Application of ML for Networking Lab 1

Student ID: 310581027

Student Name: 宋煜祥

Read Data

- Train_dataset: kddcup.data_10_percent
- Test_dataset: corrected
- Read Data: By using Pandas to convert to Data Frame.

Read Data

```python
col_names = ["duration","protocol_type","service","flag","src_bytes",
"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
"logged_in","num_compromised","root_shell","su_attempted","num_root",
"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
"is_host_login","is_guest_login","count","srv_count","serror_rate",
"srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","attack_types"]

# train_data = pd.read_table("Dataset/kddcup.data.txt",header=None, sep=',' ,on_bad_lines='skip' ,names = col_names)
train_data = pd.read_table("Dataset/kddcup.data_10_percent.txt",header=None, sep=',' ,on_bad_lines='skip' ,names = col_names)
test_data = pd.read_table("Dataset/corrected",header=None, sep=',' ,on_bad_lines='skip' ,names = col_names)
```

- And add columns to Data Frame.

```python
train_data
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_h
0	0	tcp	http	SF	181	5450	0	0	0	0	...	9	1.0	
1	0	tcp	http	SF	239	486	0	0	0	0	...	19	1.0	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29	1.0	
3	0	tcp	http	SF	219	1337	0	0	0	0	...	39	1.0	
4	0	tcp	http	SF	217	2032	0	0	0	0	...	49	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
494015	0	tcp	http	SF	310	1881	0	0	0	0	...	255	1.0	
494016	0	tcp	http	SF	282	2286	0	0	0	0	...	255	1.0	
494017	0	tcp	http	SF	203	1200	0	0	0	0	...	255	1.0	
494018	0	tcp	http	SF	291	1200	0	0	0	0	...	255	1.0	
494019	0	tcp	http	SF	219	1234	0	0	0	0	...	255	1.0	

494020 rows × 42 columns

```python
train_data
```

t_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	attack_types
0.11	0.00	0.00	0.00	0.0	0.0	normal.
0.05	0.00	0.00	0.00	0.0	0.0	normal.
0.03	0.00	0.00	0.00	0.0	0.0	normal.
0.03	0.00	0.00	0.00	0.0	0.0	normal.
0.02	0.00	0.00	0.00	0.0	0.0	normal.
...	...	...	...	...	...	...
0.01	0.05	0.00	0.01	0.0	0.0	normal.
0.17	0.05	0.00	0.01	0.0	0.0	normal.
0.06	0.05	0.06	0.01	0.0	0.0	normal.
0.04	0.05	0.04	0.01	0.0	0.0	normal.
0.17	0.05	0.00	0.01	0.0	0.0	normal.

Data Pre-Processing

- **Separate 5 labels to each attack types.**
- **normal, dos, u2r, r2l, probe.**

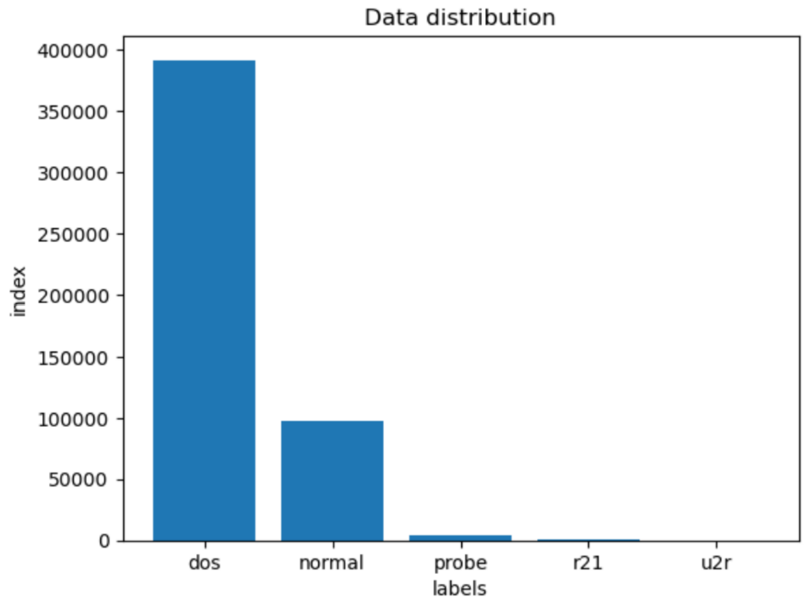
DataProcessing

```
conditions = [(train_data['attack_types'] == 'back.'),
(train_data['attack_types'] == 'buffer_overflow.'), (train_data['attack_types'] == 'ftp_write.'),
(train_data['attack_types'] == 'guess_passwd.'), (train_data['attack_types'] == 'imap.'),
(train_data['attack_types'] == 'ipsweep.'), (train_data['attack_types'] == 'land.'),
(train_data['attack_types'] == 'loadmodule.'), (train_data['attack_types'] == 'multihop.'),
(train_data['attack_types'] == 'neptune.'), (train_data['attack_types'] == 'nmap.'),
(train_data['attack_types'] == 'perl.'), (train_data['attack_types'] == 'phf.'),
(train_data['attack_types'] == 'pod.'), (train_data['attack_types'] == 'portsweep.'),
(train_data['attack_types'] == 'rootkit.'), (train_data['attack_types'] == 'satan.'),
(train_data['attack_types'] == 'smurf.'), (train_data['attack_types'] == 'spy.'),
(train_data['attack_types'] == 'teardrop.'), (train_data['attack_types'] == 'warezclient.'),
(train_data['attack_types'] == 'warezmaster.')]

choices = ['dos', 'u2r', 'r2l', 'r2l', 'r2l', 'probe', 'dos', 'u2r', 'r2l', 'dos', 'probe', 'u2r', 'r2l', 'dos', 'probe',
'u2r', 'probe', 'dos', 'r2l', 'dos', 'r2l', 'r2l'] # dos, u2r, r2l, probe

train_data['label'] = np.select(conditions, choices, default='normal')
# train_data.to_csv("Dataset/kddcup.data_10_percent.csv")
```

```
train_data
```

;ame_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	attack_types	label
0.11	0.00	0.00	0.00	0.0	0.0	normal.	normal
0.05	0.00	0.00	0.00	0.0	0.0	normal.	normal
0.03	0.00	0.00	0.00	0.0	0.0	normal.	normal
0.03	0.00	0.00	0.00	0.0	0.0	normal.	normal
0.02	0.00	0.00	0.00	0.0	0.0	normal.	normal
...	...	...	...	...	...	...	...
0.01	0.05	0.00	0.01	0.0	0.0	normal.	normal
0.17	0.05	0.00	0.01	0.0	0.0	normal.	normal
0.06	0.05	0.06	0.01	0.0	0.0	normal.	normal
0.04	0.05	0.04	0.01	0.0	0.0	normal.	normal
0.17	0.05	0.00	0.01	0.0	0.0	normal.	normal

- Data distribution



- Convert data to float
- Select the features that don’t have string.

```
num_features = ["duration", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent", "hot", "num_failed_logins",
                "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations", "num_shells",
                "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login", "count", "srv_count", "serror_rate",
                "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate",
                "dst_host_count", "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
                "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate", "dst_host_srv_serror_rate",
                "dst_host_rerror_rate", "dst_host_srv_rerror_rate"]
train_x = train_data[num_features].astype(float)
test_x = test_data[num_features].astype(float)
train_x
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	dst_host_count	dst_host_srv_
0	0.0	181.0	5450.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	9.0	
1	0.0	239.0	486.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	19.0	
2	0.0	235.0	1337.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	29.0	
3	0.0	219.0	1337.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	39.0	
4	0.0	217.0	2032.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	49.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
494015	0.0	310.0	1881.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	86.0	
494016	0.0	282.0	2286.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	6.0	
494017	0.0	203.0	1200.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	16.0	
494018	0.0	291.0	1200.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	26.0	
494019	0.0	219.0	1234.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	6.0	

494020 rows × 38 columns

Feature transformation

- Using MinMaxScaler to let data among 0,1.

```
train_y = train_data['label']
test_y = test_data['label']
```

```
#Ignoring the deprecation warnings
import warnings
warnings.filterwarnings("ignore", category = DeprecationWarning)

#Rescaling the data
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()

min_max_scaler = min_max_scaler.fit(train_x)
train_x = min_max_scaler.transform(train_x)

min_max_scaler = min_max_scaler.fit(test_x)
test_x = min_max_scaler.transform(test_x)

train_x
```

```
array([[0.00000000e+00, 2.61041764e-07, 1.05713002e-03, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 3.44690506e-07, 9.42688423e-05, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 3.38921627e-07, 2.59336301e-04, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [0.00000000e+00, 2.92770597e-07, 2.32762574e-04, ...,
         1.00000000e-02, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 4.19685930e-07, 2.32762574e-04, ...,
         1.00000000e-02, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 3.15846112e-07, 2.39357513e-04, ...,
         1.00000000e-02, 0.00000000e+00, 0.00000000e+00]])
```

Training – 1: Random Forest

- Choosing **Random Forest. (iterations = 100.)**

Training: RandomForest

```
#Training a classifier
from sklearn.ensemble import RandomForestClassifier
import time
clf = RandomForestClassifier(n_estimators=100 ,random_state = 0)
t0 = time.time()
clf.fit(train_x, train_y)
tt = time.time() - t0
pred = clf.predict(test_x)
print ("Classifier trained in {} seconds.".format(round(tt, 3)))
```

```
Classifier trained in 30.641 seconds.
```

- K-FOLD, K = 3. (K > 2).

K-FOLD

```
from sklearn.model_selection import KFold

kf =KFold(n_splits=3, shuffle=True, random_state=42)
cnt = 1

for train_index, test_index in kf.split(train_x, train_y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:{len(test_index)}')
    cnt += 1
```

```
Fold:1, Train set: 329346, Test set:164674
Fold:2, Train set: 329347, Test set:164673
Fold:3, Train set: 329347, Test set:164673
```

- **Accuracy & Cross value score**

Accuracy_score

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
acc = accuracy_score(pred, test_y)
cv_score = cross_val_score(clf, train_x, train_y, cv=kf)
print("Scores for each fold: ",cv_score)
print ("Accuracy is {}.".format(round(acc,4)))
```

```
Scores for each fold:  [0.99961135 0.99969637 0.99965993]
Accuracy is 0.9762.
```

● Confusion matrix

Confusion_matrix

```
from sklearn.metrics import confusion_matrix
label = ["dos", "normal", "probe", "r2l", "u2r"]
Confusion_Matrix = pd.DataFrame(confusion_matrix(test_y, pred,labels=label))
Confusion_Matrix.columns = label
Confusion_Matrix.index = label
print(Confusion_Matrix)
```

```
           dos  normal  probe  r2l  u2r
dos     223267      15     14    2    0
normal     596   77748    970    3    5
probe        4      12   2361    0    0
r2l         55    5598     87  251    2
u2r          0      37      0    0    2
```

● Precision score

Recall_score

```
from sklearn.metrics import recall_score
print(recall_score(test_y, pred, average='macro'))
print(recall_score(test_y, pred, average='micro'))
print(recall_score(test_y, pred, average='weighted'))
print(recall_score(test_y, pred, average=None))
```

```
0.6132902149243435
0.9762080063273842
0.9762080063273842
[0.99986117 0.98015683 0.99326883 0.0418822  0.05128205]
```

● Recall score

Recall_score

```
from sklearn.metrics import recall_score
print(recall_score(test_y, pred, average='macro'))
print(recall_score(test_y, pred, average='micro'))
print(recall_score(test_y, pred, average='weighted'))
print(recall_score(test_y, pred, average=None))
```

```
0.6132902149243435
0.9762080063273842
0.9762080063273842
[0.99986117 0.98015683 0.99326883 0.0418822  0.05128205]
```

● F1-score

F1_score

```
from sklearn.metrics import f1_score
print(f1_score(test_y, pred, average='macro'))
print(f1_score(test_y, pred, average='micro'))
print(f1_score(test_y, pred, average='weighted'))
print(f1_score(test_y, pred, average=None))
```

```
0.5861086178798002
0.9762080063273842
0.9682931783509195
[0.99846608 0.95553425 0.81287657 0.08033285 0.08333333]
```

Training – 2: SVM

- Choosing SVM

Training: SVM

```
from sklearn import svm
clf=svm.SVC(kernel='rbf',C=1,gamma='auto')
t0 = time.time()
clf.fit(train_x, train_y)
tt = time.time() - t0
pred = clf.predict(test_x)
print ("Classifier trained in {} seconds.".format(round(tt, 3)))
```

```
Classifier trained in 541.833 seconds.
```

- **Accuracy & Cross value score**

Accuracy_score

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
acc = accuracy_score(pred, test_y)
cv_score = cross_val_score(clf, train_x, train_y, cv=kf)
print("Scores for each fold: ",cv_score)
print ("Accuracy is {}.".format(round(acc,4)))
```

```
Scores for each fold:  [0.99291934 0.9930529  0.99275534]
Accuracy is 0.8751.
```

- **Confusion matrix**

Confusion_matrix

```
from sklearn.metrics import confusion_matrix
label = ["dos", "normal", "probe", "r2l", "u2r"]
Confusion_Matrix = pd.DataFrame(confusion_matrix(test_y, pred,labels=label))
Confusion_Matrix.columns = label
Confusion_Matrix.index = label
print(Confusion_Matrix)
```

	dos	normal	probe	r2l	u2r
dos	193410	29864	24	0	0
normal	1418	76838	1016	50	0
probe	15	428	1934	0	0
r2l	8	5951	26	8	0
u2r	0	38	0	1	0

● Precision score

Precision_score

```
from sklearn.metrics import precision_score
print(precision_score(test_y, pred, average='macro'))
print(precision_score(test_y, pred, average='micro'))
print(precision_score(test_y, pred, average='weighted'))
print(precision_score(test_y, pred, average=None))
```

```
D:\Users\gcobs\anaconda3\envs\MLFN\lib\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))

0.49042629184436093
0.8751273996958483

D:\Users\gcobs\anaconda3\envs\MLFN\lib\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))

0.8933971442505151
[0.99260461 0.67926697 0.64466667 0.13559322 0.        ]

D:\Users\gcobs\anaconda3\envs\MLFN\lib\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

● **Recall score**

Recall_score

```
from sklearn.metrics import recall_score
print(recall_score(test_y, pred, average='macro'))
print(recall_score(test_y, pred, average='micro'))
print(recall_score(test_y, pred, average='weighted'))
print(recall_score(test_y, pred, average=None))
```

```
0.5299604155835757
0.8751273996958483
0.8751273996958483
[0.86615196 0.9686846  0.81363063 0.00133489 0.        ]
```

● **F1-score**

F1_score

```
from sklearn.metrics import f1_score
print(f1_score(test_y, pred, average='macro'))
print(f1_score(test_y, pred, average='micro'))
print(f1_score(test_y, pred, average='weighted'))
print(f1_score(test_y, pred, average=None))
```

```
0.4891285151022008
0.8751273996958483
0.873349756535064
[0.92507695 0.79856164 0.71936024 0.00264375 0.        ]
```

Training – 3: Decision Tree

● Choosing Decision Tree

Training: DecisionTree

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
t0 = time.time()
clf.fit(train_x, train_y)
tt = time.time() - t0
pred = clf.predict(test_x)
print ("Classifier trained in {} seconds.".format(round(tt, 3)))
```

```
Classifier trained in 3.379 seconds.
```

● Accuracy & Cross value Score

Accuracy_score

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
acc = accuracy_score(pred, test_y)
cv_score = cross_val_score(clf, train_x, train_y, cv=kf)
print("Scores for each fold: ",cv_score)
print ("Accuracy is {}.".format(round(acc,4)))
```

```
Scores for each fold:  [0.99938667 0.99954455 0.9993563 ]
Accuracy is 0.9709.
```

● Confusion matrix

Confusion_matrix

```
from sklearn.metrics import confusion_matrix
label = ["dos", "normal", "probe", "r2l", "u2r"]
Confusion_Matrix = pd.DataFrame(confusion_matrix(test_y, pred,labels=label))
Confusion_Matrix.columns = label
Confusion_Matrix.index = label
print(Confusion_Matrix)
```

```
           dos  normal  probe  r2l  u2r
dos     223227      47     21    3    0
normal     755   76311   1669  434  153
probe       12      24   2092  249    0
r2l          1    4975    126  338  553
u2r         17      17      0    4    1
```

- **Precision score**

Precision_score

```
from sklearn.metrics import precision_score
print(precision_score(test_y, pred, average='macro'))
print(precision_score(test_y, pred, average='micro'))
print(precision_score(test_y, pred, average='weighted'))
print(precision_score(test_y, pred, average=None))
```

```
0.5599594429016056
0.9708708834224461
0.9650068917085983
[0.99649572 0.93778111 0.53531218 0.32879377 0.00141443]
```

- **Recall score**

Recall_score

```
from sklearn.metrics import recall_score
print(recall_score(test_y, pred, average='macro'))
print(recall_score(test_y, pred, average='micro'))
print(recall_score(test_y, pred, average='weighted'))
print(recall_score(test_y, pred, average=None))
```

```
0.5847727921120447
0.9708708834224461
0.9708708834224461
[0.99968204 0.9620408  0.88010097 0.05639913 0.02564103]
```

- **F1-score**

F1_score

```
from sklearn.metrics import f1_score
print(f1_score(test_y, pred, average='macro'))
print(f1_score(test_y, pred, average='micro'))
print(f1_score(test_y, pred, average='weighted'))
print(f1_score(test_y, pred, average=None))
```

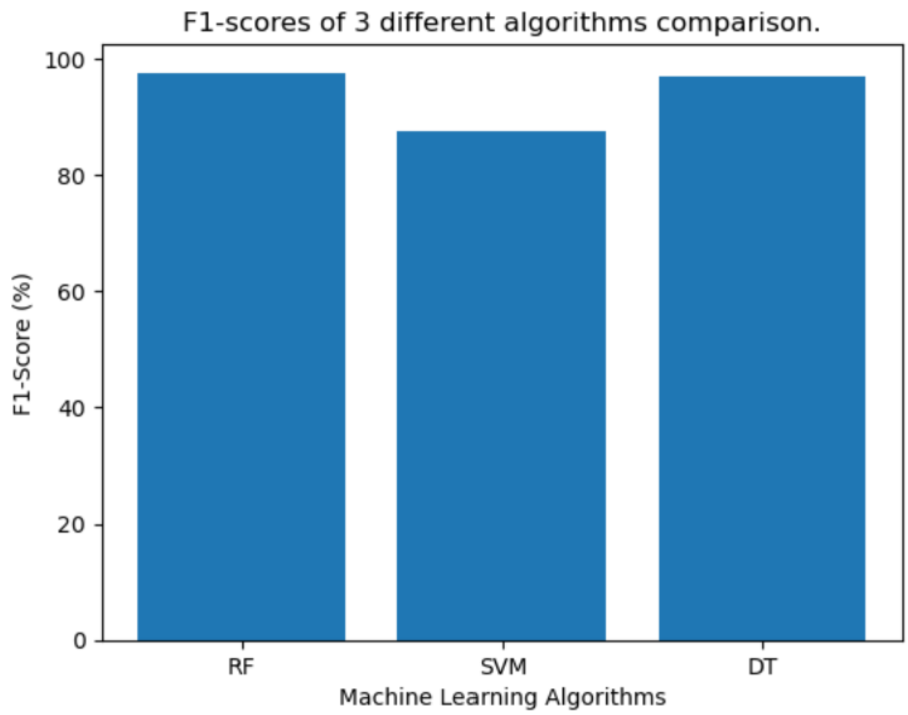
```
0.542503591634555
0.9708708834224461
0.9657194569551266
[0.99808634 0.94975606 0.66571201 0.09628258 0.00268097]
```

F1-scores of 3 different algorithms comparison.

- In this case, we use 3 different ML algorithms: Random Forest, SVM, and Decision Tree.
- According to the result, we can observe that Random Forest and Decision Tree get similar and good scores, and SVM gets a worse score.

F1-scores of 3 different algorithms comparison.

```
x = ['RF', 'SVM', 'DT']
x_len = np.arange(len(x))
y = [rf_f1_score*100, svm_f1_score*100, dt_f1_score*100 ]
plt.bar(x_len, y)
plt.xticks(x_len, x)
plt.xlabel('Machine Learning Algorithms')
plt.ylabel('F1-Score (%)')
plt.title('F1-scores of 3 different algorithms comparison.')
plt.show()
y
```




```
[97.62080063273842, 87.51273996958483, 97.01378328065871]
```