```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
%matplotlib inline

DDoS = pd.read_csv('/content/SDN_DDoS_.csv')
DDoS.head()
```

	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	Bwd Pkt Len Max	Bwd Pkt Len Min
0	245230	44	40	124937	1071	9100	0	2839.477273	1839.508257	517	0
1	1605449	107	149	1071	439537	517	0	10.009346	67.496680	27300	0
2	53078	5	5	66	758	66	0	13.200000	29.516097	638	0
3	6975	1	1	0	0	0	0	0.000000	0.000000	0	0
4	190141	13	16	780	11085	427	0	60.000000	130.042942	2596	0

```
DDoS.isnull().sum()
```

```
Flow Duration      0
Tot Fwd Pkts       0
Tot Bwd Pkts       0
TotLen Fwd Pkts    0
TotLen Bwd Pkts    0
                  ..
Idle Mean          0
Idle Std           0
Idle Max           0
Idle Min           0
Label              0
Length: 67, dtype: int64
```

```
from sklearn.model_selection import train_test_split
features = DDoS.drop('Label', axis=1)
labels = DDoS['Label']

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.4, random_s
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=4

for dataset in [y_train, y_val, y_test]:
```

```
print(round(len(dataset) / len(labels), 2))
```

```
0.6
0.2
0.2
```

```
X_train.to_csv('train_features.csv', index=False)
X_val.to_csv('val_features.csv', index=False)
X_test.to_csv('test_features.csv', index=False)

y_train.to_csv('train_labels.csv', index=False)
y_val.to_csv('val_labels.csv', index=False)
y_test.to_csv('test_labels.csv', index=False)
```

```
import joblib
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)

tr_features = pd.read_csv('train_features.csv')
tr_labels = pd.read_csv('train_labels.csv')
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))

lr = LogisticRegression()
parameters = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

cv = GridSearchCV(lr, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
cv.best_estimator_
```

```
LogisticRegression(C=1000)
```

```
joblib.dump(cv.best_estimator_, 'LR_model.pkl')
```

```
['LR_model.pkl']
```

```
from sklearn.svm import SVC
svc = SVC()
parameters = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10]
}

cv = GridSearchCV(svc, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
BEST PARAMS: {'C': 0.1, 'kernel': 'linear'}

1.0 (+/-0.0) for {'C': 0.1, 'kernel': 'linear'}
0.9 (+/-0.0) for {'C': 0.1, 'kernel': 'rbf'}
1.0 (+/-0.0) for {'C': 1, 'kernel': 'linear'}
0.9 (+/-0.0) for {'C': 1, 'kernel': 'rbf'}
1.0 (+/-0.0) for {'C': 10, 'kernel': 'linear'}
0.9 (+/-0.0) for {'C': 10, 'kernel': 'rbf'}
```

```
cv.best_estimator_
```

```
SVC(C=0.1, kernel='linear')
```

```
joblib.dump(cv.best_estimator_, 'SVM_model.pkl')
```

```
['SVM_model.pkl']
```

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
parameters = {
    'hidden_layer_sizes': [(10,), (50,), (100,)],
    'activation': ['relu', 'tanh', 'logistic'],
    'learning_rate': ['constant', 'invscaling', 'adaptive']
}

cv = GridSearchCV(mlp, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
BEST PARAMS: {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'invs

0.999 (+/-0.001) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate
0.998 (+/-0.003) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate
```

```
0.999 (+/-0.003) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'learning_rate
0.999 (+/-0.001) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate
0.999 (+/-0.0) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate':
0.999 (+/-0.001) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate
0.999 (+/-0.0) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate'
0.999 (+/-0.001) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rat
0.997 (+/-0.006) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rat
0.999 (+/-0.0) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate':
0.999 (+/-0.001) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate
0.999 (+/-0.001) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate
0.999 (+/-0.0) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate':
0.999 (+/-0.001) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate
0.999 (+/-0.0) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate':
0.999 (+/-0.001) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rat
0.999 (+/-0.001) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rat
0.999 (+/-0.0) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate'
0.999 (+/-0.001) for {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_
0.999 (+/-0.001) for {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_
0.999 (+/-0.001) for {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (50,), 'learning_ra
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (50,), 'learning_ra
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (50,), 'learning_ra
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_r
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_r
0.999 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_r
```

```
cv.best_estimator_
```

```
MLPClassifier(hidden_layer_sizes=(50,), learning_rate='invscaling')
```

```
joblib.dump(cv.best_estimator_, 'MLP_model.pkl')
```

```
['MLP_model.pkl']
```

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
parameters = {
    'n_estimators': [5, 50, 250],
    'max_depth': [2, 4, 8, 16, 32, None]
}

cv = GridSearchCV(rf, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
BEST PARAMS: {'max_depth': 8, 'n_estimators': 50}

0.994 (+/-0.004) for {'max_depth': 2, 'n_estimators': 5}
0.997 (+/-0.003) for {'max_depth': 2, 'n_estimators': 50}
```

```
0.998 (+/-0.002) for {'max_depth': 2, 'n_estimators': 250}
0.999 (+/-0.0) for {'max_depth': 4, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': 4, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 4, 'n_estimators': 250}
1.0 (+/-0.0) for {'max_depth': 8, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': 8, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 8, 'n_estimators': 250}
1.0 (+/-0.0) for {'max_depth': 16, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': 16, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 16, 'n_estimators': 250}
1.0 (+/-0.0) for {'max_depth': 32, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': 32, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': 32, 'n_estimators': 250}
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 5}
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 50}
1.0 (+/-0.0) for {'max_depth': None, 'n_estimators': 250}
```

```
cv.best_estimator_
```

```
RandomForestClassifier(max_depth=8, n_estimators=50)
```

```
joblib.dump(cv.best_estimator_, 'RF_model.pkl')
```

```
['RF_model.pkl']
```

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
parameters = {
    'n_estimators': [5, 50, 250, 500],
    'max_depth': [1, 3, 5, 7, 9],
    'learning_rate': [0.01, 0.1, 1, 10, 100]
}

cv = GridSearchCV(gb, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 5}
```

```
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 500}
0.9 (+/-0.0) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 5}
0.9 (+/-0.0) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 50}
0.9 (+/-0.0) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 250}
0.9 (+/-0.0) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 500}
0.9 (+/-0.0) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 5}
0.9 (+/-0.0) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 50}
0.9 (+/-0.0) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 250}
0.9 (+/-0.0) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 500}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 5}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 50}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 250}
1.0 (+/-0.0) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 500}
```

```
cv.best_estimator_
```

```
GradientBoostingClassifier(learning_rate=0.01, max_depth=1, n_estimators=250)
```

```
joblib.dump(cv.best_estimator_, 'GB_model.pkl')
```

```
['GB_model.pkl']
```

```
import joblib
```

```
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score
from time import time

val_features = pd.read_csv('val_features.csv')
val_labels = pd.read_csv('val_labels.csv')

te_features = pd.read_csv('test_features.csv')
te_labels = pd.read_csv('test_labels.csv')



from sklearn.metrics import accuracy_score, precision_score, recall_score
models = {}

for mdl in ['LR', 'SVM', 'MLP', 'RF', 'GB']:
    models[mdl] = joblib.load('{}_model.pkl'.format(mdl))



def evaluate_model(name, model, features, labels):
    start = time()
    pred = model.predict(features)
    end = time()
    accuracy = round(accuracy_score(labels, pred), 3)
    precision = round(precision_score(labels, pred), 3)
    recall = round(recall_score(labels, pred), 3)
    print('{} -- Accuracy: {} / Precision: {} / Recall: {} / Latency: {}ms'.format(name,
                                                                                   accuracy,
                                                                                   precision,
                                                                                   recall,
                                                                                   round((enc



for name, mdl in models.items():
    evaluate_model(name, mdl, val_features, val_labels)
```

```
 LR -- Accuracy: 1.0 / Precision: 1.0 / Recall: 1.0 / Latency: 24.4ms
 SVM -- Accuracy: 1.0 / Precision: 0.999 / Recall: 1.0 / Latency: 30.9ms
 MLP -- Accuracy: 1.0 / Precision: 0.999 / Recall: 0.999 / Latency: 31.5ms
 RF -- Accuracy: 1.0 / Precision: 1.0 / Recall: 1.0 / Latency: 71.3ms
 GB -- Accuracy: 1.0 / Precision: 1.0 / Recall: 1.0 / Latency: 52.9ms
```

```
evaluate_model('Random Forest', models['RF'], te_features, te_labels)
```

```
 Random Forest -- Accuracy: 1.0 / Precision: 1.0 / Recall: 0.999 / Latency: 56.4ms
```

✓ 0s completed at 11:25 PM