ML2016 Final Project

Project Name

Cyber Security Attack Defender

Team Name

NTU_b02902041_我的一半是由雷姆的愛 另一半是由雷姆的思戀組成的 這樣說也不過分哦

Members

B02902014 林家緯
B02902036 徐誌鴻
B02902041 徐朝駿
B02902099 葉政杰

Work Division

四個人共同討論後，決定先嘗試看看NN和Random Forest這兩種作法。徐朝駿和林家緯負責NN，徐誌鴻和葉政杰負責Random Forest，最後以徐誌鴻找到的XGBoost過了strong baseline。

Preprocessing and Feature Engineering

在觀察本task的敘述以及資料後，很明顯可以發現是一個分類問題，於是我們首先需要將資料處理成某些特定格式，並且挑取可能會是重要的部份來讓機器辨識。以下將介紹我們嘗試了哪些方法來處理資料並抽取特徵。

原本的data set有41維，我們將它分成兩個部分，一個是把原先是用字串表示的protocol type、service、flag這三維直接移除，當做data A；另一個是把這三種資料做成one-hot encoding的形式放在資料中，當做data B。之所以沒有直接將字串編號，轉換成數值使用，是為了避免因為數值大小的差異造成每種類型有不一樣的影響力。

	data A	data B
dimension	38	122

雖然data A捨棄了非常多重要的資訊，不過以Random Forest做出來的accuracy卻跟data B差不多；對於NN來說才有比較大的影響，這部份在Experiments and Discussion會再詳細說明。

至於training的target，我們也嘗試了兩種選擇，一種是先找出data set中的answer屬於最底層的哪一種分類，之後再對應回上傳時所需的五種答案其一；另外一

種則是直接將training sample的正確答案先分類好，最後testing時就直接predict成上傳時所需的五種。

Model Description

我們總共嘗試了下列四種常見於分類問題的model，以下將簡介之並說明我們使用的參數以及一些有助於訓練的技巧。

1. Random Forest
    a. 使用套件：sklearn中的RandomForestClassifier
    b. 概念：利用resampling的概念，種出一把不同的decision tree，最後再由這些樹的分類結果中選擇出現次數最多的類別作為最後答案
    c. 參數：n_estimators=100、max_depth=None、oob_score=True,

2. XGBoost
    a. 使用套件：xgboost中的XGBClassifier
    b. 概念：將用來作regression的decision tree做一些組合之後，重新設定objective function後就可以拿來做為classification。有別於作業一與作業二時所定義的objective function都是可以數值直接表示，所以都可以直接微分來找optimal，在decision tree類型的方法中，我們必須採用別的方法來讓機器學習。而XGBoost所使用的方法就是Additive Training (Boosting)，也就是每次greedy地多種一棵樹(相當於在原來的function上多加一個function)，之後再經過一些split finding以及pruning，這些動作的意義與老師上課時說過的機器學習的本質是相同的，就是要讓我們定義出來的function能夠越fit資料越好
    c. 參數：n_estimators=100、max_depth=5

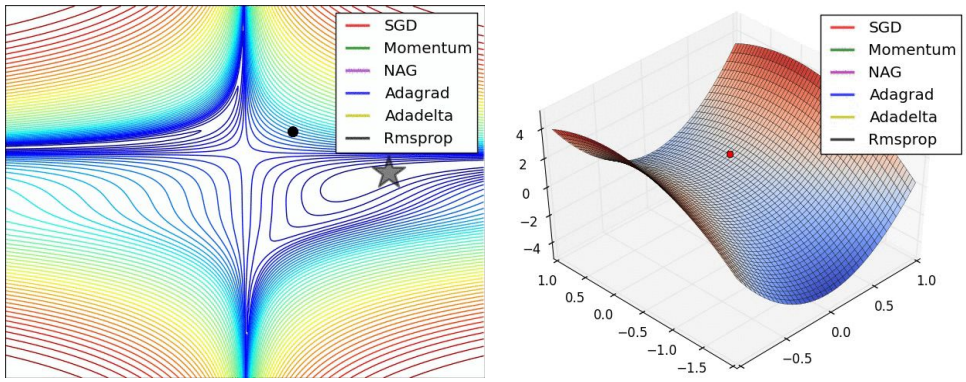
3. Neural Network
    a. Model：因為這次題目做的是一般的分類問題，所以我們採取基本的fully-connected DNN來嘗試，經過多次的調整參數與深度後最終的版本是如下圖：

```
model = Sequential()
model.add(Dense(1024, input_dim=122))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5))
model.add(Activation('softmax'))

model.compile(optimizer='adadelta',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

做成三層的dense layer，最後再經過softmax去和target做crossentropy後做weight update。

b. Updating Method：原先使用的optimizer是rmsprop，但是使用rmsprop時因為浮動較大導致loss一直將不下來。在看過關於update的討論之後，發現adadelta可能可以有比較好、快速的表現，可以看到以下的模擬：



所以最後採用adadelta也真的獲得比較穩定的loss下降跟validation正確率。
gif credit: Alec Radford.

4. Ensemble
    a. 直接組合法：將上述三種model測試的各種不同結果以相同權重直接投票後取得新的答案，若是有相同票數的問題則選擇準確率最高的那個仁model的決定作為答案，其中，我們初步的嘗試是將三份答案組合。

Experiments and Discussion

Experiment Results:

A. 各種方法對於兩種feature的比較

	data A	data B
NN	0.89926	0.95698
Random Forest	0.96034	0.96045
XGBoost	X	0.96145
Ensemble	X	0.96050

B. RF中不同數量的樹的比較：
n = 25

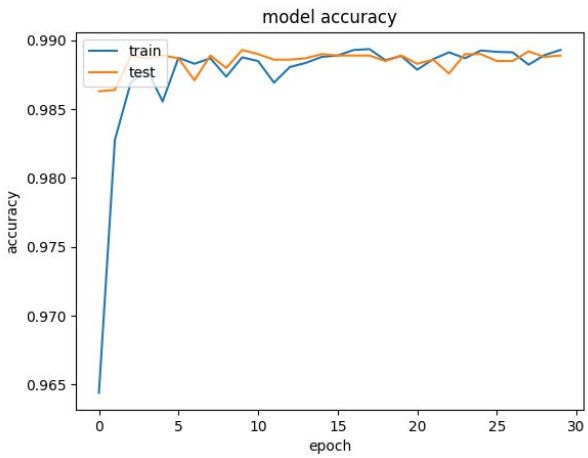
	data A	data B
target A	0.95005	X
target B	0.96034	0.95996

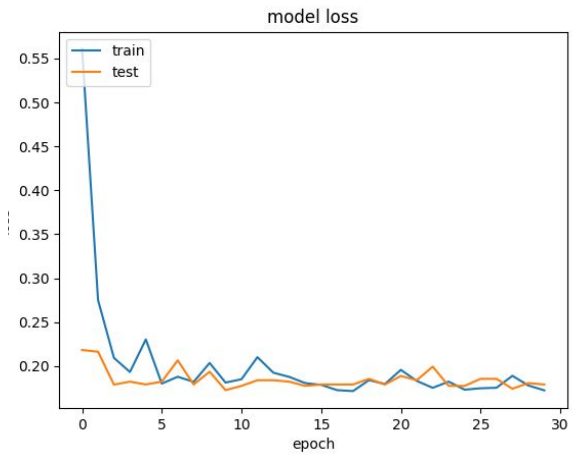
n = 100

	depth = 9	none(不限深度)
data B, target B	0.95964	0.96045

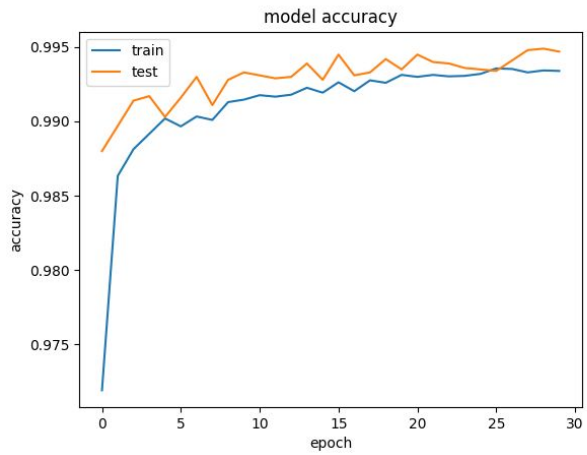
C. NN中兩種不同的optimize方法比較 （圖中test為validation）

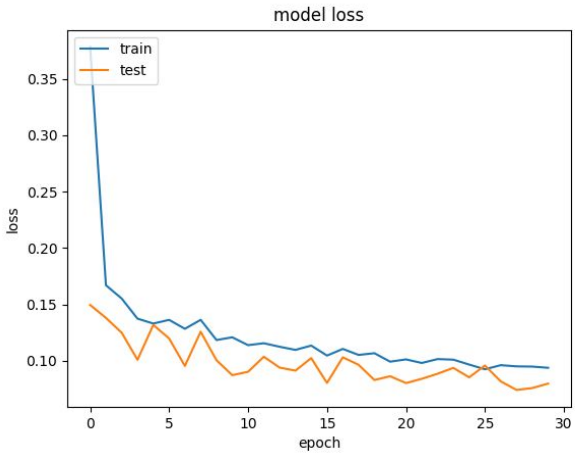
Rmsprop ：





Adadelta：





	Accuracy(train, val)	Loss(train, val)
rmsprop	**(0.9893, 0.9889)**	**(0.1723, 0.1789)**
adadelta	**(0.9947, 0.9934)**	**(0.0939, 0.0799)**

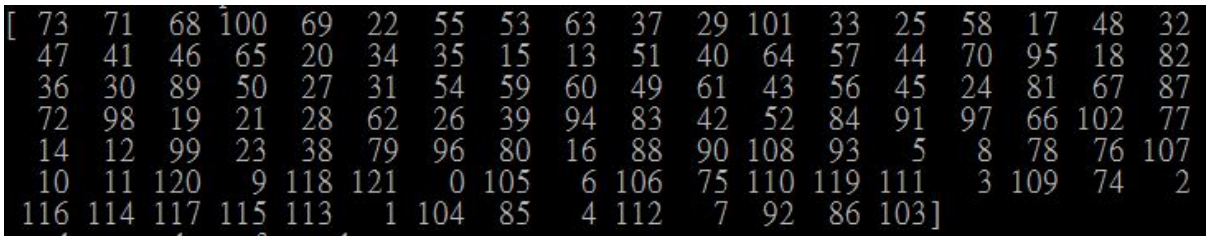
由上方的圖片與表格可以得知在介紹model時所遇建的問題，就是rmsprop會在一開始的幾個epoch內就達到幾乎最高的accuracy，然後會在後面的epoch中一直震盪，卻不會上升。反倒是用了adadelta之後，可以看到lost跟accuracy的圖片中，train的部分是穩定的下降跟上升，有明顯的進步。

Discussion:

A. Random Forest參數討論：

a. 由測試結果中可以看到，target A和target B的影響最大，差了1%多的準確率。經過討論，我們覺得可能是分成太多類，反而讓他沒有辦法分類好，因為其實有些可能是可以歸為同一類，但硬要分成不同類，導致較多的error。

b. 另外，我們透過套件中提供的feature_importances_來看各維度的重要性(越後面越重要)：



透過上圖可以看到，在我們表示one-hot的維度(2 ~ 82)中，有很多維度很重要。雖然在Random Forest中，one-hot並沒有帶來很大的提升，但我們可以從中得知，這裡面的資訊有些是很重要的。

c. 至於其他的參數，如樹的數量、最大深度，經過一些測試後，沒有看到太大的變化。

B. XGBoost：

a. XGBoost的部分，因為上傳次數有限，我們直接使用了data B和target B的data，而且因為要跑很久，因此我們在第一次train完，還沒把validation也一起train就把結果上傳，沒想到結果還不錯。經過討論之後，我們認為，一部份可能是因為XGBoost真的很厲害，另一部份，可能是因為data本身的緣故，data的部分下面有討論。

C. DNN的validation accuracy：

在過去的一些作業中我們在training時，通常會一邊training一邊看validation accuracy來看是否有overfitting的問題發生。但是在這一次的data set中，不管NN的層數或是每層的長度如何，val acc都會在前面幾

個epoch就衝到0.97、0.98，就差不多開始收斂了。所以在這次的題目中，我們的validation部分幫助並不大，還是只能依照loss來判斷什麼時候該停止。

D. 觀察這次自己做出的prediction跟原本的labeled data，我們發現其實幾乎所有的情況都是normal跟DOS，其他情況的機率可能只佔了不到1%，所以我們推測，這次結果上傳之後大家都遇上的瓶頸可能是關於其他分類的data數量太少了，才會讓在val上可以到達99%左右的成績，在public set上卻只有96%的情況發生。

E. 參考了網路上對於XGBoost套件的介紹，預期中其運算速度應該要較Neural Network以及Random Forest快，但我們實際上run起來卻得到了完全相反的結果，推論原因可能出在套件的選擇不同。該篇討論文章中所提到的套件是以C++實作，並且支援多核心的平行運算；然而我們最終使用的卻是python的版本。以及，在執行NN的運算時我們有使用到GPU，使得運算速度得到的一定的提升所致。

Reference

1. XGBoost:
   http://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
   http://cos.name/2015/03/xgboost/
2. Random Forest:
   http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. Neural Network:
   http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML16.html