MLFN Lab1 Report

309552012 洪立宇

Data Preprocessing

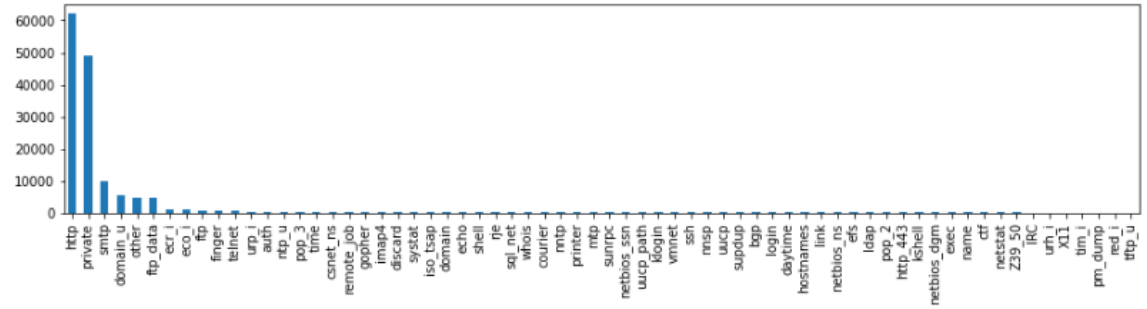
- Dataset: kddcup.data_10_percent
- Data feature: kddcup.names
- Data target: attack_types（0-4）

0	normal
1	probe
2	denial of service (DOS)
3	user-to-root (U2R)
4	remote-to-local (R2L)

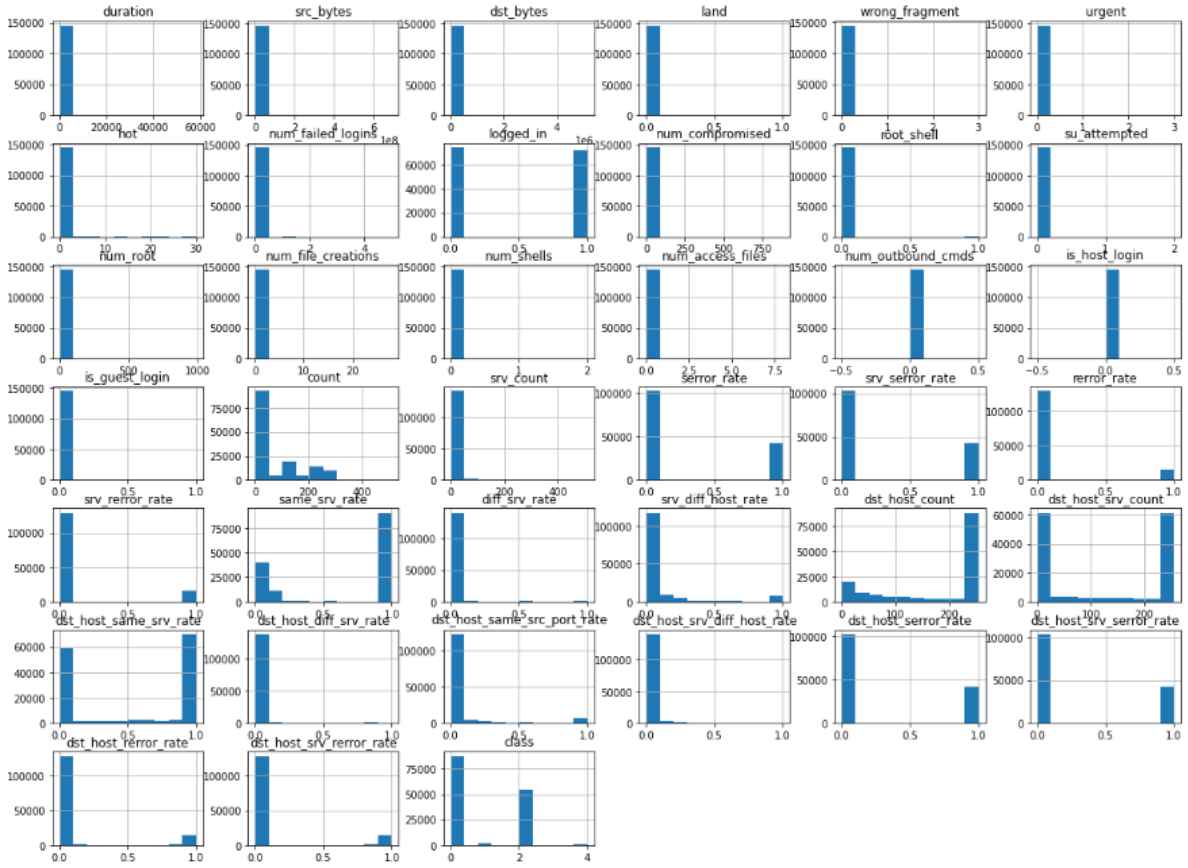
1. **Read data and get features into the dataframe**
   讀取資料發現有一行的資料過長，將那行skip掉，並且把所有的features對到對應的資料
2. **Attack type mapping**
   依照助教給的的指示將所有的attack的種類對應到其攻擊項目（normal, probe,etc.）並轉成0~4的數值
   - 注意到有一data的 attack type 是0.00,先將此data drop掉
3. **Preparing the input**
   - 用drop_duplicate() 將重複的data 刪掉, 瞬間從49萬比資料變成 14萬比資料
   - 並將資料打散隨機洗牌

Data Visualization

- 用histogram 觀察所有的資料

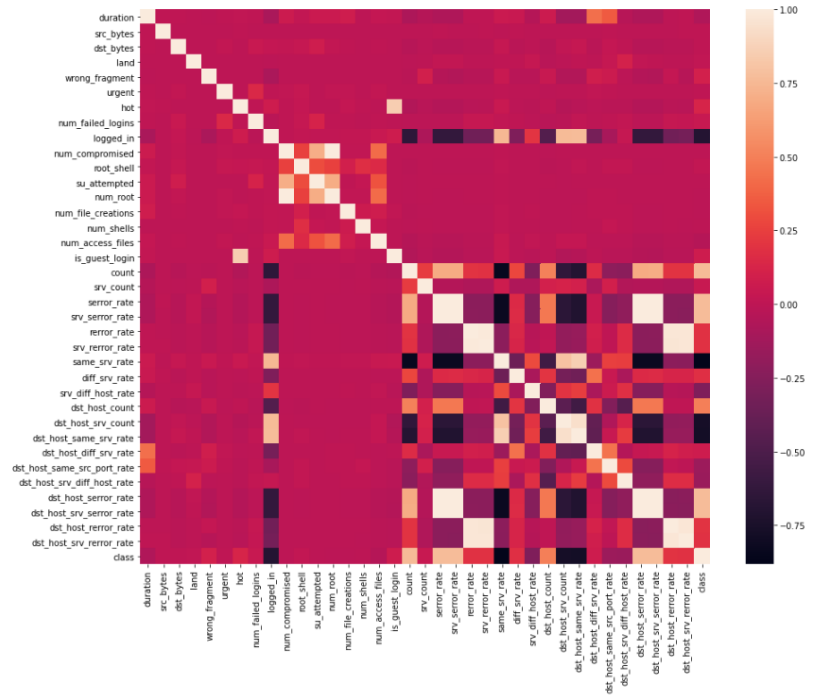


此為service的資料 因為分佈太分散,在處理時直接把這個col 去掉了

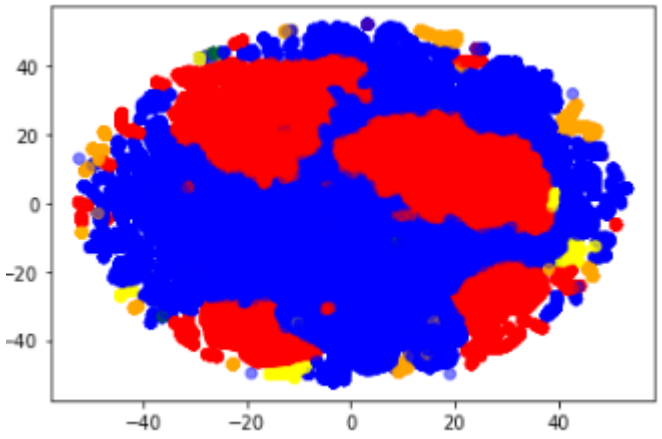


觀察後發現很多numerical 的feature, 最大最小差很多，且時常集中在某個數值
可以推測若feature值集中在某個直上代表他對target影響比較少

- 用correlation matrix 觀察數值資料之間的關聯性



顏色越黑代表correlation越低,反之則高



- 用PCA視覺化所有features 降成2維時，target type的分佈

Feature Transformation

- Onehot encoding: 對 categorical 的feature 做one hot encoding, 使 feature 上升到53個

```
df.drop('service',axis = 1,inplace= True)

### OneHotencoding
df_oh = pd.get_dummies(df)
display(df_oh)
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	flag_REJ	flag_RSTO	flag_R
111883	0	0	0	0	0	0	0	0	0	0	...	0	0	
56671	0	0	0	0	0	0	0	0	0	0	...	0	0	
454629	94	149	21090	0	0	0	0	0	1	1	...	0	0	
451952	0	0	0	0	0	0	0	0	0	0	...	1	0	
35304	0	318	1686	0	0	0	0	0	1	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
355215	0	0	0	0	0	0	0	0	0	0	...	0	0	
452458	0	740	0	0	0	0	0	0	1	0	...	0	0	
484497	0	316	943	0	0	0	0	0	1	0	...	0	0	
139189	0	317	593	0	0	0	0	0	1	0	...	0	0	
453957	2625	146	105	0	0	0	0	0	0	0	...	0	0	

145584 rows × 53 columns

- 使用minmax scalar 來 rescale data

```
sc = MinMaxScaler()
X = sc.fit_transform(X)
```

Feature Selection

- 使用REF來做feature selection

```
:
selector = RFE(DecisionTreeClassifier(), 30, step=1)
selector = selector.fit(X, y)
print(selector.support_)
print(selector.ranking_)
```

```
/home/harry/.local/lib/python3.6/site-packages/sklearn/utils/validation.py:72: FutureWarning: Pass n_features_to_sel
ect=30 as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an
error
  "will result in an error", FutureWarning)
```

```
[ True  True  True False  True False  True  True  True False  True False
  True False False False False False  True  True  True False  True False
 False  True  True False  True  True  True  True  True  True  True  True
  True  True  True False False  True False False False  True  True False
 False False False  True]
[ 1  1  1  5  1 16  1  1  1 11  1 22  1  2  8  9 14 23  1  1  1  6  1  3
 20  1  1 21  1  1  1  1  1  1  1  1  1  1  4 13  1 18 15  7  1  1 17
 10 19 12  1]
```

使用 Decision Tree的model 來套進去feature，數字是1的是selector認為比較好的 feature，其他數字的feature則為selector認定較差的,在手動將這些欄位的featrue 移除掉

Feature Engineering

除了處理原本資料重複並且將錯誤的值去掉以外,也有確定每個欄位是否有空值，dataset 裡面有原有的categorical feature 先做 onehot encoring numerical的feature則做minmax數值標準化。

因為對資料的domain knowlege 不夠，很難去找到一個方式來，現有的feature合併並創造一個新的feature，從視覺劃上也只能做出初步的判斷可能哪些和target較沒有關係，或許還是可能要套用sklearn的feature selection工具，如VarianceThreshold、SelectPercentile或是SelectFromModel去選擇feature或REF，來判斷哪種效果最好。

Training Model and Experiment

- 先將data分成training (80%)和testing(20%) set
- K-ford validation(k=5)
  從原本的training data 去做5-fold cross validation，並分出training 還有validation 來進行交叉驗證，在此使用**GridSearchCV** 來對每個**model**做超過**100**次的**iteration** 來找出最好的**model**參數**,** 最後在以那個參數去做訓練
- 總共使用了四個model來做測試，分別為Decision Tree, Random Forest, SVM, KNN

以下為做grid search後找到最好的參數結果，並使用這些參數來跑最後的預測，計算出accuracy, percision, recall, F1-score，對於不同model在使用了不同的parameters來組合（如下圖）來找出最好的對應參數。

- Decision Tree

```
parameters = {
    'max_depth' :[5, 10, 15, 20 ,30, 50, 80, 100, 120, 150],
    'criterion' :['gini', 'entropy'],
}
D_clf = GridSearchCV(DecisionTreeClassifier(random_state=rs), parameters, cv=5,verbose=2, n_jobs=-1)
D_clf.fit(X_train, Y_train)
print(D_clf.score(X_train, Y_train))
print(D_clf.best_params_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
0.999845449784059
{'criterion': 'entropy', 'max_depth': 15}
```

- Random Forest

```
rs=1

parameters = {
    'max_depth' :[5, 10, 15, 20 ,30, 50, 80, 100, 120, 150],
    'criterion' :['gini', 'entropy'],
}
R_clf = GridSearchCV(RandomForestClassifier(random_state=rs), parameters, cv=5,verbose=2, n_jobs=-1)
R_clf.fit(X_train, Y_train)
print(R_clf.score(X_train, Y_train))
print(R_clf.best_params_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
0.9999077311079444
{'criterion': 'entropy', 'max_depth': 20}
```

- KNN

```
parameters = {
    'weights' : ['uniform','distance'],
    'algorithm' : ['auto','ball_tree','kd_tree','brute'],
    'leaf_size' : list(range(1,4)),
}
KNN_clf = GridSearchCV(KNeighborsClassifier(), parameters, cv=5,verbose=2, n_jobs=-1)
KNN_clf.fit(X_train, Y_train)
print(KNN_clf.score(X_train, Y_train))
print(KNN_clf.best_params_)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
0.9999914138768922
{'algorithm': 'ball_tree', 'leaf_size': 1, 'weights': 'distance'}
```

- SVM

```
parameters = {
    'kernel' :['poly', 'rbf', 'sigmoid'],
    'C': [0.1,1, 10, 100],
    'gamma': [1,0.1],
}
SVC_clf = GridSearchCV(SVC(random_state=rs), parameters, cv=5, n_jobs=-1)

SVC_clf.fit(X_train, Y_train)
print(SVC_clf.score(X_train, Y_train))
print(SVC_clf.best_params_)
```

```
0.9985918758103154
{'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

Result

- **Target{‘normal’: 0, ‘probe’:1, ‘dos’:2, ‘u2r’:3, ‘r2l’:4}**

Accuracy, Recall, f1-score

使用metrics.classification_report(ypred, y_test)，可以直接用來算percision, recall, F1-score,support數值

- Decision Tree

Decision Tree:

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	29036
Probing	0.97	0.96	0.97	710
DOS	1.00	1.00	1.00	17971
U2R	0.79	0.65	0.71	17
R2L	0.98	0.96	0.97	309
accuracy			1.00	48043
macro avg	0.95	0.91	0.93	48043
weighted avg	1.00	1.00	1.00	48043

- Random Forest

Random Forest:

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	29036
Probing	0.99	0.97	0.98	710
DOS	1.00	1.00	1.00	17971
U2R	0.80	0.47	0.59	17
R2L	0.99	0.97	0.98	309
accuracy			1.00	48043
macro avg	0.96	0.88	0.91	48043
weighted avg	1.00	1.00	1.00	48043

- KNN

KNN:

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	29036
Probing	0.99	0.98	0.98	710
DOS	1.00	1.00	1.00	17971
U2R	0.64	0.53	0.58	17
R2L	0.93	0.96	0.95	309
accuracy			1.00	48043
macro avg	0.91	0.89	0.90	48043
weighted avg	1.00	1.00	1.00	48043

- SVM

SVD:

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	29036
Probing	0.98	0.97	0.98	710
DOS	1.00	1.00	1.00	17971
U2R	0.82	0.53	0.64	17
R2L	0.91	0.94	0.92	309
accuracy			1.00	48043
macro avg	0.94	0.89	0.91	48043
weighted avg	1.00	1.00	1.00	48043

- Acurracy 比照表

```
Accuracy:
Random Forest  : 99.893845 %
Decision Tree  : 99.820994 %
SVD            : 99.781446 %
KNN            : 99.802260 %
```

Confusion Matrix

- Decision Tree

```
Decision Tree Confusion Matrix:
 [[29005    18     7     3     3]
 [   22   683     5     0     0]
 [   10     1 17960     0     0]
 [    4     0     0    11     2]
 [    7     3     1     0   298]]
```

- Random Forest

```
Random Forest Confusion Matrix:
 [[29026     5     2     1     2]
 [   18   692     0     0     0]
 [    5     0 17966     0     0]
 [    9     0     0     8     0]
 [    8     0     0     1   300]]
```

- KNN

```
KNN Confusion Matrix:
 [[28993     6    11     4    22]
 [   17   693     0     0     0]
 [   14     1 17956     0     0]
 [    8     0     0     9     0]
 [   11     0     0     1   297]]
```

- SVM

```
SVD Confusion Matrix:
 [[28989    10    13     1    23]
 [   21   686     3     0     0]
 [    3     1 17965     0     2]
 [    6     0     0     9     2]
 [   19     0     0     1   289]]
```

Discussions and Conclusion

- 從結果來看U2R 的攻擊在dataset中較少，所以能被訓練到的次數也不多，從結果中的accuracy 和confusion matrix 都可以發現錯誤率較高
- 可能因為使用的dataset資料比數較少，感覺這些model表現都還不錯，有和其他同學的Naive Bayes的mode比較發現，Naive Bayes可能就表現就不會這麼好
- 感覺在資料Transformation上，還有機會多花點功夫，像是把極端值去掉等 或者是用PCA的方式把資料降為