IF3170 Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Tugas Besar 2
Disusun untuk memenuhi tugas mata kuliah Inteligensi Artifisial
pada Semester 1 (satu) Tahun Akademik 2024/2025

Disusun oleh:

Yasmin Farisah Salma	**(13522140)**
Ikhwan Al Hakim	**(13522147)**
Axel Santadi Warih	**(13522155)**
Mohammad Akmal Ramadan	**(13522161)**

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

DAFTAR ISI 2
BAB I
DESKRIPSI MASALAH 3
BAB II
PEMBAHASAN 4
2.1. Implementasi Algoritma KNN 4
2.1.1. Metode Fit 4
2.1.2. Metode _get_nearest_neighbours 4
2.1.3. Metode _predict_instance 4
2.1.4. Metode predict 4
2.2. Implementasi Algoritma Naive-Bayes 5
2.2.1. Metode Fit 5
2.2.2. Menghitung Probabilitas Kelas (_calculate_class_probabilities) 5
2.2.3. Menghitung Mean dan Varians Setiap Fitur per Kelas 6
2.2.4. Menghitung Probabilitas Gaussian untuk Fitur 6
2.2.5. Menghitung Probabilitas Kelas Berdasarkan Fitur-Fitur 6
2.2.6. Prediksi Kelas (predict) 7
2.3. Implementasi Algoritma ID3 7
2.3.1. Fungsi Entropi 8
2.3.2. Fungsi _split 8
2.3.3. Fungsi _select_features_to_use 8
2.3.4. Fungsi _find_best_split 8
2.3.5. Fungsi _create_tree 9
2.3.6. Fungsi train 9
2.3.7. Fungsi Prediksi 10
2.4. Tahapan Data Cleaning dan Preprocessing 11
BAB III
HASIL DAN ANALISIS 16
3.1. KNN 16
3.2. Naive Bayes 17
BAB IV
KESIMPULAN DAN SARAN 20
4.1. Kesimpulan 20
4.2. Saran 20
BAB V
LAMPIRAN 21

BAB I
DESKRIPSI MASALAH

Pembelajaran mesin adalah cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data guna membuat prediksi atau keputusan tanpa memerlukan pengkodean eksplisit. Tugas ini mengharuskan Anda untuk mengimplementasikan algoritma pembelajaran mesin menggunakan dataset UNSW-NB15, yang berisi data lalu lintas jaringan, mencakup berbagai jenis serangan siber serta aktivitas normal.

Tugas besar ini melibatkan implementasi tiga algoritma dari awal, yaitu K-Nearest Neighbors (KNN), Gaussian Naive-Bayes, dan ID3 (Iterative Dichotomiser 3), dengan spesifikasi berikut:

- **KNN from scratch:** Algoritma ini menerima minimal dua parameter, yakni jumlah tetangga dan metrik jarak. Mendukung tiga jenis metrik jarak: Euclidean, Manhattan, dan Minkowski.
- **Gaussian Naive-Bayes** from scratch.
- **ID3** from scratch, termasuk pengolahan data numerik sesuai materi kuliah.

Selain implementasi manual, algoritma-algoritma tersebut juga harus diimplementasikan menggunakan pustaka scikit-learn. Untuk ID3 dalam scikit-learn, Anda akan menggunakan DecisionTreeClassifier dengan parameter criterion='entropy', yang memiliki kemiripan dengan ID3. Hasil implementasi dari kedua versi (from scratch dan scikit-learn) akan dibandingkan.

Setiap model yang dibuat harus dapat disimpan dan dimuat kembali dalam format tertentu (misalnya, .txt atau .pkl). Hasil prediksi juga dapat dikirimkan ke platform Kaggle untuk evaluasi, dengan bonus tambahan diberikan berdasarkan peringkat leaderboard.

Langkah-langkah dalam menyelesaikan tugas ini meliputi:

1. **Pembersihan Data:** Membersihkan data dari nilai hilang, duplikasi, atau entri tidak valid.
2. **Transformasi Data:** Meliputi proses seperti encoding variabel kategori, normalisasi atau standarisasi fitur numerik, dan penanganan ketidakseimbangan data.
3. **Pemilihan Fitur:** Mengidentifikasi fitur yang relevan untuk meningkatkan kinerja model.
4. **Reduksi Dimensi:** Mengurangi jumlah fitur tanpa kehilangan informasi penting, misalnya menggunakan PCA.
5. **Modeling dan Validasi:** Melatih dan memvalidasi model untuk mengklasifikasikan kategori serangan (attack_cat) menggunakan metode validasi seperti train-test split atau k-fold cross-validation.

Notebook yang dikumpulkan harus dapat menghasilkan hasil prediksi yang konsisten dengan submisi di Kaggle, di mana hanya model KNN, Naive Bayes, dan ID3 hasil implementasi dari awal yang diperbolehkan digunakan.

BAB II
PEMBAHASAN

2.1. Implementasi Algoritma KNN

Algoritma K-Nearest Neighbors yang diimplementasikan dalam kode kami ini adalah algoritma klasifikasi berbasis instance. Algoritma ini bekerja dengan mencari sejumlah *K* tetangga terdekat dari data yang diuji berdasarkan metrik data tertentu, seperti Minkowski, Euclidean, atau Manhattan, kemudian menentukan label kelas berdasarkan tetangga tersebut.

KNN memiliki keunggulan pada bidang kesederhanaan dan keefektifan dalam berbagai masalah klasifikasi. Namun karena bersifat *lazy learning,* maka KNN menyimpan seluruh data latih sehingga membutuhkan waktu agak lama untuk memproses prediksi.

2.1.1. Metode Fit

```
def fit(self, X_train, y_train):
    if isinstance(X_train, pd.DataFrame):
        self.X_train = X_train.values.astype(float)
    else:
        self.X_train = X_train.astype(float)

    self.y_train = y_train
```

Metode fit bertanggung jawab untuk menyimpan data latih (X_train dan y_train) yang akan digunakan dalam proses prediksi. Metode ini mendukung input berupa Pandas DataFrame atau NumPy array, yang kemudian dikonversi menjadi array numerik untuk memastikan kompatibilitas dalam perhitungan jarak.

2.1.2. Metode _get_nearest_neighbours

```
distances = np.linalg.norm(self.X_train - test, ord=self.p, axis=1)
indices = np.argsort(distances)[:self.k]
```

Metode ini menghitung jarak antara data uji dengan data latih menggunakan metrik Minkowski. Jika parameter weights diatur sebagai 'uniform', semua tetangga memiliki bobot yang sama. Jika diatur sebagai 'distance', bobot dihitung berdasarkan kebalikan jarak, sehingga tetangga yang lebih dekat memiliki pengaruh lebih besar. Hasil akhirnya adalah indeks tetangga terdekat serta bobot masing-masing tetangga (jika menggunakan bobot berbasis jarak).

2.1.3. Metode _predict_instance

```
labels = [self.y_train.iloc[neighbour] for neighbour in indices]
prediction = max(set(labels), key=labels.count)
```

Metode ini bertanggung jawab untuk memprediksi label kelas untuk satu instance data uji. Jika bobot adalah 'uniform', prediksi dilakukan dengan memilih kelas yang paling sering muncul di antara tetangga. Jika bobot adalah 'distance', prediksi menggunakan rata-rata berbobot berdasarkan jarak tetangga.

2.1.4. Metode predict

```
with concurrent.futures.ProcessPoolExecutor(max_workers=self.n_jobs) as executor:
    results = list(executor.map(self._predict_instance, X_test))
```

Metode ini digunakan untuk memprediksi label kelas untuk seluruh dataset uji (X_test). Metode ini mendukung eksekusi paralel menggunakan ProcessPoolExecutor untuk meningkatkan efisiensi pada dataset besar. Jika verbose=True, proses prediksi akan menampilkan kemajuan serta waktu yang dibutuhkan. Metode save dan load:

- save(path): Metode ini menyimpan model yang telah dilatih ke dalam file menggunakan pustaka pickle.
- load(path): Metode ini memuat kembali model yang telah disimpan untuk digunakan di masa depan.

Implementasi algoritma KNN ini menawarkan fleksibilitas tinggi dengan parameter yang dapat disesuaikan, seperti pilihan metrik jarak, jumlah tetangga, dan metode pembobotan. Namun, karena sifatnya yang berbasis instance, model ini membutuhkan pengoptimalan tambahan, terutama pada dataset berskala besar, seperti penggunaan teknik reduksi dimensi atau indeksasi data untuk mempercepat perhitungan jarak.

2.2. Implementasi Algoritma Naive-Bayes

Algoritma Naive Bayes yang diimplementasikan dalam kode di atas merupakan klasifikasi berbasis probabilitas yang digunakan untuk mengklasifikasikan data berdasarkan fitur-fitur yang ada. Naive Bayes sangat berguna terutama ketika data memiliki distribusi yang dapat dianggap Gaussian (normal) dan dapat diterapkan pada berbagai macam masalah klasifikasi seperti pengenalan pola, deteksi spam, dan lainnya.

2.2.1. Metode Fit

```
def fit(self, X, y):
    if self.verbose:
        print("Fitting model...")
    self.class_probabilities = self._calculate_class_probabilities(y)
    self.mean, self.variance = self._calculate_mean_and_variance(X, y)
    if self.verbose:
        print("Model fitting complete.")
```

fit adalah metode untuk melatih model Naive Bayes. Metode ini menerima dua parameter:

- X: Data fitur (biasanya matriks data).
- y: Label kelas (output yang ingin diprediksi).

Dalam metode ini, pertama-tama dihitung probabilitas kelas menggunakan metode _calculate_class_probabilities, kemudian dihitung rata-rata dan varians untuk setiap fitur dalam setiap kelas menggunakan metode _calculate_mean_and_variance.

2.2.2. Menghitung Probabilitas Kelas (_calculate_class_probabilities)

```
def _calculate_class_probabilities(self, y):
    if self.verbose:
        print("Calculating class probabilities...")
```

```
    class_counts = defaultdict(int)
    total_samples = len(y)

    for label in y:
        class_counts[label] += 1

    class_probabilities = {label: count / total_samples for label, count in class_counts.items()}
    if self.verbose:
        print(f"Class probabilities: {class_probabilities}")
    return class_probabilities
```

Metode ini menghitung probabilitas masing-masing kelas berdasarkan frekuensi relatif dalam dataset pelatihan. Misalnya, jika ada 3 kelas dan kelas pertama muncul 30 kali dari total 100 sampel, maka probabilitas kelas pertama adalah 0.3.

2.2.3. Menghitung Mean dan Varians Setiap Fitur per Kelas

```
def _calculate_mean_and_variance(self, X, y):
    if self.verbose:
        print("Calculating mean and variance for each class...")
    unique_classes = np.unique(y)
    mean = {}
    variance = {}

    for label in unique_classes:
        class_data = X[y == label]
        mean[label] = np.mean(class_data, axis=0)
        variance[label] = np.var(class_data, axis=0)
        if self.verbose:
            print(f"Class {label}: Mean = {mean[label]}, Variance = {variance[label]}")

    return mean, variance
```

Di sini, untuk setiap kelas yang unik, dilakukan perhitungan rata-rata (*mean*) dan varians (*variance*) dari setiap fitur. Hal ini digunakan karena Naive Bayes mengasumsikan bahwa data untuk setiap kelas mengikuti distribusi Gaussian (normal), yang sepenuhnya ditentukan oleh rata-rata dan variansnya.

2.2.4. Menghitung Probabilitas Gaussian untuk Fitur

```
def _gaussian_probability(self, x, mean, variance):
    exponent = np.exp(-((x - mean) ** 2) / (2 * variance))
    return (1 / np.sqrt(2 * np.pi * variance)) * exponent
```

Fungsi ini menghitung probabilitas Gaussian untuk sebuah nilai fitur x yang diberikan rata-rata (mean) dan varians (variance). Rumus ini berasal dari distribusi normal (Gaussian):

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{1}{2}(\frac{(x-\mu)}{\sigma})^2)$$

Dimana:

- P(x): fungsi kepadatan probabilitas
- σ: simpangan baku
- μ: rata-rata

2.2.5. Menghitung Probabilitas Kelas Berdasarkan Fitur-Fitur

```
def _calculate_class_likelihood(self, features, label):
    if self.verbose:
        print(f"Calculating likelihood for class {label}...")
    log_class_probability = np.log(self.class_probabilities[label])

    for i, feature in enumerate(features):
        mean = self.mean[label][i]
        variance = self.variance[label][i]
        epsilon = 1e-10
        if variance < epsilon:
            variance = epsilon

        log_class_probability += np.log(self._gaussian_probability(feature, mean, variance))

    if self.verbose:
        print(f"Class {label} likelihood: {log_class_probability}")
    return log_class_probability
```

Untuk setiap kelas, probabilitas diberikan fitur dihitung berdasarkan distribusi Gaussian untuk setiap fitur. Karena dalam Naive Bayes kita mengasumsikan bahwa fitur-fitur adalah independen (ini adalah asumsi "naive"), maka kita menjumlahkan logaritma dari probabilitas fitur-fitur individual. Penanganan untuk nilai varians yang sangat kecil (epsilon = 1e-10) dilakukan untuk menghindari pembagian dengan nol.

2.2.6. Prediksi Kelas (predict)

```
def predict(self, X):
    if self.verbose:
        print("Making predictions...")
    predictions = []

    for sample in X.values:
        class_likelihoods = {
            label: self._calculate_class_likelihood(sample, label)
            for label in self.class_probabilities
        }

        predicted_class = max(class_likelihoods, key=class_likelihoods.get)
        if self.verbose:
            print(f"Sample: {sample}, Predicted class: {predicted_class}")
        predictions.append(predicted_class)

    if self.verbose:
        print("Prediction complete.")
    return np.array(predictions)
```

Fungsi predict mengiterasi setiap sampel dalam data uji X dan menghitung probabilitas untuk setiap kelas menggunakan fitur-fitur yang ada. Kelas dengan probabilitas tertinggi dipilih sebagai prediksi untuk sampel tersebut. Fungsi max digunakan untuk memilih kelas dengan nilai probabilitas tertinggi.

2.3. Implementasi Algoritma ID3

Algoritma Iterative Dichotomiser 3 (ID3) yang diimplementasikan dalam kode ini adalah algoritma pembelajaran berbasis pohon keputusan. ID3 membangun pohon keputusan dengan

mempartisi dataset berdasarkan fitur, menggunakan information gain sebagai metrik utama untuk menentukan pemisahan terbaik.

2.3.1. Fungsi Entropi

```
def _entropy(self, class_probabilities: list) -> float:
    return sum([-p * np.log2(p) for p in class_probabilities if p>0])

def _data_entropy(self, labels: list) -> float:
    return self._entropy(self._class_probabilities(labels))

def _partition_entropy(self, subsets: list) -> float:
    """subsets = list of label lists (EX: [[1,0,0], [1,1,1])"""
    total_count = sum([len(subset) for subset in subsets])
    return sum([self._data_entropy(subset) * (len(subset) / total_count) for subset in subsets])
```

Fungsi _entropy menghitung ketidakpastian dalam distribusi kelas, sementara fungsi _data_entropy dan _partition_entropy menghitung entropi data secara keseluruhan atau setelah partisi.

2.3.2. Fungsi _split

```
def _split(self, data: np.array, feature_idx: int, feature_val: float) -> tuple:

    mask_below_threshold = data[:, feature_idx] < feature_val
    group1 = data[mask_below_threshold]
    group2 = data[~mask_below_threshold]

    return group1, group2
```

Fungsi _split membagi data berdasarkan nilai fitur tertentu, menghasilkan dua subset data.

2.3.3. Fungsi _select_features_to_use

```
def _select_features_to_use(self, data: np.array) -> list:
    """ Randomly selects the features to use while splitting w.r.t. hyperparameter numb_of_features_splitting"""
    feature_idx = list(range(data.shape[1]-1))

    if self.numb_of_features_splitting == "sqrt":
        feature_idx_to_use = np.random.choice(feature_idx, size=int(np.sqrt(len(feature_idx))))
    elif self.numb_of_features_splitting == "log":
        feature_idx_to_use = np.random.choice(feature_idx, size=int(np.log2(len(feature_idx))))
    else:
        feature_idx_to_use = feature_idx

    return feature_idx_to_use
```

Fungsi _select_features_to_use menentukan fitur mana yang akan digunakan, mendukung metode pemilihan berbasis sqrt atau log2 untuk mengurangi dimensi data secara acak.

2.3.4. Fungsi _find_best_split

```
def _find_best_split(self, data: np.array) -> tuple:
    """Finds the best split (with the lowest entropy) given data Returns 2 splitted groups and split information
"""
    min_part_entropy = 1e9
    feature_idx_to_use =  self._select_features_to_use(data)
```

```
    for idx in feature_idx_to_use:
        feature_vals = np.percentile(data[:, idx], q=np.arange(25, 100, 25))
        for feature_val in feature_vals:
            g1, g2, = self._split(data, idx, feature_val)
            part_entropy = self._partition_entropy([g1[:, -1], g2[:, -1]])
            if part_entropy < min_part_entropy:
                min_part_entropy = part_entropy
                min_entropy_feature_idx = idx
                min_entropy_feature_val = feature_val
                g1_min, g2_min = g1, g2

    return g1_min, g2_min, min_entropy_feature_idx, min_entropy_feature_val, min_part_entropy
```

Fungsi _find_best_split mengevaluasi berbagai nilai pemisahan untuk setiap fitur, mencari kombinasi dengan entropi partisi terendah.

2.3.5. Fungsi _create_tree

```
def _create_tree(self, data: np.array, current_depth: int) -> TreeNode:
    """Recursive, depth first tree creation algorithm"""
    # Check if the max depth has been reached (stopping criteria)
    if current_depth > self.max_depth:
        return None

    # Find best split
    split_1_data, split_2_data, split_feature_idx, split_feature_val, split_entropy = self._find_best_split(data)

    # Find label probs for the node
    label_probabilities = self._find_label_probs(data)

    # Calculate information gain
    node_entropy = self._entropy(label_probabilities)
    information_gain = node_entropy - split_entropy

    # Create node
    node = TreeNode(data, split_feature_idx, split_feature_val, label_probabilities, information_gain)

    if self.min_samples_leaf > split_1_data.shape[0] or self.min_samples_leaf > split_2_data.shape[0]:
        return node
    elif information_gain < self.min_information_gain:
        return node

    current_depth += 1
    node.left = self._create_tree(split_1_data, current_depth)
    node.right = self._create_tree(split_2_data, current_depth)

    return node
```

Proses utama pembuatan pohon dilakukan secara rekursif oleh fungsi _create_tree. Berikut adalah langkah-langkahnya:

1. Memeriksa Kondisi Berhenti: Fungsi akan berhenti jika kedalaman maksimum tercapai atau jika data di simpul terlalu sedikit untuk dipisah.
2. Mencari Pemisahan Terbaik: Menggunakan entropi untuk menemukan fitur dan nilai pemisahan optimal.
3. Membuat Simpul: Membuat simpul berdasarkan hasil pemisahan, dan melanjutkan proses secara rekursif untuk simpul anak.

2.3.6. Fungsi train

```
def train(self, X_train: np.array, Y_train: np.array) -> None:
    """Trains the model with given X and Y datasets"""

    self.labels_in_train = np.unique(Y_train)
    train_data = np.concatenate((X_train, np.reshape(Y_train, (-1, 1))), axis=1)

    self.tree = self._create_tree(data=train_data, current_depth=0)

    self.feature_importances = dict.fromkeys(range(X_train.shape[1]), 0)
    self._calculate_feature_importance(self.tree)
    self.feature_importances = {k: v / total for total in (sum(self.feature_importances.values()),) for k, v in
self.feature_importances.items()}
```

Fungsi train memulai proses pembuatan pohon keputusan:

1. Data fitur dan label digabungkan menjadi satu matriks.
2. Pohon dibangun menggunakan _create_tree.
3. Menghitung feature importance secara rekursif dengan fungsi _calculate_feature_importance.

2.3.7. Fungsi Prediksi

```
def predict_proba(self, X_set: np.array) -> np.array:
    """Returns the predicted probs for a given data set"""

    pred_probs = np.apply_along_axis(self._predict_one_sample, 1, X_set)

    return pred_probs

def predict(self, X_set: np.array) -> np.array:
    """Returns the predicted labels for a given data set"""

    pred_probs = self.predict_proba(X_set)
    preds = np.argmax(pred_probs, axis=1)

    return preds

def _predict_one_sample(self, X: np.array) -> np.array:
    """Returns prediction for 1 dim array"""
    node = self.tree

    while node:
        pred_probs = node.prediction_probs
        if X[node.feature_idx] < node.feature_val:
            node = node.left
        else:
            node = node.right

    return pred_probs
```

- predict_proba: Mengembalikan probabilitas prediksi untuk setiap label.
- predict: Mengembalikan label prediksi berdasarkan probabilitas maksimum.
- _predict_one_sample: Fungsi internal untuk memprediksi satu data berdasarkan navigasi dari akar ke simpul daun.

Implementasi algoritma ID3 ini dirancang untuk fleksibilitas dan interpretabilitas tinggi. Beberapa poin penting:

- Kekuatan: Sederhana dan efektif untuk dataset kecil hingga menengah. Probabilitas prediksi memberikan wawasan lebih lanjut tentang ketidakpastian model.
- Keterbatasan: Rentan terhadap overfitting jika pohon terlalu dalam. Tidak optimal untuk dataset besar tanpa reduksi dimensi atau pruning.

2.4. Tahapan Data Cleaning dan Preprocessing

a) Data Cleaning

1. Handling Missing Data

```
missing_data = train_data.isnull().sum().to_frame().rename(columns={0:"Total No. of Missing Values"})
missing_data["% of Missing Values"] = round((missing_data["Total No. of Missing Values"]/len(train_data))*100,2)
missing_data
```

	Total No. of Missing Values	% of Missing Values
swin	0	0.0
dwin	0	0.0
stcpb	0	0.0
dtcpb	0	0.0
smean	0	0.0
dmean	0	0.0
trans_depth	0	0.0
response_body_len	0	0.0
id	0	0.0
sjit	0	0.0
djit	0	0.0
sinpkt	0	0.0
dinpkt	0	0.0
tcprtt	0	0.0
synack	0	0.0
ackdat	0	0.0
proto	0	0.0
state	0	0.0
dur	0	0.0
sbytes	0	0.0
dbytes	0	0.0
sttl	0	0.0
dttl	0	0.0

dttl	0	0.0
sloss	0	0.0
dloss	0	0.0
service	0	0.0
sload	0	0.0
dload	0	0.0
spkts	0	0.0
dpkts	0	0.0
is_sm_ips_ports	0	0.0
ct_state_ttl	0	0.0
ct_flw_http_mthd	0	0.0
is_ftp_login	0	0.0
ct_ftp_cmd	0	0.0
ct_srv_src	0	0.0
ct_srv_dst	0	0.0
ct_dst_ltm	0	0.0
ct_src_ltm	0	0.0
ct_src_dport_ltm	0	0.0
ct_dst_sport_ltm	0	0.0
ct_dst_src_ltm	0	0.0
attack_cat	0	0.0
label	0	0.0

2. Dealing with Outliers

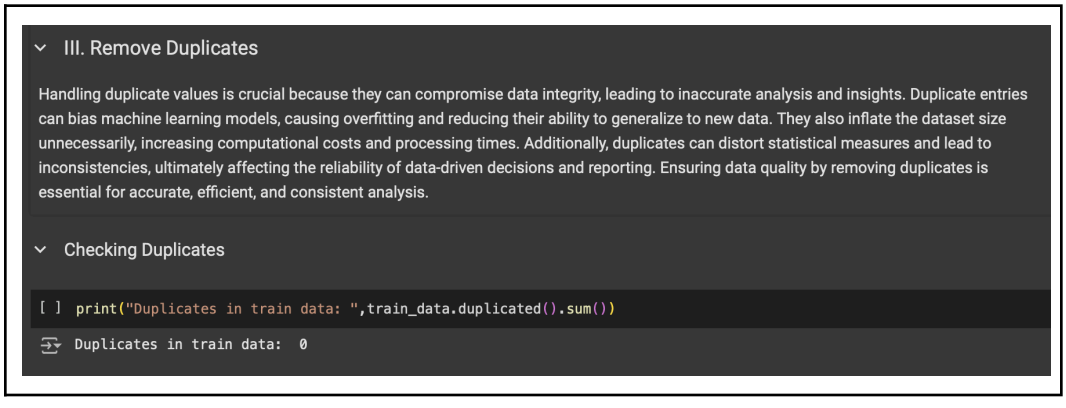
Dalam analisis data, penanganan outliers adalah langkah yang sangat penting karena outliers dapat memengaruhi kinerja model dan akurasi prediksi yang dihasilkan. Berdasarkan kode yang kami terapkan, langkah pertama yang kami lakukan adalah memilih kolom-kolom numerik dalam dataset train_data. Kami memilih kolom dengan tipe data float64 dan int64 untuk memastikan hanya kolom yang relevan yang diproses lebih lanjut. Selanjutnya, kami melakukan visualisasi distribusi data dengan

menggunakan boxplot untuk setiap kolom numerik. Boxplot ini memungkinkan kami untuk dengan mudah mengidentifikasi nilai-nilai yang jauh berbeda dari mayoritas data, yang seringkali dianggap sebagai outliers. Nilai yang terletak di luar rentang whiskers pada boxplot menandakan adanya data yang mungkin perlu ditangani.

Setelah kami mendeteksi adanya outliers, kami memutuskan untuk menyaring kolom-kolom yang tidak relevan dalam analisis, seperti kolom-kolom 'swim', 'dwim', 'stcpb', 'dtcpb', 'sttl', 'dttl'. Penyaringan ini kami lakukan untuk memastikan bahwa hanya kolom-kolom yang benar-benar penting yang dianalisis, sehingga proses analisis menjadi lebih efisien dan hasil yang diperoleh lebih akurat. Dengan mengecualikan kolom yang tidak diperlukan, kami dapat fokus pada data yang lebih relevan untuk pengolahan lebih lanjut.

Untuk menangani outliers yang kami temui, kami menggunakan metode penggantian nilai ekstrem dengan nilai median kolom tersebut. Kami menghitung nilai batas bawah dan batas atas menggunakan rumus Interquartile Range (IQR), yaitu selisih antara kuartil pertama (Q1) dan kuartil ketiga (Q3). Jika nilai data berada di luar rentang ini, kami anggap sebagai outliers dan menggantinya dengan nilai median kolom tersebut. Pendekatan ini kami pilih karena median lebih robust terhadap outliers dibandingkan rata-rata, dan penggantian ini akan mengurangi pengaruh nilai ekstrem yang dapat merusak model. Selain itu, kami juga melakukan visualisasi distribusi data dan menghitung skewness (kemiringan distribusi) untuk setiap kolom numerik. Kami menggunakan histogram dan plot kepadatan (KDE) untuk memvisualisasikan bagaimana distribusi data setelah penanganan outliers. Jika distribusi data menunjukkan skewness yang signifikan, kami menerapkan transformasi logaritma atau metode lain untuk memperbaiki distribusi data. Transformasi ini penting untuk membuat data lebih simetris, sehingga model dapat memprosesnya dengan lebih baik dan menghasilkan prediksi yang lebih akurat. Terakhir, kami menggunakan fungsi transformasi untuk menangani skewness lebih lanjut pada data. Kolom dengan skewness positif kami transformasi menggunakan logaritma, sementara kolom dengan skewness negatif kami transformasi dengan logaritma dari perbedaan antara nilai maksimum dan data tersebut. Langkah ini kami lakukan untuk mengurangi dampak skewness yang terlalu besar, sehingga distribusi data menjadi lebih seimbang dan dapat diproses lebih efektif oleh model.

3. Removing Duplicates



4. Feature Engineering

   Feature engineering adalah proses untuk menciptakan atau mengubah fitur data guna meningkatkan kinerja model machine learning. Salah satu teknik utama adalah seleksi fitur, yaitu memilih fitur yang paling relevan dan menghilangkan fitur yang tidak penting, yang membantu mencegah overfitting. Selain itu, pembuatan fitur baru seperti fitur polinomial, interaksi antar fitur, dan binning, dapat membantu model menangkap pola lebih baik. Kami juga menciptakan fitur rasio dan fitur agregat untuk menggambarkan hubungan antar variabel, serta fitur interaksi untuk menggabungkan beberapa fitur. Fitur statistik, seperti mean_pkt_size, juga ditambahkan untuk memberikan informasi tambahan. Setelah menambahkan fitur-fitur baru, kami menghapus kolom yang tidak relevan, seperti proto, state, dan service.

b) Data Preprocessing

1. Feature Encoding

   Feature encoding adalah proses mengubah data kategorikal menjadi format numerik agar dapat digunakan oleh algoritma machine learning. Data kategorikal terbagi menjadi dua jenis: nominal, yang tidak memiliki urutan (seperti warna atau nama negara), dan ordinal, yang memiliki urutan yang bermakna (seperti tingkat pendidikan). Ada beberapa metode encoding yang umum digunakan, antara lain Label Encoding, yang mengonversi setiap kategori menjadi angka unik, cocok untuk data ordinal. One-Hot Encoding membuat kolom biner untuk setiap kategori dalam data nominal, menandakan kehadiran atau ketidakhadiran kategori dengan nilai 1 atau 0. Sedangkan Target Encoding mengganti kategori dengan rata-rata variabel target untuk setiap kategori, yang berguna dalam masalah klasifikasi. Pada kasus ini, kami menggunakan Label Encoding untuk mengonversi kolom attack_cat menjadi nilai numerik. Setiap kategori dalam kolom tersebut, seperti 'Analysis' dan 'Backdoor', diberi nilai numerik yang sesuai (misalnya, 0 untuk 'Analysis' dan 1 untuk 'Backdoor'). Proses ini memungkinkan data kategorikal diproses oleh model machine learning dengan cara yang lebih efisien dan tepat.

2. Feature Scaling

   Kami menggunakan StandardScaler untuk melakukan standardisasi pada dataset. Fungsi fit_transform digunakan untuk menyesuaikan data dengan standar yang ditentukan (rata-rata 0 dan deviasi standar 1).

3. Handling Imbalanced Classes

Pada tahap ini, kami melakukan beberapa langkah untuk menangani dataset yang tidak seimbang dan fitur yang memiliki korelasi tinggi, serta memastikan kualitas data sebelum melanjutkan ke pelatihan model.

Dataset yang tidak seimbang dapat menyebabkan model menjadi bias terhadap kelas mayoritas, sehingga kinerja pada kelas minoritas terganggu. Untuk mengatasi hal ini, dilakukan teknik resampling menggunakan SMOTE (Synthetic Minority Over-sampling Technique) untuk menambah jumlah sampel pada kelas minoritas dan RandomUnderSampler untuk mengurangi sampel pada kelas mayoritas. Hasilnya, distribusi kelas menjadi seimbang, dengan setiap kelas memiliki 15.000 sampel setelah resampling, memastikan bahwa model tidak cenderung memprediksi hanya kelas mayoritas. Fitur yang memiliki korelasi tinggi dapat menyebabkan multikolinearitas, yang mempengaruhi kinerja model. Dengan menggunakan matriks korelasi, fitur yang memiliki koefisien korelasi lebih besar dari 0,75 diidentifikasi dan salah satu fitur dari setiap pasangan fitur yang sangat berkorelasi dihapus. Proses ini mengurangi redundansi data dan membantu model untuk belajar dengan lebih efisien. Selanjutnya, dilakukan analisis mutual information untuk mengukur sejauh mana setiap fitur berhubungan dengan target variabel (attack_cat). Fitur yang memiliki nilai mutual information rendah (di bawah ambang batas yang ditentukan) dihapus, karena dianggap kurang berkontribusi pada prediksi. Hasil analisis menunjukkan bahwa fitur-fitur seperti ct_flw_http_mthd, response_body_len, dan is_ftp_login memiliki nilai mutual information yang sangat rendah, sehingga dihapus untuk meningkatkan kinerja model.

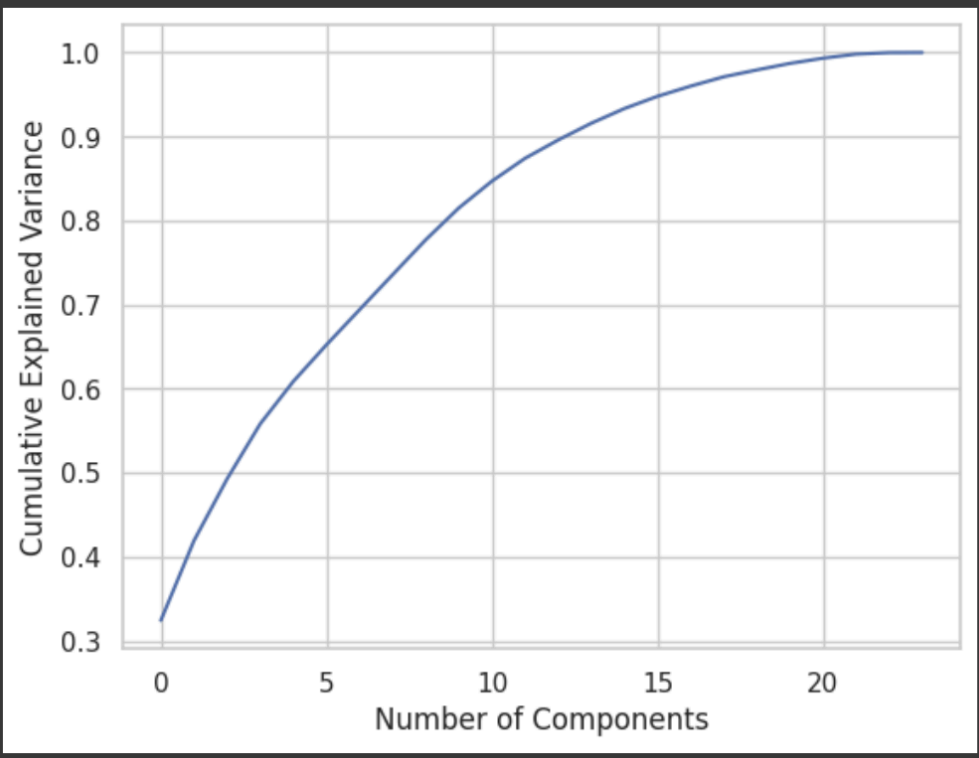
4. Dimensionality Reduction

Kami menerapkan Principal Component Analysis (PCA) untuk mengurangi dimensi data yang telah diskalakan (x_scaled). Langkah pertama adalah memfitting model PCA pada data menggunakan pca.fit(x_scaled), yang mengidentifikasi komponen utama yang menjelaskan variansi terbesar dalam data. Selanjutnya, explained_variance_ratio dihitung untuk setiap komponen utama, yang menunjukkan seberapa besar kontribusi setiap komponen dalam menjelaskan variansi data. Kemudian, kumulatif dari rasio variansi ini dihitung dengan cumsum() untuk melihat total variansi yang dijelaskan oleh sejumlah komponen utama. Hasilnya digambarkan dalam sebuah grafik yang memplot jumlah komponen utama di sumbu x dan variansi kumulatif yang dijelaskan di sumbu y. Grafik ini membantu dalam menentukan jumlah komponen utama

yang diperlukan untuk mempertahankan sebagian besar informasi penting dari data, sehingga memungkinkan pengurangan dimensi tanpa kehilangan informasi yang signifikan.

```
pca = PCA()
pca.fit(x_scaled)
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = explained_variance_ratio.cumsum()

plt.plot(cumulative_variance_ratio)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```



BAB III
HASIL DAN ANALISIS

3.1.KNN

3.2.1. KNN dengan Library

```
Accuracy: 0.7882
Confusion Matrix:
 [[   40    14   131   180     3     0    31     1     0     0]
 [   12    26   127   156     6     0     1    19     2     0]
 [   37    30   797  1468    40     8     8    56     9     0]
 [   72    35  1339  4768   212     7    66   169     8     3]
 [   22     4   164   363  2441     5   518   115     5     0]
 [    2     0    38    97    20  7834     0     5     4     0]
 [   16     3    18   146   681     0 10267    69     0     0]
 [    1    12   190   368    79     0    24  1421     3     0]
 [    0     2    13    57    25     1    10    72    47     0]
 [    0     0     0    23     3     0     0     0     0     0]]
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.20      0.10      0.13       400
      Backdoor       0.21      0.07      0.11       349
           DoS       0.28      0.32      0.30      2453
      Exploits       0.63      0.71      0.67      6679
       Fuzzers       0.70      0.67      0.68      3637
       Generic       1.00      0.98      0.99      8000
        Normal       0.94      0.92      0.93     11200
Reconnaissance       0.74      0.68      0.71      2098
     Shellcode       0.60      0.21      0.31       227
         Worms       0.00      0.00      0.00        26

      accuracy                           0.79     35069
     macro avg       0.53      0.47      0.48     35069
  weighted avg       0.79      0.79      0.79     35069
```

3.2.2. KNN Buatan Sendiri

```
Accuracy: 0.7894
Confusion Matrix:
[[   31    11    93   219     9     0    34     3     0     0]
 [    9    21   106   178     8     0     1    24     2     0]
 [   17     6   604  1652    55    17    10    81    11     0]
 [   26    13  1062  4988   235     9    93   239    11     3]
 [   17     3   124   358  2386    13   578   148    10     0]
 [    0     0    34    99    19  7838     0     5     5     0]
 [    9     1    11   133   655     0 10310    80     1     0]
 [    0     8   153   390    71     1    20  1447     8     0]
 [    0     0     5    55    23     1     9    74    60     0]
 [    0     0     0    21     4     0     0     1     0     0]]
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.28      0.08      0.12       400
      Backdoor       0.33      0.06      0.10       349
           DoS       0.28      0.25      0.26      2453
      Exploits       0.62      0.75      0.68      6679
       Fuzzers       0.69      0.66      0.67      3637
       Generic       0.99      0.98      0.99      8000
        Normal       0.93      0.92      0.93     11200
Reconnaissance       0.69      0.69      0.69      2098
     Shellcode       0.56      0.26      0.36       227
         Worms       0.00      0.00      0.00        26

      accuracy                           0.79     35069
     macro avg       0.54      0.46      0.48     35069
  weighted avg       0.78      0.79      0.78     35069
```

Hasil perbandingan antara KNN buatan sendiri dan KNN menggunakan library menunjukkan bahwa KNN library memiliki kinerja yang sedikit lebih baik dengan akurasi sebesar 0.7894 dibandingkan 0.7882 pada KNN buatan sendiri. Dari confusion matrix, terlihat bahwa pola distribusi prediksi di kedua model mirip, namun KNN library lebih unggul dalam beberapa kategori, seperti *Exploits*, dengan jumlah prediksi yang lebih tepat. Pada evaluasi metrik seperti precision, recall, dan F1-score, KNN buatan sendiri memiliki performa yang lebih rendah di kelas-kelas sulit seperti *Analysis* dan *Backdoor*, sementara KNN library menunjukkan peningkatan pada kelas-kelas ini, serta menghasilkan nilai yang lebih konsisten di sebagian besar kelas. Namun, kedua model masih mengalami kesulitan dalam memprediksi kelas tertentu seperti *Worms*, yang memiliki recall 0 di kedua hasil, kemungkinan akibat distribusi data yang tidak seimbang atau jumlah data yang terlalu sedikit. Secara keseluruhan, meskipun KNN buatan sendiri menunjukkan hasil yang kompetitif dan fleksibilitas untuk penyesuaian lebih lanjut, KNN library menawarkan keunggulan dalam akurasi dan stabilitas performa pada beberapa kelas.

3.2. Naive Bayes

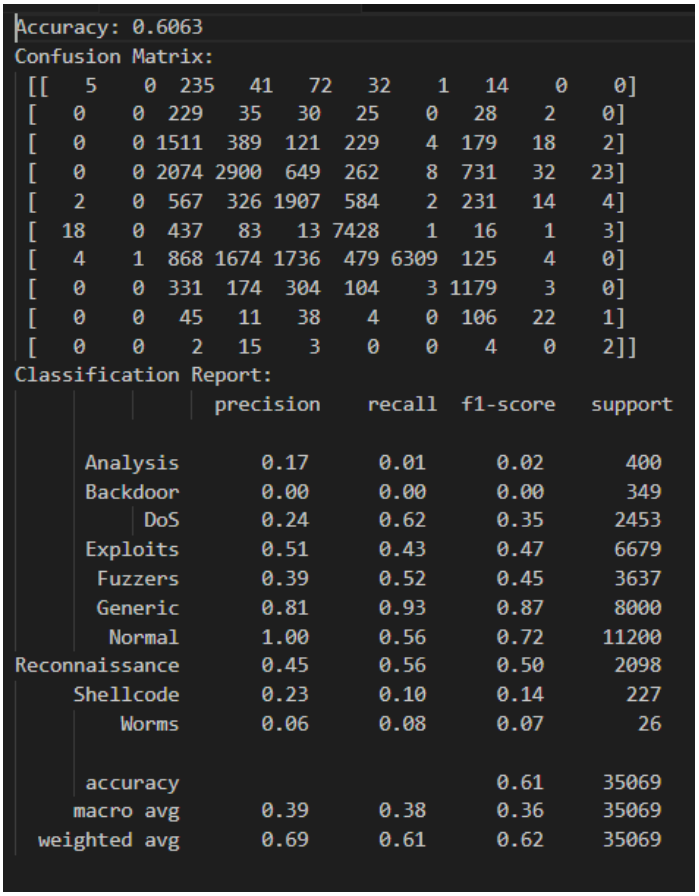
3.2.1. Naive Bayes dengan Library

```
Accuracy: 0.6064
Confusion Matrix:
 [[   5    0  235   41   72   32    1   14    0    0]
 [   0    0  229   35   30   25    0   28    2    0]
 [   0    0 1511  389  121  229    4  179   18    2]
 [   0    0 2074 2900  649  262    8  731   32   23]
 [   2    0  567  326 1907  584    2  231   14    4]
 [  18    0  437   83   13 7428    1   16    1    3]
 [   4    1  868 1674 1736  477 6311  125    4    0]
 [   0    0  331  174  304  104    3 1179    3    0]
 [   0    0   45   11   38    4    0  106   22    1]
 [   0    0    2   15    3    0    0    4    0    2]]
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.17      0.01      0.02       400
      Backdoor       0.00      0.00      0.00       349
           DoS       0.24      0.62      0.35      2453
      Exploits       0.51      0.43      0.47      6679
       Fuzzers       0.39      0.52      0.45      3637
       Generic       0.81      0.93      0.87      8000
        Normal       1.00      0.56      0.72     11200
Reconnaissance       0.45      0.56      0.50      2098
     Shellcode       0.23      0.10      0.14       227
         Worms       0.06      0.08      0.07        26

      accuracy                           0.61     35069
     macro avg       0.39      0.38      0.36     35069
  weighted avg       0.69      0.61      0.62     35069
```

3.2.2. Naive Bayes Buatan Sendiri



```
Accuracy: 0.6063
Confusion Matrix:
 [[   5    0  235   41   72   32    1   14    0    0]
 [   0    0  229   35   30   25    0   28    2    0]
 [   0    0 1511  389  121  229    4  179   18    2]
 [   0    0 2074 2900  649  262    8  731   32   23]
 [   2    0  567  326 1907  584    2  231   14    4]
 [  18    0  437   83   13 7428    1   16    1    3]
 [   4    1  868 1674 1736  479 6309  125    4    0]
 [   0    0  331  174  304  104    3 1179    3    0]
 [   0    0   45   11   38    4    0  106   22    1]
 [   0    0    2   15    3    0    0    4    0    2]]
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.17      0.01      0.02       400
      Backdoor       0.00      0.00      0.00       349
           DoS       0.24      0.62      0.35      2453
      Exploits       0.51      0.43      0.47      6679
       Fuzzers       0.39      0.52      0.45      3637
       Generic       0.81      0.93      0.87      8000
        Normal       1.00      0.56      0.72     11200
Reconnaissance       0.45      0.56      0.50      2098
     Shellcode       0.23      0.10      0.14       227
         Worms       0.06      0.08      0.07        26

      accuracy                           0.61     35069
     macro avg       0.39      0.38      0.36     35069
  weighted avg       0.69      0.61      0.62     35069
```

Hasil prediksi Gaussian Naive Bayes buatan sendiri dan yang menggunakan library menunjukkan performa yang hampir identik, dengan akurasi masing-masing sebesar 60,63% dan 60,64%. Confusion matrix kedua model memperlihatkan distribusi kesalahan klasifikasi yang serupa, di mana sebagian besar kesalahan terjadi pada kelas-kelas yang sulit seperti Analysis, Backdoor, Shellcode, dan Worms, yang memiliki precision, recall, dan f1-score sangat rendah. Sebaliknya, kelas Generic dan Normal menunjukkan performa terbaik, dengan precision tinggi masing-masing sebesar 0,81 dan 1,00, meskipun recall untuk Normal lebih rendah (0,56). Rata-rata makro (macro average) menunjukkan nilai precision, recall, dan f1-score masing-masing sebesar 0,39, 0,38, dan 0,36, menandakan performa rata-rata di semua kelas cukup rendah karena dominasi kelas mayoritas. Namun, rata-rata tertimbang (weighted average) lebih baik, dengan precision 0,69, recall 0,61, dan f1-score 0,62, karena kontribusi signifikan dari kelas mayoritas seperti Generic dan Normal. Secara keseluruhan, implementasi buatan sendiri berhasil mendekati performa model dari library, tetapi keduanya menunjukkan kelemahan dalam mengenali kelas minoritas. Untuk meningkatkan performa, diperlukan langkah-langkah seperti penyeimbangan dataset, peningkatan kualitas fitur, atau mencoba algoritma yang lebih kompleks.

3.3.ID3

3.3.1. ID3 library

```
Accuracy: 0.7777
Confusion Matrix:
 [[   57    14   157   106    34     1    19    12     0     0]
 [   20    30   107    89    24     1     4    70     4     0]
 [   98    80   832  1099   116    37    24   139    24     4]
 [  110    64  1155  4518   329    52   135   245    48    23]
 [   56    24   176   301  2237    24   638   143    34     4]
 [    0     3    37    76    24  7844     4     9     3     0]
 [   26     4    49   128   568     2 10352    58    13     0]
 [   10    78   200   252   131     6    48  1342    31     0]
 [    0     0    23    50    32     0    12    51    59     0]
 [    0     0     1    16     4     1     0     1     0     3]]
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.15      0.14      0.15       400
      Backdoor       0.10      0.09      0.09       349
           DoS       0.30      0.34      0.32      2453
      Exploits       0.68      0.68      0.68      6679
       Fuzzers       0.64      0.62      0.63      3637
       Generic       0.98      0.98      0.98      8000
        Normal       0.92      0.92      0.92     11200
Reconnaissance       0.65      0.64      0.64      2098
     Shellcode       0.27      0.26      0.27       227
         Worms       0.09      0.12      0.10        26
...
      accuracy                           0.78     35069
     macro avg       0.48      0.48      0.48     35069
  weighted avg       0.78      0.78      0.78     35069
```

BAB IV
KESIMPULAN DAN SARAN

4.1. Kesimpulan

Implementasi algoritma pembelajaran mesin (KNN, Gaussian Naive-Bayes, dan ID3) pada dataset UNSW-NB15 telah memberikan beberapa wawasan penting, yaitu

- Ketiga algoritma yang diimplementasikan mampu melakukan klasifikasi serangan siber dengan tingkat akurasi yang bervariasi. Implementasi from scratch menghasilkan performa yang sebanding dengan implementasi menggunakan scikit-learn, menunjukkan pemahaman yang baik tentang konsep dasar algoritma tersebut.
- Proses preprocessing data, termasuk penanganan missing values, normalisasi fitur numerik, dan penanganan ketidakseimbangan kelas, terbukti sangat penting dalam meningkatkan performa model. Hal ini menunjukkan bahwa kualitas data input memiliki dampak signifikan terhadap hasil klasifikasi.
- Masing-masing algoritma memiliki kelebihan dan keterbatasan tersendiri, yaitu
    - KNN menunjukkan fleksibilitas dalam mengklasifikasi pola serangan yang kompleks, namun memerlukan waktu komputasi yang lebih lama untuk dataset besar.
    - Gaussian Naive-Bayes memberikan kecepatan pemrosesan yang baik dan mudah diimplementasikan, meskipun asumsi independensi fitur tidak selalu terpenuhi.
    - ID3 menghasilkan model yang mudah diinterpretasi melalui struktur pohon keputusan, namun cenderung sensitif terhadap noise pada data.

4.2. Saran

Untuk pengembangan dan implementasi lebih lanjut, beberapa saran yang dapat dipertimbangkan, yaitu

1. Implementasi teknik paralel processing untuk meningkatkan kecepatan komputasi, terutama untuk algoritma KNN.
2. Eksplorasi teknik feature selection yang lebih advanced untuk mengurangi dimensi data tanpa kehilangan informasi penting.
3. Implementasi ensemble learning dengan mengkombinasikan ketiga algoritma untuk meningkatkan akurasi klasifikasi.
4. Pengembangan sistem pembobotan fitur yang adaptif untuk meningkatkan sensitivitas terhadap pola serangan yang jarang muncul.
5. Pengembangan interface yang user-friendly untuk memudahkan penggunaan model dalam konteks real-time.
6. Implementasi sistem logging dan monitoring untuk tracking performa model secara berkelanjutan.
7. Eksplorasi algoritma pembelajaran mesin lainnya untuk perbandingan performa yang lebih komprehensif.
8. Investigasi lebih lanjut tentang karakteristik serangan spesifik yang sulit dideteksi oleh model current.

BAB V
LAMPIRAN

5.1. Pembagian Kerja

NIM	Pembagian Kerja
13522140	• Data cleaning • Data pre-processing • Modelling • Implementasi algoritma KNN, Gaussian Naive Bayes, ID3
13522147	• Mengubah preprocessing menjadi pipeline • Implementasi ID3 • Membantu implementasi Gaussian Naive Bayes dan KNN • Modelling • Laporan
13522155	• Laporan • Membantu Implementasi ID3 • Modelling
13522161	• Laporan • Membantu Implementasi ID3 • Data splitting and processing

5.2. Referensi

- https://www.geeksforgeeks.org/sklearn-iterative-dichotomiser-3-id3-algorithms/
- https://www.geeksforgeeks.org/k-nearest-neighbours/
- https://www.geeksforgeeks.org/gaussian-naive-bayes/