

Laporan Tugas Besar 2 IF3170 Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin
Semester I Tahun 2024/2025



Disusun oleh:

- | | |
|----------------------------|----------|
| 1. Maulvi Ziadinda Maulana | 13522122 |
| 2. Ahmad Rafi Maliki | 13522137 |
| 3. Nicholas Reymond Sihite | 13522144 |
| 4. Albert Ghazaly | 13522150 |

INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

Daftar Isi.....	2
Bagian 1. Deskripsi Permasalahan.....	3
Bagian 2. Implementasi Algoritma.....	4
2.1. K-Nearest Neighbour (KNN).....	4
2.2. Gaussian Naive Bayes (NB).....	6
2.3. Iterative Dichotomiser 3 (ID3).....	11
Bagian 3. Pembersihan dan Prapemrosesan Data.....	17
3.1. Pembersihan Data.....	17
3.1.1. Penanganan Nilai Kosong.....	17
3.1.2. Penanganan Nilai Outlier.....	20
3.1.3. Penanganan Nilai Duplikat.....	20
3.1.4. Rekayasa Fitur.....	20
3.2. Prapemrosesan Data.....	22
3.2.1. Feature Scaling.....	22
3.2.2. Feature Encoding.....	23
3.2.3. Handling Imbalanced Dataset.....	25
3.2.4. Normalisasi Data.....	27
3.2.5. Pengurangan Dimensi Data.....	27
3.3. Pipeline per Algoritma.....	29
3.3.1. Pipeline k-Nearest Neighbour.....	29
3.3.2. Pipeline Gaussian Naive-Bayes.....	30
3.3.3. Pipeline Iterative Dichotomiser 3.....	30
Bagian 4. Percobaan dan Analisis.....	32
4.1. Percobaan.....	32
4.2. Analisis.....	38
Pembagian Tugas.....	39
Referensi.....	40

Bagian 1. Deskripsi Permasalahan



Gambar 1. Ilustrasi Pembelajaran Mesin

(sumber: <https://myjce.edu.in/blog/future-of-machine-learning-trends-applications/>)

Pembelajaran mesin merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit.

Dataset UNSW-NB15 adalah kumpulan data lalu lintas jaringan yang mencakup berbagai jenis serangan siber dan aktivitas normal. Pada tugas ini, mahasiswa diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah dipelajari di kuliah, yaitu KNN, *Gaussian Naive-Bayes*, dan ID3 pada *dataset* UNSW-NB15.

Bagian 2. Implementasi Algoritma

2.1. *K-Nearest Neighbour (KNN)*

Algoritma ini diimplementasikan dalam sebuah kelas yang bernama KNN dengan 3 atribut wajib dan 1 atribut opsional, yaitu: *integer* 'k' untuk menyimpan banyaknya *nearest neighbour* yang diinginkan; *string* 'distance_metric' untuk menyimpan jenis metrik pengukuran distance; DataFrame 'data' untuk menyimpan data *train*; dan atribut opsional *integer* 'p' untuk menyimpan nilai p apabila menggunakan metrik Minkowski Distance.

Konfigurasi algoritma (pengaturan nilai 'k', 'distance_metric', dan 'p') dilakukan pada konstruktor kelas. Konfigurasi data *train* dilakukan pada metode 'fit' agar sesuai dengan *pipeline* yang dibuat. Untuk memprediksi kelas sebuah DataFrame, kelas ini menggunakan sebuah metode yang bernama 'predict' yang menerima parameter 'data_to_predict' berupa DataFrame.

Proses yang terjadi pada metode ini adalah sebagai berikut: memisahkan *feature* target, yaitu *attack_cat*, dengan *feature* lainnya dari data *train*; mengambil nilai-nilai yang ada pada DataFrame (*train* dan *test*) sehingga tipe datanya menjadi np.ndarray; melakukan kalkulasi jarak antara data-data pada data *train* dan *test* dengan bantuan fungsi 'cdist' dari Scipy; mengambil sebanyak k data dari data *train* dengan 'distance' terdekat terhadap setiap data pada data *test*; mengambil kelas mayoritas dari setiap k data tersebut; dan mengembalikan sebuah *array* yang berisi prediksi kelas data *test*. Implementasi kelas KNN dapat dilihat pada potongan kode berikut.

```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self) -> None:
        self.k = 5
        self.distance_metric = "3"

        if ((self.distance_metric == "minkowski") or
            (self.distance_metric == "3")):
            self.p = 10

    def calculateManhattanDistance(self, test_data: np.ndarray,
                                  train_data: np.ndarray):
        distance = cdist(test_data, train_data, metric='cityblock')

        return distance

    def calculateEuclideanDistance(self, test_data: np.ndarray,
                                   train_data: np.ndarray):
        distance = cdist(test_data, train_data, metric='euclidean')

        return distance
```

```

    def calculateMinkowskiDistance(self, test_data: np.ndarray,
train_data: np.ndarray):
        distance = cdist(test_data, train_data, metric='minkowski',
p=self.p)

        return distance

    def calculateDistance(self, test_data: np.ndarray, train_data:
np.ndarray):
        if ((self.distance_metric == "manhattan") or
(self.distance_metric == "1")):
            return self.calculateManhattanDistance(test_data, train_data)
        elif ((self.distance_metric == "euclidean") or
(self.distance_metric == "2")):
            return self.calculateEuclideanDistance(test_data, train_data)
        elif ((self.distance_metric == "minkowski") or
(self.distance_metric == "3")):
            return self.calculateMinkowskiDistance(test_data, train_data)

    def fit(self, X, y):
        X = pd.DataFrame(X)
        y = pd.DataFrame(y)
        train_data = pd.concat([X, y], axis=1)
        sample_fraction = min(1, 10000 / len(train_data))
        train_data = train_data.groupby('attack_cat',
group_keys=False).apply(lambda x:
x.sample(frac=sample_fraction)).reset_index(drop=True)
        self.data = train_data

    def predict(self, data_to_predict: pd.DataFrame):
        class_predictions = []
        features = self.data.columns.difference(["attack_cat"])
        train_data = self.data[features].values
        train_labels = self.data['attack_cat'].values
        test_data = data_to_predict[features].values

        distances = self.calculateDistance(test_data, train_data)

        for i in range(len(test_data)):
            k_nearest_indices = distances[i].argsort()[:self.k]
            k_nearest_labels = train_labels[k_nearest_indices]
            unique, counts = np.unique(k_nearest_labels,
return_counts=True)
            class_predictions.append(unique[np.argmax(counts)])

        return class_predictions

```

2.2. Gaussian Naive Bayes (NB)

Gaussian Naive Bayes merupakan metode pembelajaran mesin variasi dari *Naive Bayes Classifier* yang digunakan untuk memprediksi kelas dari data yang fiturnya bertipe

continuous. Sama seperti NB yang standar, cara kerja algoritma ini adalah menghitung probabilitas posterior dari tiap kelas label yang diperoleh dari hasil perkalian probabilitas kondisional untuk tiap fitur dan menjadikan kelas dengan probabilitas tertinggi sebagai hasil dari prediksi. Bedanya adalah pada *Gaussian Naive Bayes*, nilai probabilitas posteriornya diperoleh dari hasil kalkulasi yang melibatkan mean dan juga standar deviasi dari data latih.

$$p(C_k | \mathbf{x}) = \frac{p(C_k) p(\mathbf{x} | C_k)}{p(\mathbf{x})}$$

Gambar 2. Probabilitas Posterior Naive Bayes

Probabilitas posterior dibaca sebagai probabilitas kelas C_k terjadi bila fitur bernilai x . Persamaan inilah yang akan menjadi landasan untuk menghitung probabilitas gabungan untuk membentuk model Naive Bayes. Probabilitas prior $p(C_k)$ merupakan peluang kejadian data memiliki kelas C_k . *Likelihood* $p(x|C_k)$ merupakan peluang kejadian data memiliki fitur x jika diketahui kelas C_k . *Evidence* $p(x)$ merupakan peluang kejadian data memiliki fitur x .

$$\begin{aligned} p(C_k | x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &= p(C_k) p(x_1 | C_k) p(x_2 | C_k) p(x_3 | C_k) \dots \\ &= p(C_k) \prod_{i=1}^n p(x_i | C_k), \end{aligned}$$

Gambar 3. Probabilitas Gabungan Naive Bayes

Berikut merupakan cara untuk menghitung nilai probabilitas gabungan dari sebuah kelas. Nilai probabilitas gabungan diperoleh dengan cara mengalikan probabilitas prior dan juga probabilitas kondisional tiap fitur menggunakan *chain rule*. Hal tersebut dilakukan untuk semua kelas yang mungkin dan kelas yang memiliki nilai probabilitas tertinggi dijadikan sebagai hasil dari prediksi.

Seperti yang sudah dijelaskan sebelumnya, perbedaan *Gaussian Naive Bayes* terletak di cara menghitung probabilitas kondisional. Pada kasus ini tidak mungkin menghitung nilai probabilitas kondisional dengan cara yang sudah dijelaskan untuk tiap fitur karena fiturnya tidak bernilai diskrit, tetapi *continuous*.

$$p(x = v | C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Gambar 4. Probabilitas Kondisional Gaussian Naive Bayes

Pada *Gaussian Naive Bayes*, nilai probabilitas kondisional diperoleh dengan menggunakan formula pada Gambar 4. Formula merupakan formula dari distribusi normal (*gaussian*) yang menggunakan nilai mean dan juga standar deviasi dari fitur terkait. Perhitungan probabilitas gabungan kemudian dilanjutkan seperti halnya Naive Bayes yang standar.

Pada implementasi kode, ada sedikit penyesuaian untuk cara perhitungan probabilitas gabungungan. Karena nilai standar deviasi tidak jarang senganget kecil (diantara nol dan satu), perhitungan perkalian diubah menjadi penjumlahan dalam basis logaritma untuk menghindari terjadinya error akibat perkalian dengan angka yang sangat kecil.

```
class CustomGaussianNB(BaseEstimator, ClassifierMixin):

    def __init__(self):

        self.smoothing = 1e-9
        self.label_encoder = LabelEncoder()
        self.model = {}
        self.classes = None
        self.class_priors = None
        self.fitted = False

    def fit(self, X, y):

        if self.fitted:
            return self

        if isinstance(X, pd.DataFrame):
            X = X.to_numpy()
        if isinstance(y, pd.Series):
            y = y.to_numpy()

        y_encoded = self.label_encoder.fit_transform(y)
        self.classes = self.label_encoder.classes_
        class_counts = np.bincount(y_encoded)
        self.class_priors = np.log(class_counts / len(y_encoded))

        self.model = {}
        for label in np.unique(y_encoded):
            class_data = X[y_encoded == label]
            self.model[label] = {}

            for i in range(X.shape[1]):
                feature_data = class_data[:, i]
                mean = np.mean(feature_data)
                std = np.std(feature_data)

                std = max(std, self.smoothing)
                self.model[int(label)][int(i)] = {'mean': mean, 'std':

std}

        self.fitted = True
        return self
```

```

def predict(self, X):

    if isinstance(X, pd.DataFrame):
        X = X.to_numpy()

    predictions = []

    for row in X:
        log_likelihoods = {}

        predicted_label_encoded = None
        max_log_likelihood = float('-inf')

        for label in self.model.keys():
            log_likelihood = self.class_priors[label]

            for i in range(len(row)):
                mean = self.model[label][i]['mean']
                std = self.model[label][i]['std']

                coefficient = -np.log(std * np.sqrt(2 * np.pi))
                exponent = -0.5 * ((row[i] - mean) / std) ** 2

                log_likelihood += coefficient + exponent

            if predicted_label_encoded == None or log_likelihood >
max_log_likelihood:
                max_log_likelihood = log_likelihood
                predicted_label_encoded = label

            log_likelihoods[label] = log_likelihood

        predicted_class =
self.label_encoder.inverse_transform([predicted_label_encoded])[0]
        predictions.append(predicted_class)

    return np.array(predictions)

def save_model(self, model_path, download=False):

    if not self.fitted:
        raise ValueError("Model is not fitted. Please fit the model
before saving.")

    model_data = {
        'class_priors': self.class_priors.tolist(),
        'classes': self.classes.tolist(),
        'model': {},
    }

    for label in self.model:
        model_data['model'][str(label)] = {}
        for i in self.model[label]:
            model_data['model'][str(label)][str(i)] = {

```



```

        'mean': str(self.model[label][i]['mean']),
        'std': str(self.model[label][i]['std'])
    }

    with open(model_path, 'w') as f:
        json.dump(model_data, f, indent=4)

    print(f"\nModel saved to: {model_path}")

    if download:
        files.download(model_path)

def load_model(self, model_path):
    with open(model_path, 'r') as f:
        model_data = json.load(f)

    self.class_priors = np.array(model_data['class_priors'])
    self.classes = np.array(model_data['classes'])
    self.model = model_data['model']

    new_model = {}
    for label_str, features in self.model.items():
        label = int(label_str)
        new_model[label] = {}
        for i_str, feature_data in features.items():
            i = int(i_str)
            feature_data['mean'] = float(feature_data['mean'])
            feature_data['std'] = float(feature_data['std'])
            new_model[label][i] = feature_data

    self.model = new_model
    self.label_encoder.classes_ =
np.array(np.array(model_data['classes']))

    self.fitted = True

```

Potongan kode di atas merupakan implementasi kami terhadap Gaussian Naive Bayes dengan mekanisme seperti yang sudah dijelaskan sebelumnya. Implementasi kami wujudkan dalam bentuk kelas CustomGaussianNB yang meng-inherit kelas BaseEstimator dan ClassifierMixin dari pustaka sklearn. Kelas ini memiliki lima buah atribut:

1. ***smoothing***

Yaitu nilai “penghalus” untuk mengganti nilai standar deviasi yang bernilai nol akan diubah menjadi nilai smoothing ini yang merupakan nilai sangat kecil yaitu 10^{-9} .

2. ***label_encoder***

Yaitu objek dari pustaka sklearn yang akan digunakan untuk mengkonversi nilai label dari *string* ke *integer*.

3. ***model***
Yaitu sebuah data bertipe *dictionary* untuk menyimpan hasil pembelajaran.
4. ***classes***
Yaitu sebuah *list* yang menyimpan semua kelas label yang mungkin.
5. ***class_priors***
Yaitu sebuah *list* yang menyimpan nilai probabilitas prior dari tiap kelas.
6. ***fitted***
Yaitu sebuah variabel bertipe *boolean* untuk menandakan apakah objek sudah belajar atau belum. Nilai atribut ini dapat berubah setelah melakukan pembelajaran atau setelah melakukan pemuatan hasil pembelajaran dari *external file*.

Selain atribut yang disebutkan di atas, kelas ini juga memiliki lima buah metode:

1. **`__init__(self)`**
Metode konstruktor.
2. **`fit(self, X, y)`**
Metode ini merupakan metode yang dipanggil untuk melakukan pembelajaran. Terdapat dua buah argumen yaitu *X* yang merupakan *list of tuples* yang tiap *tuple* nya merupakan nilai fitur dari data dan juga *y* yang merupakan kelas label dari *tuple* yang sejajar. Method ini akan mempopulasi nilai dari atribut *models*.

Setelah metode ini dijalankan akan dihasilkan *models* yang merupakan sebuah *dictionary* dengan data nilai mean dan standar deviasi untuk setiap fitur dari setiap label.
3. **`predict(self, X)`**
Metode ini merupakan metode yang dipanggil untuk melakukan prediksi kelas dari sebuah dataset. Terdapat sebuah argumen *X* yang merupakan *list of tuples* yang tiap *tuple* nya merupakan nilai fitur dari data. Metode ini mengembalikan list label kelas hasil prediksi.
4. **`save_model(self, model_path, download=False)`**
Metode ini digunakan untuk menyimpan model hasil pembelajaran ke *external file*.
5. **`load_model(self, model_path)`**
Metode ini digunakan untuk memuat model hasil pembelajaran dari *external file*.

2.3. Iterative Dichotomiser 3 (ID3)

Algoritma *Iterative Dichotomiser 3* (ID3) merupakan salah satu jenis algoritma yang digunakan untuk membuat *decision tree* dari sebuah dataset. Algoritma ini bekerja dengan memanfaatkan beberapa parameter atau variabel penilaian (*measurement*), yakni *Entropy* (H) dan *Information Gain* (IG). *Entropy* adalah suatu pengukuran (*measurement*) ketidakpastian pada suatu data atau set.

Dalam mengukur *Entropy*, jika kita memiliki data atau set S , *Entropy* dari set tersebut $H(S)$ dapat dihitung:

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

Gambar 5 Perhitungan Entropy pada Set S

Penjelasan dari persamaan tersebut adalah *Entropy* dari set S adalah penjumlahan dari negatif proporsi kelas x dari seluruh set S dikali \log_2 dari proporsi kelas x dari seluruh set S . Hasil dari perhitungan tersebut (*Entropy*) akan merepresentasikan ketidakpastian dan digunakan sebagai salah satu komponen dalam menghitung *Information Gain* (IG). *Information Gain* sendiri merupakan komponen yang digunakan untuk menentukan atribut terbaik yang akan digunakan sebagai *node* dalam tree.

Dalam menghitung nilai *Information Gain*, jika kita memiliki atribut A pada sebuah set atau data S , *Information Gain* dari atribut tersebut $IG(S, A)$, dapat dihitung:

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) = H(S) - H(S|A).$$

Gambar 6 Perhitungan Entropy pada Atribut A

Penjelasan dari persamaan tersebut adalah *Information Gain* dari atribut A adalah hasil perhitungan *Entropy* S , $H(S)$, dikurang penjumlahan dari perkalian proporsi dari kelas t dan total data di set S dan *Entropy* dari kelas t . Persamaan tersebut dapat juga dinyatakan sebagai *Entropy* dari set S dikurang *Entropy* S terhadap atribut A .

```
class ID3DecisionTree(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.tree: Optional[Dict[str, Any]] = None
        self.label: str = ""

    def calculate_entropy(self, data: pd.DataFrame) -> float:
        _, counts = np.unique(data.iloc[:, -1], return_counts=True)
        probabilities = counts / counts.sum()
        return -np.sum(probabilities * np.log2(probabilities + 1e-9))

    def calculate_information_gain(self, data: pd.DataFrame, feature:
str) -> float:
```

```

        total_entropy = self.calculate_entropy(data)
        unique_values = data[feature].unique()
        weighted_entropy = 0

        for value in unique_values:
            subset = data[data[feature] == value]
            weighted_entropy += (len(subset) / len(data)) *
self.calculate_entropy(subset)

        return total_entropy - weighted_entropy

    def find_best_splits(self, data: pd.DataFrame, feature: str,
max_splits: int) -> List[float]:
        sorted_data = data.sort_values(feature)
        unique_values = sorted_data[feature].unique()

        print(f"Unique value length of column '{feature}' is
{len(unique_values)}")

        if len(unique_values) <= 1:
            return []

        max_splits = min(max_splits, len(unique_values) - 1)

        split_points = []
        last_split = None
        for i in range(1, max_splits + 1):
            quantile = i / (max_splits + 1)
            split_value = np.quantile(unique_values, quantile)

            if last_split is None or not np.isclose(split_value,
last_split):
                split_points.append(split_value)
                last_split = split_value

        split_points = sorted(set(split_points))

        if len(split_points) < 1:
            return []

        return split_points

    def calculate_information_gain_on_split(self, data: pd.DataFrame,
left: pd.DataFrame, right: pd.DataFrame) -> float:
        total_entropy = self.calculate_entropy(data)
        left_weight = len(left) / len(data)
        right_weight = len(right) / len(data)
        return total_entropy - (left_weight *
self.calculate_entropy(left) + right_weight *
self.calculate_entropy(right))

    def convert_numeric_to_categorical(self, data: pd.DataFrame) ->
pd.DataFrame:

```

```

        for column in data.columns[:-1]:
            if pd.api.types.is_numeric_dtype(data[column]):
                print(f"Processing numeric column: {column}")

                unique_values = data[column].unique()
                if len(unique_values) < 20:
                    max_split = len(unique_values)
                else:
                    max_split = int(len(data) / 100)
                min_val = data[column].min()
                max_val = data[column].max()

                split_points = self.find_best_splits(data, column,
max_splits=max_split)
                if split_points:
                    bins = [min_val] + split_points + [max_val]
                    bins = sorted(set(bins))
                    labels = [f'<= {split_points[0]]' + \
                        [f'{split_points[i]} - {split_points[i + 1]]'
for i in range(len(split_points) - 1)] + \
                        [f'> {split_points[-1]]'

                    data[column] = pd.cut(data[column], bins=bins,
labels=labels, include_lowest=True)

                return data

    def best_feature_to_split(self, data: pd.DataFrame) -> str:
        gains = {feature: self.calculate_information_gain(data, feature)
for feature in data.columns[:-1]}
        return max(gains, key=gains.get)

    def build_tree(self, data: pd.DataFrame) -> Dict[str, Any]:
        target = data.columns[-1]

        if len(data[target].unique()) == 1:
            return data[target].iloc[0]

        if len(data.columns) == 1:
            return data[target].mode().iloc[0]

        best_feature = self.best_feature_to_split(data)
        print(f"Feature {best_feature} processed")

        tree = {best_feature: {}}

        majority_class = data[target].mode().iloc[0]

        for value in data[best_feature].unique():
            subset = data[data[best_feature] ==
value].drop(columns=[best_feature])
            tree[best_feature][value] = self.build_tree(subset)

        tree[best_feature]["unknown"] = majority_class

```

```

        return tree

    def fit(self, X: pd.DataFrame, y: pd.Series) -> Dict[str, Any]:
        self.label = 'attack_cat'

        if isinstance(X, np.ndarray):
            X = pd.DataFrame(X)
            print(f"Converted X to pandas DataFrame. Type of X: {type(X)}; Shape of X: {X.shape}")

        if len(X) != len(y):
            raise ValueError("The number of rows in X and the number of elements in y must be the same.")

        data = X.copy()
        print(f"Target column 'attack_cat':\n{y.head()}")

        try:
            data[self.label] = y.values
            print(f"Data after adding target column:\n{data.head()}")
        except Exception as e:
            print(f"Error occurred while adding target column: {e}")

        processed_data = self.convert_numeric_to_categorical(data)

        print("Building tree")

        columns = [col for col in processed_data.columns if col != self.label]
        columns.append(self.label)

        processed_data = processed_data[columns]
        print(f"Columns after reordering: {processed_data.columns}")

        self.tree = self.build_tree(processed_data)

    def predict_sample(self, tree: dict, sample: pd.Series) -> Any:
        while isinstance(tree, dict):
            feature = next(iter(tree))
            value = sample.get(feature)

            if isinstance(value, pd.Series):
                value = value.iloc[0]

            if pd.isna(value):
                return "default_class"

            branches = tree[feature]

            found_match = False
            for key in branches:
                if isinstance(key, str) and ' - ' in key:
                    lower_bound, upper_bound = key.split(' - ')
                    lower_bound = float(lower_bound)
                    upper_bound = float(upper_bound)

```

```

        if isinstance(value, (int, float)):
            if lower_bound <= value <= upper_bound:
                tree = branches[key]
                found_match = True
                break
            else:
                continue

        elif isinstance(key, str) and any(op in key for op in
['<=', '>=', '<', '>', '==', '!=']):
            operator, threshold = key.split(' ', 1)
            threshold = float(threshold) if operator not in
['==', '!='] else threshold

        if isinstance(value, (int, float)):
            if operator == '<=':
                if value <= threshold:
                    tree = branches[key]
                    found_match = True
                    break
            elif operator == '<':
                if value < threshold:
                    tree = branches[key]
                    found_match = True
                    break
            elif operator == '>=':
                if value >= threshold:
                    tree = branches[key]
                    found_match = True
                    break
            elif operator == '>':
                if value > threshold:
                    tree = branches[key]
                    found_match = True
                    break

            elif operator == '==':
                if value == threshold:
                    tree = branches[key]
                    found_match = True
                    break
            elif operator == '!=':
                if value != threshold:
                    tree = branches[key]
                    found_match = True
                    break
        else:
            continue

    elif isinstance(value, (str, int, float)):
        if value == key:
            tree = branches[key]
            found_match = True
            break
    else:

```

```

        continue

    if not found_match:
        if "unknown" in branches:
            # return "Normal"
            return branches["unknown"]
        else:
            return "default_class"

    return tree

def predict(self, df: pd.DataFrame) -> pd.Series:
    if isinstance(df, np.ndarray):
        df = pd.DataFrame(df)
        print(f"Converted df to pandas DataFrame. Type of df: {type(df)}; Shape of df: {df.shape}")

    predictions = df.apply(lambda row: self.predict_sample(self.tree,
row), axis=1)

    return predictions

def save_model(self, file_path: str) -> None:
    if self.tree is None:
        raise ValueError("Model is not trained yet, cannot save.")
    with open(file_path, 'w') as f:
        json.dump(self.tree, f, indent=4)

def load_model(self, file_path: str) -> None:
    with open(file_path, 'r') as f:
        self.tree = json.load(f)

```


Bagian 3. Pembersihan dan Prapemrosesan Data

3.1. Pembersihan Data

Pembersihan data adalah langkah awal yang paling penting dalam mempersiapkan *dataset* untuk melatih sebuah model *Artificial Intelligence*. Data awal seringkali mengandung berbagai kesalahan termasuk nilai kosong, nilai yang duplikat, dan data yang inkonsisten. Proses pembersihan data meliputi menangani nilai kosong dengan imputasi atau penghapusan; menangani *outlier* dengan menghapus atau mentransformasi nilai; memvalidasi data agar konsisten dan sesuai tipe yang diharapkan; menghapus data duplikat yang dapat mempengaruhi hasil pelatihan; serta melakukan perubahan fitur, seperti normalisasi, perubahan skala nilai, atau *encoding*, untuk meningkatkan relevansi data dalam model.

3.1.1. Penanganan Nilai Kosong

Untuk menangani nilai kosong, kami melakukan *Missing Value Analysis* terlebih dahulu. Berdasarkan analisis pada tabel, beberapa kolom memiliki persentase nilai yang hilang sekitar 5%, dengan tipe data yang beragam, termasuk numerik (*float* dan *integer*) serta kategori (nominal). Kolom-kolom numerik, seperti *dur*, *sbytes*, dan *dbytes*, memiliki nilai yang hilang yang signifikan namun tidak dominan, sedangkan kolom kategori, seperti *statedan service*, juga menunjukkan pola kehilangan serupa. Berikut adalah hasil lengkap dari *Missing Value Analysis* yang kami lakukan.

Tabel 1. Hasil *Missing Value Analysis*

No	Name	Missing Count	Missing Percentage	Type
0	state	8805	5.021644	nominal
1	dur	8722	4.974307	float
2	sbytes	8561	4.882486	integer
3	dbytes	8869	5.058144	integer
4	sttl	8825	5.03305	integer
5	dttl	8654	4.935526	integer
6	sloss	8794	5.01537	integer
7	dloss	8978	5.120308	integer
8	service	8791	5.013659	nominal
9	sload	8786	5.010808	float
10	dload	8837	5.039894	float
11	spkts	8654	4.935526	integer

12	dpkts	8686	4.953776	integer
14	is_sm_ips_ports	8746	4.987995	binary
15	ct_state_ttl	8635	4.92469	integer
16	ct_flw_http_mthd	8647	4.931533	integer
17	is_ftp_login	8647	4.931533	binary
18	ct_ftp_cmd	8842	5.042745	integer
19	ct_srv_src	8851	5.047878	integer
20	ct_srv_dst	8774	5.003964	integer
21	ct_dst_ltm	8738	4.983432	integer
22	ct_src_ltm	8823	5.031909	integer
23	ct_src_dport_ltm	8775	5.004534	integer
24	ct_dst_sport_ltm	8788	5.011948	integer
25	ct_dst_src_ltm	8895	5.072972	integer
26	swin	8740	4.984573	integer
27	dwin	8779	5.006815	integer
28	stcpb	8672	4.945791	integer
29	dtcpb	8803	5.020503	integer
30	smean	8788	5.011948	integer
31	dmean	8855	5.050159	integer
32	trans_depth	8785	5.010237	integer
33	response_body_len	8791	5.013659	integer
34	proto	8826	5.03362	nominal
35	sjit	8738	4.983432	float
36	djit	8846	5.045027	float
37	sinpkt	8707	4.965752	float
38	dinpkt	8734	4.981151	float
39	tcprtt	8836	5.039323	float
40	synack	8736	4.982292	float
41	ackdat	8595	4.901877	float

Kami merancang kelas *CustomImputer* untuk menangani nilai yang kosong pada data. Untuk kolom numerik, kami mengganti nilai dengan rata-rata (mean) agar tidak mengubah distribusi data secara signifikan. Sedangkan untuk kolom kategori, kami mengganti nilai yang kosong dengan nilai yang paling sering muncul. Selain itu, kami juga mengubah nilai '-' menjadi NaN untuk memudahkan pengolahan. Dengan pendekatan ini, kami memastikan bahwa dataset yang dihasilkan bebas dari nilai hilang, konsisten, dan siap untuk analisis atau modeling lebih lanjut.

```
class CustomImputer(BaseEstimator, TransformerMixin):
    def __init__(self, numeric_columns=None,
categorical_columns=None):
        self.numeric_columns = numeric_columns
        self.categorical_columns = categorical_columns
        self.numeric_imputer_ = SimpleImputer(strategy='mean')
        self.categorical_imputer_ =
SimpleImputer(strategy='most_frequent')

    def fit(self, X, y=None):
        if self.numeric_columns is None:
            self.numeric_columns =
X.select_dtypes(include=['int64', 'float64']).columns.tolist()

            if self.categorical_columns is None:
                self.categorical_columns =
X.select_dtypes(include=['object', 'category']).columns.tolist()

                if len(self.numeric_columns) > 0: # Check if
numeric_columns is not empty
                    self.numeric_imputer_.fit(X[self.numeric_columns])

                    if len(self.categorical_columns) > 0: # Check if
categorical_columns is not empty
                        self.categorical_imputer_.fit(X[self.categorical_columns])

                    return self

    def transform(self, X):
        X_transformed = X.copy()

        X_transformed.replace('-', np.nan, inplace=True)

        if len(self.numeric_columns) > 0: # Check if
numeric_columns is not empty
            X_transformed[self.numeric_columns] =
self.numeric_imputer_.transform(X_transformed[self.numeric_columns]
)

            if len(self.categorical_columns) > 0: # Check if
categorical_columns is not empty
                X_transformed[self.categorical_columns] =
self.categorical_imputer_.transform(X_transformed[self.categorical_
```

```
columns])

    return X_transformed

def fit_transform(self, X, y=None):
    return self.fit(X, y).transform(X)
```

3.1.2. Penanganan Nilai Outlier

Dalam analisis dataset ini, kami memutuskan untuk tidak melakukan penanganan outlier karena mempertimbangkan kasus yang berkaitan dengan serangan siber. Kami berpikir bahwa *outlier*, yang biasanya dianggap sebagai data yang menyimpang, justru dapat memiliki makna yang signifikan dalam kasus ini. Dalam serangan siber, outlier sering kali merepresentasikan aktivitas yang tidak normal atau pola serangan yang berbeda dari lalu lintas jaringan yang biasa. Misalnya, lonjakan nilai pada kolom seperti *sbytes* atau *dbytes* dapat mencerminkan anomali seperti serangan *Denial of Service* (DoS).

Dengan tidak menghapus atau mengubah outlier, kami memastikan bahwa informasi penting yang dapat mengindikasikan serangan tidak hilang. Penanganan yang salah terhadap outlier berisiko menghilangkan pola penting yang dapat membantu dalam mendeteksi atau mengklasifikasikan serangan. Oleh karena itu, pendekatan ini kami ambil untuk menjaga integritas data dan memastikan bahwa analisis atau model klasifikasi yang dibangun dapat menangkap anomali dengan lebih baik, yang merupakan inti dari deteksi serangan siber.

3.1.3. Penanganan Nilai Duplikat

Menghapus duplikat data merupakan langkah penting untuk menjaga integritas data dan memastikan analisis yang akurat. Duplikasi dapat menyebabkan bias dalam model *Artificial Intelligence*, meningkatkan risiko *overfitting*, dan memperbesar ukuran dataset secara tidak perlu, sehingga meningkatkan biaya komputasi. Fungsi *removeDuplicates* menggunakan metode *drop_duplicates()* untuk menghapus entri yang sama dalam dataset, memastikan data lebih bersih, efisien, dan konsisten untuk analisis lebih lanjut.

```
def removeDuplicates(df):
    df = df.drop_duplicates()
    return df
```

3.1.4. Rekayasa Fitur

Rekayasa fitur atau *Feature engineering* adalah proses membuat, mengubah, atau mengurangi fitur dalam dataset untuk meningkatkan kemampuan model

Artificial Intelligence dalam belajar pola dan membuat prediksi yang akurat. Berikut adalah kelas *FeatureEngineer* yang kami buat.

```
class FeatureEngineer(BaseEstimator, TransformerMixin):
    def __init__(self, numeric_columns=None,
categorical_columns=None):
        self.numeric_columns = numeric_columns
        self.categorical_columns = categorical_columns
        self.columns_to_drop = []

    def fit(self, X, y=None):
        # Ensure target column is passed as y
        if y is None:
            raise ValueError("The target variable 'y' must be
provided for the feature engineering process.")

        # Step 1: Identify numeric columns with low variance
        if len(self.numeric_columns) > 0:
            numeric_data = X[self.numeric_columns]
            variances = numeric_data.var()
            low_variance_columns = variances[variances <
0.01].index.tolist()
        else:
            low_variance_columns = []

        # Step 2: Remove irrelevant columns
        irrelevant_columns = ['id']

        # Step 3: Feature importance analysis (numeric vs
categorical target)
        if len(self.numeric_columns) > 0:
            numeric_data = X[self.numeric_columns]
            anova_f_values, _ = f_classif(numeric_data, y)
            low_importance_columns = [
                col for col, score in zip(self.numeric_columns,
anova_f_values) if score < 0.1
            ]
        else:
            low_importance_columns = []

        # Step 4: Redundancy analysis (highly correlated numeric
features)
        if len(self.numeric_columns) > 0:
            corr_matrix = X[self.numeric_columns].corr().abs()
            upper_triangle =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(bool))
            redundant_columns = [column for column in
upper_triangle.columns if any(upper_triangle[column] > 0.9)]
        else:
            redundant_columns = []

        # Combine all columns to drop
        self.columns_to_drop = list(set(
            low_variance_columns + irrelevant_columns +
low_importance_columns + redundant_columns
```

```

    ))

    return self

    def transform(self, X):
        # Drop identified columns from X
        X_transformed = X.drop(columns=self.columns_to_drop,
errors='ignore')
        return X_transformed

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)

```

Kelas *FeatureEngineer* yang kami buat ini bertujuan untuk mengidentifikasi fitur yang tidak relevan, memiliki variansi rendah, atau redundan, sekaligus memastikan bahwa fitur yang digunakan memiliki hubungan signifikan dengan fitur target yaitu *attack_cat*.

Langkah-langkah dalam kelas ini digunakan untuk mengoptimalkan relevansi fitur, mengurangi risiko *overfitting*, dan meningkatkan efisiensi pelatihan model. Identifikasi fitur dengan variansi rendah dilakukan karena fitur tersebut tidak memberikan informasi yang cukup untuk membedakan target, sehingga akan menyederhanakan model. Kolom yang tidak relevan, seperti 'id', dihapus karena tidak memiliki kontribusi terhadap prediksi model. Kami juga menganalisis pentingnya fitur menggunakan ANOVA F-test mengevaluasi hubungan antara fitur numerik dan target untuk mengidentifikasi fitur dengan pengaruh kecil. Selain itu, analisis korelasi digunakan untuk menghapus fitur yang sangat berkorelasi (>0.9) guna mengurangi redundansi informasi. Pendekatan berbasis statistik dan korelasi ini memastikan dataset berkualitas lebih baik sebelum modeling.

3.2. Prapemrosesan Data

3.2.1. *Feature Scaling*

Feature scaling adalah teknik prapemrosesan data yang bertujuan untuk menyamakan skala dari fitur-fitur dalam dataset sehingga algoritma *Artificial Intelligence* dapat bekerja secara efektif. Skala fitur yang berbeda dapat mempengaruhi kinerja model, terutama untuk algoritma yang sensitif terhadap skala seperti *K-Nearest Neighbors* (KNN). Beberapa metode umum dalam *feature scaling* adalah normalisasi (Min-Max Scaling), standarisasi (Z-score Scaling), *robust scaling* (berbasis IQR), dan *Log Transformation*.

Kelas *FeatureScaler* yang kami buat dirancang untuk mengotomatiskan proses *scaling* pada fitur numerik dalam dataset. Dengan parameter scaler, tiap algoritma dapat menentukan metode *scaling* yang diinginkan, seperti *StandardScaler*, *MinMaxScaler*, atau lainnya dari library *sklearn*. Jika tidak diberikan, fungsi secara

default menggunakan *StandardScaler*, yang merupakan metode standarisasi berbasis Z-score. Kelas ini akan memastikan bahwa fitur numerik dalam dataset memiliki skala yang seragam. Berikut adalah implementasi kelas yang kami buat.

```
class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, scaler=None, numeric_columns=None):
        self.scaler = scaler if scaler else StandardScaler()
        self.columns = numeric_columns

    def fit(self, X, y=None):
        self.columns = X.select_dtypes(include=['int64',
'float64']).columns if self.columns is None else self.columns
        self.scaler.fit(X[self.columns])
        return self

    def transform(self, X):
        X = X.copy()
        X[self.columns] = self.scaler.transform(X[self.columns])
        return X

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)
```

3.2.2. *Feature Encoding*

Data yang digunakan untuk pembelajaran mesin seringkali memiliki tipe data yang berbeda. Tipe data string merupakan tipe data yang tidak cocok jika digunakan untuk melakukan perhitungan. Maka dari itu, fitur dengan tipe data ini kami *encode* menjadi bentuk numerik yang dapat lebih mudah diolah. Beberapa metode *encoding* yang kami implementasikan adalah *one-hot encoding*, *label encoding*, dan *target encoding*. Berikut adalah kelas *FeatureEncoder* yang kami buat.

```
class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, categorical_columns=None,
encoding_strategy='onehot'):
        self.categorical_columns = categorical_columns
        self.encoding_strategy = encoding_strategy
        self.encoders = {}

    def fit(self, X, y=None):
        X = X.copy()

        if self.encoding_strategy not in ['target', 'label',
'onehot']:
            raise ValueError("encoding_strategy must be one of
'target', 'label', or 'onehot'")

        # Automatically detect categorical columns if not specified
        if self.categorical_columns is None:
            self.categorical_columns =
X.select_dtypes(include=['object', 'category']).columns.tolist()
```

```

        # Fit the encoder for each column
        for col in self.categorical_columns:
            if self.encoding_strategy == 'onehot':
                enc = OneHotEncoder(sparse=False,
handle_unknown='ignore')
            elif self.encoding_strategy == 'label':
                enc = LabelEncoder()
            elif self.encoding_strategy == 'target':
                if y is None:
                    raise ValueError("Target encoding requires the
target variable y.")
                enc = TargetEncoder()

            if self.encoding_strategy == 'label':
                enc.fit(X[col].astype(str))
            else:
                enc.fit(X[[col]].astype(str), y)

            self.encoders[col] = enc # Save the fitted encoder for
the column

        return self

    def transform(self, X):
        X = X.copy()

        for col, enc in self.encoders.items():
            if self.encoding_strategy == 'label':
                transformed =
pd.Series(enc.transform(X[col].astype(str)), index=X.index,
name=col)
                X[col] = transformed
            else:
                transformed = enc.transform(X[[col]].astype(str))

            # Convert the transformed array or sparse matrix
into a DataFrame
            if hasattr(transformed, "toarray"): # Handle
sparse matrices
                transformed = pd.DataFrame(
                    transformed.toarray(),
                    columns=[f"{col}_{i}" for i in
range(transformed.shape[1])],
                    index=X.index,
                )
            elif isinstance(transformed, np.ndarray): # Handle
numpy arrays
                transformed = pd.DataFrame(
                    transformed,
                    columns=[f"{col}_{i}" for i in
range(transformed.shape[1])],
                    index=X.index,
                )
            elif isinstance(transformed, pd.DataFrame): #
Handle DataFrames

```



```

        transformed.columns = [f"{col}_{c}" for c in
transformed.columns]

        # Replace the column or concatenate with the
DataFrame
        X = pd.concat([X.drop(columns=[col]), transformed],
axis=1)

        return X

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)

```

3.2.3. *Handling Imbalanced Dataset*

Dataset yang tidak seimbang artinya sebagian besar dari *tuple* data memiliki kelas yang sama dan ada kelas yang hanya diwakili oleh sedikit *tuple*. Hal ini dapat mempengaruhi proses pembelajaran yang mengakibatkan model hanya bagus dalam memprediksi kelas tertentu. Ada dua metode utama untuk mengatasi masalah ini, *oversampling* dan *undersampling*.

Oversampling dilakukan ketika ingin menyeimbangkan data dengan kelas minoritas. Hal ini dapat dicapai dengan menggunakan kelas SMOTE dari pustaka *sklearn*. Kelas ini mampu menciptakan data sintetis dari data minoritas yang ada sehingga frekuensi data dengan kelas minoritas tadi dapat menyaingi kelas mayoritas. Metode ini dapat mengurangi *overfitting* terhadap kelas mayoritas dan sering dipakai ketika ada ketidak seimbangan data yang signifikan.

Undersampling dilakukan ketika ingin menyeimbangkan data dengan kelas mayoritas. Hal ini dapat dicapai dengan kelas *RandomUnderSampler* dari pustaka *sklearn*. Kelas ini bekerja dengan cara mengurangi data dengan kelas mayoritas pada dataset secara acak sehingga jumlahnya bisa setara dengan kelas minoritas. Keuntungan metode ini adalah mudah dilakukan dan juga mengurangi beban komputasi karena data yang diolah akan lebih sedikit. Metode ini sering digunakan ketika *dataset* terlalu besar dan perlu penyeimbangan. Berikut adalah implementasi dari *SamplingHandler* yang kami buat.

```

class SamplingHandler(BaseEstimator, TransformerMixin):
    def __init__(self, target_column, strategy='both',
sampling_ratio=1.0, random_state=42):
        if strategy not in ['undersample', 'oversample', 'both']:
            raise ValueError("Strategy must be 'undersample',
'oversample', or 'both'.")

```

```

        self.target_column = target_column
        self.strategy = strategy
        self.sampling_ratio = sampling_ratio
        self.random_state = random_state
        self.smote = SMOTE(sampling_strategy=sampling_ratio,
random_state=random_state)

    def fit(self, X, y=None):
        # SMOTE requires X and y, so it needs to be initialized
with proper data during fitting.
        if self.strategy in ['oversample', 'both'] and y is not
None:
            self.smote.fit(X, y)
            return self

    def transform(self, X, y=None):
        if y is None:
            raise ValueError("y (target column) must be provided
during transform.")

        X, y = X.copy(), y.copy()
        majority_class = y.value_counts().idxmax()
        minority_class = y.value_counts().idxmin()

        # Perform undersampling
        if self.strategy in ['undersample', 'both']:
            majority_data = X[y == majority_class]
            minority_data = X[y == minority_class]

            n_majority_samples = int(len(minority_data) /
self.sampling_ratio)
            majority_data_resampled = resample(
                majority_data,
                replace=False,
                n_samples=n_majority_samples,
                random_state=self.random_state
            )

            X = pd.concat([majority_data_resampled, minority_data])
            y = pd.concat([y.loc[majority_data_resampled.index],
y[y == minority_class]])

        # Perform oversampling
        if self.strategy in ['oversample', 'both']:
            X, y = self.smote.fit_resample(X, y)

        # Shuffle the data to ensure randomness
        X = pd.DataFrame(X).reset_index(drop=True)
        y = pd.Series(y).reset_index(drop=True)

        return X, y

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X, y)

```

3.2.4. Normalisasi Data

Normalisasi data adalah proses untuk mengubah nilai fitur ke dalam skala tertentu (misalnya, antara 0 dan 1) atau ke distribusi tertentu seperti distribusi normal. Hal ini berguna untuk memastikan bahwa model atau proses yang mengasumsikan normalitas data dapat bekerja dengan baik. Selain itu, normalisasi juga membantu mengurangi pengaruh besar kecilnya skala fitur (*magnitude effect*) dalam model. Berikut adalah implementasi dari *DataNormalizer* yang kami buat.

```
class DataNormalizer(BaseEstimator, TransformerMixin):
    def __init__(self, feature_range=(0, 1),
numerical_columns=None, scaler=StandardScaler()):
        self.feature_range = feature_range
        self.numeric_columns = numerical_columns
        self.method = StandardScaler()
        self.scaler = scaler

    def fit(self, X, y=None):
        X = X.copy()

        # Identify numeric columns if not provided
        if self.numeric_columns is None:
            self.numeric_columns =
X.select_dtypes(include=['int64', 'float64']).columns

        # Fit the scaler to the selected numeric columns
        self.scaler.fit(X[self.numeric_columns])

        return self

    def transform(self, X):
        X = X.copy()

        if self.scaler is None:
            raise ValueError("The normalizer must be fitted before
transforming the data.")

        # Apply normalization to numeric columns
        X[self.numeric_columns] =
self.scaler.transform(X[self.numeric_columns])

        return X

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)
```

3.2.5. Pengurangan Dimensi Data

Mengurangi dimensi data dapat mempercepat proses pembelajaran model serta membuat keterikatan nilai antarfitur semakin kuat. Setelah melalui *pipeline* pemrosesan ini, seluruh fitur yang ada sudah berubah menjadi fitur baru hasil pengolahan fitur aslinya.

Pada implementasi kode, kami menggunakan kelas PCA dari pustaka sklearn. Kelas ini dapat dikostumisasi dengan cara memberi *key argument* “n_components” yang bernilai *integer* pada konstruktornya. Nilai ini menyatakan jumlah fitur yang ingin dihasilkan setelah melewati komponen *pipeline* ini. Kelas dari pustaka ini mampu membuat fitur baru hasil reduksi dimensi secara otomatis.

```
class DimensionalityReduction(BaseEstimator, TransformerMixin):
    def __init__(self, method='pca', n_components=2,
random_state=None):
        if method not in ['pca', 'tsne', 'autoencoder']:
            raise ValueError("Method must be one of 'pca', 'tsne',
or 'autoencoder'.")

        self.method = method
        self.n_components = n_components
        self.random_state = random_state
        self.model = None

    def fit(self, X, y=None):
        X = X.copy()

        if self.method == 'pca':
            self.model = PCA(n_components=self.n_components,
random_state=self.random_state)
            self.model.fit(X)
        elif self.method == 'tsne':
            # t-SNE does not require fitting; it works directly in
transform.
            self.model = TSNE(n_components=self.n_components,
random_state=self.random_state)
        elif self.method == 'autoencoder':
            self.model = self._build_autoencoder(X.shape[1],
self.n_components)
            self.model.fit(X, X, epochs=50, batch_size=32,
verbose=0)

        return self

    def transform(self, X):
        X = X.copy()

        if self.model is None:
            raise ValueError("The model must be fitted before
transforming the data.")

        if self.method == 'pca':
            return self.model.transform(X)
        elif self.method == 'tsne':
            return self.model.fit_transform(X)
        elif self.method == 'autoencoder':
            encoder = self.model.get_layer('encoder')
            return encoder.predict(X)
```

```

def fit_transform(self, X, y=None):
    return self.fit(X, y).transform(X)

def _build_autoencoder(self, input_dim, latent_dim):
    # Encoder
    input_layer = layers.Input(shape=(input_dim,))
    encoded = layers.Dense(64, activation='relu')(input_layer)
    encoded = layers.Dense(32, activation='relu')(encoded)
    latent = layers.Dense(latent_dim, activation='relu',
name='encoder')(encoded)

    # Decoder
    decoded = layers.Dense(32, activation='relu')(latent)
    decoded = layers.Dense(64, activation='relu')(decoded)
    output_layer = layers.Dense(input_dim,
activation='linear')(decoded)

    # Autoencoder Model
    autoencoder = Model(inputs=input_layer,
outputs=output_layer)
    autoencoder.compile(optimizer='adam', loss='mse')

    return autoencoder

```

3.3. Pipeline per Algoritma

Setiap algoritma membutuhkan pembersihan data, tetapi prapemrosesan data yang belum tentu sama. Oleh sebab itu, diperlukan *pipeline* yang berbeda-beda untuk setiap algoritma. Berikut merupakan spesifikasinya.

3.3.1. Pipeline *k-Nearest Neighbour*

Pemrosesan data yang digunakan pada algoritma kNN adalah *imputer* dan *encoder*. *Imputer* digunakan untuk mengganti *value* yang hilang agar data dapat lebih akurat diprediksi oleh kNN. *Encoder* digunakan untuk mengubah data kategorikal menjadi numerikal agar fitur-fitur datanya seragam. Awalnya, akan digunakan *scaling* untuk mengubah skala fitur-fitur yang sudah dibuat menjadi numerikal menjadi 0-1 agar perhitungan *distance* lebih *scalable*. Namun, setelah dilakukan eksperimen, ternyata *scaling* membuat hasil prediksi menjadi lebih buruk.

```

pipe = Pipeline([
    ('imputer', CustomImputer(
        numeric_columns=numeric_columns,
        categorical_columns=categorical_columns_filtered
    )),
    ('encoder', FeatureEncoder(
        categorical_columns=categorical_columns_filtered,
        encoding_strategy='target'
    )),
    ('classifier', classifier)
])

```

```
])
```

3.3.2. Pipeline Gaussian Naive-Bayes

Setelah dilakukan eksperimen, diperoleh konfigurasi pipeline yang paling optimal untuk Gaussian Naive Bayes adalah sebagai berikut. *Imputer* adalah *pipeline* yang wajib digunakan karena berfungsi untuk mengatasi nilai yang hilang pada data. Pada kasus ini digunakan kelas SimpleImputer dari pustaka sklearn dengan strategi mean untuk tipe data numerik dan juga strategi modus untuk tipe data kategorikal. *Encoder* juga wajib digunakan untuk model ini karena penting bagi model ini untuk data yang diolah bertipe data numerik. Sayangnya, masih banyak fitur yang bertipe data kategorikal dan perlu di-*encode* terlebih dahulu. *Scaler* dan *Normalizer* perlu digunakan juga agar nilai data numerik rentangnya seragam untuk berbagai fitur agar hasil perhitungannya lebih seimbang. *Dim Reduction* juga digunakan agar beban komputasi lebih rendah akibat berkurangnya dimensi data.

```
pipe = Pipeline([
    ('imputer', CustomImputer(
        numeric_columns=numeric_columns,
        categorical_columns=categorical_columns_filtered
    )),
    ('encoder', FeatureEncoder(
        categorical_columns=categorical_columns_filtered,
        encoding_strategy='target'
    )),
    ('scaler', FeatureScaler(
        scaler=MinMaxScaler(),
        numeric_columns=numeric_columns
    )),
    ('normalizer', DataNormalizer(
        numerical_columns=numeric_columns
    )),
    ('dim_reduction', DimensionalityReduction(
        method='pca',
        n_components=5,
        random_state=42
    )),
    ('classifier', classifier)
])
```

3.3.3. Pipeline Iterative Dichotomiser 3

Melalui beberapa hipotesis dan hasil eksperimen, kita menyeleksi beberapa kriteria untuk digunakan dalam pipeline. Yang pertama, *imputer*, ini sudah jelas kami implementasikan karena banyaknya nilai kosong pada data set baik *training* maupun *testing*. Oleh karena itu, kami implementasikan *imputer* pada *pipeline*. Selain itu, kami juga menggunakan *encoder* untuk meng-*encode* fitur-fitur kategorik menjadi numerik agar data menjadi seragam. Selanjutnya, data tersebut bisa dijalankan algoritma ID3 menggunakan *classifier*.

```
pipe = Pipeline([
    ('imputer', CustomImputer(
        numeric_columns=numeric_columns,
        categorical_columns=categorical_columns_filtered
    )),
    ('encoder', FeatureEncoder(
        categorical_columns=categorical_columns_filtered,
        encoding_strategy='target'
    )),
    ('classifier', classifier)
])
```

Bagian 4. Percobaan dan Analisis

4.1. Percobaan

Tabel 2. Perbandingan Hasil Prediksi Algoritma terhadap Hasil Prediksi Pustaka

No.	Percobaan dan Prediksi																																																																										
1.	Data yang digunakan: data <i>validation</i> yang di- <i>split</i> dari data <i>train</i> Rasio pemisahan <i>train</i> : <i>validation</i> = 66 : 33																																																																										
	Spesifikasi algoritma KNN: k = 5 distance_metric = <i>euclidean</i>																																																																										
	Model yang dibuat sendiri																																																																										
	Prediksi Algoritma KNN	<div>Prediction Validity Evaluation: Accuracy: 0.6371</div> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>Analysis</td><td>0.07</td><td>0.05</td><td>0.06</td><td>442</td></tr><tr><td>Backdoor</td><td>0.02</td><td>0.01</td><td>0.01</td><td>386</td></tr><tr><td>DoS</td><td>0.29</td><td>0.33</td><td>0.31</td><td>2712</td></tr><tr><td>Exploits</td><td>0.39</td><td>0.51</td><td>0.44</td><td>7383</td></tr><tr><td>Fuzzers</td><td>0.34</td><td>0.29</td><td>0.31</td><td>4020</td></tr><tr><td>Generic</td><td>0.95</td><td>0.97</td><td>0.96</td><td>8844</td></tr><tr><td>Normal</td><td>0.79</td><td>0.77</td><td>0.78</td><td>12382</td></tr><tr><td>Reconnaissance</td><td>0.65</td><td>0.34</td><td>0.45</td><td>2320</td></tr><tr><td>Shellcode</td><td>0.17</td><td>0.03</td><td>0.05</td><td>250</td></tr><tr><td>Worms</td><td>0.00</td><td>0.00</td><td>0.00</td><td>29</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.64</td><td>38768</td></tr><tr><td>macro avg</td><td>0.37</td><td>0.33</td><td>0.34</td><td>38768</td></tr><tr><td>weighted avg</td><td>0.64</td><td>0.64</td><td>0.63</td><td>38768</td></tr></tbody></table>					precision	recall	f1-score	support	Analysis	0.07	0.05	0.06	442	Backdoor	0.02	0.01	0.01	386	DoS	0.29	0.33	0.31	2712	Exploits	0.39	0.51	0.44	7383	Fuzzers	0.34	0.29	0.31	4020	Generic	0.95	0.97	0.96	8844	Normal	0.79	0.77	0.78	12382	Reconnaissance	0.65	0.34	0.45	2320	Shellcode	0.17	0.03	0.05	250	Worms	0.00	0.00	0.00	29	accuracy			0.64	38768	macro avg	0.37	0.33	0.34	38768	weighted avg	0.64	0.64	0.63	38768
		precision	recall	f1-score	support																																																																						
Analysis	0.07	0.05	0.06	442																																																																							
Backdoor	0.02	0.01	0.01	386																																																																							
DoS	0.29	0.33	0.31	2712																																																																							
Exploits	0.39	0.51	0.44	7383																																																																							
Fuzzers	0.34	0.29	0.31	4020																																																																							
Generic	0.95	0.97	0.96	8844																																																																							
Normal	0.79	0.77	0.78	12382																																																																							
Reconnaissance	0.65	0.34	0.45	2320																																																																							
Shellcode	0.17	0.03	0.05	250																																																																							
Worms	0.00	0.00	0.00	29																																																																							
accuracy			0.64	38768																																																																							
macro avg	0.37	0.33	0.34	38768																																																																							
weighted avg	0.64	0.64	0.63	38768																																																																							

		<div>Confusion Matrix:</div> <div><pre>[[21 9 137 196 20 11 39 9 0 0] [10 3 127 196 18 6 24 2 0 0] [67 54 882 1263 140 76 171 55 4 0] [113 84 1231 3748 832 135 1076 161 2 1] [41 19 274 1549 1154 61 830 75 17 0] [7 6 54 112 28 8575 49 12 0 1] [28 9 144 1622 893 59 9526 93 8 0] [28 13 192 754 200 59 285 785 4 0] [2 2 13 69 75 21 50 11 7 0] [0 0 1 7 12 0 9 0 0 0]]</pre></div>																																																																						
	Prediksi Algoritma GNB	<div>Prediction Validity Evaluation: Accuracy: 0.5244</div> <div>Classification Report:</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>Analysis</td><td>0.00</td><td>0.00</td><td>0.00</td><td>660</td></tr><tr><td>Backdoor</td><td>0.03</td><td>0.72</td><td>0.06</td><td>576</td></tr><tr><td>DoS</td><td>0.03</td><td>0.01</td><td>0.01</td><td>4047</td></tr><tr><td>Exploits</td><td>0.43</td><td>0.05</td><td>0.08</td><td>11020</td></tr><tr><td>Fuzzers</td><td>0.29</td><td>0.66</td><td>0.40</td><td>6001</td></tr><tr><td>Generic</td><td>0.95</td><td>0.95</td><td>0.95</td><td>13200</td></tr><tr><td>Normal</td><td>0.79</td><td>0.70</td><td>0.74</td><td>18480</td></tr><tr><td>Reconnaissance</td><td>0.04</td><td>0.00</td><td>0.00</td><td>3462</td></tr><tr><td>Shellcode</td><td>0.00</td><td>0.00</td><td>0.00</td><td>374</td></tr><tr><td>Worms</td><td>0.00</td><td>0.00</td><td>0.00</td><td>43</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.52</td><td>57863</td></tr><tr><td>macro avg</td><td>0.26</td><td>0.31</td><td>0.22</td><td>57863</td></tr><tr><td>weighted avg</td><td>0.59</td><td>0.52</td><td>0.51</td><td>57863</td></tr></table> <div>Confusion Matrix:</div> <div><pre>[[0 448 0 25 117 6 61 3 0 0] [0 415 2 25 97 8 25 4 0 0] [0 2920 22 180 575 59 273 18 0 0] [0 3771 57 507 3849 69 2737 30 0 0] [0 1359 154 211 3976 199 87 15 0 0] [0 455 10 9 107 12522 74 23 0 0] [0 1342 498 198 3200 343 12895 4 0 0] [0 1613 1 35 1718 16 75 4 0 0] [0 157 1 2 208 0 6 0 0 0] [0 2 3 0 32 0 6 0 0 0]]</pre></div>		precision	recall	f1-score	support	Analysis	0.00	0.00	0.00	660	Backdoor	0.03	0.72	0.06	576	DoS	0.03	0.01	0.01	4047	Exploits	0.43	0.05	0.08	11020	Fuzzers	0.29	0.66	0.40	6001	Generic	0.95	0.95	0.95	13200	Normal	0.79	0.70	0.74	18480	Reconnaissance	0.04	0.00	0.00	3462	Shellcode	0.00	0.00	0.00	374	Worms	0.00	0.00	0.00	43	accuracy			0.52	57863	macro avg	0.26	0.31	0.22	57863	weighted avg	0.59	0.52	0.51	57863
	precision	recall	f1-score	support																																																																				
Analysis	0.00	0.00	0.00	660																																																																				
Backdoor	0.03	0.72	0.06	576																																																																				
DoS	0.03	0.01	0.01	4047																																																																				
Exploits	0.43	0.05	0.08	11020																																																																				
Fuzzers	0.29	0.66	0.40	6001																																																																				
Generic	0.95	0.95	0.95	13200																																																																				
Normal	0.79	0.70	0.74	18480																																																																				
Reconnaissance	0.04	0.00	0.00	3462																																																																				
Shellcode	0.00	0.00	0.00	374																																																																				
Worms	0.00	0.00	0.00	43																																																																				
accuracy			0.52	57863																																																																				
macro avg	0.26	0.31	0.22	57863																																																																				
weighted avg	0.59	0.52	0.51	57863																																																																				

Prediksi Algoritma ID3

Prediction Validity Evaluation:				
Accuracy: 0.7230				
Classification Report:				
	precision	recall	f1-score	support
Analysis	0.06	0.03	0.04	660
Backdoor	0.11	0.07	0.09	576
DoS	0.31	0.24	0.27	4047
Exploits	0.52	0.64	0.57	11020
Fuzzers	0.49	0.47	0.48	6001
Generic	0.99	0.97	0.98	13200
Normal	0.85	0.85	0.85	18480
Reconnaissance	0.80	0.67	0.73	3462
Shellcode	0.35	0.13	0.19	374
Worms	0.11	0.02	0.04	43
accuracy			0.72	57863
macro avg	0.46	0.41	0.42	57863
weighted avg	0.72	0.72	0.72	57863

Confusion Matrix:										
[23	12	140	420	14	2	22	26	1	0]
[17	40	132	296	35	3	37	16	0	0]
[115	111	968	2250	257	19	203	117	6	1]
[161	153	1347	7076	1053	29	993	190	15	3]
[14	15	211	1415	2807	17	1375	113	33	1]
[1	8	34	184	38	12861	65	7	2	0]
[11	12	89	1176	1341	22	15698	102	26	3]
[26	14	192	735	91	3	82	2313	6	0]
[0	0	6	104	101	1	93	21	48	0]
[0	0	0	31	5	0	6	0	0	1]]




Model *scikit-learn*

Prediksi Algoritma KNN

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
```

		<pre>Running with KNN Classifier: Prediction Validity Evaluation: Accuracy: 0.6346 Classification Report: precision recall f1-score support Analysis 0.10 0.07 0.08 660 Backdoor 0.03 0.01 0.02 576 DoS 0.28 0.33 0.31 4047 Exploits 0.39 0.50 0.44 11020 Fuzzers 0.35 0.31 0.33 6001 Generic 0.96 0.96 0.96 13200 Normal 0.79 0.76 0.78 18480 Reconnaissance 0.68 0.32 0.44 3462 Shellcode 0.30 0.07 0.12 374 Worms 0.00 0.00 0.00 43 accuracy 0.63 0.63 0.63 57863 macro avg 0.39 0.33 0.35 57863 weighted avg 0.64 0.63 0.63 57863 Confusion Matrix: [[47 11 198 295 32 15 57 5 0 0] [12 8 200 275 29 14 29 9 0 0] [83 62 1339 1918 228 112 247 50 8 0] [197 101 1904 5508 1279 148 1700 177 6 0] [56 35 386 2264 1837 78 1192 122 31 0] [7 12 109 218 60 12708 66 18 2 0] [47 21 225 2485 1350 86 14121 135 10 0] [31 22 326 1120 343 50 442 1123 5 0] [3 2 17 124 103 16 60 22 27 0] [1 0 4 14 9 1 12 2 0 0]]</pre>
	Prediksi Algoritma GNB	<pre>gaussian_nb_classifier = GaussianNB()</pre>

	<pre>Running with Gaussian Naive Bayes Classifier: Prediction Validity Evaluation: Accuracy: 0.4145 Classification Report: precision recall f1-score support Analysis 0.00 0.00 0.00 660 Backdoor 0.00 0.00 0.00 576 DoS 0.05 0.00 0.00 4047 Exploits 0.61 0.02 0.04 11020 Fuzzers 0.25 0.34 0.29 6001 Generic 0.44 0.95 0.60 13200 Normal 0.82 0.44 0.57 18480 Reconnaissance 0.09 0.28 0.14 3462 Shellcode 0.00 0.00 0.00 374 Worms 0.01 0.02 0.01 43 accuracy 0.41 0.41 0.37 57863 macro avg 0.23 0.21 0.17 57863 weighted avg 0.51 0.41 0.37 57863 Confusion Matrix: [[0 0 0 3 84 455 15 103 0 0] [0 0 0 2 48 452 20 54 0 0] [0 1 5 57 383 3131 96 367 0 7] [0 0 25 249 2186 4464 875 3201 0 20] [0 0 0 57 2026 1805 239 1874 0 0] [0 0 3 0 107 12567 414 108 0 1] [0 0 77 35 2596 3876 8177 3588 0 131] [0 0 0 4 614 1753 129 962 0 0] [0 0 0 0 95 170 17 92 0 0] [0 0 1 1 13 3 5 19 0 1]]</pre>
Prediksi Algoritma ID3	<pre>id3_classifier = DecisionTreeClassifier(criterion="entropy", random_state=42)</pre>

		<pre> Running with ID3 Classifier: Prediction Validity Evaluation: Accuracy: 0.7803 Classification Report: precision recall f1-score support Analysis 0.14 0.16 0.15 660 Backdoor 0.15 0.14 0.14 576 DoS 0.32 0.35 0.33 4047 Exploits 0.67 0.67 0.67 11020 Fuzzers 0.65 0.64 0.64 6001 Generic 0.98 0.98 0.98 13200 Normal 0.91 0.90 0.90 18480 Reconnaissance 0.78 0.75 0.76 3462 Shellcode 0.45 0.45 0.45 374 Worms 0.35 0.30 0.33 43 accuracy 0.78 0.78 0.78 57863 macro avg 0.54 0.53 0.54 57863 weighted avg 0.78 0.78 0.78 57863 Confusion Matrix: [[105 31 168 244 24 4 70 14 0 0] [39 78 167 235 16 4 4 29 4 0] [208 142 1403 1835 175 47 47 164 24 2] [249 216 2042 7360 324 111 213 438 51 16] [40 33 246 397 3817 29 1335 31 73 0] [6 5 68 129 18 12944 17 10 2 1] [59 4 50 200 1411 12 16675 24 45 0] [34 23 273 475 38 8 12 2588 10 1] [0 4 17 58 80 2 31 8 170 4] [0 0 2 23 2 0 0 3 0 13]] </pre>
2.	Data yang digunakan: data <i>test</i> yang di-submit ke Kaggle	
	Spesifikasi algoritma KNN: k = 5 distance_metric = <i>manhattan</i>	
	Model yang dibuat sendiri	
	Prediksi Algoritma KNN	<div>  knn_prediction_manhattan.csv <div>Complete · 13522144 Nicholas Reymond Sihite · 16m ago</div> 0.28689 </div>
	Prediksi Algoritma GNB	<div>  predicted_label.csv <div>Complete · Ahmad Rafi Maliki · 15h ago</div> 0.18879 </div>
	Prediksi Algoritma ID3	<div>  ID3_Prediction6.csv <div>Complete · Albert150 · 3h ago</div> 0.39533 </div>
	Model <i>scikit-learn</i>	
	Prediksi Algoritma KNN	Tidak dapat diuji karena dilarang pada spesifikasi

	Prediksi Algoritma GNB	Tidak dapat diuji karena dilarang pada spesifikasi
	Prediksi Algoritma ID3	Tidak dapat diuji karena dilarang pada spesifikasi

4.2. Analisis

Berdasarkan hasil kedua percobaan, algoritma yang paling baik dalam memprediksi *attack_cat* data (*validation* dan *test*) adalah algoritma *Iterative Dichotomiser 3* dengan skor prediksi *validation* 72.3% dan *test* 39.53%. Setelah itu, algoritma terbaik kedua adalah *k-Nearest Neighbour* dengan skor prediksi *validation* 63.3% dan *test* 28.69%. Algoritma terbaik ketiga adalah *Gaussian Naive Bayes* dengan skor prediksi *validation* adalah 52.44% dan *test* 18.879%. Hasil yang didapat dengan menggunakan model dari *library* 'scikit' juga tidak berbeda dengan hasil yang didapat dengan menggunakan model yang dibuat sendiri, yaitu urutan keakuratan modelnya adalah ID3 - KNN - GNB.

Selain itu, berdasarkan hasil percobaan terhadap algoritma KNN, tidak ada perbedaan signifikan antara hasil prediksi ketiga metrik pengukuran *distance*. Selain itu, tidak ada juga perbedaan signifikan antara 5 *neighbour* dengan 10 *neighbour*.

Pembagian Tugas

Tabel 3. Kontribusi Anggota

NIM - Nama	Pekerjaan
13522122 - Maulvi Ziadinda Maulana	Pembersihan dan prapemrosesan data
13522137 - Ahmad Rafi Maliki	Algoritma <i>Gaussian Naive-Bayes</i>
13522144 - Nicholas Reymond Sihite	Algoritme <i>k-Nearest Neighbour</i>
13522150 - Albert Ghazaly	Algoritma <i>Iterative Dichotomiser 3</i>

Referensi

- Elmenschawii, F. 2022. *KNN From Scratch* di <https://www.kaggle.com/code/fareselmenschawii/knn-from-scratch> (diakses 15 Desember 2024).
- Wikipedia kontributor. 2024. *Klasifikasi Naive Bayes* di https://id.wikipedia.org/wiki/Klasifikasi_Naive_Bayes (diakses 15 Desember 2024).