Tugas Project 2 Mata Kuliah Sistem Keamanan Cerdas

Malware Detection in Network Traffic Data In IoT Using Machine Learning

Disusun oleh:

Seagata Ade Pratama Barus (1301210371)
Andi Ahmad Ramadhan Makmur Perkasa (1301213166)

Bandung

2024

Deskripsi Proyek

Proyek ini bertujuan untuk melakukan deteksi malware pada traffic jaringan IoT menggunakan machine learning demi mendeteksi traffic yang bisa saja berupa malware.

Dataset

Stratosphere Laboratory. A labeled dataset with malicious and benign IoT network traffic. January 22th. Agustin Parmisano, Sebastian Garcia, Maria Jose Erquiaga. https://www.stratosphereips.org/datasets-iot23

Algoritma

1. Support Vector Machine (SVM)

Mesin vektor pendukung (bahasa Inggris: Support-vector machine atau disingkat SVM) adalah sebuah algoritma klasifikasi untuk data linear dan non-linear. SVM menggunakan mapping non-linear untuk mentransformasikan training data awal ke dimensi yang lebih tinggi.
(https://id.wikipedia.org/wiki/Mesin_vektor_pendukung)

2. Isolation Forest

Isolation Forest adalah sebuah algoritma untuk deteksi anomali data yang awalnya dikembangkan oleh Fei Tony Liu pada tahun 2008. Isolation Forest mendeteksi anomali dengan menggunakan pohon biner.
(https://en.wikipedia.org/wiki/Isolation_forest)

3. Random Forest

Random forests atau random decision forests adalah metode pembelajaran ensemble untuk klasifikasi, regresi, dan tugas-tugas lain yang beroperasi dengan membangun banyak pohon keputusan pada waktu pelatihan.
(https://en.wikipedia.org/wiki/Random_forest)

4. XGBoost

XGBoost adalah optimized distributed gradient boosting library yang dirancang agar sangat efisien, fleksibel, dan portable. Library ini mengimplementasikan algoritma pembelajaran mesin di bawah kerangka kerja Gradient Boosting.
(https://xgboost.readthedocs.io/en/stable/)

5. Naïve Bayes(Gaussian)

Naive bayes merupakan metode pengklasifikasian berdasarkan probabilitas sederhana dan dirancang agar dapat dipergunakan dengan asumsi antar variabel penjelas saling bebas (independen).
(https://dqlab.id/mengenal-naive-bayes-sebagai-salah-satu-algoritma-data-science)

Distribusi normal atau distribusi Gaussian adalah jenis distribusi probabilitas kontinu untuk variabel acak bernilai riil.
https://en.wikipedia.org/wiki/Normal_distribution

Implementasi

Dataset

kami memakai perbandingan 19000 baris data malware dan 17000 baris data benign

```
df_1 = df_1.sample(n=19000, random_state=13)
df_0 = df_0.sample(n=17000, random_state=13)
```

Preprocess

Drop kolom yang memiliki nilai NULL yang banyak

```
df.drop(['service','orig_bytes','resp_bytes','local_orig','local_resp',
'tunnel_parents'],axis=1,inplace=True)
```

Drop kolom yang tidak penting

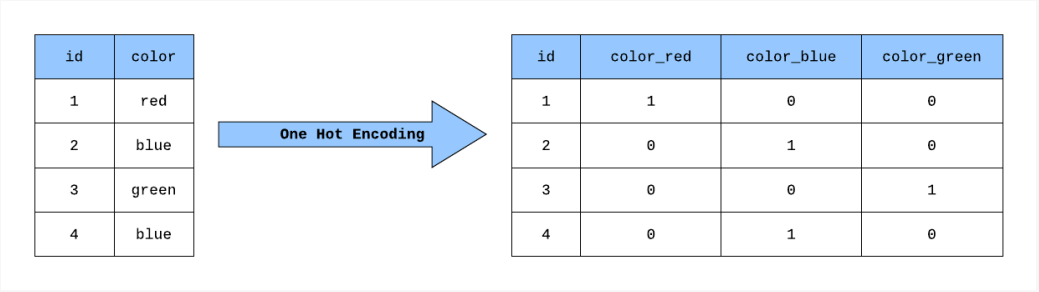
```
df['id.orig_h']= label_encoder.fit_transform(df['id.orig_h'])
df['id.resp_h']= label_encoder.fit_transform(df['id.resp_h'])
df.drop(['missed_bytes'],axis=1,inplace=True)
df.drop(['detailed-label'],axis=1,inplace=True)
df.drop(['uid'],axis=1,inplace=True)
df.drop(['ts'],axis=1,inplace=True)
df['conn_state']= label_encoder.fit_transform(df['conn_state'])
```

Melakukan encoding kolom yang memiliki nilai bukan numerik menjadi numerik

```
label_encoder = preprocessing.LabelEncoder()
df['label']= label_encoder.fit_transform(df['label'])
df['duration'] = pd.to_numeric(df['duration'])
```

Memakai One-hot encoder untuk kolom proto

```
onehot = pd.get_dummies(df['proto'])
df = df.join(onehot)
df.drop(['proto'],axis=1,inplace=True)
```



Pembagian Train & Test

```
X = df.drop('label', axis=1)
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=34)
```

Melakukan Normalisasi data frame train & test

```
scaler = Normalizer()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

SVM

```
SVM = SVC()
```

Hyperparameter Tuning

```
param = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
         'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
         'tol': [0.001, 0.01, 0.1, 1],
         'kernel': ['rbf']}

random_search = RandomizedSearchCV(SVM,
                                   param,
                                   n_iter=5,
                                   scoring='accuracy',
                                   n_jobs=-1,
                                   random_state=34)
```

Isolation Forest

```
IsoForest = IsolationForest(n_estimators=100, contamination=0.1)
```

Hyperparameter Tuning

```
param = {'n_estimators': list(range(100, 1000, 5)),
         'contamination': [0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5],
         'max_samples': ['auto'],
         'max_features': [1],
         'bootstrap': [True, False]}

random_search = RandomizedSearchCV(IsoForest,
                                   param,
                                   n_iter=10,
                                   scoring='accuracy',
                                   n_jobs=-1,
                                   cv=5,
                                   random_state=34)
```

Random Forest

```
rf = RandomForestClassifier()
```

Hyperparameter Tuning

```
param = {'n_estimators': list(range(100, 1000, 5)),
         'max_features': ['auto', 'sqrt'],
         'min_samples_split': list(range(2, 10, 1)),
         'min_samples_leaf': [1, 2],
         'bootstrap': [True, False]}

random_search = RandomizedSearchCV(rf,
                                   param,
                                   n_iter=5,
                                   scoring='accuracy',
                                   n_jobs=-1,
                                   cv=5,
                                   random_state=34)
```

XGBoost

```
XGB = XGBClassifier()
```

Hyperparameter Tuning

```
ran = range(1,10,1)

param = {'n_estimators': list(range(50, 1000, 5)),
               'max_depth': list(range(1,10,1)),
               'learning_rate': [0.001,0.01, 0.05, 0.1, 0.2, 0.3],
               'subsample': [i / 10.0 for i in ran],
               'colsample_bytree': [i / 10.0 for i in ran],
               'gamma': [i / 10.0 for i in ran]}

random_search = RandomizedSearchCV(XGB,
                                   param,
                                   n_iter=5,
                                   scoring='accuracy',
                                   n_jobs=-1,
                                   cv=5,
                                   random_state=34)
```

Naïve Bayes(Gaussian)

```
NB = GaussianNB()
```

Hyperparameter Tuning

```
cv_method = RepeatedStratifiedKFold(n_splits=5,
                                    n_repeats=3)

ran = range(1,1000000,1)
param = {'var_smoothing': [i / 1000000.0 for i in ran]}

random_search = RandomizedSearchCV(NB,
                           param,
                           scoring='accuracy',
                           n_jobs=-1,
                           cv=5,
                           random_state=34)
```

XGBoost 2

memakai 305.000 baris benign dan 300.000 malware dikarenakan akurasi benign yang lebih rendah dibandingkan malware dengan kata lain False Positive pada confusion matrix XGBoost sebelumnya

```
df_0 = df_0.sample(n=305000, random_state=13)
df_1 = df_1.sample(n=300000, random_state=13)
```

Preprocessing & Split data frame, sama seperti sebelumnya

```
df.replace('-', pd.NA, inplace=True)
df.drop(['service','orig_bytes','resp_bytes','local_orig','local_resp',
'tunnel_parents'],axis=1,inplace=True)
label_encoder = preprocessing.LabelEncoder()

df['label']= label_encoder.fit_transform(df['label'])
df['duration'] = pd.to_numeric(df['duration'])

df.drop(['duration'],axis=1,inplace=True)
df.dropna(subset=['history'], inplace=True)
df['history']= label_encoder.fit_transform(df['history'])

onehot = pd.get_dummies(df['proto'])
df = df.join(onehot)
df.drop(['proto'],axis=1,inplace=True)

df['id.orig_h']= label_encoder.fit_transform(df['id.orig_h'])
df['id.resp_h']= label_encoder.fit_transform(df['id.resp_h'])
df.drop(['missed_bytes'],axis=1,inplace=True)
df.drop(['detailed-label'],axis=1,inplace=True)
df.drop(['uid'],axis=1,inplace=True)
df.drop(['ts'],axis=1,inplace=True)
df['conn_state']= label_encoder.fit_transform(df['conn_state'])

X = df.drop('label', axis=1)
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=34)
scaler = Normalizer()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Memakai parameter terbaik yang sudah didapatkan pada random search sebelumnya dan di modif sedikit

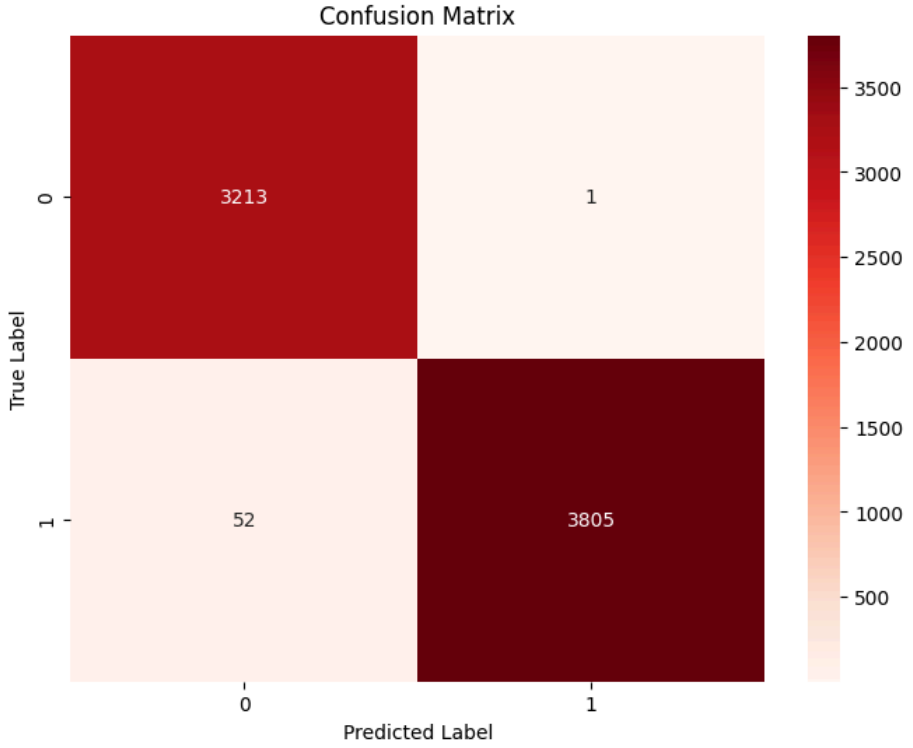
```
XGB = XGBClassifier(subsample= 0.9, n_estimators= 1001, max_depth= 10,
learning_rate= 0.98, gamma= 0.3, colsample_bytree= 0.75)
```

Hasil & Diskusi

Algoritma	Akurasi	Precision		F1-Score	
		0	1	0	1
SVM	0.90595	0.80	1.0	0.89	0.92
Isolation Forest	0.54504	0.2	1.0	0.03	0.70
Random Forest	0.99250	0.98	1.0	0.99	0.99
XGBoost	0.99604	0.99	1.0	1.0	1.0
Naïve Bayes (Gaussian)	0.98557	0.97	1.0	0.98	0.99
XGBoost 2	0.99852	1.0	1.0	1.0	1.0

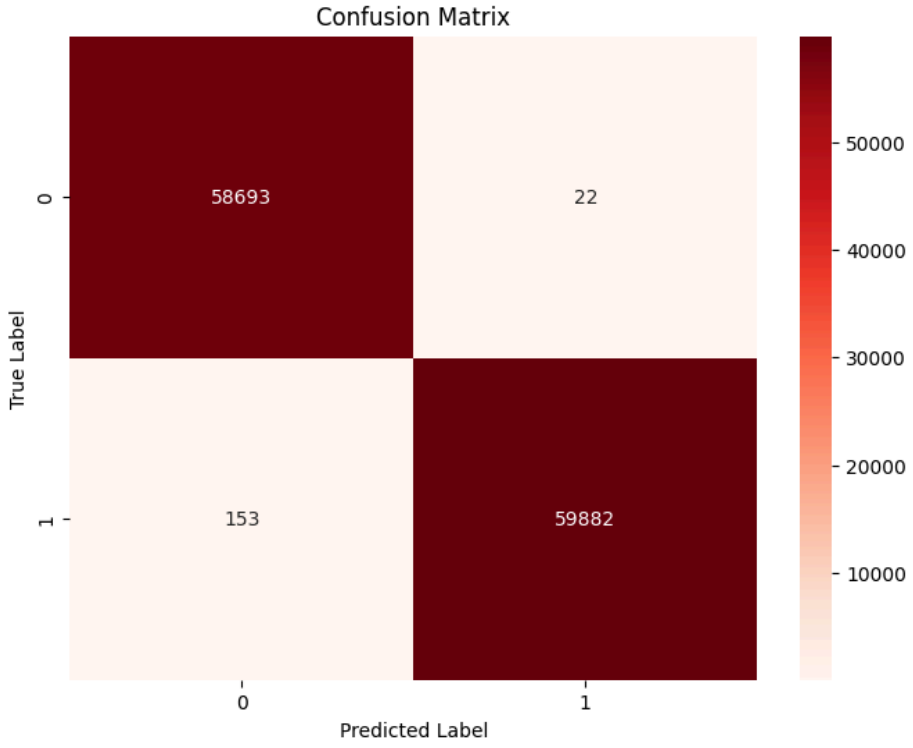
Berdasarkan ke-5 algoritma yang telah dipakai kami menemukan algoritma yang memiliki akurasi tertinggi dan Precision serta F1-Score tertinggi adalah XGBoost 2 dengan 0.99852 dan disusul oleh Random Forest dengan 0.99250 walaupun akurasinya dapat dibilang dekat namun F1-Score dari keduanya berbeda 0.1 pada 0 dan 1. walaupun begitu XGBoost 2 tidak berarti menjadi algoritma tanpa kesalahan dan hal tersebut dapat dilihat pada confusion matrix dibawah

Confusion Matrix Random Forest



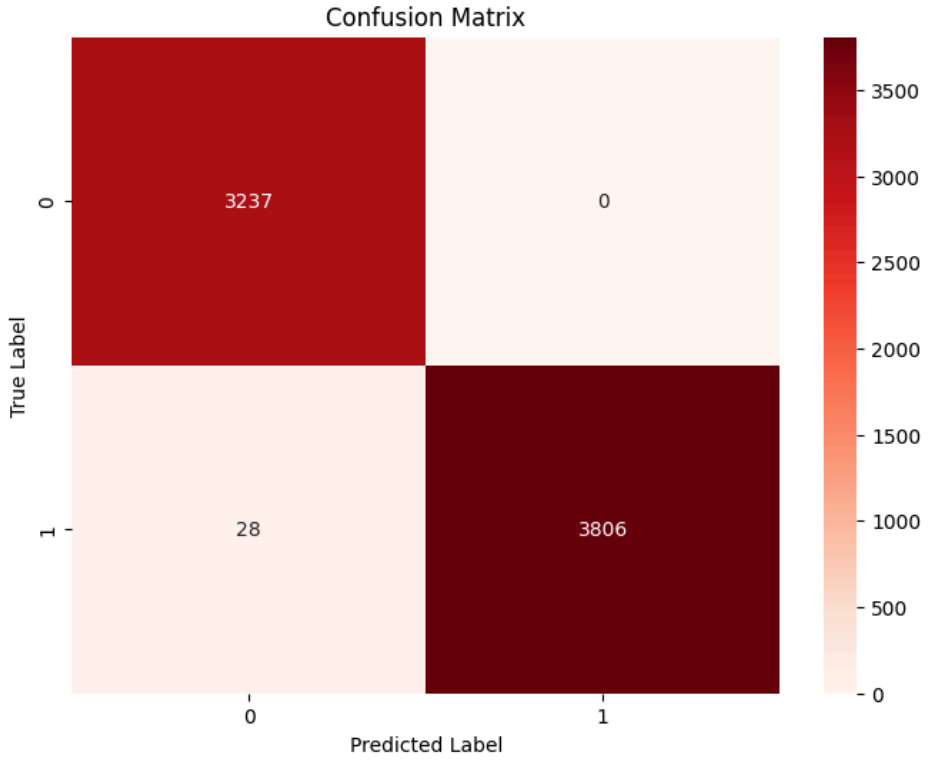
TP = 3805
TN = 3213
FN = 52
FP = 1

Confusion Matrix XGBoost 2



TP = 59882
TN = 58693
FN = 153
FP = 22

Confusion Matrix XGBoost



TP = 3806
TN = 3237
FN = 28
FP = 0

Seperti yang terlihat diatas meskipun XGBoost 2 memiliki akurasi yang lebih tinggi namun masih terdapat banyak kesalahan karena data tesnya yang lebih besar akan tetapi jika dibandingkan dengan random forest maka XGBoost masih menjadi opsi yang lebih baik walaupun pada paper yang kami jadikan acuan masih terdapat algoritma yang lebih baik dimana mereka menggabungkan beberapa algoritma menjadi 1 namun hasil yang kami dapatkan lebih tinggi akurasinya dibanding Fusion model yang mereka pakai

Algoritma	Precision	Recall
XGBoost	99.8	100
Fusion Model	97.2	98.1

Kesimpulan

XGBoost merupakan algoritma terbaik untuk kasus dan dataset yang kami ambil ini dan hal tersebut bisa saja overfitting ataupun kebetulan saja, dan yang menjadi algoritma terbaik nomor 2 untuk kasus ini adalah Random Forest.

Referensi

- Wang X. and Lu X., A host-based anomaly detection framework using XGBoost and LSTM for IoT devices, *Wireless Communications and Mobile Computing*. (2020) 2020, 13, 8838571, https://doi.org/10.1155/2020/8838571.
- Jamal, A., Hayat, M. F., & Nasir, M. (2022). Malware detection and classification in IoT network using ANN. Mehran University Research Journal Of Engineering & Technology, 41(1), 80–91. https://search.informit.org/doi/10.3316/informit.263296849285942
- Almazroi, Abdulwahab Ali and Ayub, Nasir, Deep learning hybridization for improved malware detection in smart Internet of Things. 2024, 38570575, https://www.scopus.com/record/display.uri?eid=2-s2.0-85189918564&origin=resultslist&sort=plf-f&src=s&sid=80da78a6ad8795c693fcd5403b73626c&sot=b&sdt=b&s=TITLE-ABS-KEY%28IOT+malware+detection%29&sl=36&sessionSearchId=80da78a6ad8795c693fcd5403b73626c&relpos=0