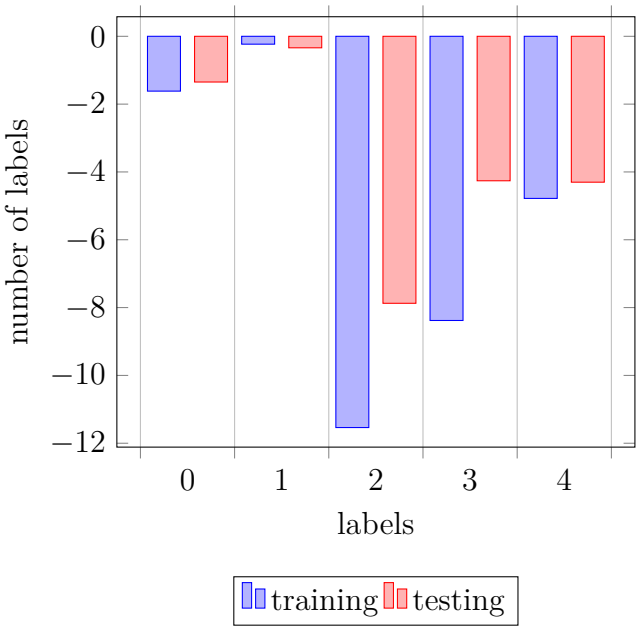

Figure 2: Label distribution in log scale

故在參數調整時，會令輸出結果中 labels 的分佈接近上面的統計結果。

在 training data 的各 labels 的分佈也類似於 testing data 的分佈，每個 label 的個數從 0 到 4 分別是 `875363, 3495181, 43, 1011, 36989`，而我們的預測結果中，label 為 0 or 1 的準確率很高，故 cross validation 的效果並不好，各個 Model 中的參數調整與比較就會依賴於上傳 testing labels 的 public scores。

以下比較各個 Model 間的差異。比較的方式是以 Kaggle 上的 Public Score (Classification Accuracy) 作為測量的基準。由於 Kaggle 上傳的限制，可以得到的數據不足。主要是比對幾種不同方法的準確度差異，以及同一方法不同參數的差異。

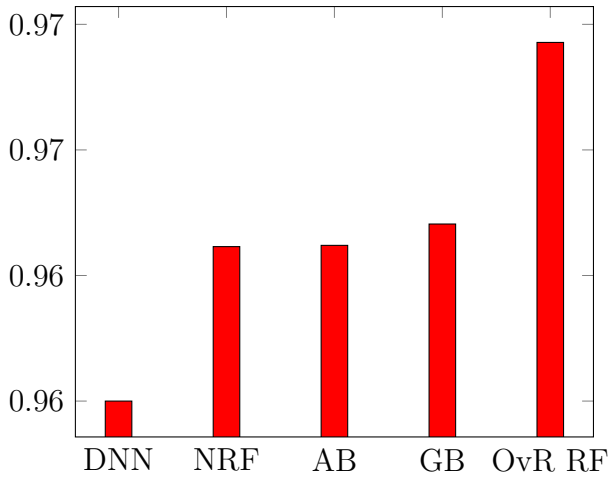

Figure 3: Scores of Models

各個 model 產出最好 Public Score 的各 label distribution table 如下:

Model	label 0	label 1	label 2	label 3	label 4
DNN	182002	416610	0	0	8167
NRF	180754	417678	6	696	7645
AdaBoost	180891	417739	8	504	7637
GradBoost	180046	418261	20	585	7867
OvRRF	173650	417399	48	7844	7838

因為 kaggle 上傳的限制，我們主要用上面的分佈來討論下面的 experiment。

觀察前四個 labels distribution，其中 label 3 與 label 4 的數量其實是差不多的，但預測出來的數量卻遠低於 label 4。發現這四個 Models 的分佈與 training data 的分佈比例相似，可能是 training 時 overfit 了，故 label 3 比例低於最佳解，而 label 0 數量比最佳解多一些，極有可能 label 3 features 與 label 0 相似度高，使 Model 都因條件機率將本是 label 3 的都歸類為 label 0。

DNN:

整體來說效果不是很好，準確度在 0.95 到 0.96 之間。準確度可能是因為 epoch 不夠，或是 model layer 的調整不夠而有進步的空間。

Normal Random Forest:

由於 Random Forest 的隨機性，在同樣的參數下會出現不一樣的結果。小量的樹的個數會使得 overfit 的機會增加，在數字增大到一定數字後出現個數愈多準確度提高但幅度減少。準確度的範圍在 0.96055-0.96115 之間。

AdaBoost:

在幾次上傳的結果中，不同的 estimator 數量並不會有很大的差別，最好的結果也只是剛好可以過 strong baseline 的 0.9612。

Gradient Boost:

這個方法在測試時主要是調整 iteration 的次數，分別嘗試過 200 跟 700，對應的分數是 0.96181, 0.96205. 推測他 iteration 的次數增加會使準確度提高，但在 700 時還沒有出現 overfit 的狀況。

SVM-SVC:

這個 Model 在實作上遇到了問題，由於資料的數量太龐大，一般的 SVM 似乎無法完成，data 的降維在 training 時無法找到收斂的點。於是我們便只使用部份的 training data，即便如此 training 也無法結束。後來我們使用了額外的 program 使用 gpu 執行 libsvm 做嘗試，但還是沒成功。

OvR Random Forest:

這個方法主要是把 5 個 Random Forest Binary Classifier 作 ensemble。令每一個 Random Forest Classifier 輸出該筆 Data 屬於對應的 Class 的機率，最後再 Ensemble 來決定最終輸出的結果。Ensemble 是以 Weighted max 的方式實作。

實驗後觀察輸出結果，發現 label 0 的輸出過多，因此我們把 label 0 的 weight 從 1 調到 0.6，而且 label 3 的機率由於 training data 中 label 3 過少而使得分類器得出的 probaility 很低。最終我們決定當 label 3 的機率不為 0 時，就使輸出結果較為跟 testing data 接近。而藉由增加 tree 的數量就可以有效的提高 train 出來 label3 的數量。

最終在 Kaggle 上的分數最好為 0.96928。

References

[1] AdaBoost: http://blog.csdn.net/marvin521/article/details/9319459