Laporan Tugas Besar 2 IF3170

Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Kelompok 13 :

1.	Eduardus Alvito Kristiadi	13522004
2.	Rici Trisna Putra	13522026
3.	Francesco Michael Kusuma	13522038
4.	Keanu Amadius Gonza Wrahatno	13522082
5.	Dimas Bagoes Hendrianto	13522112

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

2024

DAFTAR ISI

BAB 1

DESKRIPSI DATASET

Dataset UNSW-NB15 merupakan dataset berisi raw network packets yang dibuat menggunakan IXIA PerfectStorm oleh Cyber Range Lab UNSW Canberra. Dataset ini terdiri dari 10 jenis aktivitas (9 jenis attack dan 1 aktivitas normal). Sembilan jenis attack yang termasuk ke dalam dataset ini adalah Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode dan Worms. Dataset yang akan digunakan (beserta metadata-nya) pada tugas ini dapat anda akses di tautan berikut (Catatan: Terdapat perubahan dataset dari yang telah digunakan di Tugas Kecil 2, sehingga untuk Tugas Besar 2, gunakan data yang diberikan pada tautan tersebut). Untuk tugas ini, variabel target yang akan digunakan hanya attack_cat saja.

Untuk menghasilkan prediksi yang berkualitas, Anda diharuskan untuk melakukan beberapa tahap berikut ini (tahapan lebih lengkap dapat dilihat di template notebook):

1. Data Cleaning
   Tahap ini bertujuan untuk membersihkan dataset dari nilai yang hilang (missing values), data duplikat, atau data yang tidak valid sehingga dataset siap digunakan untuk analisis.
   Untuk mengatasi missing value, dilakukan dengan SimpleImputer pada pipeline.
   Untuk mengatasi outlier, dilakukan dengan teknik clip

```python
def handle_outliers(X):
    df_numeric = X.select_dtypes(include=[np.number])
    Q1 = df_numeric.quantile(0.25)
    Q3 = df_numeric.quantile(0.75)
    IQR = Q3 - Q1

    lower_limit = Q1 - 1.5 * IQR
    upper_limit = Q3 + 1.5 * IQR

    for column in df_numeric.columns:
        X[column] = np.clip(X[column], lower_limit[column], upper_limit[column])

    return X
```

   Untuk mengatasi data yang duplicat digunakan metode drop_duplicate()
   Lalu yang terakhir, proses Feature Engineering

```python
def feature_engineering(X):
    if 'is_ftp_login' in X.columns:
        X['is_ftp_login'] = X['is_ftp_login'].replace({0: 'false', 1: 'true'})

    if 'is_sm_ips_ports' in X.columns:
        X['is_sm_ips_ports'] = X['is_sm_ips_ports'].replace({0: 'false', 1: 'true'})

    X['network_bytes'] = X['sbytes'] + X['dbytes']

    correlation_matrix = X.select_dtypes(include=[np.number]).corr()

    upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))

    highly_correlated_features = [column for column in upper_triangle.columns if any(upper_triangle[column] > 0.8)]

    X.drop(columns=highly_correlated_features, inplace=True, errors='ignore')

    return X, highly_correlated_features
```

Maka, feature yang di drop adalah `['ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'sloss', 'dloss', 'spkts', 'dpkts', 'dwin', 'dmean', 'sjit', 'djit', 'sinpkt', 'dinpkt', 'tcprtt', 'synack', 'ackdat', 'network_bytes']`
Selain itu, menggabungkan sbytes dan dbytes menjadi network_bytes

2. Data Preprocessing

   Feature Encoding
   Ini dilakukan dengan menggunakan OneHotEncoder. Ini akan mengubah feature categorical menjadi binary 0, 1. Jika ada data baru, maka di ignore atau diabaikan

   Data Normalization
   StandardScaler diterapkan untuk melakukan standarisasi data. Proses ini mengubah fitur-fitur dalam dataset sehingga memiliki distribusi dengan mean 0 dan standar deviasi 1.

   DimensiReduction
   Jika dataset memiliki jumlah fitur yang besar, reduksi dimensi dapat digunakan untuk mengurangi dimensi tanpa kehilangan informasi penting. Teknik seperti Principal Component Analysis (PCA) sering digunakan pada tahap ini. PCA dilakukan pada pipeline

3. Compile Preprocessing Pipeline

```python
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('yeo_johnson', PowerTransformer(method='yeo-johnson')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='unknown')),
    ('onehot', OneHotEncoder(sparse_output=False, handle_unknown='ignore'))
])
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, X_train_transformed.select_dtypes(include=[np.number]).columns),
        ('cat', categorical_transformer, X_train_transformed.select_dtypes(exclude=[np.number]).columns)
    ], remainder='passthrough'
)
```

```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('var_filter', VarianceThreshold(threshold=0.01)),
    ('feature_selection', SelectKBest(score_func=f_classif, k=20)),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95))
])
```

4. Modeling dan Validation
   Pada tahap ini, algoritma pembelajaran mesin seperti K-Nearest Neighbors (KNN), Naive Bayes, dan ID3 (Iterative Dichotomiser 3) diterapkan pada dataset. Anda akan melatih model pembelajaran mesin yang akan mengklasifikasi kategori attack (attack_cat) berdasarkan fitur-fitur lain yang telah diberikan. Model yang telah dibuat divalidasi menggunakan metode seperti train-test split atau k-fold cross-validation untuk memastikan kinerja yang optimal.
   KNN

BAB 2

PEMBAHASAN

2.1. Penjelasan Implementasi

2.1.1. KNN

K-Nearest Neighbor atau KNN adalah sebuah algoritma machine learning yang bekerja berdasarkan prinsip bahwa objek yang nilai atribut-atributnya mirip cenderung berada dalam jarak yang dekat satu sama lain. Dengan kata lain, data yang memiliki karakteristik serupa akan cenderung saling bertetangga dalam ruang fitur (feature space). Algoritma K-Nearest Neighbor (KNN) juga memiliki karakteristik sebagai algoritma yang bersifat non-parametric dan lazy learning.

Salah satu karakteristik utama KNN adalah sifat non-parametric-nya. Konsep non-parametric dalam KNN menggambarkan sifat algoritma ini yang tidak membuat asumsi tertentu tentang distribusi data yang digunakan. Dalam kata lain, KNN tidak memiliki parameter tertentu atau estimasi parameter yang harus diatur pada modelnya, terlepas dari seberapa banyak data yang digunakan. Ini menjadikan KNN sebagai algoritma yang sangat fleksibel dan mampu menangani berbagai jenis data.

KNN juga dikenal juga sebagai algoritma lazy learning. Ini berarti algoritma ini tidak melibatkan fase pelatihan yang signifikan seperti algoritma machine learning lainnya. Dalam KNN, hampir tidak ada pembentukan model yang dilakukan menggunakan data pelatihan. Sebaliknya, seluruh data pelatihan digunakan saat menguji atau melakukan klasifikasi data baru. Hal ini membuat proses pelatihan berjalan lebih cepat, tetapi proses pengujian memerlukan lebih banyak waktu dan sumber daya, termasuk penggunaan memori yang lebih besar.

Prinsip dasar algoritma KNN mengasumsikan bahwa objek yang mirip akan berada dalam jarak yang dekat satu sama lain. Dengan kata lain, data yang memiliki karakteristik serupa akan cenderung terletak berdekatan. KNN menggunakan seluruh data yang tersedia dalam pengambilan keputusan. Ketika ada data baru yang perlu diklasifikasikan, algoritma mengukur tingkat kemiripan atau fungsi jarak antara data baru tersebut dengan data yang sudah ada. Data baru kemudian ditempatkan dalam kelas yang paling banyak dimiliki oleh data tetangga terdekatnya.

```python
class KNNScratch:
    def __init__(self, neighbors=5, metric='euclidean', p=2):
        self.neighbors = neighbors
        self.metric = metric
        self.p = p

    def fit(self, X_train, y_train):
```

```
        self.X_train = X_train.to_numpy() if hasattr(X_train, 'to_numpy') else X_train
        self.y_train = y_train.to_numpy() if hasattr(y_train, 'to_numpy') else y_train

        self.tree = BallTree(self.X_train, metric=self.metric)

    def predict(self, X_test):
        X_test = X_test.to_numpy() if hasattr(X_test, 'to_numpy') else X_test

        predictions = []
        for x_test in X_test:
            distances, indices = self.tree.query([x_test], k=self.neighbors)
            k_nearest_labels = self.y_train[indices[0]]
            most_common = Counter(k_nearest_labels).most_common(1)
            predictions.append(most_common[0][0])

        return np.array(predictions)
```

1. **Inisialisasi (*__init__ method*)**
   Tujuan : Mengatur konfigurasi awal algoritma k-Nearest Neighbors (k-NN).
   Parameter :
   - neighbors (default : 5) : Jumlah tetangga terdekat yang akan digunakan untuk prediksi.
   - metric (default : euclidean) : Metode pengukuran jarak yang digunakan. Opsi:
     - Euclidean : Menggunakan jarak Euclidean.
     - Manhattan : Menggunakan jarak Manhattan.
     - Minkowski : Menggunakan jarak Minkowski
   - p (default : 2): Parameter pangkat untuk jarak Minkowski.

2. **Melatih Model (*fit method*)**
   Input :
   - X_train: Fitur data pelatihan (numpy array atau DataFrame).
   - y_train: Label data pelatihan (numpy array atau Series).

   Proses :
   - Data pelatihan dikonversi menjadi numpy array jika masih berbentuk DataFrame/Series.
   - Membuat BallTree dari X_train, yang mempercepat pencarian tetangga berdasarkan metrik jarak yang diberikan.

   Hasil :
   - Data pelatihan (X_train, y_train) disimpan.
   - Sebuah objek BallTree diinisialisasi untuk membantu pencarian tetangga lebih efisien.

3. **Prediksi (*predict method*)**
   Tujuan : Memperkirakan label untuk data uji.
   Input : X_test: Data baru yang ingin diprediksi.
   Proses :
   - Data uji (X_test) dikonversi menjadi numpy array jika masih berbentuk DataFrame.
   - Untuk setiap sampel dalam X_test:
     - Menggunakan BallTree untuk mencari tetangga terdekat (k tetangga) berdasarkan metrik jarak.
     - Mengambil label dari tetangga terdekat (k_nearest_labels).
     - Menentukan label yang paling umum (most common) dari tetangga-tetangga tersebut menggunakan Counter.

   Hasil : Prediksi untuk semua data uji, dikembalikan sebagai array numpy.

2.1.2. Naive-Bayes

Naive Bayes Classifier atau NB adalah sekumpulan algoritma yang didasarkan pada Teorema Bayes. Dengan kata lain, algoritma ini bukan algoritma tunggal melainkan satu grup algoritma dimana masing-masing memiliki prinsip kerja yang mirip. Pengertian dan Contoh Algoritma Naive Bayes Classifier Algoritma ini bekerja berdasarkan prinsip probabilitas bersyarat, seperti yang diberikan oleh Teorema Bayes. Teorema Bayes menemukan probabilitas atau kemungkinan suatu peristiwa akan terjadi dengan memberikan probabilitas peristiwa lain yang telah terjadi. Dalam istilah yang lebih sederhana, Teorema Bayes adalah metode untuk menemukan probabilitas ketika kita mengetahui probabilitas tertentu lainnya. Teorema Bayes dinyatakan secara matematis dalam persamaan berikut,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

dimana P(B) != 0

Pada dasarnya, kita mencoba mencari peluang kejadian A, apabila kejadian B bernilai benar. Kejadian B juga disebut sebagai bukti. P(A) adalah apriori dari A (probabilitas sebelumnya, yaitu probabilitas peristiwa sebelum bukti terlihat). Bukti adalah nilai atribut dari instance yang tidak diketahui (peristiwa B). P(A|B) adalah probabilitas posteriori dari B, yaitu probabilitas kejadian setelah bukti terlihat. Ciri utama dari algoritma Naive Bayes Classifier adalah adanya asumsi yg sangat kuat (naif) akan
independensi dari masing-masing kondisi atau kejadian.

Namun pada kasus ini, kami menggunakan Gaussian Naive Bayes yang merupakan modifikasi dari Naive Bayes biasa. Naive Bayes biasa hanya cocok untuk data dengan fitur kategori atau data yang bisa diasumsikan sebagai distribusi diskrit seperti teks atau klasifikasi biner. Di sisi lain, Gaussian Naive Bayes lebih cocok untuk data dengan fitur-fitur numerik yang diasumsikan terdistribusi normal atau mendekati distribusi normal.

```python
class GaussianNaiveBayesScratch:
    def gauss_dist(self, class_idx, x):
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator

    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = np.zeros((len(self.classes), X.shape[1]))
        self.var = np.zeros((len(self.classes), X.shape[1]))
        self.priors = np.zeros(len(self.classes))

        for idx, c in enumerate(self.classes):
            X_c = X[y == c]
            self.mean[idx, :] = X_c.mean(axis=0)
            self.var[idx, :] = X_c.var(axis=0)
            self.priors[idx] = X_c.shape[0] / X.shape[0]

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        likelihoods = []
        for idx, c in enumerate(self.classes):
            prior = np.log(self.priors[idx])
            likelihood = np.sum(np.log(self.gauss_dist(idx, x)))
            likelihood += prior
            likelihoods.append(likelihood)
        return self.classes[np.argmax(likelihoods)]
```

1. **Metode gauss_dist**

   Tujuan : Menghitung probabilitas densitas Gaussian untuk suatu nilai fitur x dalam kelas tertentu.

   Parameter :

   - class_idx : Indeks kelas yang sedang dihitung.
   - x : Nilai fitur yang dihitung densitasnya.

   Logika :

   - Menggunakan formula distribusi Gaussian :

$$P(x|c) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Hasil : Mengembalikan nilai probabilitas densitas untuk x

2. **Metode fit**

   Tujuan : Melatih model dengan data pelatihan.

   Parameter :

   - X: Data fitur pelatihan.
   - y: Label kelas pelatihan

   Logika:

   - Mengidentifikasi kelas unik dalam y dan menyimpannya dalam self.classes.
   - Menginisialisasi :
     - self.mean : Array untuk menyimpan rata-rata fitur setiap kelas.
     - self.var : Array untuk menyimpan variansi fitur setiap kelas.
     - self.priors : Array untuk menyimpan probabilitas prior setiap kelas.
   - Untuk setiap kelas c:
     - Ambil subset data fitur X_c yang memiliki label c.
     - Hitung rata-rata (mean) dan variansi (var) setiap fitur dalam kelas c dan simpan.
     - Hitung probabilitas prior kelas c sebagai :

$$P(c) = \frac{jumlah\ sampel\ dalam\ kelas\ c}{total\ jumlah\ sampel}$$

3. **Metode predict**

   Tujuan : Membuat prediksi untuk data uji X.

   Logika :

   - Untuk setiap sampel dalam X, panggil metode _predict untuk memprediksi kelas.
   - Hasilkan array prediksi (y_pred).

4. **Metode _predict**

   Tujuan : Memprediksi kelas untuk satu sampel x.

   Logika :

   - Untuk setiap kelas c :
     - Hitung prior dalam logaritma :

$$log\ P(c)$$

     - Hitung log likelihood :

$$\Sigma log P(x|c)$$

     - Tambahkan log prior ke log likelihood untuk mendapatkan total skor probabilitas.
   - Simpan total skor probabilitas untuk setiap kelas dalam likelihoods.
   - Pilih kelas dengan skor probabilitas tertinggi menggunakan np.argmax.

2.1.3. ID3

```python
class DecisionTreeClassifierScratch:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.tree = None

    def entropy(self, y):
        """Calculate the entropy of a dataset."""
        classes, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        return -np.sum(probabilities * np.log2(probabilities + 1e-9))  # Added epsilon to avoid log(0)

    def information_gain(self, y, y_left, y_right):
        """Calculate information gain from a split."""
        parent_entropy = self.entropy(y)
        n = len(y)
        n_left, n_right = len(y_left), len(y_right)
        if n_left == 0 or n_right == 0:
            return 0
        # Weighted average of the entropy of the children
        child_entropy = (n_left / n) * self.entropy(y_left) + (n_right / n) * self.entropy(y_right)
        return parent_entropy - child_entropy

    def split(self, X, y, feature_index, threshold):
        """Split the dataset into left and right based on a threshold."""
        left_idxs = X[:, feature_index] <= threshold
        right_idxs = X[:, feature_index] > threshold
        return X[left_idxs], X[right_idxs], y[left_idxs], y[right_idxs]

    def best_split(self, X, y):
        """Find the best feature and threshold to split on using information gain."""
        best_gain = -1
        best_index, best_threshold = None, None

        for feature_index in range(X.shape[1]):
            thresholds = np.unique(X[:, feature_index])
            for threshold in thresholds:
                X_left, X_right, y_left, y_right = self.split(X, y, feature_index, threshold)

                if len(y_left) == 0 or len(y_right) == 0:
                    continue

                gain = self.information_gain(y, y_left, y_right)

                if gain > best_gain:
                    best_gain = gain
                    best_index = feature_index
                    best_threshold = threshold
```

```
        return best_index, best_threshold

    def build_tree(self, X, y, depth=0):
        """Recursively build the decision tree."""
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        if n_labels == 1 or n_samples == 0 or (self.max_depth is not None and depth >= self.max_depth):
            return np.bincount(y).argmax()

        feature_index, threshold = self.best_split(X, y)
        if feature_index is None:
            return np.bincount(y).argmax()

        left_idxs = X[:, feature_index] <= threshold
        right_idxs = X[:, feature_index] > threshold
        left = self.build_tree(X[left_idxs], y[left_idxs], depth + 1)
        right = self.build_tree(X[right_idxs], y[right_idxs], depth + 1)

        return {'feature_index': feature_index, 'threshold': threshold, 'left': left, 'right': right}

    def fit(self, X, y):
        """Build the decision tree from the training data."""
        self.tree = self.build_tree(X, y)

    def predict_one(self, x, tree):
        """Predict a single instance using the decision tree."""
        if not isinstance(tree, dict):
            return tree
        if x[tree['feature_index']] <= tree['threshold']:
            return self.predict_one(x, tree['left'])
        else:
            return self.predict_one(x, tree['right'])

    def predict(self, X):
        """Predict multiple instances using the decision tree."""
         return np.array([self.predict_one(x, self.tree) for x in X])
```

1. **Inisialisasi (__init__ method)**
   Tujuan : Mengatur parameter awal model.
   Parameter :
   - max_depth : Kedalaman maksimum pohon keputusan. Digunakan untuk mencegah overfitting.
   - tree : Menyimpan struktur pohon keputusan setelah dilatih.

2. **Entropi (entropy method)**
   Tujuan : Menghitung entropi dataset y, yaitu ukuran ketidakpastian kelas dalam data.

Formula:

$$H(y) = -\sum_i P(c_i) \cdot log_2(P(c_i))$$

Detail :

- Kelas unik dihitung menggunakan np.unique.
- Probabilitas dihitung sebagai jumlah kemunculan setiap kelas dibagi jumlah total sampel.
- Epsilon (1e-9) ditambahkan untuk menghindari kesalahan saat menghitung log(0).

3. **Information Gain (information_gain method)**
   Tujuan : Mengukur pengurangan ketidakpastian (entropi) setelah dataset dibagi berdasarkan suatu fitur dan threshold.
   Formula :

$$IG = H(parent) - (\frac{n_{left}}{n} H(left) + \frac{n_{right}}{n} H(right)$$

   Hasil : Mengembalikan nilai Information Gain.

4. **Split Dataset (split method)**
   Tujuan : Membagi dataset menjadi subset kiri dan kanan berdasarkan fitur (feature_index) dan threshold tertentu.
   Detail :
   - Data yang memenuhi kondisi X[:, feature_index] <= threshold masuk ke subset kiri.
   - Data lainnya masuk ke subset kanan.

   Hasil : Mengembalikan subset kiri (X_left, y_left) dan kanan (X_right, y_right).

5. **Best Split (best_split method)**
   Tujuan : Mencari fitur dan threshold terbaik untuk memisahkan dataset berdasarkan Information Gain tertinggi.
   Langkah-langkah :
   - Iterasi melalui semua fitur.
   - Untuk setiap fitur, coba semua nilai unik sebagai threshold.
   - Bagi dataset dengan threshold tersebut.
   - Hitung Information Gain untuk split tersebut.
   - Simpan fitur dan threshold dengan Information Gain tertinggi.

   Hasil : Mengembalikan indeks fitur (feature_index) dan nilai threshold terbaik

6. **Membangun Pohon (build_tree method)**
   Tujuan : Membangun pohon keputusan secara rekursif.
   Langkah-langkah :
   - Kondisi Terminasi :
     - Jika semua sampel dalam satu kelas.

    - Jika tidak ada sampel tersisa.
    - Jika kedalaman maksimum tercapai (max_depth).
    - Mengembalikan label mayoritas menggunakan np.bincount(y).argmax().
- Cari fitur dan threshold terbaik menggunakan best_split.
- Bagi dataset menjadi subset kiri dan kanan.
- Rekursif membangun subtree untuk subset kiri dan kanan.
- Struktur pohon disimpan sebagai dictionary

7. **Melatih Model (fit method)**
   Tujuan : Membangun pohon keputusan dari data pelatihan (X, y) dengan memanggil metode build_tree.
   Hasil : Menyimpan struktur pohon dalam atribut self.tree.

8. **Prediksi Satu Sampel (predict_one method)**
   Tujuan : Melakukan prediksi untuk satu sampel x menggunakan pohon keputusan.
   Logika :
   - Jika simpul saat ini adalah simpul daun (bukan dictionary), kembalikan label.
   - Jika x[feature_index] <= threshold, teruskan ke subtree kiri.
   - Jika tidak, teruskan ke subtree kanan.

9. **Prediksi Banyak Sampel (predict method)**
   Tujuan : Melakukan prediksi untuk beberapa sampel dalam X.
   Detail :
   - Iterasi melalui setiap sampel dalam X.
   - Prediksi dilakukan untuk setiap sampel dengan memanggil predict_one.

   Hasil : Mengembalikan array label prediksi.

2.1.4. Cleaning and Preprocessing

Pembersihan data atau data cleaning adalah proses mengidentifikasi dan mengoreksi kesalahan, konsistensi, dan akurasi data. Proses ini dilakukan dengan tujuan meningkatkan kualitas data. Kualitas data berpengaruh besar pada kinerja model. Terdapat beberapa teknik data cleansing yang kami gunakan, yaitu sebagai berikut.

Imputer untuk Data Numerikal dengan Median

Dalam pengolahan data numerikal, kami menggunakan metode imputer median untuk menangani nilai-nilai yang hilang. Imputer median bekerja dengan menggantikan nilai yang hilang pada suatu kolom dengan nilai tengah (median) dari data pada kolom tersebut. Metode ini dipilih karena median memberikan representasi yang lebih stabil, terutama ketika data memiliki pencilan (outlier) atau distribusi yang tidak simetris. Berbeda dengan rata-rata (mean) yang bisa

terpengaruh oleh nilai ekstrem, median tetap mempertahankan karakteristik distribusi data yang lebih representatif.

Imputer untuk Data Kategorikal dengan Constant

Untuk menangani nilai yang hilang (missing values) pada data kategorikal, kami menggunakan imputer dengan strategi 'constant'. Dalam metode ini, nilai yang hilang akan digantikan dengan nilai tetap yang telah ditentukan, misalnya *'unknown'*. Ada beberapa alasan yang mendasari pemilihan antara lain, mengisi nilai hilang dengan nilai tetap seperti *'unknown'* membantu menjaga format data agar tetap terstruktur, nilai seperti *'unknown'* jelas menunjukkan bahwa data tersebut tidak tersedia, tanpa memperkenalkan asumsi yang tidak perlu, metode ini efektif digunakan pada kolom data kategorikal di mana pengisian dengan nilai rata-rata atau median tidak relevan.

Clipping untuk Data Numerikal dengan IQR

Clipping dengan IQR adalah metode untuk menangani nilai ekstrim (outlier) dalam data numerikal dengan membatasi nilai ke dalam rentang tertentu berdasarkan interquartile range (IQR). Teknik ini membantu menjaga distribusi data tetap representatif tanpa harus menghapus nilai ekstrim sepenuhnya.

$$\text{Batas bawah} : Q1 - 1.5 \times IQR$$
$$\text{Batas atas} : Q3 + 1.5 \times IQR$$

2.2. Hasil eksperimen dan analisis

Berdasarkan implementasi algoritma yang telah kami kerjakan didapat perbandingan prediksi algoritma yang telah kami implementasi dengan algoritma pustaka sebagai berikut:

KNN Scratch

```
from main import KNNScratch

knn = KNNScratch(neighbors=21, metric='euclidean')

knn.fit(train_X_preprocessor, y_train)
knn_pred = knn.predict(val_X_preprocessor)
print("Accuracy: ", accuracy_score(y_val, knn_pred))
print("Classification Report: \n", classification_report(y_val, knn_pred))
```

✓ 54.9s

```
Accuracy:  0.7840226984330667
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.32      0.03      0.06       780
      Backdoor       0.56      0.04      0.07       694
           DoS       0.32      0.19      0.24      4924
      Exploits       0.61      0.81      0.70     13386
       Fuzzers       0.61      0.67      0.64      7285
       Generic       0.99      0.98      0.99     16007
        Normal       0.91      0.88      0.90     22341
Reconnaissance       0.75      0.69      0.72      4241
     Shellcode       0.51      0.22      0.31       426
         Worms       0.00      0.00      0.00        53

      accuracy                           0.78     70137
     macro avg       0.56      0.45      0.46     70137
  weighted avg       0.78      0.78      0.77     70137
```

KNN Pustaka

```
knn2 = KNeighborsClassifier(n_neighbors=21, metric='euclidean')

knn2.fit(train_X_preprocessor, y_train)
knn_pred = knn2.predict(val_X_preprocessor)
print("Accuracy: ", accuracy_score(y_val, knn_pred))
print("Classification Report: \n", classification_report(y_val, knn_pred))
```

✓ 8.7s

```
Accuracy:  0.782725237748977
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.31      0.04      0.07       780
      Backdoor       0.51      0.04      0.07       694
           DoS       0.32      0.22      0.26      4924
      Exploits       0.61      0.80      0.69     13386
       Fuzzers       0.60      0.68      0.64      7285
       Generic       1.00      0.98      0.99     16007
        Normal       0.92      0.88      0.90     22341
Reconnaissance       0.75      0.68      0.71      4241
     Shellcode       0.52      0.20      0.29       426
         Worms       0.00      0.00      0.00        53

      accuracy                           0.78     70137
     macro avg       0.55      0.45      0.46     70137
  weighted avg       0.78      0.78      0.77     70137
```

Bisa dilihat bahwa terdapat perbedaan performa dikarenakan tidak adanya optimasi seperti yang ada pada algoritma pustaka standar seperti ngan struktur data canggih seperti KD-tree atau Ball Tree, serta paralelisasi untuk pencarian tetangga yang lebih efisien. Algoritma kami juga belum dapat memangkas pencarian dengan efektif. Perbedaan ini juga bisa disebabkan karena adanya perbedaan pemilihan data saat ada jarak yang sama.

Naive Bayes Scratch

```
from main import GaussianNaiveBayesScratch

gnb_scratch = GaussianNaiveBayesScratch()
gnb_scratch.fit(train_X_preprocessor, y_train)
gnb_scratch_pred = gnb_scratch.predict(val_X_preprocessor)
print("Accuracy: ", accuracy_score(y_val, gnb_scratch_pred))
print("Classification Report: \n", classification_report(y_val, gnb_scratch_pred))
```

✓ 7.9s

```
Accuracy:  0.6819367808717225
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.00      0.00      0.00       780
      Backdoor       0.06      0.03      0.04       694
           DoS       0.33      0.68      0.44      4924
      Exploits       0.59      0.39      0.47     13386
       Fuzzers       0.48      0.56      0.52      7285
       Generic       1.00      0.96      0.98     16007
        Normal       0.80      0.79      0.80     22341
Reconnaissance       0.51      0.49      0.50      4241
     Shellcode       0.18      0.11      0.13       426
         Worms       0.04      0.34      0.07        53

      accuracy                           0.68     70137
     macro avg       0.40      0.43      0.39     70137
  weighted avg       0.70      0.68      0.68     70137
```

Naive Bayes Pustaka

```
gnb = GaussianNB()
gnb.fit(train_X_preprocessor, y_train)
gnb_pred = gnb.predict(val_X_preprocessor)
print("Accuracy: ", accuracy_score(y_val, gnb_pred))
print("Classification Report: \n", classification_report(y_val, gnb_pred))
```

[2] ✓ 1.1s

```
Accuracy:  0.6819367808717225
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.00      0.00      0.00       780
      Backdoor       0.06      0.03      0.04       694
           DoS       0.33      0.68      0.44      4924
      Exploits       0.59      0.39      0.47     13386
       Fuzzers       0.48      0.56      0.52      7285
       Generic       1.00      0.96      0.98     16007
        Normal       0.80      0.79      0.80     22341
Reconnaissance       0.51      0.49      0.50      4241
     Shellcode       0.18      0.11      0.13       426
         Worms       0.04      0.34      0.07        53

      accuracy                           0.68     70137
     macro avg       0.40      0.43      0.39     70137
  weighted avg       0.70      0.68      0.68     70137
```

Dapat dilihat dari hasil akurasi antara naive bayes implementasi dan pustaka, nilainya sangat mirip sehingga dapat disimpulkan bahwa algoritma implementasi sudah berhasil mereplikasikan performa akurasi algoritma pustaka.

Decision Tree Pustaka

```
id3 = DecisionTreeClassifier()
id3.fit(train_X_preprocessor, y_train)
id3_pred = gnb_scratch.predict(val_X_preprocessor)
print("Accuracy: ", accuracy_score(y_val, id3_pred))
print("Classification Report: \n", classification_report(y_val, id3_pred))
```

✓ 10.7s

```
Accuracy:  0.6819367808717225
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.00      0.00      0.00       780
      Backdoor       0.06      0.03      0.04       694
           DoS       0.33      0.68      0.44      4924
      Exploits       0.59      0.39      0.47     13386
       Fuzzers       0.48      0.56      0.52      7285
       Generic       1.00      0.96      0.98     16007
        Normal       0.80      0.79      0.80     22341
Reconnaissance       0.51      0.49      0.50      4241
     Shellcode       0.18      0.11      0.13       426
         Worms       0.04      0.34      0.07        53

      accuracy                           0.68     70137
     macro avg       0.40      0.43      0.39     70137
  weighted avg       0.70      0.68      0.68     70137
```

Karena algoritma Decision Tree kami terlalu lama untuk dijalankan maka tidak ada komparasi yang dapat dilakukan, namun dari prediksi kami akan lebih lambat karena decision tree pustaka menggunakan Gini Impurity sebagai pengganti entropy yang lebih bagus, selain itu pustaka juga menggunakan algoritma CART yang berbeda dengan ID3 dan memiliki optimalitas lebih tinggi.

BAB 3

KESIMPULAN DAN SARAN

3.1 Kesimpulan

Berdasarkan hasil eksperimen dan analisis kami, didapatkan bahwa dalam implementasi dan Perbandingan Algoritma dalam tugas besar ini telah berhasil diimplementasikan tiga algoritma machine learning dari awal (from scratch) yaitu k-Nearest Neighbors (kNN), Naive Bayes, dan Decision Tree untuk klasifikasi cyber attack, implementasi algoritma-algoritma from scratch dibandingkan dengan implementasi menggunakan library scikit-learn untuk memvalidasi kebenaran implementasi dan mengukur performa.

Dataset yang digunakan adalah UNSW-NB15 yang berisi berbagai fitur serangan siber Dilakukan berbagai tahap preprocessing termasuk penanganan missing values menggunakan imputation, feature selection menggunakan SelectKBest dan VarianceThreshold, feature scaling menggunakan StandardScaler, dimensionality reduction menggunakan PCA.

KNN mencapai akurasi tertinggi sebesar 0.784 dengan parameter k=21, Naive Bayes mencapai akurasi 0.681. Decision Tree mencapai akurasi 0.68. Feature engineering dan preprocessing data terbukti sangat penting untuk meningkatkan performa model.

PCA membantu mengurangi dimensi data dengan menghasilkan 10 komponen optimal. Secara keseluruhan, proyek ini berhasil mengimplementasikan dan membandingkan tiga algoritma klasifikasi dasar, dengan hasil yang cukup baik terutama untuk klasifikasi kelas mayoritas. Penggunaan teknik preprocessing yang tepat terbukti penting dalam meningkatkan performa model.

KNN memiliki akurasi paling tinggi karena ia menghitung jarak perbedaan. Walaupun waktu yang diperlukan relatif lama, namun hasilnya paling baik.

3.2 Saran

Terkait kinerja kami mendapat bahwa seharusnya kami mulai mengerjakan sejak jauh hari dan tidak deadliner supaya pengerjaan dapat dilakukan dengan pengujian secara benar dan tertata.

BAB 4

PEMBAGIAN TUGAS

No	Anggota	Tugas
1	Eduardus Alvito Kristiadi (13522004)	Split Training Set and Validation Set, Data Cleaning and Preprocesing, Compile Preprocessing Pipeline, Modeling and Validation
2	Rici Trisna Putra (13522026)	Split Training Set and Validation Set, Data Cleaning and Preprocesing, Compile Preprocessing Pipeline, Modeling and Validation
3	Francesco Michael Kusuma (13522038)	Split Training Set and Validation Set, Data Cleaning and Preprocesing, Compile Preprocessing Pipeline, Modeling and Validation
4	Keanu Amadius Gonza Wrahatno (13522082)	Split Training Set and Validation Set, Data Cleaning and Preprocesing, Compile Preprocessing Pipeline, Modeling and Validation
5	Dimas Bagoes Hendrianto (13522112)	Split Training Set and Validation Set, Data Cleaning and Preprocesing, Compile Preprocessing Pipeline, Modeling and Validation

REFERENSI

https://github.com/FrancescoMichael/Tubes2-AI