

TUGAS AKHIR - IF184802

PENGUNAAN DATA KOTOR UNTUK MELATIH SISTEM DETEKSI SERANGAN BERBASIS JARINGAN

KRISNA BADRU WIJAYA

NRP 05111740000048

Dosen Pembimbing

Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D.

NIP 198702182014041001

Program Studi Sarjana (S-1)

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022



TUGAS AKHIR - IF184802

PENGGUNAAN DATA KOTOR UNTUK MELATIH SISTEM DETEKSI SERANGAN BERBASIS JARINGAN

KRISNA BADRU WIJAYA

NRP 05111740000048

Dosen Pembimbing

Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D.

NIP 198702182014041001

Program Studi Sarjana (S-1)

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022



FINAL PROJECT - IF184802

THE USAGE OF NOISY DATA FOR TRAINING NETWORK-BASED INTRUSION DETECTION SYSTEM

KRISNA BADRU WIJAYA

NRP 05111740000048

Advisor

Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D.

NIP 198702182014041001

Bachelor Degree Program

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2022

LEMBAR PENGESAHAN

PENGUNAAN DATA KOTOR UNTUK MELATIH SISTEM DETEKSI SERANGAN BERBASIS JARINGAN

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **KRISNA BADRU WIJAYA**

NRP. 05111740000048

Disetujui oleh Tim Penguji Tugas Akhir:

- | | |
|---|---------------|
| 1. Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D. | Pembimbing |
| 2. Nama dan gelar ko-pembimbing/penguji | Ko-pembimbing |
| 3. Nama dan gelar penguji | Penguji |
| 4. Nama dan gelar penguji | Penguji |
| 5. Nama dan gelar penguji | Penguji |

SURABAYA

Juli, 2022

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : Krisna Badru Wijaya / 05111740000048
Program studi : Teknik Informatika
Dosen Pembimbing / NIP : Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D. /
198702182014041001

dengan ini menyatakan bahwa Tugas Akhir dengan judul “Penggunaan Data Kotor untuk Melatih Sistem Deteksi Serangan berbasis Jaringan” adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, Juli 2022

Mengetahui
Dosen Pembimbing

Mahasiswa

Baskoro Adi Purnomo, S.Kom., M.Kom.,
Ph.D.
NIP. 198702182014041001

Krisna Badru Wijaya
NRP. 05111740000048

ABSTRAK

PENGUNAAN DATA KOTOR UNTUK MELATIH SISTEM DETEKSI SERANGAN BERBASIS JARINGAN

Nama Mahasiswa / NRP : Krisna Badru Wijaya / 05111740000048
Departemen : Teknik Informatika FTEIC- ITS
Dosen Pembimbing : Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D.

Abstrak

Keamanan sistem dan jaringan merupakan hal yang sangat penting untuk dijaga oleh setiap perangkat yang terhubung ke jaringan. Kemudahan akses pada jaringan menjadi pintu terbuka untuk pelaku-pelaku kejahatan siber melakukan serangan terhadap perangkat targetnya. Berbagai jenis serangan dapat ditemui secara sadar atau tidak sadar saat mengakses data pada jaringan. Dikarenakan banyaknya jenis data dan koneksi yang terjadi di jaringan, sangat sulit untuk membedakan antara koneksi yang normal dan koneksi yang tidak normal pada jaringan. Untuk mencegah terjadinya kejahatan siber yang dapat membahayakan perangkat pengguna, setiap perangkat yang terhubung pada jaringan harus bisa mengidentifikasi keamanan koneksi yang masuk ke dalam perangkat.

Sistem deteksi serangan berbasis jaringan atau *Network Intrusion Detection System* (NIDS) adalah salah satu jenis sistem pendeteksi serangan yang bekerja pada jaringan komputer. NIDS berfungsi untuk memonitor, mengidentifikasi, dan menganalisa lalu lintas jaringan komputer dari potensi serangan pada keamanan jaringan. Data-data koneksi yang masuk pada jaringan akan diperiksa oleh NIDS. Apabila ditemukan keanehan-keanehan saat dilakukannya analisa, NIDS akan langsung menotifikasi *host*.

Salah satu metode yang digunakan pada NIDS untuk melatih sistem dalam mendeteksi serangan pada jaringan adalah dengan menggunakan algoritma *machine learning*. NIDS dilatih untuk mengenali koneksi-koneksi normal dan koneksi-koneksi anomali yang ada pada jaringan. Normalnya, data latih yang digunakan untuk melatih NIDS adalah data yang sama sekali tidak mengandung unsur serangan. Akan tetapi dalam prakteknya di dunia nyata, data seperti itu sangat sulit untuk didapatkan.

Tugas Akhir ini berjudul “Penggunaan Data Kotor untuk Melatih Sistem Deteksi Serangan berbasis Jaringan”. Tujuan utama pembuatan tugas akhir ini adalah untuk

membangun sistem deteksi serangan cerdas yang bisa mendeteksi serangan dengan menggunakan data kotor atau data campuran yang di dalamnya masih berpotensi mengandung unsur serangan.

Tugas akhir ini akan membahas tentang bagaimana cara kerja sistem, metode-metode yang digunakan untuk mengolah data latih pada NIDS, alat - alat yang digunakan dalam penelitian, serta perbandingan performa pada masing-masing metode yang digunakan dengan mempertimbangkan parameter jenis metode, jenis fitur, dan dataset yang dipakai dalam penelitian.

Kata kunci: NIDS, *Machine Learning*, Data Kotor, Data latih.

ABSTRACT

THE USAGE OF NOISY DATA FOR TRAINING NETWORK-BASED INTRUSION DETECTION SYSTEM

Student Name / NRP : Krisna Badru Wijaya / 05111740000048
Department : Teknik Informatika FTEIC- ITS
Advisor : Baskoro Adi Purnomo, S.Kom., M.Kom., Ph.D.

Abstract

Every device connected to the network need to maintain their network security system very well. Ease of access to the network is an open door for cybercriminals to carry out attacks on their target devices. Various types of attacks can be encountered consciously or unconsciously when accessing data on the network. Due to the large number of variety data and connections that occur on the network, it is very difficult to distinguish between normal connections and abnormal connections on the network. In order to prevent cybercrimes that can cause harm to user devices, every device connected to the network must be able to identify the security of the incoming connection.

Network Intrusion Detection System (NIDS) is intrusion detection system that works on computer networks. NIDS serves to monitor, identify, and analyze computer network traffic from potential attacks on network security. The incoming connection data on the network will be checked by NIDS. If anomalies are found during analysis, NIDS will immediately notify the host.

One of the methods used in NIDS to train the system to detect attacks on the network is to use machine learning algorithms. NIDS is trained to recognize normal connections and anomalous connections that exist on the network. Normally, the training data used to train NIDS is data that does not contain any elements of attack. However, in the real-time practice, such data is very difficult to obtain.

This Final Project is entitled " The Usage of Noisy Data for Training Network-Based Intrusion Detection System ". The main objective of making this final project is to build an intelligent intrusion detection system that can detect attacks using noisy data or mixed data which still has the potential to contain elements of attack in it.

This final project will discuss how the system works, the methods used to process training data on NIDS, the tools used in research, and the performance comparison of each method used by considering the parameters of the method type, feature type, and datasets used in research.

Keywords: NIDS, *Machine Learning*, Noisy Data, Data Train.

DAFTAR ISI

| | |
|--|-----|
| LEMBAR PENGESAHAN | i |
| PERNYATAAN ORISINALITAS | ii |
| ABSTRAK | iii |
| ABSTRACT | v |
| DAFTAR ISI | vii |
| DAFTAR GAMBAR | ix |
| DAFTAR TABEL | x |
| DAFTAR KODE PROGRAM | xi |
| BAB 1 PENDAHULUAN | 12 |
| 1.1 Latar Belakang | 12 |
| 1.2 Rumusan Masalah | 13 |
| 1.3 Batasan Masalah | 13 |
| 1.4 Tujuan | 13 |
| 1.5 Manfaat | 14 |
| BAB 2 Dasar Teori | 15 |
| 2.1 NIDS | 15 |
| 2.2 Python | 15 |
| 2.3 Machine Learning | 15 |
| 2.4 Sckit-learn | 16 |
| 2.5 PCAP | 16 |
| 2.6 Pcap | 17 |
| 2.7 Payload | 17 |
| BAB 3 METODOLOGI | 19 |
| 3.1 Analisa Metode Penelitian | 19 |
| 3.2 Perlengkapan Uji Coba Penelitian | 20 |
| 3.2.1 Perangkat Keras | 20 |
| 3.2.2 Perangkat Lunak | 20 |
| 3.2.3 Data Latih dan Data Testing | 21 |
| 3.3 Algoritma <i>Unsupervised Learning</i> | 23 |
| 3.3.1 One-class SVM | 23 |
| 3.3.2 Local Outlier Factor | 24 |
| 3.3.3 Isolation Forest | 25 |
| 3.4 Urutan Pelaksanaan Penelitian | 25 |

| | | |
|-----------------|--|----|
| 3.4.1 | Pra Proses Data..... | 26 |
| 3.4.2 | Proses Latih..... | 27 |
| 3.4.3 | Proses Testing..... | 28 |
| 3.4.4 | Proses Evaluasi Performa..... | 28 |
| 3.5 | Implementasi Sistem | 30 |
| 3.5.1 | Implementasi Pra Proses Data..... | 30 |
| 3.5.2 | Implementasi Proses Latih..... | 31 |
| 3.5.3 | Implementasi Proses Testing..... | 32 |
| 3.5.4 | Implementasi Evaluasi Performa..... | 33 |
| BAB 4 | Hasil dan Pembahasan..... | 36 |
| 4.1 | Skenario Evaluasi Hasil Uji Coba | 36 |
| 4.2 | Evaluasi Data Hasil Percobaan..... | 36 |
| 4.2.1 | Hasil Pengujian Protokol FTP..... | 36 |
| 4.2.2 | Hasil Pengujian Protokol SMTP..... | 39 |
| 4.2.3 | Hasil Pengujian Protokol HTTP..... | 43 |
| 4.2.4 | Hasil Evaluasi Keseluruhan..... | 46 |
| BAB 5 | Kesimpulan dan Saran..... | 49 |
| 5.1 | Kesimpulan..... | 49 |
| 5.2 | Saran | 50 |
| DAFTAR PUSTAKA | | 51 |
| BIODATA PENULIS | | 52 |

DAFTAR GAMBAR

| | |
|--|----|
| Gambar 2.1 <i>Packet capture</i> dengan Aplikasi Wireshark | 16 |
| Gambar 3.1 Diagram Alur Penelitian..... | 19 |
| Gambar 3.2 Gambaran One-class SVM | 23 |
| Gambar 3.3 Gambaran Local Outlier Factor | 24 |
| Gambar 3.4 Gambaran Isolation Forest | 25 |
| Gambar 3.5 Urutan Pelaksanaan Penelitian | 26 |
| Gambar 3.6 Urutan Pelaksanaan Penelitian | 26 |
| Gambar 3.7 Diagram Alur Proses Latih..... | 27 |
| Gambar 3.8 Diagram Alur Proses Testing | 28 |
| Gambar 3.9 Hasil Pra Proses Data | 31 |
| Gambar 3.10 Eksekusi Program Proses Latih | 32 |
| Gambar 3.11 Data Hasil Prediksi | 33 |
| Gambar 4.1 Grafik F ₂ -score Hasil Pengujian Protokol FTP | 37 |
| Gambar 4.2 Grafik DR Hasil Pengujian Protokol FTP | 38 |
| Gambar 4.3 Grafik FPR Hasil Pengujian Protokol FTP | 39 |
| Gambar 4.4 Grafik F ₂ -score Hasil Pengujian Protokol SMTP..... | 40 |
| Gambar 4.5 Grafik DR Hasil Pengujian Protokol SMTP | 41 |
| Gambar 4.6 Grafik FPR Hasil Pengujian Protokol SMTP | 42 |
| Gambar 4.7 Grafik F ₂ -score Hasil Pengujian Protokol HTTP | 44 |
| Gambar 4.8 Grafik DR Hasil Pengujian Protokol HTTP | 45 |
| Gambar 4.9 Grafik FPR Hasil Pengujian Protokol HTTP | 46 |

DAFTAR TABEL

| | |
|---|----|
| Tabel 2.1 Format Datagram IP | 17 |
| Tabel 3.1 Spesifikasi PC | 20 |
| Tabel 3.2 Daftar Perangkat Lunak | 20 |
| Tabel 3.3 Daftar Pustaka Python..... | 21 |
| Tabel 3.4 Jenis-jenis Serangan | 21 |
| Tabel 3.5 Jenis-jenis Protokol | 22 |
| Tabel 3.6 Tabel Penjelasan Confusion Matrix | 28 |
| Tabel 3.7 Tabel Evaluasi Performa | 35 |
| Tabel 4.1 Sampel Data Confusion Matrix Protokol FTP | 36 |
| Tabel 4.2 Detail Nilai F_2 -score Protokol FTP | 38 |
| Tabel 4.3 Sampel Data Confusion Matrix Protokol SMTP | 40 |
| Tabel 4.4 Detail Nilai F_2 -score Protokol SMTP | 41 |
| Tabel 4.5 Sampel Data Confusion Matrix Protokol HTTP | 43 |
| Tabel 4.6 Detail Nilai F_2 -score Protokol HTTP | 44 |
| Tabel 4.7 Rata-rata Waktu Eksekusi Program | 46 |
| Tabel 4.8 Rata-rata Nilai F_2 -score | 46 |
| Tabel 4.9 Rata-rata Nilai DR..... | 47 |
| Tabel 4.10 Rata-rata Nilai FPR | 47 |

DAFTAR KODE PROGRAM

| | |
|--|----|
| Kode Program 3.1 Implementasi Pra Proses Data | 30 |
| Kode Program 3.2 Implementasi Perhitungan Byte Frequency..... | 31 |
| Kode Program 3.3 Implementasi Proses Latih | 32 |
| Kode Program 3.4 Implementasi Proses Testing..... | 33 |
| Kode Program 3.5 Implementasi Evaluasi Hasil Prediksi | 34 |
| Kode Program 3.6 Implementasi Confusion Matrix..... | 34 |

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Semakin tinggi tingkat penggunaan teknologi informasi, semakin tinggi pula risiko dan ancaman keamanan jaringan. Berdasarkan data Badan Siber dan Sandi Negara (BSSN) periode Januari-April 2020, telah terjadi 88 juta kasus serangan siber di Indonesia. Total serangan siber yang terjadi di Indonesia meningkat dari tahun ke tahun. Jenis serangan yang berbeda-beda mempersulit penanganan kasus serangan siber di Indonesia. Untuk mencegah agar serangan siber tidak terjadi, pengguna jaringan harus bisa memastikan koneksi-koneksi yang terhubung ke jaringan merupakan koneksi yang telah teridentifikasi aman [1].

Salah satu cara untuk mendeteksi serangan siber pada perangkat yang dilindungi adalah dengan menggunakan sistem deteksi intrusi berbasis jaringan atau *Network Intrusion Detection System* (NIDS). NIDS berfungsi untuk memonitor, mengidentifikasi, dan menganalisa lalu lintas jaringan komputer dari potensi serangan pada keamanan jaringan. Agar NIDS dapat mendeteksi serangan dengan baik, NIDS harus dilatih terlebih dahulu mengenali koneksi-koneksi yang normal dan koneksi-koneksi yang bersifat anomali.

Salah satu cara untuk melatih NIDS adalah dengan menggunakan algoritma *machine learning*. Algoritma *machine learning* dapat melakukan prediksi terhadap koneksi-koneksi yang masuk terhadap perangkat. Agar *machine learning* dapat mengidentifikasi data dengan baik, *machine learning* dilatih menggunakan data latih yang disiapkan terlebih dahulu. Dari data latih tersebut, *machine learning* dapat menentukan pola data yang selanjutnya menjadi model dalam melakukan prediksi data. Model yang terbentuk menjadi patokan saat melakukan prediksi data test. Kinerja algoritma *machine learning* diukur dengan melihat patokan nilai-nilai dari kesuksesan algoritma dalam memprediksi data dengan benar.

Pada umumnya, data yang digunakan untuk melatih NIDS adalah data latih yang berisi koneksi-koneksi bersih dan jelas. Setiap data latih yang dipakai, terlebih dahulu diklasifikasikan sebagai serangan atau tidak dengan memberikan pelabelan pada setiap data. Selanjutnya dengan menggunakan *machine learning*, sistem akan membuat aturan-aturan umum (*generalized rule*) yang dipakai untuk mendeteksi serangan. NIDS ini dapat dikategorikan sebagai *Signature-based Intrusion Detection System* (SIDS). Salah satu kelemahan terbesar pada algoritma ini adalah algoritma hanya dapat dilatih menggunakan data yang telah memiliki label atau data yang telah terlebih dahulu diklasifikasikan.

Namun pada prakteknya di dunia nyata, data yang bersih dan sudah diklasifikasikan sangat sulit untuk didapatkan. Data yang ada biasanya berupa data campuran yang terdapat data kotor atau data yang masih berpotensi mengandung unsur serangan di dalamnya. Umumnya untuk mendapatkan data latih yang bersih dan berlabel, perlu dilakukan proses klasifikasi secara manual data-data jaringan yang ukurannya sangat besar. Ukuran data yang sangat besar tersebut, membuat proses pengklasifikasian data secara manual sangat sulit dan melelahkan untuk dilakukan.

Salah satu alternatif adalah dengan menggunakan *Anomaly-based Intrusion Detection System* (AIDS). Berbeda dengan SIDS, AIDS mendeteksi data dengan memperhitungkan nilai-nilai pada data yang memiliki pola tidak normal. Beberapa algoritma yang dapat digunakan untuk melatih AIDS adalah *one-class SVM*, *local outlier factor*, dan *isolation forest*.

Algoritma-algoritma tersebut tidak memerlukan data yang berlabel dalam melakukan prediksi, sehingga memungkinkan untuk menggunakan data kotor untuk melatih algoritma tersebut.

Data yang digunakan adalah data yang berformat PCAP. PCAP berisi data paket-paket yang masuk pada protokol TCP/IP dan UDP. Data dapat berupa IP *address*, protokol, *payload*, sumber dan destinasi dari paket yang terdeteksi. *Payload* adalah konten dari paket yang masuk. Seluruh data *payload* pada file PCAP akan diproses lagi menjadi data *byte frequency*. *Byte frequency* merupakan data frekuensi setiap *byte* pada masing-masing *payload* paket. Dari analisa *byte frequency* tersebut, dapat dibuat sebuah model *machine learning* untuk mencari anomali yang terjadi pada data test yang digunakan.

Berdasarkan uraian diatas, laporan tugas akhir berjudul “Penggunaan Data Kotor untuk Melatih Sistem Deteksi Serangan berbasis Jaringan” ini memiliki tujuan utama untuk membangun sistem deteksi serangan cerdas yang bisa mendeteksi serangan dengan menggunakan data kotor atau data yang belum diklasifikasikan berdasarkan ada tidaknya potensi serangan.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara melatih sistem deteksi serangan untuk mendeteksi serangan yang masuk pada jaringan, meski data latih yang digunakan mengandung data kotor?
2. Seberapa jauh penurunan performa pendeteksian serangan jika data latih yang digunakan mengandung data serangan?

1.3 Batasan Masalah

Batasan-batasan yang ada pada tugas akhir ini, yaitu:

1. Menggunakan bahasa pemrograman Python disertai dengan standar pustaka Python.
2. *Machine learning* menggunakan pustaka scikit-learn untuk algoritma *One-Class SVM*, *Local Outlier Factor*, dan *Isolation Forest*.
3. *Unsupervised learning* untuk menganalisis dataset yang tidak berlabel.
4. Data yang digunakan merupakan data dari protokol port 21 (FTP), port 25 (SMTP), dan port 80 (HTTP).
5. Menggunakan pustaka Pcap.
6. Data latih dan data testing berupa data berformat PCAP.
7. *Machine learning* menggunakan *byte frequency payload* pada masing-masing paket data untuk pembuatan model.
8. Menggunakan dataset UNSW-NB15 (Moustafa & Slay, 2015) untuk data latih dan data testing

1.4 Tujuan

Tujuan penulisan tugas akhir ini adalah sebagai berikut:

1. Implementasi sistem deteksi serangan cerdas yang bisa mendeteksi serangan meski dilatih dengan data yang mengandung data kotor.

2. Mencari metode *machine learning* yang optimal berdasarkan hasil akurasi, F_2 -score, dan waktu yang dibutuhkan saat eksekusi program.
3. Melakukan simulasi deteksi serangan pada jaringan menggunakan bahasa Python.

1.5 Manfaat

1. Bagi Akademik
 - Menambah pustaka baru sebagai pembandingan terhadap penelitian-penelitian yang sudah ada sebelumnya.
 - Sebagai bahan referensi bagi mahasiswa lainnya dalam penyusunan tugas akhir dibidang keamanan komputer jaringan
2. Bagi Penulis
 - Meningkatkan wawasan dan pengetahuan dalam memahami cara kerja sistem deteksi serangan berbasis jaringan.
 - Dapat menerapkan *machine learning* dalam pembuatan program NIDS untuk dapat mendeteksi serangan dalam paket data jaringan.
 - Memahami struktur data pada paket berformat PCAP
 - Menambah pengalaman menggunakan bahasa Python untuk pembuatan program berbasis jaringan

BAB 2 Dasar Teori

2.1 NIDS

Network Intrusion Detection System (NIDS) atau sistem deteksi serangan berdasarkan jaringan adalah salah satu jenis *Intrusion Detection System* (IDS) yang bekerja untuk memonitor lalu lintas jaringan. NIDS ditempatkan pada titik strategis yang ada pada jaringan. Selanjutnya, NIDS akan melakukan analisis seluruh paket data yang melintas pada sub-jaringan NIDS ditempatkan. Paket data yang dianalisa NIDS akan dicocokkan dengan daftar pustaka serangan yang dimiliki NIDS. Bila paket teridentifikasi sebagai serangan, NIDS akan memberikan peringatan ke administrator jaringan [2].

Salah satu jenis metode deteksi dalam NIDS adalah *anomaly-based* NIDS. Sistem deteksi intrusi berbasis anomali bertujuan untuk mendeteksi serangan yang sebelumnya tidak diketahui jenis serangannya dikarenakan adanya jenis *malware* yang berbeda-beda disetiap waktunya. Pendekatan dasar *anomaly-based* NIDS adalah menggunakan *machine learning* untuk membuat model aturan deteksi sistem, yang selanjutnya digunakan untuk membandingkan perilaku paket data yang masuk dengan model yang terbentuk.

Keunggulan dari metode *machine learning* yaitu model yang terbentuk bisa dilatih sesuai kriteria sistem dan perangkat keras yang diinginkan. Karena itu bila dibandingkan dengan *traditional signature-based* IDS, metode *machine learning* memiliki properti umum yang lebih baik. Salah satu kekurangan pada metode *machine learning* adalah paket yang terdeteksi mudah sekali dinyatakan menjadi *false positive* [3].

2.2 Python

Python adalah bahasa pemrograman tingkat tinggi yang dirancang agar mudah dibaca, dan memiliki beberapa kesamaan dengan bahasa inggris. Salah satu keunggulan Python adalah luasnya daftar pustaka yang dimiliki oleh Python. Hal ini memudahkan programmer dalam menjalankan fungsi-fungsi kompleks pada program yang dibuat. Selain itu, fitur-fitur dan modul yang ada pada Python juga tidak dikenakan biaya dalam penggunaannya. Python dipilih sebagai basis bahasa pemrograman dalam tugas akhir ini karena Python memiliki fungsi-fungsi dan daftar pustaka yang mendukung dalam kegiatan analisis data jaringan. Simulasi *sniffing* dan *capture* paket menjadi mudah dilakukan dengan menggunakan fungsi-fungsi yang ada [4].

2.3 Machine Learning

Machine learning adalah algoritma komputer yang secara otomatis dapat mengembangkan dirinya berdasar data latih yang diberikan. Untuk menyelesaikan tugas yang diberikan, algoritma *machine learning* membangun model berdasarkan data sampel, yang dikenal sebagai "data latih", untuk membuat prediksi atau keputusan tanpa diprogram secara eksplisit untuk melakukannya. Beberapa jenis machine learning berhubungan erat dengan komputer statistik. Dari data hasil statistik, machine learning dapat memprediksi keluaran yang sesuai dengan tugas yang diberikan [5].

Machine learning dibagi menjadi 2 tipe berdasarkan cara algoritma tersebut bekerja dalam membentuk sebuah model. Kedua tipe tersebut adalah *supervised learning* dan *unsupervised learning*. *Supervised learning* adalah sebuah pendekatan *machine learning* dengan cara melatih algoritma *machine learning* menggunakan data yang sudah berlabel untuk dapat memprediksi

hasil keluaran data testing. Model yang terbentuk dapat mendeteksi pola dan hubungan antara data latih dan data testing.

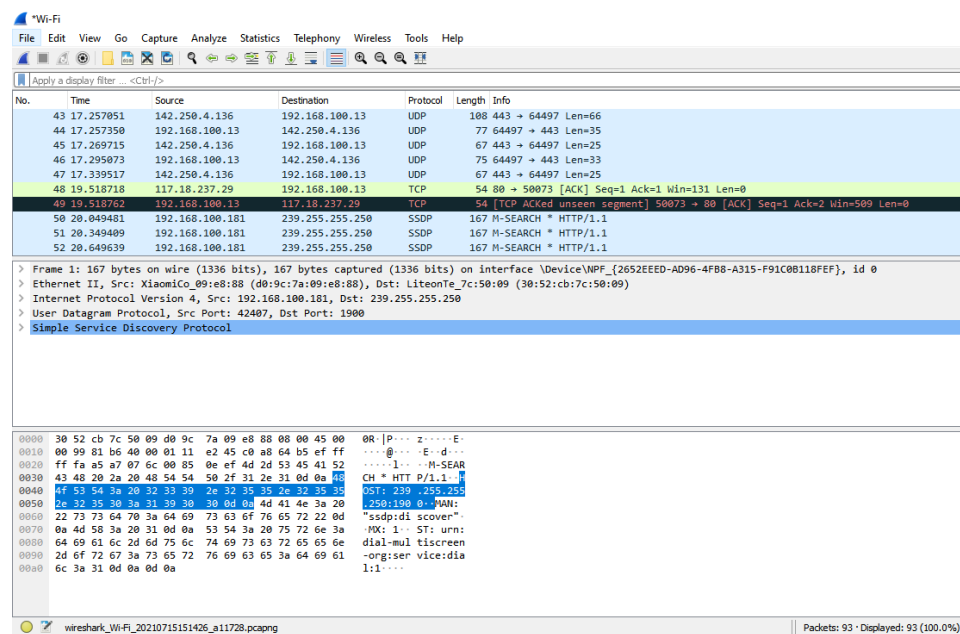
Unsupervised learning adalah teknik *machine learning* di mana pengguna tidak perlu melakukan pengawasan secara manual saat proses pembuatan model. Sebaliknya, algoritma *unsupervised learning* memungkinkan model bekerja sendiri menemukan pola dari informasi yang diberikan. Umumnya, algoritma ini digunakan untuk membuat model bagi data yang tidak memiliki label atau data yang masih terdapat data kotor di dalam data yang diujikan. Algoritma *unsupervised learning* memungkinkan pengguna untuk melakukan tugas pemrosesan data yang lebih kompleks dibanding algoritma *machine learning* yang lain. Tetapi, hasil keluaran *unsupervised learning* terkadang susah diprediksi dan tidak sesuai dari keluaran harapan [6].

2.4 Sckit-learn

Scikit-learn adalah pustaka *machine learning* yang bersifat *open-source*. Scikit-learn membantu pengguna dalam pembuatan algoritma *supervised learning* dan *unsupervised learning*. Scikit-learn juga menyediakan berbagai alat untuk pencocokan model, pra-pemrosesan data, pemilihan dan evaluasi model, dan banyak utilitas lainnya. Di dalam pustaka scikit-learn sudah ada *built-in* algoritma *machine learning* yang membantu dalam melakukan pemrosesan data [7].

2.5 PCAP

PCAP (singkatan dari Packet Capture) adalah nama API yang biasa digunakan untuk merekam metrik paket. File PCAP sangat membantu karena dapat digunakan untuk merekam data lalu lintas *multilayer*, menangkap paket yang berasal dari lapisan *data link* sampai ke lapisan aplikasi. Data rekaman dari PCAP selanjutnya bisa digunakan untuk kegiatan analisa paket jaringan. Data latih dan data testing NIDS biasanya menggunakan file PCAP. Gambaran PCAP dapat dilihat pada Gambar 2.1.



Gambar 2.1 Packet capture dengan Aplikasi Wireshark

Alat analisis PCAP memungkinkan pengguna untuk secara konsisten merekam data lalu lintas di beberapa lapisan OSI, tanpa memperlambat aliran jaringan. Dengan menggunakan paket data, pengguna dapat mengekstrak informasi penting tentang kondisi kesehatan dan kinerja jaringan yang ada. Selain itu PCAP juga bisa digunakan untuk memecahkan masalah kinerja dengan melacak paket data yang tidak biasa kembali ke asalnya [9].

Dari data berformat PCAP tersebut, bisa didapatkan informasi *header* dan *payload* setiap koneksi yang ada di dalam PCAP. Informasi tersebut akan digunakan untuk melatih dan melakukan uji coba terhadap sistem deteksi intrusi yang akan dibuat.

2.6 Pcapy

Pcapy adalah modul ekstensi Python yang berhubungan dengan *packet capture* dari pustaka libpcap. Pcapy memungkinkan skrip Python untuk menangkap paket di jaringan. Pcapy sangat efektif bila digunakan bersama dengan modul kelas Python lainnya untuk membangun dan menangani paket. Salah satu hal yang membedakan Pcapy dengan modul lain adalah Pcapy mendukung penggunaan Python *threads* dalam menjalankan fungsinya. Pcapy juga menyediakan *object oriented* API untuk memudahkan penggunaan *object oriented* pada program [10].

Pcapy juga berjalan dengan baik terhadap pustaka Python yang lain. Dengan dipadukan dengan pustaka impacket, proses ekstraksi data pada PCAP menjadi lebih mudah. Dari data yang diperoleh, data akan direkonstruksi ulang dan dihitung nilai *byte frequency* pada masing-masing *payload* yang didapatkan.

2.7 Payload

Paket-paket data dalam protokol IP dikirimkan dalam bentuk datagram. *Network intrusion detection system* dibagi menjadi 2 tipe berdasarkan bagian datagram yang dianalisis yaitu NIDS berdasarkan *header* dan NIDS berdasarkan *payload*. *Payload* merupakan isi atau muatan dari sebuah paket jaringan yang akan ditransmisikan, dimana *payload length* menentukan panjang dari isi data yang dienkapsulasi di dalam paket dalam sebuah *byte* [8]. Bentuk struktur payload dapat dilihat pada Tabel 2.1.

Tabel 2.1 Format Datagram IP

| | | | |
|---|---------------------------|-------------------|-------------------------------------|
| <i>Header</i> Protokol Antarmuka Jaringan | <i>Header</i> Protokol IP | <i>Payload</i> IP | Trailer Protokol Antarmuka jaringan |
|---|---------------------------|-------------------|-------------------------------------|

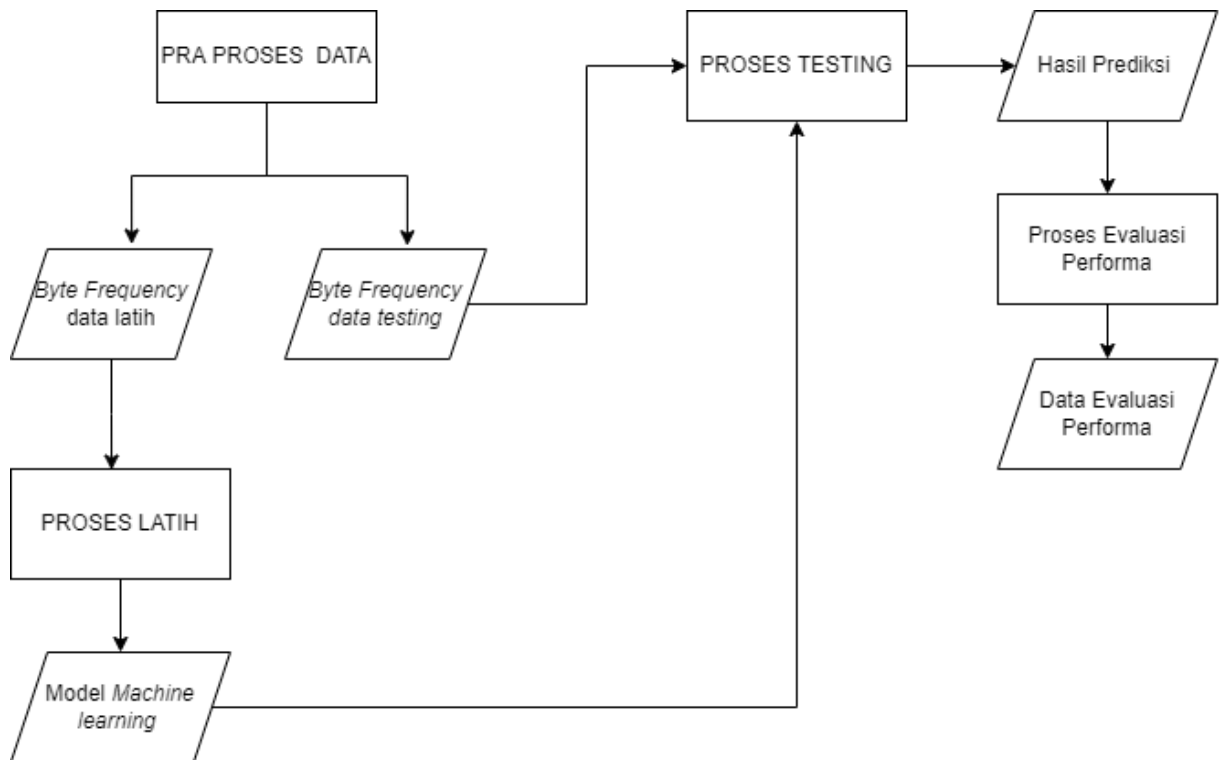
NIDS berdasarkan *payload* melakukan analisa muatan paket pada datagram IP. Apabila ditemukan anomali pada muatan paket yang diperiksa, NIDS akan mengirimkan peringatan ke administrator. Proses analisis *payload* ada beberapa cara, salah satunya dengan menggunakan *machine learning*. Algoritma *machine learning* dilatih menggunakan data latih yang berisi paket-paket jaringan. Karena itu, hasil keluaran dari algoritma *machine learning* sangat berpengaruh terhadap data latih yang dipakai.

Data *payload* yang didapatkan dari ekstraksi file berformat PCAP akan dikonversikan terlebih dahulu menjadi *byte frequency* sebelum digunakan untuk proses latih dan proses testing. *Byte frequency* adalah sebuah data yang menampilkan frekuensi munculnya setiap tipe *byte* yang berbeda-beda pada satu payload. Dari nilai-nilai *byte frequency* dapat ditentukan pola data normal dan data anomali.

BAB 3 METODOLOGI

3.1 Analisa Metode Penelitian

Pada laporan tugas akhir ini akan dibuat sebuah sistem deteksi intrusi jaringan yang dilatih menggunakan data yang mengandung data kotor pada algoritma *machine learning*. Sistem menggunakan algoritma *unsupervised learning* dengan membandingkan nilai-nilai anomali pada setiap data latih yang diberikan. Sistem diharapkan mampu memprediksi dengan baik apakah data yang sedang diuji termasuk data serangan atau data normal. Pada Gambar 3.1 di bawah ini dapat dilihat alur diagram penelitian yang menggambarkan sebagian besar proses-proses yang dilakukan saat melakukan penelitian.



Gambar 3.1 Diagram Alur Penelitian

Dalam penelitian tugas akhir ini, ada 4 proses utama yang akan dilakukan. Hal yang paling awal dilakukan adalah melakukan pra proses data. Pra proses data bertujuan untuk melakukan ekstraksi *payload* dari data berformat PCAP hingga menjadi data berupa *byte frequency*. Data-data yang digunakan untuk data latih dan data testing dibagi menjadi 3 jenis berdasar jenis protokol yang dimiliki pada masing-masing data, yaitu FTP, SMTP, dan HTTP. Data-data *payload* yang sudah diproses menjadi *byte frequency* selanjutnya akan digunakan untuk menjalankan proses latih dan proses testing.

Pada proses latih, data *byte frequency* dari data latih akan digunakan untuk membentuk model pada *machine learning*. Model dibentuk berdasarkan analisa nilai-nilai parameter yang menentukan tingkat anomali data *byte frequency*. Proses pembentukan model berdasarkan analisa data latih disebut sebagai proses *fitting*. Model yang terbentuk dari proses *fitting* akan digunakan untuk memprediksi data pada saat dilakukan proses testing.

Ada 2 komponen utama yang diperlukan untuk menjalankan proses testing, yaitu data testing yang sudah diproses menjadi data *byte frequency* dan model yang didapatkan dari proses latih. Data testing yang telah diproses menjadi data *byte frequency* merupakan data mentah yang belum memiliki label untuk membedakan apakah data tersebut merupakan data serangan atau data normal. Sistem akan melakukan prediksi untuk memberikan pelabelan pada masing-masing data latih. Apabila sistem menganggap data tersebut adalah data yang teridentifikasi serangan, sistem akan memberikan label -1 (*outlier*). Data yang teridentifikasi sebagai data normal akan diberikan label 1 (*inlier*).

Hasil proses testing selanjutnya akan dievaluasi secara keseluruhan untuk melihat performa algoritma dalam melakukan proses prediksi. Evaluasi dilakukan dengan melihat akurasi dan F_2 -score pada hasil percobaan masing-masing algoritma. Selanjutnya, dari hasil tersebut dapat disimpulkan bagaimana peningkatan dan penurunan performa algoritma pada setiap penggunaan protokol dan data yang berbeda-beda saat pengujian.

3.2 Perlengkapan Uji Coba Penelitian

Pada subbab ini akan dijelaskan perangkat-perangkat serta data-data yang dibutuhkan untuk melakukan percobaan. Perangkat berupa perangkat keras dan perangkat lunak. Data dibagi menjadi data latih dan data testing.

3.2.1 Perangkat Keras

Tugas akhir ini menggunakan perangkat keras berupa *personal computer* (PC) untuk menjalankan percobaan serta pembuatan dokumen laporan tugas akhir. Spesifikasi PC yang digunakan untuk mengerjakan tugas akhir ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi PC

| No | Komponen | Spesifikasi |
|----|----------|--|
| 1 | Prosesor | Intel(R) Core(TM) i5-4670 CPU @ 3.40Ghz (4CPUs), ~3.4Ghz |
| 2 | RAM | 12 GB |
| 3 | Display | NVIDIA GeForce GT 1030 |

3.2.2 Perangkat Lunak

Perangkat lunak yang digunakan untuk menjalankan aplikasi, pengerjaan laporan, serta pustaka-pustaka Python yang diperlukan untuk menjalankan program dapat dilihat pada Tabel 3.2.

Tabel 3.2 Daftar Perangkat Lunak

| No | Nama Software | Kegunaan |
|----|--------------------|---|
| 1 | Windows 10 | <i>Operating System</i> (OS) yang digunakan untuk menjalankan program. |
| 2 | Visual studio code | Compiler yang digunakan untuk menjalankan program. |
| 3 | Microsoft Excel | Software yang digunakan untuk menyimpan dan menganalisa hasil penelitian. |
| 4 | Microsoft Document | Software yang digunakan untuk pembuatan proposal dan laporan tugas akhir. |

| | | |
|---|------------|--|
| 5 | Python 3.9 | Bahasa pemrograman yang digunakan pada implementasi sistem deteksi intrusi menggunakan <i>machine learning</i> . |
|---|------------|--|

Selain pustaka standar yang ada pada Python, ada juga pustaka-pustaka lain yang digunakan untuk membantu pembuatan program penelitian. Pustaka-pustaka tersebut dapat dilihat pada Tabel 3.3.

Tabel 3.3 Daftar Pustaka Python

| No | Nama Pustaka | Kegunaan |
|----|--------------|---|
| 1 | Scikit-learn | Pustaka yang memiliki fungsi dalam menjalankan algoritma <i>machine learning</i> yang bersifat <i>open source</i> . Scikit-learn mendukung dalam menjalankan algoritma <i>supervised learning</i> dan <i>unsupervised learning</i> . Scikit-learn memiliki beberapa fungsi seperti <i>model fitting</i> , pra proses data, pemilihan model, dan evaluasi model. |
| 2 | Pcap | Pustaka yang berfungsi untuk membaca file PCAP dan melakukan <i>capture</i> paket-paket di jaringan. Pcap digunakan bersama pustaka-pustaka lainnya untuk membuat dan menangani paket data. |
| 3 | Impacket | Pustaka yang digunakan untuk mendapatkan <i>payload</i> pada paket serta melakukan klasifikasi berdasarkan protokol pada masing-masing paket. |
| 4 | Numpy | Pustaka Python yang berguna dalam perhitungan matematika pada matrik data. Data <i>byte frequency</i> dikumpulkan pada numpy array yang selanjutnya akan digunakan dalam proses latih. |

3.2.3 Data Latih dan Data Testing

Data latih dan data testing yang digunakan untuk penelitian ini adalah data file berformat PCAP dari UNSW-NB15 [11]. Data PCAP ini berisikan paket campuran antara aktivitas jaringan normal dan paket yang berisikan perilaku serangan. Total data sebesar 100 GB file PCAP yang berupa tcpdump. Di dalam dataset mengandung 4 jenis serangan yaitu, *Backdoor*, *Exploits*, *Shellcode*, dan *Worm*. Penjelasan lebih lanjut 4 jenis serangan tersebut dapat dilihat pada Tabel 3.4.

Tabel 3.4 Jenis-jenis Serangan

| NO | Nama Serangan | _Penjelasan |
|----|-----------------|--|
| 1 | <i>Backdoor</i> | Jenis serangan yang bertujuan untuk mendapatkan hak akses terhadap sistem dengan melewati mekanisme keamanan sistem. Pelaku serangan akan mendapatkan <i>remote</i> akses terhadap sistem yang selanjutnya dapat digunakan untuk mendapatkan sumber daya berupa data pada database atau <i>web server</i> pada target. |
| 2 | <i>Exploits</i> | <i>Exploit</i> adalah sebuah serangan pada sistem komputer dengan memanfaatkan kelemahan |

| | | |
|---|------------------|--|
| | | dari sistem. <i>Exploit</i> bertujuan untuk menginisiasi serangan seperti <i>denial-of-service</i> (DoS) atau penginstalan <i>malware</i> pada komputer target. Dapat disimpulkan exploit bukanlah <i>malware</i> , tapi sebuah cara yang digunakan untuk mengirimkan <i>malware</i> ke komputer target. |
| 3 | <i>Shellcode</i> | <i>Shellcode</i> adalah kode yang ditanam pada <i>payload</i> untuk mengeksploitasi komputer target. <i>Shellcode</i> dimasukkan ke dalam kode <i>exploit</i> dengan tujuan melewati keamanan sistem dan menjalankan fungsi pada <i>shellcode</i> . |
| 4 | <i>Worm</i> | Sebuah <i>malware</i> yang memiliki kemampuan menggandakan dirinya sendiri dalam sistem komputer. Sebuah worm dapat menggandakan diri dengan memanfaatkan jaringan internet tanpa perlu campur tangan dari pengguna. |

Dari data mentah UNSW-NB15, data PCAP diklasifikasikan ulang menjadi 3 jenis data berdasarkan protokolnya. Penjelasan protokol-protokol yang akan diuji dapat dilihat pada Tabel 3.5.

Tabel 3.5 Jenis-jenis Protokol

| NO | Protokol | Port | _Penjelasan |
|----|----------|------|--|
| 1 | FTP | 21 | <i>File Tranfer Protocol</i> (FTP) adalah protokol yang digunakan untuk melakukan pertukaran data dua arah antar komputer dalam jaringan. |
| 2 | SMTP | 25 | <i>Simple Mail Tranfer Protocol</i> (SMTP) adalah protokol standar yang digunakan untuk pengiriman email. Pengiriman email dilakukan baik dari lokal email ke email server maupun sebaliknya. |
| 3 | HTTP | 80 | <i>Hypertext Tranfer Protocol</i> (HTTP) merupakan protokol yang berperan sebagai penghubung antara client dengan <i>web server</i> . <i>Web server</i> merupakan server yang dibuat untuk memenuhi kebutuhan website dan situs pada internet. |

Total ada 26 data latih berformat PCAP yang digunakan pada proses latih. Data tersebut terbagi menjadi 9 paket FTP, 8 paket SMTP, dan 9 paket HTTP. Setiap data latih yang digunakan mengandung data kotor yang memiliki kadar berbeda-beda. Data kotor tersebut merupakan data yang bersifat berbahaya dan memiliki unsur serangan. Secara berurutan, protokol FTP memiliki data dengan kadar data kotor sebesar 0%, 0,05%, 0,1%, 0,15%, 0,2%, 0,25%, 0,3%, 0,35%, 0,39%. Pada protokol SMTP terdapat data dengan kadar data kotor sebesar 0%, 0,1%, 0,2%, 0,3%, 0,4%, 0,5%, 0,6%, 0,7%. Sedangkan pada protokol HTTP terdapat data kotor dengan kadar data kotor sebesar 0%, 0,1%, 0,2%, 0,3%, 0,4%, 0,5%, 0,6%, 0,7%, 0,8%.

3.3 Algoritma *Unsupervised Learning*

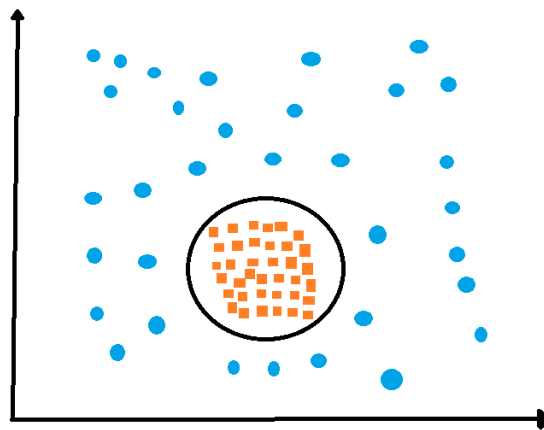
Dalam menjalankan tugasnya, aplikasi NIDS harus dapat menentukan apakah data yang terdeteksi merupakan data *malicious* ataukah data bersih. Salah satu cara melatih NIDS untuk dapat mengenali data dengan baik adalah dengan menggunakan algoritma *machine learning*. Algoritma *machine learning* yang akan digunakan pada penelitian kali ini adalah algoritma *unsupervised learning*. *Unsupervised learning* digunakan untuk mendeteksi data-data yang bersifat anomali pada data yang digunakan pada proses testing. Deteksi anomali dilakukan dengan melihat tingkat densitas persebaran data. Data pada area dengan tingkat densitas rendah diasumsikan sebagai data *outlier* atau data anomali.

Salah satu keunggulan *unsupervised learning* adalah data latih yang digunakan dapat berupa data latih yang masih mengandung data kotor. Pada penelitian ini akan disediakan data latih dengan kadar data kotor yang berbeda-beda. Selanjutnya, akan dilakukan analisa pengaruh data kotor terhadap performa algoritma *unsupervised learning* yang digunakan.

Pada penelitian ini ada 3 algoritma *unsupervised learning* yang akan digunakan, yaitu *one-class SVM*, *local outlier factor*, dan *isolation forest*. Performa masing-masing algoritma dinilai dengan melihat nilai F_2 -score pada setiap percobaan. Percobaan dilakukan beberapa kali untuk melihat ada tidaknya perubahan hasil apabila data latih yang digunakan memiliki kadar data kotor yang berbeda beda.

3.3.1 One-class SVM

One-class SVM (OCSVM) adalah salah satu model *unsupervised machine learning* untuk deteksi *outlier* atau anomali pada suatu dataset. Tidak seperti algoritma *supervised SVM* lainnya, OCSVM tidak memerlukan label pada data yang digunakan untuk proses latih. OCSVM menentukan sendiri batasan-batasan parameter yang membedakan antara data anomali dan data normal. Penentuan batasan tersebut dilakukan pada proses latih. Gambaran cara kerja OCSVM dapat dilihat pada Gambar 3.2.



Gambar 3.2 Gambaran *One-class SVM*

Dari proses latih, algoritma OCSVM mendapatkan sebuah pembatas berbentuk bulat yang membatasi antara data anomali dan data normal. Pusat dan jari-jari dari lingkaran pembatas ditentukan dengan menggunakan kombinasi persamaan *support vector*. Data dianggap normal apabila data yang diuji memiliki jarak lebih kecil atau sama dengan jari-jari lingkaran pembatas. Perhitungan jarak antara satu titik dengan titik yang lain dihitung menggunakan operasi *dot-*

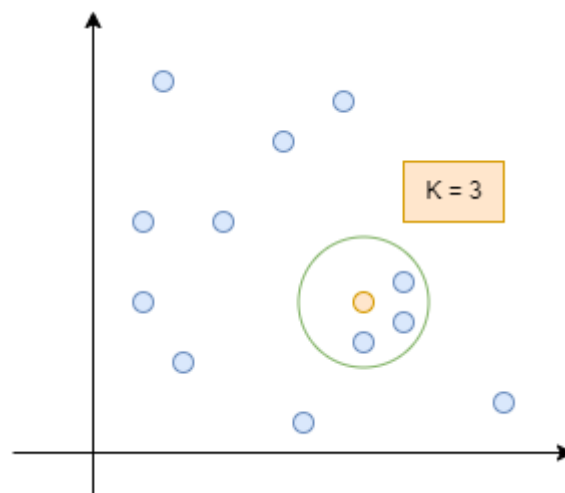
product. Secara *default*, OCSVM akan menggunakan *rbf kernel* dalam menentukan jarak antar titik. Operasi *dot-product* dapat dilihat pada Persamaan 3.1.

$$\exp(-\gamma \|x - x'\|^2) \quad (3.1)$$

Dimana γ adalah parameter gamma. Parameter gamma digunakan untuk menentukan seberapa besar pengaruh yang dimiliki sebuah data latih. Semakin besar gamma, semakin besar pula pengaruh dari data latih yang digunakan. Karena itu, penentuan nilai gamma yang digunakan dapat berpengaruh pula pada hasil prediksi data testing.

3.3.2 Local Outlier Factor

Local outlier factor (LOF) adalah salah satu *unsupervised machine learning* untuk deteksi *outlier* atau data anomali dengan menghitung deviasi densitas lokal dari titik data tertentu terhadap tetangganya. Data-data yang memiliki nilai densitas lebih rendah dari tetangganya dianggap sebagai *outlier*. Gambaran algoritma LOF dapat dilihat pada Gambar 3.3.



Gambar 3.3 Gambaran *Local Outlier Factor*

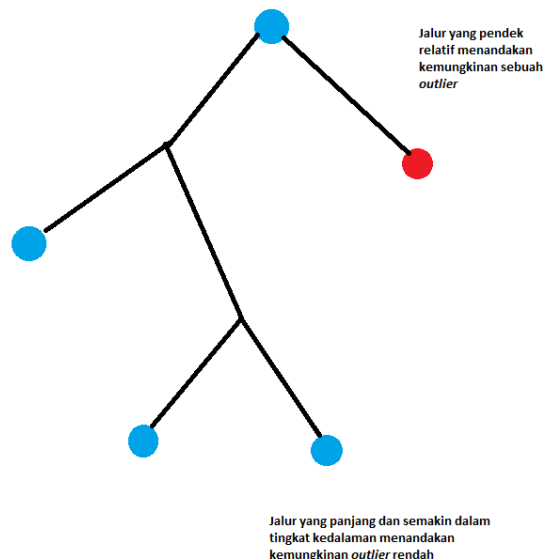
Untuk mencari jarak antar observasi, perlu ditentukan hyperparameter k . Nilai k di sini menentukan berapa banyak tetangga yang dilihat, jarak dihitung menurut nilai k tersebut. Misalnya, jika $k=3$, dicari 3 tetangga terdekat dari titik yang ditentukan. Jika nilai k yang ditentukan sangat kecil, algoritma menjadi sensitif terhadap *noise*, dan jika diberikan nilai yang besar, anomali lokal akan susah untuk dikenali.

Nilai anomali pada setiap sampel data disebut sebagai LOF. Nilai ini yang menggambarkan seberapa besar nilai densitas objek tersebut terhadap tetangganya. Dengan membandingkan nilai masing-masing LOF, dapat diketahui sebuah data memiliki nilai densitas lebih kecil dari tetangganya.

Dalam model LOF, terdapat 3 pilihan algoritma yang dapat digunakan untuk menghitung pembagian cluster setiap data. Ketiga algoritma tersebut adalah BallTree, KDTree, dan *brute-force*. Pengaturan *default* LOF secara otomatis menentukan algoritma yang cocok dengan melihat data latih yang digunakan saat proses *fitting*.

3.3.3 Isolation Forest

Isolation forest (ISOF) adalah model *unsupervised machine learning* yang dibentuk berdasarkan *decision tree*. Di dalam model ISOF, data latih yang digunakan secara acak akan diproses menjadi sebuah basis data berstruktur *tree* yang disebut *isolation tree*. Pembuatan struktur *tree* diawali dengan pengambilan secara acak salah satu fitur yang ada pada data latih. Dari fitur yang diambil secara acak tadi, dilakukan proses percabangan struktur *tree* dengan menentukan nilai ambang batas secara acak sesuai nilai minimum dan maksimum dari fitur yang dipilih. Jika nilai sebuah fitur dari data yang dipilih lebih kecil dari nilai ambang batas yang telah ditentukan, data tersebut akan diklasifikasikan ke bagian kiri *tree*. Sebaliknya jika nilai fitur data melebihi ambang batas, data tersebut akan diklasifikasikan ke bagian kanan *tree*. Proses ini akan dilanjutkan secara rekursif hingga semua data dapat terisolasi atau hingga mencapai kedalaman maksimum. Gambaran algoritma *isolation forest* dapat dilihat pada Gambar 3.4.



Gambar 3.4 Gambaran *Isolation Forest*

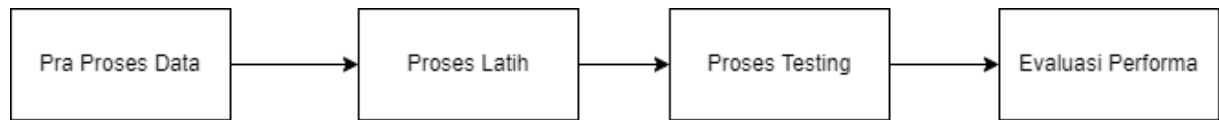
Semakin dalam tingkat kedalaman node pada *isolation tree*, semakin kecil kemungkinan data sebagai *outlier*. Sebaliknya, data yang terletak pada tingkat kedalaman relatif dangkal akan lebih berpotensi sebagai *outlier*. Maksimum kedalaman tiap diukur berdasarkan Persamaan 3.2.

$$\lceil \log_2 (n) \rceil \quad (3.2)$$

Nilai n adalah nilai banyaknya sampel yang digunakan dalam pembuatan model *isolation tree*.

3.4 Urutan Pelaksanaan Penelitian

Pada subbab ini akan dijelaskan urutan dari pelaksanaan penelitian. Diagram alur urutan pelaksanaan penelitian dapat dilihat pada Gambar 3.5.

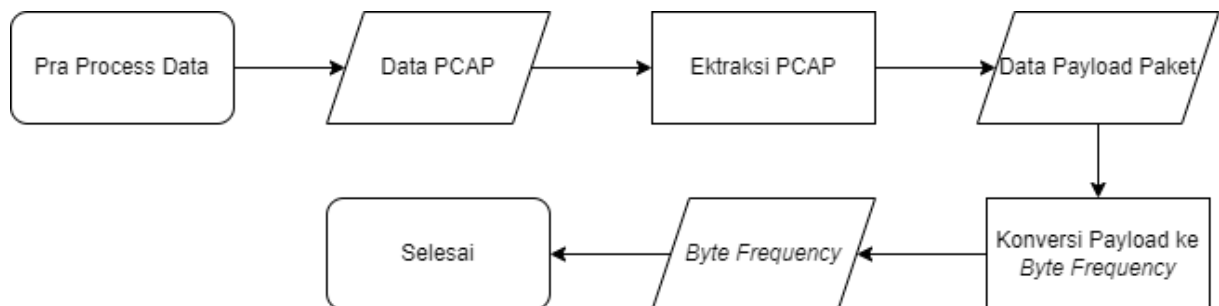


Gambar 3.5 Urutan Pelaksanaan Penelitian

Penelitian diawali dengan melakukan pra proses data. Pra proses data bertujuan untuk mendapatkan data latih dan data testing. Data latih akan digunakan untuk melatih algoritma pada proses latih. Selesai proses latih, program akan menghasilkan keluaran berupa model *machine learning* yang berfungsi menjadi model untuk memprediksi data testing. Pada proses testing, program akan memprediksi dengan memberikan pelabelan pada setiap data testing. Pelabelan bergantung pada parameter yang terdapat pada model yang digunakan. Hasil prediksi akan digunakan untuk mengukur performa dari masing-masing algoritma.

3.4.1 Pra Proses Data

Pada pra proses data, terdapat 2 kegiatan yang akan dilakukan untuk memproses file PCAP hingga menjadi *byte frequency*. Proses pertama yang dilakukan adalah ekstraksi PCAP. Pada proses ini, sistem akan membaca file PCAP untuk mendapatkan *header* dan *payload* dari data PCAP tersebut. Setelah didapatkan data *payload*, *payload* akan dikonversikan menjadi *byte frequency*. Masing-masing data latih dan data testing akan dikonversikan menjadi *byte frequency* sebelum digunakan pada proses latih dan proses testing. Urutan pelaksanaan penelitian dapat dilihat pada Gambar 3.6.



Gambar 3.6 Urutan Pelaksanaan Penelitian

3.4.1.1 Ekstraksi PCAP

Protokol pada jaringan sering kali mengangkut file yang sangat besar ketika melakukan pengiriman file. Protokol tranfer biasanya tidak dapat mengirim file data yang terlalu besar. Untuk menangani hal ini, paket-paket akan dipecah terlebih dahulu sebelum dikirim melalui protokol tranfer. Kegiatan pemecahan paket ini disebut *IP fragmentation*. Hal ini membuat data *payload* dan *header* menjadi terpecah-pecah saat membaca file PCAP.

Untuk menangani permasalahan dalam *IP fragmentation*, aplikasi program harus dapat mengidentifikasi paket-paket yang identik dan merekonstruksi paket tersebut menjadi satu kesatuan. Salah satu cara untuk menangani masalah ini adalah dengan menyimpan masing-masing bagian paket yang terpisah ke dalam antrian *buffer* lalu menggabungkannya menjadi satu. Dengan cara mengidentifikasi informasi *header* masing-masing paket, aplikasi dapat

mengidentifikasi pecahan-pecahan paket yang satu kesatuan untuk menggabungkannya menjadi 1 *payload*. Setelah *payload* sudah didapatkan secara utuh, langkah selanjutnya adalah mengkonversi menjadi *byte frequency* [13].

3.4.1.2 Konversi Payload ke Byte Frequency

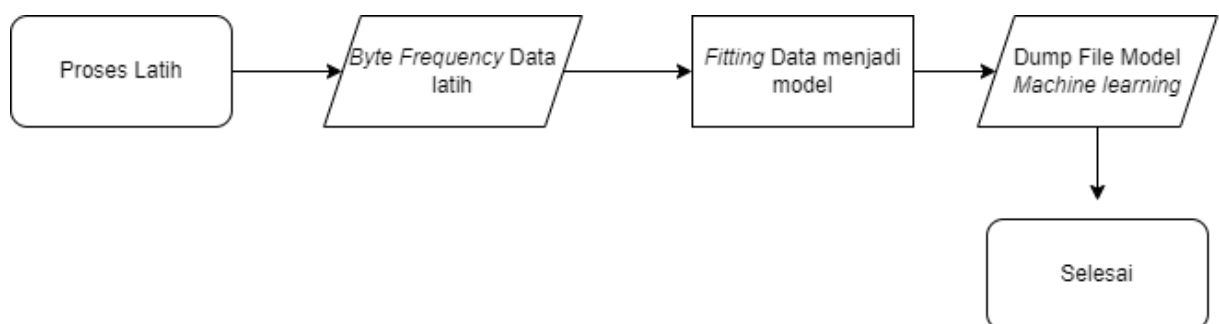
Setelah paket direkonstruksi menjadi *payload*, langkah selanjutnya adalah proses konversi dari *payload* menjadi *byte frequency*. *Byte* tersendiri terdiri dari 0 sampai 256 jenis yang berbeda. Setiap *byte* yang terdapat pada *payload* akan dihitung dan diakumulasikan menjadi satu numpy array berindeks 256. Setiap data *payload* yang telah didapatkan dari proses ekstraksi *payload* memiliki ukuran panjang yang berbeda-beda. Hal ini membuat nilai *byte frequency* pada *payload* yang panjang jauh lebih besar dibandingkan nilai *byte frequency* pada data yang lebih pendek. Untuk menyeimbangkan nilai *byte frequency* masing-masing *payload*, perlu dilakukan normalisasi data dengan membagi nilai frekuensi tiap *byte* terhadap panjang *payload*. Persamaan untuk menormalisasi data dapat dilihat pada Persamaan 3.3

$$x = \frac{\text{frekuensi tiap byte}}{\text{panjang(payload)}} \quad (3.3)$$

Data *byte frequency* hasil normalisasi selanjutnya digunakan untuk proses latih dan proses testing.

3.4.2 Proses Latih

Byte frequency data latih yang didapatkan dari hasil pra proses akan digunakan untuk pembuatan model *machine learning* pada proses latih. Pembuatan model *machine learning* dengan menggunakan data latih disebut sebagai proses *fitting*. Dalam proses *fitting*, data *byte frequency* dianalisa hingga menghasilkan sebuah nilai-nilai ambang batas yang digunakan dalam pembuatan model. Nilai ambang batas ini yang selanjutnya menjadi patokan saat melakukan prediksi data menggunakan model yang telah terbentuk. Diagram alur proses latih dapat dilihat pada Gambar 3.7.



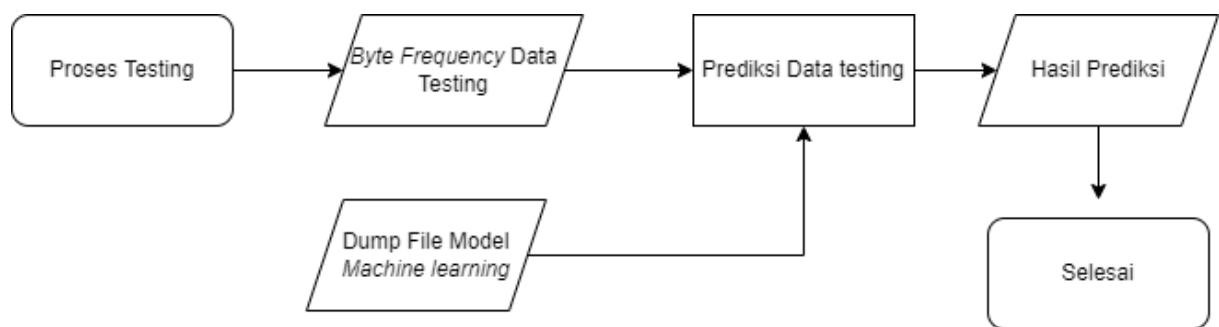
Gambar 3.7 Diagram Alur Proses Latih

Proses *fitting* dilakukan sebanyak 3 kali dengan menggunakan 3 algoritma yang berbeda. Sehingga pada akhir proses latih akan terbentuk 3 model *machine learning* yang berbeda antara satu dengan yang lain. Dari ketiga model yang berbeda ini, akan dilakukan uji performa dengan membandingkan hasil prediksi masing-masing model.

Dalam proses latih, model yang terbentuk disimpan di dalam sebuah *dump file*. Setiap akan dilakukan proses testing, sistem akan membaca *dump file* untuk mendapatkan model yang akan dipakai untuk proses testing. Data *dump file* akan diperbaharui setiap kali dilakukan proses latih baru dengan data yang berbeda.

3.4.3 Proses Testing

Ada 2 komponen utama yang diperlukan untuk menjalankan proses testing, yaitu data testing yang sudah diproses menjadi data *byte frequency* dan model yang didapatkan dari proses latih. Data testing yang telah diproses menjadi data *byte frequency* merupakan data mentah yang belum memiliki label untuk membedakan apakah data tersebut merupakan data serangan atau data normal. Sistem akan melakukan prediksi untuk memberikan pelabelan pada masing-masing data latih. Diagram alur proses testing dapat dilihat pada Gambar 3.8.



Gambar 3.8 Diagram Alur Proses Testing

Prediksi dilakukan dengan membandingkan fitur-fitur yang ada pada data latih dengan nilai-nilai ambang batas yang ada pada model. Apabila nilai-nilai dari fitur data testing tidak melebihi dari nilai ambang batas yang ditetapkan oleh model, sistem akan menganggap data tersebut merupakan data normal (*inlier*). Nilai-nilai fitur pada data yang melebihi ambang batas melebihi model akan dianggap sebagai data anomali (*outlier*).

3.4.4 Proses Evaluasi Performa

Evaluasi uji coba dilakukan dengan melihat hasil dari prediksi data testing. Hasil prediksi data testing disimpulkan menjadi *confusion matrix*. *Confusion matrix* digunakan untuk membandingkan hasil prediksi yang dilakukan oleh program dengan hasil prediksi yang benar. Dari *confusion matrix* didapatkan nilai *true positive*, *true negative*, *false positive*, dan *false negative* dari hasil prediksi. Penjelasan dari confusion matrix dijelaskan lebih lanjut pada Tabel 3.6.

Tabel 3.6 Tabel Penjelasan *Confusion Matrix*

| No | Nama | Penjelasan |
|----|----------------------------|--|
| 1 | <i>True Positive (TP)</i> | Hasil prediksi terdeteksi sebagai <i>outlier</i> dan hasil aktual terdeteksi sebagai <i>outlier</i> juga. |
| 2 | <i>True Negative (TN)</i> | Hasil prediksi terdeteksi sebagai <i>inlier</i> , dan hasil aktual terdeteksi sebagai <i>inlier</i> juga. |
| 3 | <i>False Positive (FP)</i> | Hasil prediksi terdeteksi sebagai <i>outlier</i> , sedangkan hasil aktual terdeteksi sebagai <i>inlier</i> . |

| | | |
|---|----------------------------|--|
| 4 | <i>False Negative</i> (FN) | Hasil prediksi terdeteksi sebagai <i>inlier</i> , sedangkan hasil aktual terdeteksi sebagai <i>outlier</i> . |
|---|----------------------------|--|

Hasil Confusion matrix ini yang selanjutnya digunakan untuk menentukan nilai dari F_2 -score masing-masing hasil data. Nilai F_2 -score selanjutnya digunakan untuk pembandingan performa masing-masing algoritma. Persamaan untuk mencari nilai F_2 -score dapat dilihat pada Persamaan 3.4.

$$F2 = \frac{5 \times TP}{5 \times TP + 4 \times FN + FP} \quad (3.4)$$

Di dalam sistem NIDS, hal yang paling utama untuk dicegah adalah sebuah kegagalan dalam mendeteksi serangan yang masuk. Serangan yang masuk tanpa terdeteksi akan sangat membahayakan keamanan data pada sistem. Maka dari itu, pada pembuatan sistem NIDS nilai FN jauh lebih krusial daripada nilai FP. Pada rumus diatas, terlihat bahwa nilai FN empat kali lebih besar dibandingkan nilai FP.

Nilai FN memiliki bobot lebih besar dibandingkan dengan nilai FP. Hal ini juga merupakan alasan mengapa digunakan F_2 -score sebagai nilai pembandingan performa algoritma. Dengan nilai bobot setiap unsur *confusion matrix* yang tidak seimbang, nilai akurasi pada sebuah algoritma kurang cocok digunakan untuk pembandingan performa algoritma untuk NIDS. Nilai akurasi jauh lebih cocok digunakan untuk algoritma *machine learning* dengan nilai setiap unsur pada *confusion matrix* memiliki bobot yang sama.

Selain menggunakan F_2 -score, evaluasi juga akan dilakukan dengan menggunakan nilai *detection rate* (DR) dan *false positive rate* (FPR). DR digunakan untuk melihat seberapa baik sebuah algoritma dapat mendeteksi data yang bersifat *malicious*. Sedangkan FPR, merupakan nilai pembandingan antara nilai FP dengan nilai TN. Nilai DR dan FPR dihitung dengan menggunakan persamaan yang ada pada Persamaan 3.5 dan Persamaan 3.6.

$$DR = \frac{TP}{TP + FN} \quad (3.5)$$

$$F2 = \frac{FP}{TN + FP} \quad (3.6)$$

Nilai DR didapatkan dengan membandingkan nilai TP dan FN dari hasil prediksi. Semakin besar nilai DR menandakan semakin banyak data *malicious* yang terdeteksi saat hasil prediksi. Nilai FPR menandakan sebesara sering sebuah program memberikan hasil FP saat proses prediksi. Semakin kecil nilai FPR menandakan semakin kecil pula total data FP yang dilakukan oleh algoritma.

3.5 Implementasi Sistem

Pada subbab ini akan dijelaskan bagaimana implementasi dari masing-masing metode yang digunakan dalam penelitian tugas akhir ini. Implementasi program dilakukan dengan menggunakan bahasa Python 3.9.

3.5.1 Implementasi Pra Proses Data

Pada pra proses data, tujuan utama dalam menjalankan proses ini adalah untuk mendapatkan nilai *byte frequency* pada *payload* data yang berformat PCAP. File PCAP terlebih dahulu dilakukan ekstraksi untuk mendapatkan informasi *header* dan *payload* yang tergantung di dalamnya. Setelah itu, langkah selanjutnya adalah melakukan konversi informasi *payload* menjadi data-data *byte frequency*. Implementasi pra proses dapat dilihat pada Kode Program 3.1.

```
1. def get_data_bytefreq():
2.
3.     filename = "noisy_port-80_percentage-0.003.pcap"
4.     protocol = "tcp"
5.     prt = StreamReaderThread(filename, protocol, "80")
6.     prt.delete_read_connections = True
7.     prt.start()
8.
9.     counter = 0
10.    final_list= []
11.
12.    while not prt.done or prt.has_ready_message():
13.        if not prt.has_ready_message():
14.            time.sleep(0.0001)
15.            continue
16.        buffered_packets = prt.pop_connection()
17.        #buffered_packets.get_byte_frequency("server")
18.        if buffered_packets is not None:
19.            #print(buffered_packets.get_byte_frequency("client"))
20.            counter += 1
21.            #list_temp=[]
22.            byte_frequency = buffered_packets.get_byte_frequency("server")
23.            #list_temp.extend(byte_frequency)
24.            final_list.append(byte_frequency)
25.            sys.stdout.write("\r{} flows.".format(counter))
26.            sys.stdout.flush()
27.
28.    #print(final_list)
29.    return final_list
```

Kode Program 3.1 Implementasi Pra Proses Data

Pada implementasi kode diatas file PCAP akan dibaca dan dilakukan ekstraksi *payload* dengan menggunakan modul StreamReaderThread dan modul TcpStream. Di dalam modul StreamReaderThread, informasi paket-paket *payload* akan disimpan dalam sebuah *buffer* yang selanjutnya akan dilakukan rekonstruksi kembali pecahan-pecahan paket *payload* untuk menjadi satu kesatuan. Di dalam modul TcpStream, sebuah paket akan diidentifikasi apakah sebuah paket termasuk bagian dari paket lain atau merupakan bagian paket tersendiri. Selanjutnya, program dapat menyatukan pecahan paket tersebut menjadi satu kesatuan *payload*.

Setelah sebuah *payload* selesai didapatkan, proses selanjutnya adalah dengan mengkonversi data *payload* tersebut menjadi sebuah *byte frequency*. Implementasi kode program untuk menghitung nilai *byte frequency* dapat dilihat pada Kode Program 3.2.

```

1. def __calculate_byte_frequency__(payload, length):
2.     byte_frequency = [0] * 256
3.
4.     if length > 0:
5.         for i in range(0, 256):
6.             byte_frequency[i] = float(payload.count(i)) / length
7.
8.     #print(byte_frequency)
9.     return byte_frequency

```

Kode Program 3.2 Implementasi Perhitungan *Byte Frequency*

Pada masing-masing *payload*, nilai frekuensi masing-masing *byte* akan dihitung dan dimasukkan menjadi sebuah array berdimensi 256. Data pada array tersebut selanjutnya akan dilakukan normalisasi. Mengingat data *payload* yang ada pada file PCAP memiliki ukuran yang berbeda-beda, data yang memiliki *payload* yang panjang akan memiliki nilai *byte frequency* yang relatif lebih besar dari *payload* yang lainnya. Untuk menyeimbangkan bobot masing-masing data, perlu dilakukan normalisasi terhadap data *byte frequency* masing-masing *payload*. Normalisasi bertujuan agar setiap data *byte frequency payload* memiliki bobot yang seimbang.

Pra proses data berakhir dengan didapatkannya sebuah data yang berisikan seluruh *header* dan *payload* dari file PCAP yang telah diproses. Gambaran hasil pra proses data dapat dilihat pada Gambar 3.9.

```

192.168.100.13,53942,74.125.24.105,443,[0,,0,4,4,7,9,2,2,2,8,8,1,8,1,3,2,7,6,,,0,,0,1,6,1,8,9,9,6,2,2,3,4,2,1,4,7,8,,,
192.168.100.13,64546,142.250.4.132,443,[0,,0,5,8,7,8,0,3,0,8,5,9,6,6,2,0,1,3,,,0,,0,2,0,5,7,3,1,0,8,0,0,8,8,1,7,0,4,7,,,
192.168.100.13,50271,172.217.194.95,443,[0,,0,1,0,9,7,7,4,4,3,6,0,9,0,2,2,5,5,7,,,0,,0,3,3,0,8,2,7,0,6,7,6,6,9,1,7,2,9,,,
192.168.100.13,63824,74.125.24.119,443,[0,,0,2,1,1,7,0,1,3,0,8,6,9,8,9,9,2,2,,,0,,0,1,6,9,3,6,1,0,4,6,9,5,9,1,9,9,3,7,,,
192.168.100.13,55430,74.125.24.93,443,[0,,0,4,6,6,4,3,7,1,7,7,2,8,0,5,5,0,8,,,0,,0,3,7,8,6,5,7,4,8,7,0,9,1,2,2,2,,,0,,0,1
192.168.100.13,60025,172.217.194.94,443,[0,,0,1,0,8,7,6,1,3,2,9,3,0,5,1,3,5,9,5,,,0,,0,3,3,2,3,2,6,2,8,3,9,8,7,9,1,5,4,,,
192.168.100.13,52930,172.217.194.94,443,[0,,0,1,1,0,2,7,1,9,0,3,3,2,3,2,6,2,8,4,,,0,,0,3,4,7,4,3,2,0,2,4,1,6,9,1,8,4,3,,,
192.168.100.13,55373,142.251.10.91,443,[0,,0,1,1,7,9,4,4,3,9,7,6,4,1,1,1,2,0,4,,,0,,0,1,3,0,5,8,1,2,9,7,3,8,8,3,7,4,0,4,,,
192.168.100.13,55683,74.125.24.155,443,[0,,0,3,6,4,6,8,3,3,0,1,3,4,3,5,7,0,0,5,,,0,,0,1,5,8,3,4,9,3,2,8,2,1,4,9,7,1,2,,,
192.168.100.13,55373,142.251.10.91,443,[0,,0,0,4,7,5,3,1,0,2,7,1,9,8,3,1,0,0,1,,,0,,0,0,5,2,8,1,2,2,5,2,4,4,2,5,6,6,6,7,
192.168.100.13,59679,172.217.194.148,443,[0,,0,1,2,7,1,3,8,2,3,3,9,3,4,3,5,0,4,5,,,0,,0,1,3,4,0,7,3,0,4,6,6,9,4,4,0,5,9,2,
192.168.100.13,54825,172.217.194.94,443,[0,,0,1,0,7,2,5,0,7,5,5,2,8,7,0,0,9,0,6,,,0,,0,3,3,2,3,2,6,2,8,3,9,8,7,9,1,5,4,,,
192.168.100.13,53052,74.125.24.105,443,[0,,0,3,7,4,2,3,0,2,2,2,6,4,3,2,9,7,0,4,,,0,,0,1,3,2,6,3,8,5,5,9,9,2,4,2,0,6,5,4,
192.168.100.13,63825,74.125.24.94,443,[0,,0,4,9,7,6,3,0,3,3,1,7,5,3,5,5,4,5,,,0,,0,1,7,1,8,0,0,9,4,7,8,6,7,2,9,8,5,8,,,
192.168.100.13,60710,74.125.24.94,443,[0,,0,3,9,1,6,0,8,3,9,1,6,0,8,3,9,1,6,4,,,0,,0,1,3,9,8,6,0,1,3,9,8,6,0,1,3,9,8,6,,,
192.168.100.13,54118,142.251.10.132,443,[0,,0,1,3,3,5,2,0,0,7,4,6,9,6,5,4,5,2,8,,,0,,0,0,9,8,0,3,9,2,1,5,6,8,6,2,7,4,5,,,

```

Gambar 3.9 Hasil Pra Proses Data

Informasi pada *header* yang diambil adalah informasi IP *address* sumber, protokol sumber, IP *address* tujuan, dan protokol tujuan. Data-data informasi ini yang nantinya digunakan untuk melakukan perbandingan hasil prediksi dengan data aktual yang berisikan daftar serangan. *Byte frequency* yang terbentuk nantinya akan digunakan untuk proses latih dan proses testing.

3.5.2 Implementasi Proses Latih

Setelah didapatkannya *byte frequency* data latih, proses selanjutnya adalah proses pembuatan model dengan menggunakan data latih tersebut. Proses pembuatan model dengan menggunakan data latih disebut juga dengan proses *fitting*. Data *byte frequency* yang didapatkan akan digunakan untuk proses *fitting* masing-masing algoritma. Implementasi dari proses *fitting* dapat dilihat pada Kode Program 3.3.

```

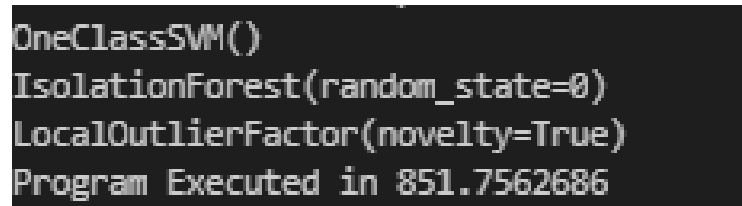
1. def ocsvm(data_latih):
2.     svm = OneClassSVM()
3.     print(svm)
4.     svm.fit(data_latih)
5.     joblib.dump(svm, 'ocsvm.pkl')
6.
7. def IsoForest(data_latih):
8.     svm = IsolationForest(random_state=0)
9.     print(svm)
10.    svm.fit(data_latih)
11.    joblib.dump(svm, 'IsoForest.pkl')
12.
13. def lof(data_latih):
14.     svm = LocalOutlierFactor(novelty=True)
15.     print(svm)
16.     svm.fit(data_latih)
17.     joblib.dump(svm, 'lof.pkl')

```

Kode Program 3.3 Implementasi Proses Latih

Pada proses *fitting* tersebut, terdapat 3 algoritma yang akan diujikan. Algoritma tersebut adalah *one-class SVM*, *isolation forest*, dan *local outlier factor*. Ketiga algoritma tersebut digunakan karena dapat mendukung proses *unsupervised learning*. Dari ketiga algoritma tersebut, akan terbentuk 3 model *machine learning* yang akan digunakan untuk memprediksi data pada saat proses testing. Hasil model yang terbentuk akan disimpan di dalam sebuah *dump file* berformat pkl. Setiap kali akan dilakukan proses testing, sistem akan terlebih dahulu membaca *dump file* yang terbentuk untuk mendapatkan model yang akan digunakan untuk proses prediksi.

Dari pemrosesan data latih hingga proses latih selesai dilakukan, program akan mencatat lama waktu eksekusi program. Gambaran eksekusi program dapat dilihat pada Gambar 3.10.



```

OneClassSVM()
IsolationForest(random_state=0)
LocalOutlierFactor(novelty=True)
Program Executed in 851.7562686

```

Gambar 3.10 Eksekusi Program Proses Latih

Waktu eksekusi program digunakan untuk melihat lama waktu yang dibutuhkan untuk program selesai berjalan pada masing-masing percobaan. Dengan data latih dan testing yang berbeda-beda, dapat dilihat bagaimana performa program dari awal program berjalan sampai program berhenti. Agar waktu eksekusi tidak terpengaruhi oleh faktor-faktor yang tidak diinginkan, program akan dijalankan pada perangkat komputer yang memiliki spesifikasi sama.

3.5.3 Implementasi Proses Testing

Setelah kedua proses sebelumnya selesai, diperoleh 2 komponen utama yang akan digunakan untuk proses testing. Komponen tersebut adalah model *machine learning* dan *byte frequency* data testing. Proses testing dilakukan dengan memprediksi nilai label pada data testing. Apabila sistem menyatakan sebuah data sebagai *outlier*, sistem akan memberikan data nilai label -1. Sedangkan data yang dianggap sebagai data normal akan diberikan label dengan nilai 1. Implementasi proses prediksi data dapat dilihat pada Kode Program 3.4.

```

1. def ocsvm(data_testing):
2.
3.     svm = joblib.load('ocsvm.pkl')
4.     pred = svm.predict(data_testing)
5.     print(pred)
6.     csv_name="ocsvm_result.csv"
7.     result_csv(pred,csv_name)
8.
9. def IsoForest(data_testing):
10.
11.     svm = joblib.load('IsoForest.pkl')
12.     pred = svm.predict(data_testing)
13.     print(pred)
14.     csv_name="Isoforest_result.csv"
15.     result_csv(pred,csv_name)
16.
17. def lof(data_testing):
18.
19.     svm = joblib.load('lof.pkl')
20.     pred = svm.predict(data_testing)
21.     print(pred)
22.     csv_name="lof_result.csv"
23.     result_csv(pred,csv_name)

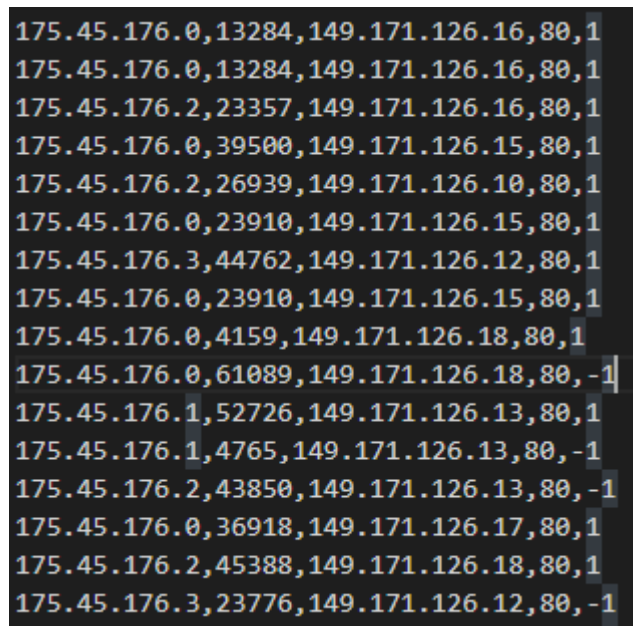
```

Kode Program 3.4 Implementasi Proses Testing

Hasil prediksi akan disimpan pada file berformat csv yang selanjutnya akan digunakan untuk pembuatan *confusion matrix*. *Confusion matrix* ini yang selanjutnya digunakan untuk evaluasi performa masing-masing algoritma.

3.5.4 Implementasi Evaluasi Performa

Setelah proses latih dan proses testing telah selesai, masing-masing algoritma akan dievaluasi performanya dengan melihat hasil dari prediksi data testing. Hasil prediksi berupa file csv yang berisi informasi IP *host*, protokol *host*, IP tujuan, protokol tujuan, dan hasil pelabelan. Gambaran data hasil prediksi dapat dilihat pada Gambar 3.11.



```

175.45.176.0,13284,149.171.126.16,80,1
175.45.176.0,13284,149.171.126.16,80,1
175.45.176.2,23357,149.171.126.16,80,1
175.45.176.0,39500,149.171.126.15,80,1
175.45.176.2,26939,149.171.126.10,80,1
175.45.176.0,23910,149.171.126.15,80,1
175.45.176.3,44762,149.171.126.12,80,1
175.45.176.0,23910,149.171.126.15,80,1
175.45.176.0,4159,149.171.126.18,80,1
175.45.176.0,61089,149.171.126.18,80,-1
175.45.176.1,52726,149.171.126.13,80,1
175.45.176.1,4765,149.171.126.13,80,-1
175.45.176.2,43850,149.171.126.13,80,-1
175.45.176.0,36918,149.171.126.17,80,1
175.45.176.2,45388,149.171.126.18,80,1
175.45.176.3,23776,149.171.126.12,80,-1

```

Gambar 3.11 Data Hasil Prediksi

Data hasil prediksi akan dibandingkan dengan data daftar serangan NUSW-NB15_GT untuk melihat apakah nilai pelabelan yang diberikan sudah sesuai dengan informasi pada dataset tersebut. Implementasi proses perbandingan hasil prediksi dengan data aktual dapat dilihat pada Kode Program 3.5

```

1. with open('NUSW-NB15_GT.csv') as csvfile:
2.     csvReader = csv.reader(csvfile, delimiter=',')
3.     for row in csvReader:
4.         sourceport=row[8]
5.         if(sourceport=='80'):
6.             list_atk_source.append(row[5])
7.             list_atk_sourceport.append(row[6])
8.             list_atk_destinationip.append(row[7])
9.             list_atk_destinationport.append(row[8])
10.            list_atk_name.append(row[9])
11.
12. #print(list_atk_source)
13. data_source=[]
14. data_sourceport=[]
15. data_destinationip=[]
16. data_destinationport=[]
17. data_prediksi=[]
18. with open('Isoforest_result.csv') as csvfile:
19.     csvReader = csv.reader(csvfile, delimiter=',')
20.     for row in csvReader:
21.         data_source.append(row[0])
22.         data_sourceport.append(row[1])
23.         data_destinationip.append(row[2])
24.         data_destinationport.append(row[3])
25.         data_prediksi.append(row[4])
26.
27. #print(data_prediksi)

```

Kode Program 3.5 Implementasi Evaluasi Hasil Prediksi

Perbandingan dilakukan dengan membuat sebuah *confusion matrix*. Nilai-nilai pada *confusion matrix* selanjutnya digunakan untuk mendapatkan nilai F_2 -score, *detection rate*, dan *false positive rate*. Implementasi pembuatan *confusion matrix* dapat dilihat pada Kode Program 3.6.

```

1. for data in data_source:
2.     if (data in list_atk_source):
3.         if(data_prediksi[total_data]=='-1'):
4.             tp=tp+1
5.         else:
6.             fn=fn+1
7.     else:
8.         if(data_prediksi[total_data]=='-1'):
9.             fp=fp+1
10.        else:
11.            tn=tn+1
12.        total_data=total_data+1

```

Kode Program 3.6 Implementasi *Confusion Matrix*

Hasil akhir performa berbentuk sebuah tabel yang berisi informasi-informasi yang didapatkan dari proses evaluasi. Contoh hasil akhir dapat dilihat pada Tabel 3.7.

Tabel 3.7 Tabel Evaluasi Performa

| Algoritma | Parameter | TP | TN | FP | FN | Total Data | F2-score | DR | FPR |
|-----------|----------------|-----|-------|------|-----|------------|----------|----------|----------|
| OCSVM | <i>Default</i> | 288 | 23251 | 482 | 161 | 24182 | 0,561185 | 0,641425 | 0,020309 |
| LOF | <i>Default</i> | 266 | 23454 | 279 | 183 | 24182 | 0,568133 | 0,592428 | 0,011756 |
| ISOF | <i>Default</i> | 449 | 21407 | 2326 | 0 | 24182 | 0,49114 | 1 | 0,098007 |

Dari informasi tersebut akan dibuat sebuah grafik yang memperlihatkan seluruh proses uji coba. Grafik tersebut menjadi sebuah patokan bagaimana performa setiap algoritma apabila data latih yang digunakan memiliki kadar data kotor yang berbeda-beda.

BAB 4 Hasil dan Pembahasan

4.1 Skenario Evaluasi Hasil Uji Coba

Evaluasi uji coba dilakukan dengan melihat hasil dari prediksi data testing. Hasil prediksi data testing disimpulkan menjadi *confusion matrix*. *Confusion matrix* digunakan untuk membandingkan hasil prediksi yang dilakukan oleh program dengan hasil prediksi yang benar. Dari *confusion matrix* didapatkan nilai *true positive*, *true negative*, *false positive*, dan *false negative* dari hasil prediksi. Hasil *Confusion matrix* ini yang selanjutnya digunakan untuk menentukan nilai dari F_2 -score masing-masing hasil data. Nilai F_2 -score selanjutnya digunakan untuk pembanding performa masing-masing algoritma.

Setiap data PCAP dengan kadar yang berbeda akan menghasilkan data hasil prediksi yang berupa *confusion matrix* yang selanjutnya digunakan untuk analisa performa masing-masing algoritma. Analisa dilakukan dengan melihat nilai rata-rata F_2 -score untuk 3 kali perulangan pada 3 algoritma yang berbeda. Dari sini terlihat bagaimana penurunan dan kenaikan dari nilai F_2 -score apabila data latih yang digunakan memiliki kadar data kotor yang berbeda-beda.

Ada 3 jenis algoritma yang digunakan dalam pengujian. Algoritma yang digunakan adalah *one-class SVM* (OCSVM), *local outlier factor* (LOF), dan *isolation forest* (ISOF). Ketiga algoritma berikut merupakan algoritma *unsupervised learning*. Sehingga memungkinkan untuk melakukan proses latih dengan data latih yang tidak memiliki label.

4.2 Evaluasi Data Hasil Percobaan

Percobaan dibedakan menjadi 3 jenis berdasarkan protokol masing-masing dataset. Setiap percobaan dilakukan sebanyak 3 kali perulangan dengan 3 algoritma *machine learning* yang berbeda. Pada subbab ini akan dipaparkan data hasil percobaan dan analisa berdasarkan nilai F_2 -score masing-masing algoritma. Selain nilai F_2 -score, performa algoritma juga akan diuji dengan menggunakan nilai dari *detection rate* (DR) dan *false positive rate* (FPR).

4.2.1 Hasil Pengujian Protokol FTP

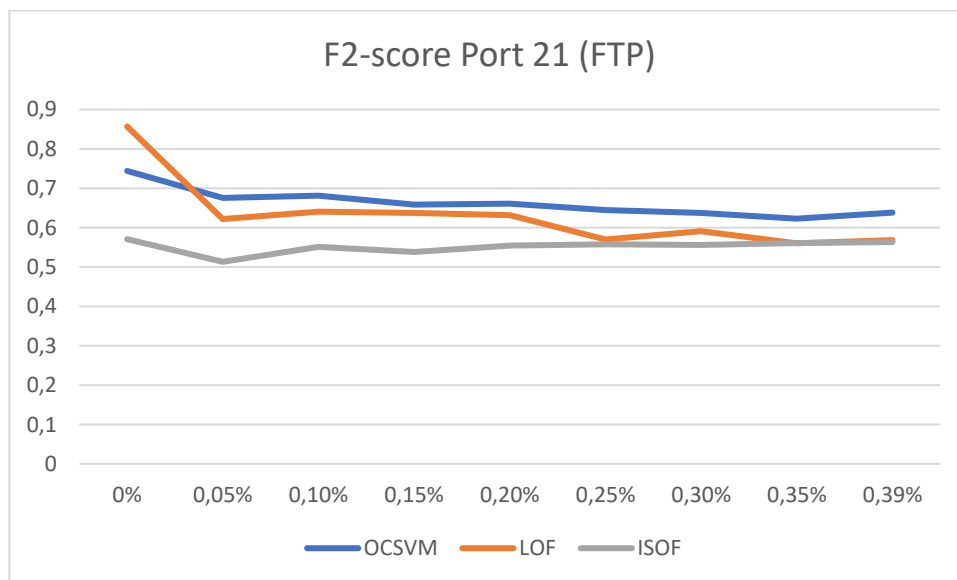
Pada protokol FTP terdapat total 9 data latih PCAP dengan kadar yang berbeda-beda. Masing-masing data PCAP memiliki ukuran dengan rata-rata 228 MB. Proses eksekusi program membutuhkan waktu rata-rata 10 menit dari proses latih hingga proses testing. Pengujian dilakukan sebanyak 3 kali percobaan pada setiap 3 algoritma yang berbeda. Tiap algoritma menggunakan parameter *default* dari pustaka *scikit-learn*. Hasil total terdapat 81 data *confusion matrix*. Sampel data *confusion matrix* hasil percobaan dapat dilihat pada Tabel 4.1.

Tabel 4.1 Sampel Data *Confusion Matrix* Protokol FTP

| Kadar Data kotor | Port | Algoritma | Parameter | TP | TN | FP | FN |
|------------------|------|-----------|----------------|-----|-------|------|-----|
| 0.000 | 21 | OCSVM | <i>Default</i> | 350 | 22511 | 475 | 88 |
| | | LOF | <i>Default</i> | 273 | 22687 | 299 | 165 |
| | | ISOF | <i>Default</i> | 438 | 21553 | 1433 | 0 |
| 0.0005 | 21 | OCSVM | <i>Default</i> | 350 | 22525 | 464 | 92 |
| | | LOF | <i>Default</i> | 254 | 22700 | 289 | 188 |
| | | ISOF | <i>Default</i> | 442 | 21594 | 1395 | 0 |

| | | | | | | | |
|--------|----|-------|---------|-----|-------|------|-----|
| 0.001 | 21 | OCSVM | Default | 388 | 22544 | 462 | 50 |
| | | LOF | Default | 291 | 22738 | 268 | 147 |
| | | ISOF | Default | 438 | 21202 | 1804 | 0 |
| 0.0015 | 21 | OCSVM | Default | 381 | 22471 | 545 | 60 |
| | | LOF | Default | 301 | 22707 | 309 | 140 |
| | | ISOF | Default | 441 | 21343 | 1673 | 0 |
| 0.002 | 21 | OCSVM | Default | 371 | 22534 | 482 | 67 |
| | | LOF | Default | 297 | 22704 | 312 | 141 |
| | | ISOF | Default | 438 | 21321 | 1695 | 0 |
| 0.0025 | 21 | OCSVM | Default | 357 | 22556 | 491 | 81 |
| | | LOF | Default | 271 | 22715 | 332 | 167 |
| | | ISOF | Default | 438 | 21530 | 1517 | 0 |
| 0.003 | 21 | OCSVM | Default | 360 | 22450 | 545 | 79 |
| | | LOF | Default | 277 | 22690 | 305 | 162 |
| | | ISOF | Default | 439 | 21520 | 1475 | 0 |
| 0.0035 | 21 | OCSVM | Default | 350 | 22525 | 464 | 92 |
| | | LOF | Default | 254 | 22700 | 289 | 188 |
| | | ISOF | Default | 442 | 21594 | 1395 | 0 |
| 0.0039 | 21 | OCSVM | Default | 350 | 22511 | 475 | 88 |
| | | LOF | Default | 273 | 22687 | 299 | 165 |
| | | ISOF | Default | 438 | 21553 | 1433 | 0 |

Dari data *confusion matrix* yang didapatkan diatas, dapat dihitung nilai masing-masing F_2 -score, DR, dan FPR. Visualisasi nilai F_2 -score dari data hasil pengujian dapat dilihat pada Gambar 4.1.



Gambar 4.1 Grafik F_2 -score Hasil Pengujian Protokol FTP

Dari data diatas dapat dilihat nilai puncak F_2 -score pada masing-masing data terdapat pada saat dilakukan percobaan pada dataset dengan kadar 0%. Penurunan F_2 -score tampak jelas terlihat pada grafik seiring dengan kenaikan kadar data kotor pada data yang digunakan. Pada algoritma *One-Class SVM* puncak tertinggi terjadi pada kadar 0% dengan nilai F_2 -score sebesar 0,74. Lalu terjadi penurunan nilai F_2 -score seiring dengan meningkatnya kadar data kotor dengan

rentang nilai 0,68-0,62, puncak terendah terjadi pada data yang memiliki kadar 0.35% dengan nilai F_2 -score sebesar 0.62.

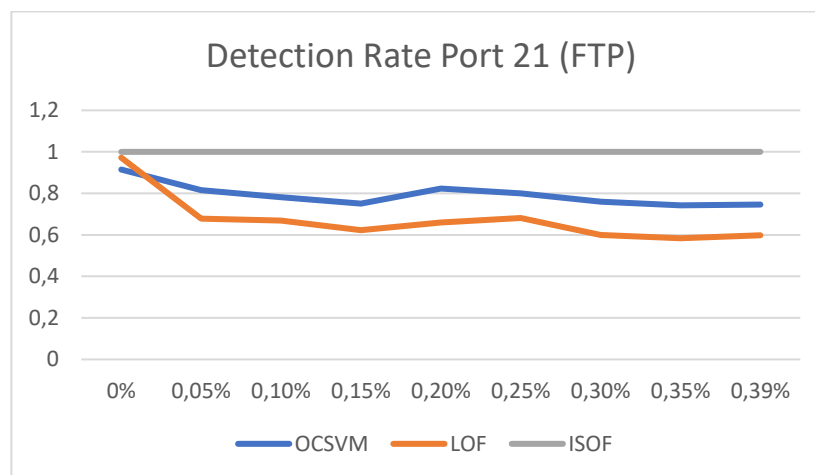
Pada algoritma *local outlier factor*, puncak tertinggi nilai F_2 -score juga terjadi pada data yang memiliki kadar 0% dengan nilai F_2 -score sebesar 0,85. Dari nilai F_2 -score sebesar 0,85, nilai F_2 -score data pada algoritma LOF mengalami penurunan dengan titik terendah pada nilai 0,5602 pada data dengan kadar data kotor sebesar 0.35%.

Sedangkan pada algoritma *isolation forest*, puncak tertinggi memiliki nilai F_2 -score sebesar 0,57 yang terjadi pada data dengan kadar data kotor sebesar 0%. Puncak terendah terjadi pada data dengan kadar data kotor sebesar 0,05% dengan nilai F_2 -score sebesar 0,51. Lebih detail nilai F_2 -score Protokol FTP dapat dilihat pada Tabel 4.2.

Tabel 4.2 Detail Nilai F_2 -score Protokol FTP

| Kadar Data Kotor | OCSVM | LOF | ISOF |
|------------------|----------|-------------|-------------|
| 0% | 0,744039 | 0,856936639 | 0,570712542 |
| 0,05% | 0,675212 | 0,62188819 | 0,513176392 |
| 0,10% | 0,681534 | 0,640530879 | 0,550927165 |
| 0,15% | 0,658513 | 0,637435585 | 0,538198215 |
| 0,20% | 0,661011 | 0,631361069 | 0,55434825 |
| 0,25% | 0,644925 | 0,569594236 | 0,557829109 |
| 0,30% | 0,636961 | 0,590967456 | 0,556118658 |
| 0,35% | 0,622802 | 0,560224547 | 0,561180129 |
| 0,39% | 0,63851 | 0,568342775 | 0,563702567 |

Selain nilai F_2 -score, observasi juga dilakukan dengan melihat nilai dari DR dan FPR masing-masing algoritma. Gambaran nilai DR dapat dilihat pada Gambar 4.2.



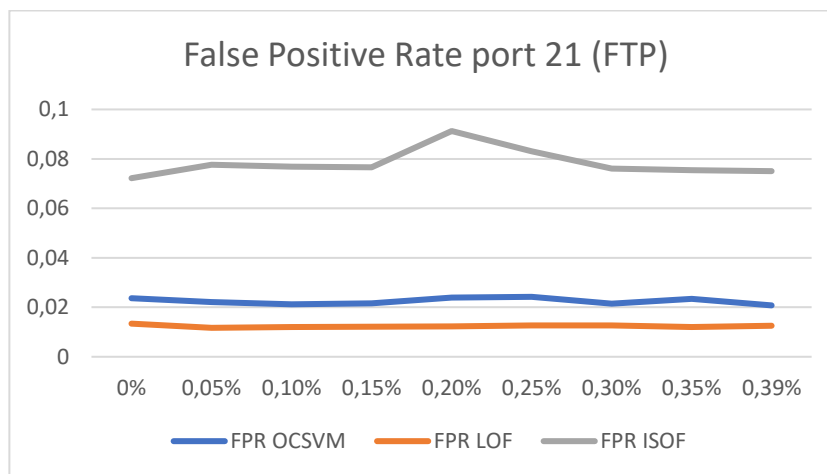
Gambar 4.2 Grafik DR Hasil Pengujian Protokol FTP

Dilihat dari nilai DR masing-masing algoritma, algoritma *isolation forest* memiliki nilai DR yang relatif stabil sebesar 1 pada setiap hasil percobaan. Hal ini menandakan hasil percobaan memiliki nilai FN sebesar 0, menandakan algoritma mampu mendeteksi seluruh data yang bersifat *malicious*.

Pada algoritma *local outlier factor*, nilai DR memiliki rentang antara 0,97-0,58. Nilai tertinggi terletak pada percobaan menggunakan kadar data kotor sebesar 0%. Sedangkan nilai terendah terjadi pada saat percobaan menggunakan kadar data kotor sebesar 0,35%. Berbeda dengan algoritmya sebelumnya, algoritma LOF tampak mengalami penurunan dalam mendeteksi data *malicious*. Penurunan tampak terjadi dengan seiring meningkatnya kadar data kotor yang ada pada data latih yang digunakan.

Pada algoritma *one-class SVM*, nilai DR memiliki rentang antara 0,91-0,74. Hampir sama seperti algoritma LOF, nilai DR algoritma OCSVM tampak turun seiring meningkatnya kadar data kotor pada data latih yang digunakan. Nilai DR tertinggi terletak pada data dengan kadar data kotor sebesar 0%. Sedangkan nilai terendah terdapat pada saat percobaan menggunakan kadar data kotor sebesar 0,35%.

Selain DR, nilai FPR pada masing-masing algoritma juga dilihat. Nilai FPR masing-masing algoritma dapat dilihat pada Gambar 4.3.



Gambar 4.3 Grafik FPR Hasil Pengujian Protokol FTP

Dari grafik gambar diatas dapat dilihat bahwa algoritma *isolation forest* memiliki rentang nilai FPR yang relatif lebih tinggi dibanding algoritmya yang lain. Nilai FPR pada algoritma ISOF memiliki rentang dari 0,091-0,072. Sedangkan pada algoritma *one-class SVM* dan *isolation forest*, memiliki nilai FPR relatif jauh lebih rendah. Pada algoritma *one-class SVM* nilai FPR berada pada rentang 0,024-0,020. Sedangkan pada algoritma *local outlier factor*, nilai FPR berada pada rentang 0,013-0,012.

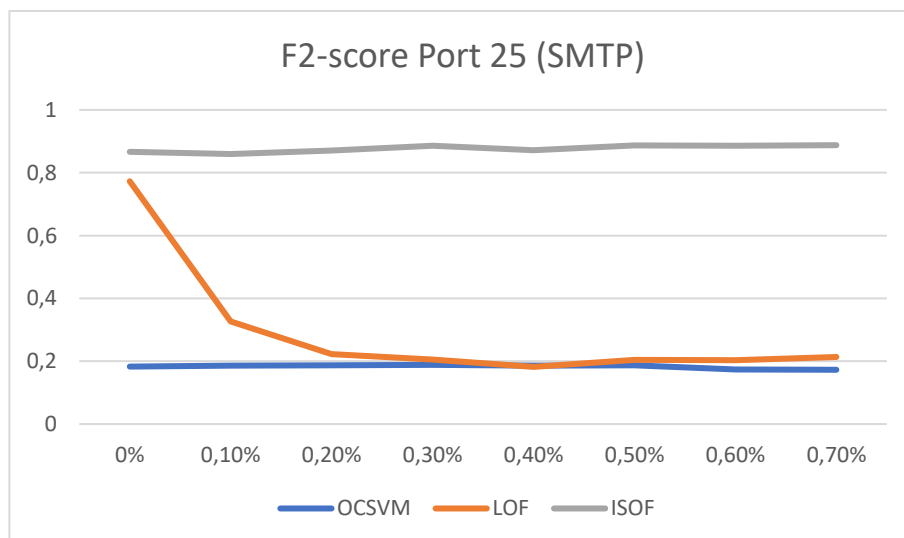
4.2.2 Hasil Pengujian Protokol SMTP

Pada protokol SMTP terdapat total 8 data latih PCAP dengan kadar yang berbeda-beda. Masing-masing data PCAP memiliki ukuran dengan rata-rata 1,9 GB. Pengujian dilakukan sebanyak 3 kali percobaan pada setiap 3 algoritma yang berbeda. Tiap algoritma menggunakan parameter *default* dari pustaka *skit-learn*. Proses eksekusi program membutuhkan waktu rata-rata 1 jam 30 menit dari proses latih hingga proses testing. Hasil total terdapat 72 data *confusion matrix*. Sampel data *confusion matrix* hasil percobaan dapat dilihat pada Tabel 4.3.

Tabel 4.3 Sampel Data *Confusion Matrix* Protokol SMTP

| Kadar Data kotor | Port | Algoritma | Parameter | TP | TN | FP | FN |
|------------------|------|-----------|----------------|-----|-------|-------|-----|
| 0.00 | 25 | OCSVM | <i>Default</i> | 716 | 25718 | 15869 | 2 |
| | | LOF | <i>Default</i> | 717 | 40524 | 1063 | 1 |
| | | ISOF | <i>Default</i> | 718 | 41058 | 529 | 0 |
| 0.001 | 25 | OCSVM | <i>Default</i> | 710 | 25689 | 15918 | 2 |
| | | LOF | <i>Default</i> | 270 | 40546 | 1061 | 442 |
| | | ISOF | <i>Default</i> | 712 | 40993 | 614 | 0 |
| 0.002 | 25 | OCSVM | <i>Default</i> | 727 | 25950 | 15615 | 2 |
| | | LOF | <i>Default</i> | 186 | 40521 | 1044 | 543 |
| | | ISOF | <i>Default</i> | 729 | 41038 | 527 | 0 |
| 0.003 | 25 | OCSVM | <i>Default</i> | 708 | 26276 | 15320 | 3 |
| | | LOF | <i>Default</i> | 168 | 40538 | 1058 | 543 |
| | | ISOF | <i>Default</i> | 711 | 41095 | 501 | 0 |
| 0.004 | 25 | OCSVM | <i>Default</i> | 717 | 26138 | 15448 | 3 |
| | | LOF | <i>Default</i> | 152 | 40527 | 1059 | 568 |
| | | ISOF | <i>Default</i> | 720 | 41045 | 541 | 0 |
| 0.005 | 25 | OCSVM | <i>Default</i> | 711 | 26465 | 15107 | 10 |
| | | LOF | <i>Default</i> | 164 | 40516 | 1056 | 557 |
| | | ISOF | <i>Default</i> | 721 | 41138 | 434 | 0 |
| 0.006 | 25 | OCSVM | <i>Default</i> | 668 | 27171 | 14411 | 62 |
| | | LOF | <i>Default</i> | 173 | 40527 | 1055 | 557 |
| | | ISOF | <i>Default</i> | 730 | 41154 | 428 | 0 |
| 0.007 | 25 | OCSVM | <i>Default</i> | 622 | 27186 | 14406 | 93 |
| | | LOF | <i>Default</i> | 171 | 40532 | 1060 | 544 |
| | | ISOF | <i>Default</i> | 715 | 41084 | 508 | 0 |

Dari data *confusion matrix* yang didapatkan diatas dapat dihitung nilai masing-masing F_2 -score, DR, dan FPR. Visualisasi nilai F_2 -score dari data hasil pengujian dapat dilihat pada Gambar 4.4.

**Gambar 4.4** Grafik F_2 -score Hasil Pengujian Protokol SMTP

Pada algoritma *Isolation Forest* tampak nilai F_2 -score relatif konstan berkisar pada rentang 0,85 – 0,88. Perbedaan F_2 -score tidak terlalu signifikan dengan selisih batas atas dan batas bawah hanya 0,03.

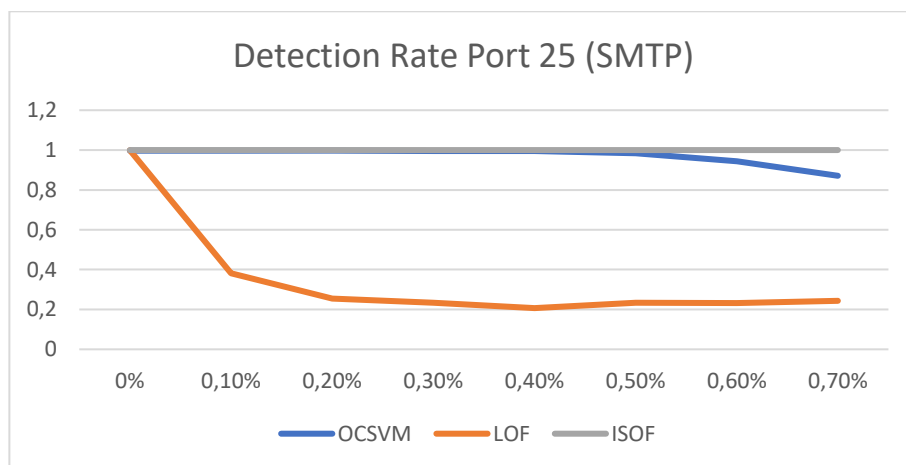
Pada algoritma *Local Outlier Factor*, titik puncak terjadi pada data dengan kadar data kotor sebesar 0% dengan nilai F_2 -score sebesar 0,77. Namun, penurunan nilai F_2 -score tampak sangat jelas dibandingkan algoritma *One-Class SVM*. Dari angka 0,77, nilai F_2 -score turun drastis menjadi 0,32. Titik terendah terjadi pada data testing dengan kadar data sebesar 0,4% dengan nilai F_2 -score sebesar 0,18.

Sedangkan pada algoritma *One-Class SVM*, tampak nilai F_2 -score relatif stabil tetapi rata-rata nilainya terlalu rendah dibanding dengan algoritma lain. Nilai F_2 -score hanya berkisar pada rentang 0,17-0,18. Sangat rendah dibanding dengan algoritma yang lain. Titik tertinggi terjadi pada data testing dengan kadar 0,6%. Dan titik terendah terjadi pada data testing dengan kadar 0,7%. Detail nilai F_2 -score protokol SMTP dapat dilihat pada Tabel 4.4.

Tabel 4.4 Detail Nilai F_2 -score Protokol SMTP

| Kadar Data Kotor | OCSVM | LOF | ISOF |
|------------------|----------|-------------|-------------|
| 0% | 0,18287 | 0,772757828 | 0,866105613 |
| 0,1% | 0,186234 | 0,326722944 | 0,859494838 |
| 0,2% | 0,187221 | 0,223044539 | 0,870948655 |
| 0,3% | 0,18852 | 0,205572386 | 0,885918339 |
| 0,4% | 0,185642 | 0,181995322 | 0,871593739 |
| 0,5% | 0,187419 | 0,204427568 | 0,887231535 |
| 0,6% | 0,174137 | 0,203317481 | 0,88630814 |
| 0,7% | 0,17288 | 0,213212364 | 0,887685586 |

Selain nilai F_2 -score, observasi juga dilakukan dengan melihat nilai dari DR dan FPR masing-masing algoritma. Grafik nilai DR protokol SMTP dapat dilihat pada Gambar 4.5.



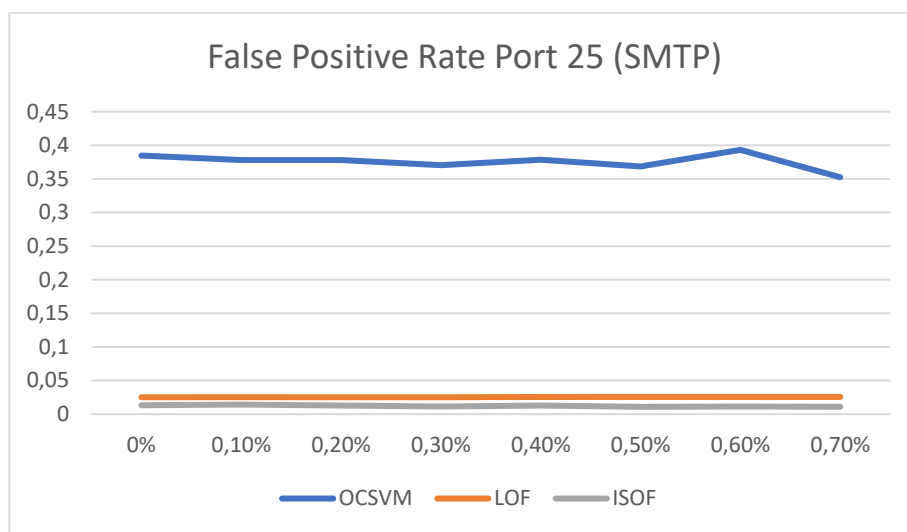
Gambar 4.5 Grafik DR Hasil Pengujian Protokol SMTP

Dilihat dari nilai DR masing-masing algoritma, algoritma *isolation forest* memiliki nilai DR yang relatif stabil sebesar 1 pada setiap hasil percobaan. Hal ini menandakan hasil percobaan memiliki nilai FN sebesar 0, menandakan algoritma mampu mendeteksi seluruh data yang bersifat *malicious*.

Pada algoritma *local outlier factor*, nilai DR memiliki rentang antara 0,99-0,24. Nilai tertinggi terletak pada percobaan menggunakan kadar data kotor sebesar 0%. Sedangkan nilai terendah terjadi pada saat percobaan menggunakan kadar data kotor sebesar 0,4%. Berbeda dengan algoritmya sebelumnya, algoritma LOF tampak mengalami penurunan dalam mendeteksi data *malicious*. Penurunan tampak terjadi dengan seiring meningkatnya kadar data kotor yang ada pada data latih yang digunakan.

Pada algoritma *one-class SVM*, nilai DR memiliki rentang antara 0,99-0,87. Hampir sama seperti algoritma LOF, nilai DR algoritma OCSVM tampak turun seiring meningkatnya kadar data kotor pada data latih yang digunakan. Nilai DR tertinggi terletak pada data dengan kadar data kotor sebesar 0%. Sedangkan nilai terendah terdapat pada saat percobaan menggunakan kadar data kotor sebesar 0,7%.

Selain DR, nilai FPR pada masing-masing algoritma juga dilihat. Nilai FPR masing-masing algoritma dapat dilihat pada Gambar 4.6.



Gambar 4.6 Grafik FPR Hasil Pengujian Protokol SMTP

Dari grafik gambar diatas dapat dilihat bahwa algoritma *one-class SVM* memiliki rentang nilai FPR yang sangat tinggi dibanding algoritmya yang lain. Nilai FPR pada algoritma OCSVM memiliki rentang dari 0,39-0,36. Hal ini menandakan ada lebih dari 30% prediksi data yang terdeteksi mengalami FP. Nilai FP yang sangat besar dapat juga diartikan nilai ambang batas dari model yang terbentuk tidak dapat mencakup seluruh data normal yang mengakibatkan banyaknya data normal dikategorikan sebagai *outlier*

Sedangkan pada algoritma *isolation forest* dan *local outlier factor*, memiliki nilai FPR relatif jauh lebih rendah. Pada algoritma *isolation forest* nilai FPR berada pada rentang 0,014-0,010. Sedangkan pada algoritma *local outlier factor*, nilai FPR berada pada rentang 0,0254-

0,0252. Nilai FPR pada LOF sangat stabil, tidak terlalu terlihat efek perubahan kadar data kotor pada data testingnya.

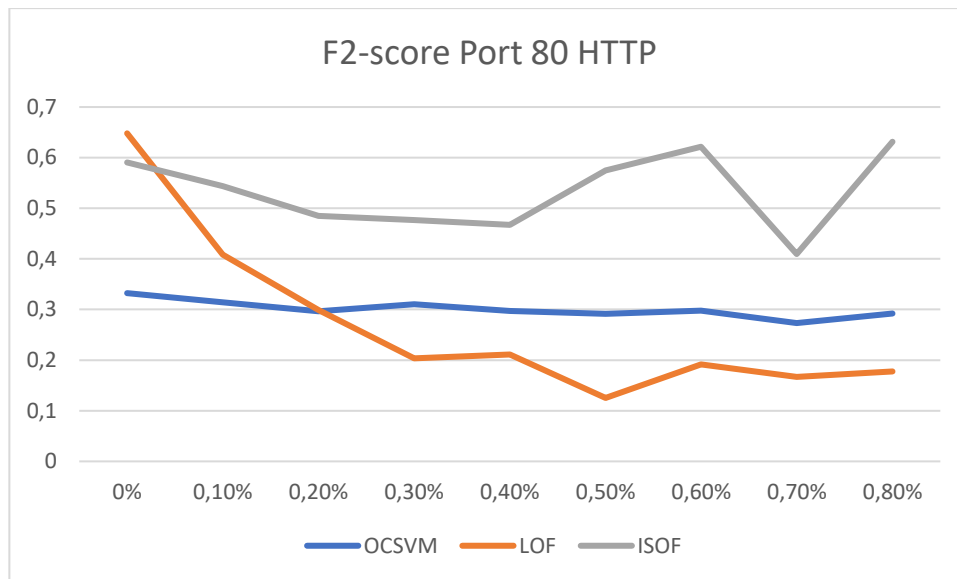
4.2.3 Hasil Pengujian Protokol HTTP

Pada protokol SMTP terdapat total 9 data latih PCAP dengan kadar yang berbeda-beda. Masing-masing data PCAP memiliki ukuran dengan rata-rata 11,4 GB. Pengujian dilakukan sebanyak 3 kali percobaan pada setiap 3 algoritma yang berbeda. Tiap algoritma menggunakan parameter *default* dari pustaka *skit-learn*. Proses eksekusi program membutuhkan waktu rata-rata 10 jam dari proses latih hingga proses testing. Hasil total terdapat 81 data *confusion matrix*. Sampel data *confusion matrix* hasil percobaan dapat dilihat pada Tabel 4.5.

Tabel 4.5 Sampel Data *Confusion Matrix* Protokol HTTP

| Kadar Data kotor | Port | Algoritma | Parameter | TP | TN | FP | FN |
|------------------|------|-----------|----------------|------|--------|-------|------|
| 0.00 | 80 | OCSVM | <i>Default</i> | 1116 | 118550 | 2025 | 2296 |
| | | LOF | <i>Default</i> | 2766 | 115647 | 4928 | 646 |
| | | ISOF | <i>Default</i> | 2798 | 113342 | 7233 | 614 |
| 0.001 | 80 | OCSVM | <i>Default</i> | 1083 | 120756 | 2489 | 2335 |
| | | LOF | <i>Default</i> | 1672 | 118112 | 5133 | 1746 |
| | | ISOF | <i>Default</i> | 2640 | 115272 | 7973 | 778 |
| 0.002 | 80 | OCSVM | <i>Default</i> | 1045 | 118276 | 2936 | 2366 |
| | | LOF | <i>Default</i> | 1161 | 116607 | 4605 | 2250 |
| | | ISOF | <i>Default</i> | 2243 | 113982 | 7230 | 1168 |
| 0.003 | 80 | OCSVM | <i>Default</i> | 1047 | 114420 | 2123 | 2377 |
| | | LOF | <i>Default</i> | 913 | 108693 | 7850 | 2511 |
| | | ISOF | <i>Default</i> | 2115 | 110173 | 6370 | 1309 |
| 0.004 | 80 | OCSVM | <i>Default</i> | 1001 | 119774 | 2153 | 2418 |
| | | LOF | <i>Default</i> | 833 | 116736 | 5191 | 2586 |
| | | ISOF | <i>Default</i> | 2174 | 114521 | 7406 | 1245 |
| 0.005 | 80 | OCSVM | <i>Default</i> | 1006 | 122987 | 2579 | 2415 |
| | | LOF | <i>Default</i> | 740 | 110462 | 15104 | 2681 |
| | | ISOF | <i>Default</i> | 2883 | 117064 | 8502 | 538 |
| 0.006 | 80 | OCSVM | <i>Default</i> | 944 | 113608 | 1272 | 2467 |
| | | LOF | <i>Default</i> | 707 | 110798 | 4082 | 2704 |
| | | ISOF | <i>Default</i> | 2828 | 108610 | 6270 | 583 |
| 0.007 | 80 | OCSVM | <i>Default</i> | 955 | 120907 | 2859 | 2461 |
| | | LOF | <i>Default</i> | 680 | 117765 | 6001 | 2736 |
| | | ISOF | <i>Default</i> | 1985 | 115185 | 8581 | 1431 |
| 0,008 | 80 | OCSVM | <i>Default</i> | 941 | 112847 | 1522 | 2474 |
| | | LOF | <i>Default</i> | 661 | 110076 | 4293 | 2754 |
| | | ISOF | <i>Default</i> | 2768 | 108878 | 5491 | 647 |

Dari data *confusion matrix* yang didapatkan diatas dapat dihitung nilai masing-masing F_2 -score, DR, dan FPR. Visualisasi nilai F_2 -score dari data hasil pengujian dapat dilihat pada Gambar 4.7.



Gambar 4.7 Grafik F_2 -score Hasil Pengujian Protokol HTTP

Pada algoritma *One-Class SVM*, tampak nilai F_2 -score relatif stabil pada kisaran 0,33 – 0,27. Puncak tertinggi terjadi pada data dengan kadar data kotor sebesar 0% dengan nilai F_2 -score sebesar 0,33. Sedangkan puncak terendah terjadi pada data dengan kadar data kotor sebesar 0,7% dengan nilai F_2 -score sebesar 0,27.

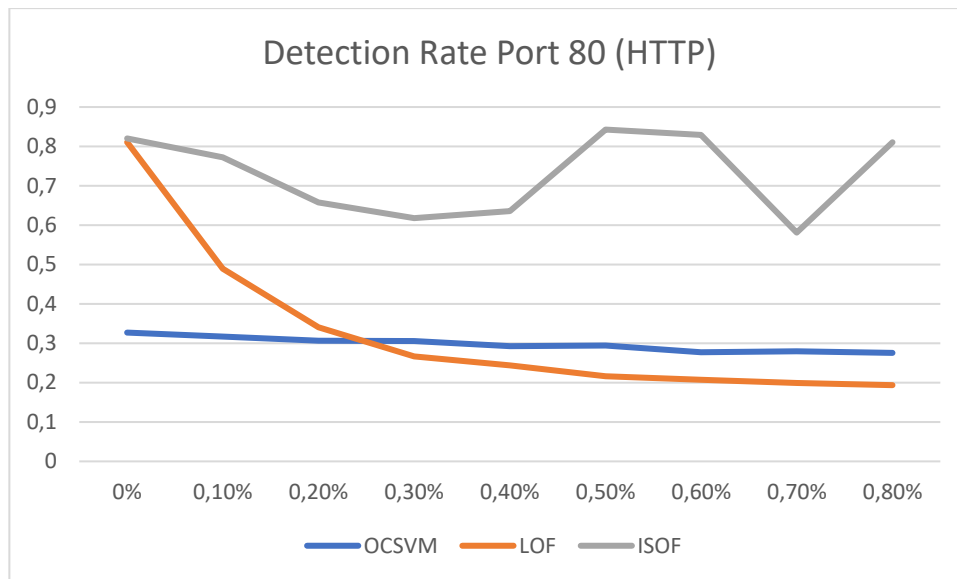
Pada algoritma *Local Outlier Factor*, titik puncak terjadi pada data dengan kadar data kotor sebesar 0% dengan nilai F_2 -score sebesar 0,64. Tampak nilai F_2 -score mengalami penurunan drastis seiring bertambahnya kadar data kotor dengan penurunan dari 0,64 sampai dengan nilai puncak terendah sebesar 0,12 yang terjadi pada data dengan kadar data kotor sebesar 0,05%.

Sedangkan pada algoritma *Isolation Forest*, tampak nilai bergelombang dengan rentan nilai F_2 -score antara 0,63-0,40. Nilai puncak terjadi pada data testing dengan kadar data kotor sebesar 0,8%. Sedangkan puncak terendah terjadi pada data testing dengan kadar data kotor sebesar 0,7%. Detail nilai F_2 -score protokol HTTP dapat dilihat pada Tabel 4.6.

Tabel 4.6 Detail Nilai F_2 -score Protokol HTTP

| Kadar Data Kotor | OCSVM | LOF | ISOF |
|------------------|----------|-------------|-------------|
| 0% | 0,33236 | 0,648017993 | 0,590818869 |
| 0,10% | 0,314022 | 0,408262929 | 0,543545398 |
| 0,20% | 0,296454 | 0,299072643 | 0,485140805 |
| 0,30% | 0,310388 | 0,203259272 | 0,476759389 |
| 0,40% | 0,297386 | 0,21142132 | 0,467406261 |
| 0,50% | 0,291273 | 0,125304795 | 0,575012964 |
| 0,60% | 0,297604 | 0,19177562 | 0,621757101 |
| 0,70% | 0,273201 | 0,167117228 | 0,409616178 |
| 0,80% | 0,291819 | 0,177554529 | 0,631415667 |

Selain nilai F_2 -score, observasi juga dilakukan dengan melihat nilai dari DR dan FPR masing-masing algoritma. Gambaran nilai DR protokol HTTP dapat dilihat pada Gambar 4.8.



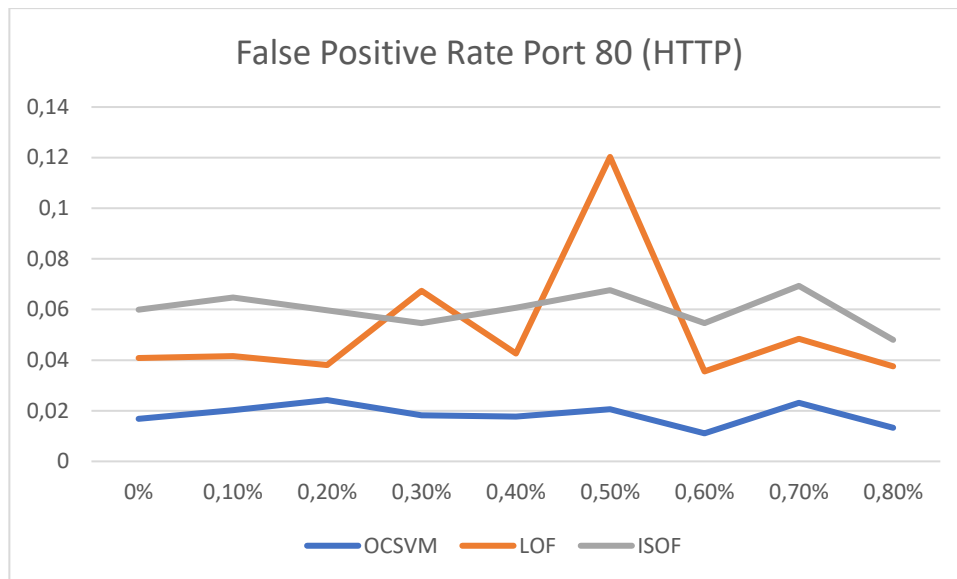
Gambar 4.8 Grafik DR Hasil Pengujian Protokol HTTP

Dilihat dari nilai DR masing-masing algoritma, algoritma *isolation forest* memiliki nilai DR yang tampak tidak stabil dibandingkan nilai DR algoritma yang lain. Nilai DR algoritma ISOF antara rentang 0,84-0,58. Nilai DR tertinggi terletak pada percobaan menggunakan data dengan kadar data kotor sebesar 0,5%. Sedangkan nilai terendah terdapat pada percobaan menggunakan data dengan kadar data kotor sebesar 0,7%.

Pada algoritma *local outlier factor*, nilai DR memiliki rentang antara 0,81-0,19. Nilai tertinggi terletak pada percobaan menggunakan kadar data kotor sebesar 0%. Sedangkan nilai terendah terjadi pada saat percobaan menggunakan kadar data kotor sebesar 0,8%. Berbeda dengan algoritmya sebelumnya, algoritma LOF tampak mengalami penurunan dalam mendeteksi data *malicious*. Penurunan tampak terjadi dengan seiring meningkatnya kadar data kotor yang ada pada data latih yang digunakan. Penurunan DR sangat signifikan antara percobaan dengan kadar 0% terhadap percobaan yang lain.

Pada algoritma *one-class SVM*, nilai DR memiliki rentang antara 0,32-0,27. Hampir sama seperti algoritma LOF, nilai DR algoritma OCSVM tampak turun seiring meningkatnya kadar data kotor pada data latih yang digunakan. Tetapi penurunan yang terjadi tidak signifikan yang terjadi pada algoritma LOF. Nilai DR tertinggi terletak pada data dengan kadar data kotor sebesar 0%. Sedangkan nilai terendah terdapat pada saat percobaan menggunakan kadar data kotor sebesar 0,8%.

Selain DR, nilai FPR pada masing-masing algoritma juga dilihat. Nilai FPR masing-masing algoritma dapat dilihat pada Gambar 4.9.



Gambar 4.9 Grafik FPR Hasil Pengujian Protokol HTTP

Dari grafik gambar diatas terlihat bahwa algoritma *local outlier factor* memiliki rentang nilai FPR yang tampak tidak stabil dibandingkan dengan algoritmya yang lain. Nilai FPR pada algoritma LOF memiliki rentang dari 0,12-0,03.

Sedangkan pada algoritma *isolation forest* dan *one-class SVM*, memiliki nilai FPR relatif lebih stabil. Pada algoritma *isolation forest* nilai FPR berada pada rentang 0,067-0,048. Sedangkan pada algoritma OCSVM, nilai FPR berada pada rentang 0,024-0,011.

4.2.4 Hasil Evaluasi Keseluruhan

Berdasarkan rata-rata waktu eksekusi program, waktu eksekusi meningkat seiring besarnya data yang diproses meningkat juga. Informasi waktu eksekusi dapat dilihat pada Tabel 4.7.

Tabel 4.7 Rata-rata Waktu Eksekusi Program

| Protokol | Ukuran file | Waktu Eksekusi |
|----------|-------------|----------------|
| FTP | 228 MB | 10 menit |
| SMTP | 1,9 GB | 90 menit |
| HTTP | 11,4 GB | 10 jam |

Semakin besar ukuran data yang diproses, waktu yang dibutuhkan juga semakin meningkat. Perhitungan waktu dilakukan dari awal pra proses data hingga selesai proses evaluasi performa program.

Dari hasil evaluasi nilai F_2 -score, DR, dan FPR masing-masing algoritma, dapat dipetik sebuah kesimpulan terhadap performa dari masing-masing algoritma. Kesimpulan ditarik dengan melihat rata-rata pada setiap percobaan. Rata-rata nilai F_2 -score dapat dilihat pada Tabel 4.8.

Tabel 4.8 Rata-rata Nilai F_2 -score

| Protokol | OCSVM | LOF | ISOF |
|----------|-------|-----|------|
|----------|-------|-----|------|

| | | | |
|-----------|---------|---------|---------|
| FTP | 0,66261 | 0,63081 | 0,5518 |
| SMTP | 0,18312 | 0,29138 | 0,87691 |
| HTTP | 0,3005 | 0,2702 | 0,5335 |
| Rata-Rata | 0,38208 | 0,39746 | 0,65407 |

Dari rata-rata nilai F_2 -score masing-masing algoritma pada Tabel 4.4, algoritma *isolation forest* memiliki nilai F_2 -score tertinggi dengan nilai 0,65407. Sedangkan pada algoritma *one-class SVM* dan *local outlier factor*, rata-rata nilai F_2 -score relatif sama pada kisaran 0,38-0,39. Algoritma *isolation forest* memiliki rata-rata performa terbaik dibandingkan dengan 2 algoritma yang lain. Perbedaan performa tersebut tampak jelas terjadi saat diuji pada protokol SMTP. Saat pengujian protokol SMTP, nilai F_2 -score algoritma *isolation forest* sebesar 0,87. Terdapat selisih yang sangat jauh apabila dibandingkan dengan nilai F_2 -score algoritma yang lain.

Selain rata-rata nilai F_2 -score, performa juga dilihat menggunakan rata-rata nilai DR. Detail rata-rata nilai DR dapat dilihat pada Tabel 4.9.

Tabel 4.9 Rata-rata Nilai DR

| Protokol | OCSVM | LOF | ISOF |
|-----------|---------|---------|---------|
| FTP | 0,79276 | 0,67395 | 1 |
| SMTP | 0,97276 | 0,34799 | 1 |
| HTTP | 0,2972 | 0,32963 | 0,72967 |
| Rata-Rata | 0,68757 | 0,45052 | 0,90989 |

Pada Tabel 4.5 tampak terlihat bahwa nilai DR pada algoritma *isolation forest* memiliki rata-rata tertinggi dibandingkan dengan 2 algoritma lainnya. Nilai terendah terjadi pada algoritma *local outlier factor* dengan nilai rata-rata DR sebesar 0,45. Semakin tinggi nilai DR menandakan semakin tinggi pula rasio *outlier* yang dapat terdeteksi. Dalam sistem NIDS, mendeteksi *outlier* merupakan tujuan utama dari sistem.

Performa masing-masing algoritma juga dinilai berdasarkan nilai FPR. Rata-rata nilai FPR tiap algoritma dapat dilihat pada Tabel 4.10.

Tabel 4.10 Rata-rata Nilai FPR

| Protokol | OCSVM | LOF | ISOF |
|-----------|---------|---------|---------|
| FTP | 0,02249 | 0,01238 | 0,07824 |
| SMTP | 0,37555 | 0,02539 | 0,0122 |
| HTTP | 0,01835 | 0,05248 | 0,05993 |
| Rata-Rata | 0,1388 | 0,03008 | 0,05012 |

Bila dilihat pada Tabel 4.6, tampak bahwa algoritma *local outlier factor* memiliki nilai FPR terendah dari ketiga algoritma. Sedangkan nilai tertinggi terjadi pada algoritma *one-class SVM*. Semakin tinggi nilai FPR menandakan semakin tinggi pula nilai *false positive* yang dihasilkan. Nilai FPR tertinggi terjadi pada algoritma *one-class SVM* dengan nilai 0,13. Tampak ada

kenaikan yang sangat menonjol pada hasil percobaan algoritma *one-class* SVM saat diuji dengan protokol SMTP.

Dilihat dari data rata-rata nilai F_2 -score dan DR seluruh hasil uji coba, algoritma *isolation forest* memiliki performa terbaik dengan nilai rata-rata F_2 -score sebesar 0,65 dan DR sebesar 0,9. Algoritma *one-class* SVM dan *local outlier factor* memiliki rata-rata nilai F_2 -score yang hampir sama dengan nilai 0,38 untuk *one-class* SVM dan 0,39 untuk *local outlier factor*. Dapat disimpulkan bahwa *isolation forest* memiliki rata-rata performa terbaik dibanding 2 algoritma lainnya.

BAB 5 Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan data dari hasil percobaan, terdapat beberapa kesimpulan yang telah didapatkan untuk menjawab rumusan masalah pada subbab 1.2. Berikut merupakan poin-poin kesimpulan yang telah didapatkan:

1. Salah satu cara melatih NIDS untuk dapat mengenali data dengan baik adalah dengan menggunakan algoritma *machine learning*. Algoritma *machine learning* yang digunakan pada penelitian ini adalah algoritma *unsupervised learning*. *Unsupervised learning* digunakan untuk mendeteksi data-data yang bersifat anomali pada data yang digunakan pada proses testing. Deteksi anomali dilakukan dengan melihat tingkat densitas persebaran data. Data pada area dengan tingkat densitas rendah diasumsikan sebagai data *outlier* atau data anomali. Salah satu keunggulan *unsupervised learning* adalah data latih yang digunakan dapat berupa data latih yang masih mengandung data kotor.
2. Terdapat 4 proses utama yang ada dalam sistem deteksi intrusi yang terbentuk, yaitu pra proses data, proses latih, proses testing, dan proses evaluasi. Pra proses data adalah proses ekstraksi data berformat PCAP menjadi sebuah *byte frequency*. Dari pra proses data didapatkan *byte frequency* masing-masing data latih dan data testing. Pada proses latih, data latih yang berbentuk *byte frequency* akan diproses lebih lanjut hingga terbentuk sebuah model *machine learning*. Model digunakan untuk memprediksi label data testing pada proses testing. Hasil dari prediksi dilakukan evaluasi untuk menilai performa masing-masing algoritma.
3. Hasil percobaan pada protokol FTP menunjukkan bahwa nilai F_2 -score tertinggi masing-masing algoritma terletak pada data latih dengan kadar data kotor sebesar 0%. Seiring meningkatnya kadar data kotor, tampak terlihat adanya penurunan pada nilai F_2 -score masing-masing algoritma. Pada algoritma *one-class SVM* dan *local outlier factor*, nilai F_2 -score terendah terjadi pada data latih dengan kadar data kotor sebesar 0,35%. Sedangkan pada algoritma *isolation forest*, nilai F_2 -score terendah terjadi pada data latih dengan kadar data kotor sebesar 0,05%. Dari data diatas dapat disimpulkan bahwa untuk percobaan FTP, data latih dengan kadar data kotor sebesar 0% memiliki nilai F_2 -score terbesar. Hal ini menandakan adanya penurunan performa setiap algoritma karena naiknya kadar data kotor pada data latih yang digunakan.
4. Hasil percobaan pada protokol SMTP memiliki performa yang berbeda pada tiap algoritma. Pada algoritma *local outlier factor*, nilai F_2 -score tertinggi terletak pada data dengan kadar data kotor 0%. Dengan adanya kenaikan kadar data kotor pada data latih yang digunakan, nilai F_2 -score algoritma *local outlier factor* tampak mengalami penurunan. Puncak penurunan terjadi pada data dengan kadar data kotor sebesar 0,6%. Sedangkan pada algoritma *one-class SVM* dan *isolation forest*, nilai F_2 -score relatif stabil untuk setiap data. Tidak ada efek yang signifikan pada nilai F_2 -score saat dilatih menggunakan data latih dengan kadar yang berbeda-beda. Untuk algoritma *isolation forest*, nilai F_2 -score relatif tinggi pada rentang 0,85-0,88. Sedangkan pada algoritma *one-class SVM*, nilai F_2 -score relatif rendah pada rentang 0,17-0,18. Dari hasil diatas dapat disimpulkan bahwa untuk percobaan pada protokol SMTP, kenaikan kadar data kotor berpengaruh pada performa algoritma *local outlier factor*. Sedangkan pada algoritma *one-class SVM* dan *isolation forest*, nilai F_2 -score relatif stabil tidak ada

perubahan yang signifikan meski kadar data kotor yang digunakan berbeda-beda.

5. Pada percobaan protokol HTTP, masing-masing algoritma memiliki performa yang berbeda-beda. Hasil percobaan algoritma *one-class SVM* dan *local outlier factor* menunjukkan bahwa nilai F_2 -score tertinggi terletak pada data latih dengan kadar data kotor sebesar 0%. Nilai F_2 -score mengalami penurunan seiring naiknya kadar data kotor. Sedangkan pada algoritma *isolation forest*, nilai F_2 -score tampak tidak stabil dan membentuk kurva bergelombang pada rentang nilai 0,40-0,63. Pada percobaan protokol HTTP dapat disimpulkan bahwa kenaikan kadar data kotor pada data latih mempengaruhi performa algoritma *one-class SVM* dan *isolation forest*. Sedangkan pada algoritma *isolation forest*, performa algoritma tampak tidak stabil.
6. Dilihat dari data rata-rata nilai F_2 -score seluruh hasil uji coba, algoritma *isolation forest* memiliki performa terbaik dengan nilai rata-rata F_2 -score sebesar 0,65. Algoritma *one-class SVM* dan *local outlier factor* memiliki rata-rata nilai F_2 -score yang hampir sama dengan nilai 0,38 untuk *one-class SVM* dan 0,39 untuk *local outlier factor*.

5.2 Saran

Saran didapatkan dari hasil analisa kekurangan sistem dan potensi pengembangan sistem kedepannya. Berikut merupakan saran dalam improvisasi sistem untuk penelitian mendatang.:

1. Melakukan pengujian secara mendalam terhadap parameter yang digunakan pada masing-masing algoritma. Hal ini memungkinkan untuk mendapatkan nilai parameter yang maksimal untuk digunakan pada setiap algoritma.
2. Menggunakan dataset yang lebih bervariasi. Untuk mendapatkan data yang lebih akurat diperlukan dataset yang lebih beragam dengan jumlah yang lumayan banyak.
3. Uji coba dengan pembandingan fitur lain. Dalam uji coba kali ini fitur pembandingnya adalah dengan menggunakan F_2 -score.
4. Implementasi sistem NIDS secara langsung terhadap perangkat komputer.
5. Melakukan simulasi serangan dan deteksi serangan secara real time.

DAFTAR PUSTAKA

- [1] “Rekap Serangan Siber (Januari – April 2020)” Badan Siber dan Sandi Negara, 20 April 2020. [Online] Available: <https://bssn.go.id/rekap-serangan-siber-januari-april-2020/> [Accessed 20 Juni 2021]
- [2] Leonid Portnoy, “Intrusion detection with unlabeled data using clustering” Departemen of Computer Science Columbia University, 2001
- [3] Rowayda, A. Sadek; M Sami, Soliman; Hagar, S Elsayed. "Effective anomaly intrusion detection system based on neural network with indicator variable and rough set reduction". International Journal of Computer Science Issues (IJCSI), 2013.
- [4] “Python Language advantages and applications” GeeksforGeeks, 30 Juni 2021. [Online] Available: <https://www.geeksforgeeks.org/python-language-advantages-applications/> [Accessed 22 Juni 2021]
- [5] Mitchell, Tom (1997). “Machine Learning”. New York: McGraw Hill. ISBN 0-07-042807-7. OCLC 36417892.
- [6] “Unsupervised Machine Learning: What is, Algorithms, Example” Guru99. [Online] Available: <https://www.guru99.com/unsupervised-machine-learning.html> [Accessed 25 Juni 2021]
- [7] “Getting started scikit-learn” Scikit-learn. [Online] Available: https://scikit-learn.org/stable/getting_started.html [Accessed 25 Juni 2021]
- [8] Bangun, Chandra Sampe, “Uji Perbandingan Sistem Deteksi Intrusi Berdasarkan Sumber Data Header dan Payload” Program Studi Teknik Informatika FTI-UKSW, 2013
- [9] “PCAP Packet Capture Analysis” IDNS Stuff, 24 Februari 2021. [Online] Available: <https://www.dnsstuff.com/pcap-analysis> [Accessed 30 Juni 2021]
- [10] “Pcap python extension module github” IDNS Stuff, 3 Juli 2019. [Online] Available: <https://github.com/helpsystems/pcapy> [Accessed 1 Juli 2021]
- [11] Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.
- [12] “2.7 Novelty and Outlier Detection” Scikit-learn. [Online] Available: https://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection [Accessed 1 Juni 2022]
- [13] Pratomo, B.A., Burnap, P. & Theodorakopoulos, G. “Unsupervised Approach for Detecting Low Rate Attacks on Network Traffic with Autoencoder” 2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2018, IEEE, 2018

BIODATA PENULIS



Penulis dilahirkan di Banyuwangi, 3 Februari 1999, merupakan anak pertama dari 4 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Dharma Wanita Tembokrejo Muncar, SDN 1 Jajag, SMPN 2 Gambiran dan SMAN 1 Genteng. Setelah lulus dari SMAN tahun 2017, Penulis mengikuti SNMPTN dan diterima di Departemen Teknik Informatika FTEIC - ITS pada tahun 2017 dan terdaftar dengan NRP 05111740000048.

Di Departemen Teknik Informatika Penulis sempat aktif di beberapa kegiatan Seminar yang diselenggarakan oleh Departemen dan staff kegiatan SCHEMATICS 2018-2019.

Penulis dapat dihubungi melalui email krisnabadru1@gmail.com