Machine Learning Final Project

Topic: Cyber Security Attack Defender

隊伍名稱：NTU_加油好嗎
組員： R04942114 王彙智（Data preprocessing , Feature Engineering)
R04942131 蔡銘穎（Learning model design , model selection)

Work division

王彙智（Data preprocessing , Feature Engineering)

蔡銘穎（Learning model design , model selection)

1. Preprocessing and Feature Engineering

一開始拿到的訓練資料大概有四百多萬筆，總共分成五個類別，分別為「normal」、「dos」、「u2r」、「r2l」、「probe」，而在這筆 training set 主要發現了兩個問題。

第一，部分 feature 並非使用數值描述而是使用類別描述，包括 protocol type, service, 以及 status flag 等等，而在做 training 之前，必須把這些類別的 feature 轉換成數值的 feature。

第二，在這些 training set 中，有些 label 的 data 數量過少，形成 imbalance data set 的問題，像是在本次專題中所使用的 data set，normal 標記的數量有 875363 筆，dos 有 3495181 筆，u2r 有 43 筆，r2l 有 1011 筆，probe 有 36989 筆，可以看見 label 的量非常極度的不平衡，而我們參考 SMOTE 這個 resampling data 的方式，試圖去解決 imbalance data set 的問題，詳細的步驟會在下面說明。

(1) 將 categorical feature 轉換成 numerical feature

在給定的 training set 中，protocol type, service，以及 status flag，這三種 feature 是以類別的方式呈現，為了可以利用這三個 feature 來訓練 learning model，我們必須將這幾個 feature 轉換為數值的 feature，其中我們使用的方式為將每個類別 feature encode 成多維的 binary feature，每一維只會出現 0 與 1，每一維代表的就是類別 feature 中的其中一個 level，舉例來說，protocol type 的這個 feature 總共出現過三種 level，分別是 icmp，tcp，以及 udp，其中 icmp 會被 encode 為 [1 0 0]，tcp 會被 encode 為 [0 1 0]，udp 會被 encode 為 [0 0 1]，如果使用此種方式去處理 feature，最後會從原本的 41 個維度擴增為 100 多個維度，

而在 implement 轉換的過程之中，發現類別 feature 中有些 level 分得太細，像是在 service 這個 feature 當中，總共有 70 個 level，然而各個 level 所擁有的 training data 數量非常極度不平均，有些 level 的數量非常稀少，這會影響整個分類過程的 performance，為了解決這個問題，我們不使用全部可能出現的 level 來做 encoding，我們使用的方法為去看各個 level 中 training data 的數量，保留其中 k 個擁有最多 data 數量的 level，其他剩餘的 data 全都歸類成一個 level，取名為 others，透過這個方法，可以有效地去平衡 training data 中每個 categorical feature 中每個 level 的 data 數量，在我們實作的過程中，針對 protocol type，service， 以及 status flag 這三種 feature，我們保留 protocol type 的所有 level，只保留 10 個 service 的 level，剩下的歸類為 others，只保留 5 個 status flag 的 level，剩下的也歸類為 others。

(2) Data re-sampling by SMOTE

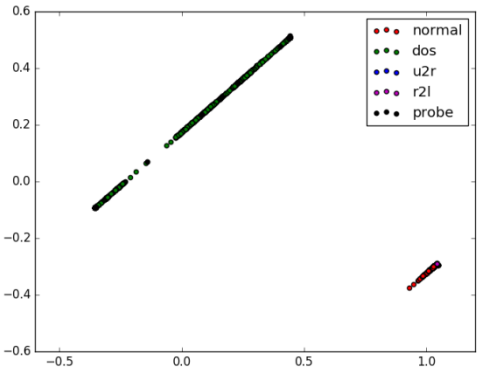
在 training data set 中各個 label 的數量如果不是 balanced ，將會影響最後 Classifier 的 performance，在這次的 project 中， normal 標記的數量有 875363 筆，dos 有 3495181 筆，u2r 有 43 筆，r2l 有 1011 筆，probe 有 36989 筆，可以看到 u2r 以及 r2l 這兩個 label 數量與其他類別比起來相差懸殊，為了讓每個類別的 training data 幾乎等量，我們必須使用 Resampling 的技巧來 generate 更多的 data，在這次的 project 中，我們使用 Synthetic Minority Over-Sampling Technique(SMOTE) [1] 的方法來做 Resampling，其中的原理為針對每一筆 minority class，去尋找 k nearest neighbors，再從這 k 個 nearest neighbors 中隨機選取一筆 data，利用隨機取出的 data 與原本的 minority class data 筆所形成的連線，隨機在線上取出一點，當作最後 generate 出來新的 data，最後即可得到 balanced 的 data set，下圖為經過 smote 之後所得到的結果。在實作上我們直接使用 sklearn 中 imbalanced-learn 的 smote function，為了不影響整體的 data 數量，先求出平均每個 class 的 data 數量，用這個平均值與各個 class 的數量做比較，如果超過這個值，表示這個 class 的 data 數目過多，會把多餘的 data 丟掉直到跟此平均數相同，若是比這個平均值還要少，則會利用 SMOTE，把不足的 sample 內插出來。

(3) Feature Normalization

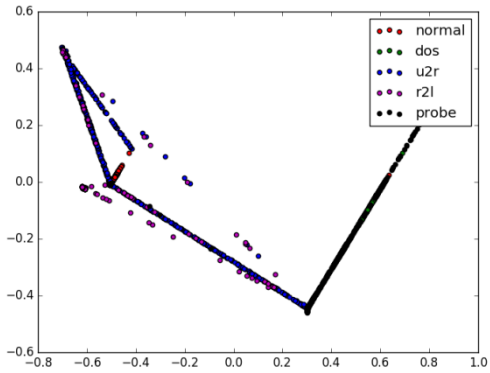
為了不讓 data 之間的數值差異過大，也為了讓收斂的速度增加，我們使用 minmax scaling 讓各個 feature 的數值皆落在 [0 1] 之間，可讓 classifier 對於數值的變化較不敏感，能夠分得更好。

(4) Data visualization

在這裡我們將會展示做完 feature engineering 後 data 間分布的情形，利用 PCA 將 data 壓縮至二維分佈觀察每個 class 的資料分布



Original data set



Resampling by SMOTE

由這兩張圖可以看出資料再經過 resampling 之後明顯的有被分開的趨勢，這是因為 data 分佈變的比較平均，在判斷分類區間時不會因為某些 class 過少量的 data 而產生誤判。

2. Model Description

在 classifier 的選擇上，我們試了幾種不同的方法，有 Random forest, SVM 以及 Deep learning，以下我們會一一介紹各種方式的優劣。

(1) Random forest

之所以使用 Random forest 的原因我想不用多說，Random forest 是近年來 Machine learning 中 Classifier 表現非常優異的，不管在什麼樣的分類問題，準確率都有相當的優勢，而且不必像 DNN 以及其他的 Classifier 一樣需要調整很多的參數，只要在 feature 上處理得夠好，準確率可以達到非常高的水平，而另外一個最大的優點是他的運算速度非常快，尤其這次的資料量非常龐大，其他的方法都要花上數倍的時間，最後我們跑完的時間大概落在 10 分鐘左右，

已經是相當快的速度，但是他比較大的缺點在於容易 Overfitting，所幸我們在這次的實驗中並沒有發生這樣的問題，最後的準確率在 Kaggle 上的分數是 0.96115，也有達到 Strong baseline 的標準。

```
"""Training random forest"""
if train_method == 'rf':
    # train_num = Xtrain.shape[0]
    # val_num = Xval.shape[0]
    # print "training on Random Forest Classifier..."
    # print "Training Num : %d  Validation Num : %d" % (train_num, val_num)
    clf = RandomForestClassifier()
    clf.fit(Xtrain, y_train)
    # train_err = clf.score(Xtrain, y_train)
    # val_err = clf.score(Xval, y_val)
    # print "Training error rate : %f" % train_err
    # print "Validation error rate : %f" % val_err
```

(2)SVM

SVM 是我們一開始所使用的方法，因為課程剛好上到這個部分，所以就直接實現看看，由於一開始並沒有做 feature selection，所以我們直接把 100 多維的 feature 丟進去 fit，最後的結果就是跑了一整個晚上，正確率也是不盡理想，雖然 SVM 適合處理高維的問題，但卻不適合處理大數量的樣本數據，所以就算後來做了 dimension reduction，還是沒辦法解決運算速度的問題，最後只好捨棄這個方法，在 Kaggle 上的分數為 0.88492 。

```
def train_svm(Xtrain,y_train,Xval,y_val,train_learning_model=False):
    train_num=Xtrain.shape[0]
    val_num=Xval.shape[0]
    train_model_file = 'learning_model_svm_smote.pkl'
    if train_learning_model is False:
        print "training on SVM multi-classifier..."
        print "Training Num : %d  Validation Num : %d" % (train_num, val_num)
        clf = OneVsRestClassifier(SVC())
        clf.fit(Xtrain, y_train)

        print "Write out learning model ..."
        pk.dump(clf, open(train_model_file, 'w'))
    else:
        print "Load SVM model ..."
        clf = pk.load(open(train_model_file, 'r'))

    print "Accuracy Measure on training set and validation set"
    y_train_predict = clf.predict(Xtrain)
    y_val_predict = clf.predict(Xval)
    train_err = accuracy_score(y_train, y_train_predict)
    val_err = accuracy_score(y_val, y_val_predict)
    print "Training error rate : %f" % train_err
    print "Validation error rate : %f" % val_err

    return clf
```

(3)DNN

DNN 是在所有方法中最為需 fine tune 的方法，可以調整整個架構、Dense、Dropout 數等等，而且每次運行的時間非常的長，所以勢必得花非常多的時間在上面，我們參考了之前的隊伍，自動化的方式，每次調整一個參數並記錄最好的結果，運行了 2.3 天的時間後，結果並沒有比 Random forest 來的優異，最後使用的參數總數為 3589 個，Optimizer 為 Adam，Batch size = 128，跑了 50 個 epoch，而在 Kaggle 上的分數也只有 0.9 左右。

```
def train_dnn(Xtrain,y_train,Xval,y_val):
    from keras.models import Sequential
    from keras.layers import Activation,Dense,Dropout
    from keras.optimizers import Adam
    from keras.utils import np_utils
    print "Training Model by DNN...."
    nb_classes=5
    y_train=np_utils.to_categorical(y_train.astype(int),nb_classes)
    y_val=np_utils.to_categorical(y_val.astype(int),nb_classes)
    model=Sequential()
    model.add(Dense(64,input_shape=(Xtrain.shape[1],)))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    model.add(Dense(32))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    model.add(Dense(5))
    model.add(Activation('softmax'))
    model.compile(optimizer=Adam(),loss='categorical_crossentropy',metrics=['accuracy'])
    model.summary()
    model.fit(Xtrain,y_train,batch_size=128,nb_epoch=50,validation_data=(Xval,y_val))
    return model
```

3. Experiments and Discussion

其實這次的 task 不算太簡單，以往的作業並沒有要求達到很高的準確率，不過這次一開始的準確率就已經在 0.7 左右了，也就是說進步的幅度很小，需要精確的抓到誤差是在哪裡發生，又該如何去修正它。

以這次的 topic 來說，問題就發生在 dataset 本身是 unbalanced，一開始都著墨在使用不一樣的 structure 以及 classifier，但準確率始終無法得到很大的提升，於是我們開始反觀是不是 feature 取的不好，太多的 noise 或是維度太高或太低，以及是否有 overfitting 的現象，直到該考慮的都考慮完之後，開始觀察 dataset 本身是否和以往的不相同。

首先一開始當然是 categorical feature 轉換成 numerical feature 的問題，但這個並不是太大的問題，也不是問題的癥結點，真正的問題是部分 Class 數目太少產生的 unbalanced 現象，而這點在上課中並沒有提到太多相關的知識，所以後來開始查文獻看看過去是否有類似的問題發生，

所幸這個問題早已有人試著去解決，也就是使用 Data re-sampling 的技術，去增加某樣本數目太少的問題，而解決了這個問題後，我們的 performance 有著飛躍性的成長，順利的通過 strong baseline。

經過這次的 project，我們學到在機器學習的議題中，該如何觀察我們的資料，進而發現我們的問題，而什麼樣的問題該用什麼樣的技術去解決，才是最恰當且精準的，尤其在這種 big data 的案例中，每次的測試時間都是非常耗時費力的，所以必須在每次的測試中得到該有的資訊，找到問題的核心，才能順利的解決問題。

Reference

[1] Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." *Journal of artificial intelligence research* 16 (2002): 321-357.

[2] Buczak, Anna L., and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection." *IEEE Communications Surveys & Tutorials* 18.2 (2015): 1153-1176.