Tugas Besar 2 IF3170 Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:

Wilson Yusda	13522019
Filbert	13522021
Farel Winalda	13522047
Benardo	13522055

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I
DESKRIPSI MASALAH

Tugas Besar 2 IF3170 Inteligensi Artifisial ini bertujuan untuk memberikan pengalaman langsung dalam mengimplementasikan algoritma pembelajaran mesin menggunakan dataset UNSW-NB15, sebuah kumpulan data lalu lintas jaringan yang berisi aktivitas normal dan berbagai jenis serangan siber. Peserta diminta untuk mengimplementasikan algoritma K-Nearest Neighbors (KNN), Gaussian Naive-Bayes, dan ID3 secara mandiri (from scratch) serta membandingkan hasilnya dengan implementasi pustaka scikit-learn. Implementasi KNN harus mendukung beberapa parameter, termasuk jumlah tetangga dan berbagai metrik jarak, sedangkan ID3 memerlukan pengolahan data numerik sesuai materi kuliah. Tugas ini mencakup tahap-tahap seperti data cleaning, transformasi data, pemilihan fitur, reduksi dimensi, modeling, validasi, dan analisis hasil. Selain itu, model yang dikembangkan harus dapat disimpan dan dimuat ulang. Sebagai bonus, peserta dapat mengikuti kompetisi Kaggle untuk menguji performa model pada leaderboard. Hasil pengerjaan tugas dikumpulkan dalam bentuk repository GitHub yang mencakup source code, laporan dokumentasi, dan deskripsi lengkap implementasi serta perbandingan hasil algoritma.

Bab II
IMPLEMENTASI ALGORITMA

1.1. Implementasi Algoritma KNN

K-Nearest Neighbors (KNN) adalah algoritma machine learning berbasis instance-based learning yang bekerja berdasarkan prinsip kemiripan atau kedekatan data. Algoritma ini digunakan untuk klasifikasi atau regresi, tetapi lebih umum digunakan untuk tugas klasifikasi. Cara kerja algoritma KNN secara umum dimulai dengan inisiasi parameter dengan parameter utama adalah jumlah tetangga terdekat yang akan digunakan untuk prediksi(k) dan metrik jarak yang digunakan untuk mengukur kedekatan. Kemudian akan dilanjutkan dengan pelatihan data ,dimana algoritma ini bersifat lazy learning , sehingga pada tahap ini hanya menyimpan data latih tanpa membuat model eksplisit. Setelah selesai training, akan dilakukan perhitungan jarak untuk setiap sampel yang akan diprediksi dengan menggunakan metrik yang dipilih dengan semua data latih. Kemudian akan memilih k tetangga terdekat dan kemudian akan ditentukan mayoritas label dari tetangga-tetangga tersebut(untuk klasifikasi). Akhirnya, dilakukan pengukuran akurasi model dengan matrik evaluasi seperti kurasi, precision, recall, atau F1-score, tergantung pada jenis tugas yang diselesaikan.

Sedikit informasi, perlu dilakukan *batching* karena konsep algoritma *lazy learning* memakan RAM dalam jumlah besar, sehingga perlu dilakukan *batching* untuk mengurangi jumlah interaksi pencarian k. Untuk menjalankan algoritma ini, cukup memanggil fungsi ini, melakukan fitting, dan mengubah metrik sesuai keinginan. Opsi metrik dapat disesuaikan dengan *library cdist*.

Implementasi dari algoritma KNN pada tugas besar kali ini dapat dijelaskan sebagai berikut.

1. Inisiasi kelas (__init__)

   Pada tahap ini, akan dilakukan inisiasi kelas yang dapat menerima parameter berupa nilai k, metric penilaian, dan ukuran batch.

2. Tahap Pelatihan dan Preprocess data (fit dan _preprocess_data)

   Karena algoritma ini bersifat lazy learning , maka pada tahap fit ini hanya dilakukan untuk memproses data latih dengan menghilangkan nilai non-numerik atau NaN.

3. Mencari nilai K terbaik (_batch_predict)

   Pada metode ini, akan dimulai dengan menghitung matriks jarak antara setiap data dalam batch(x_batch) dan data latih (x_train). Kemudian akan mengambil jarak dan nilai k tetangga terdekat, dan menghitung kelas mayoritas dari label tetangga untuk membuat prediksi.

4. Prediksi data

   Pada tahap ini, akan dilakukan batch processing , dengan memecah data menjadi batch kecil untuk menghemat memori, dan kemudian setiap batch, akan dihitung jarak dan prediksi menggunakan fungsi **_batch predict.**

5. Evaluasi

   Pada tahap ini , akan dilakukan evaluasi terhadap model dengan menghitung akurasi dan nilai F1 makro.

```
class KNN:
    def __init__(self, k, metric='euclidean', batch_size=100):
        self.k = k
        self.metric = metric
        self.batch_size = batch_size

    def fit(self, X, y):
        self.X_train = self._preprocess_data(X)
        self.y_train = y

    def _preprocess_data(self, X):
        if isinstance(X, pd.DataFrame):
            X = X.apply(pd.to_numeric, errors='coerce')
            X = X.fillna(0)
        return X

    def _batch_predict(self, X_batch):
        distances = cdist(X_batch, self.X_train.values,
metric=self.metric)
        k_indices = np.argsort(distances, axis=1)[:, :self.k]
        k_nearest_labels = [self.y_train.iloc[indices] for indices in
k_indices]
        y_pred_batch = [
            Counter(labels).most_common(1)[0][0] for labels in
k_nearest_labels
        ]
        return y_pred_batch

    def predict(self, X):
        X = self._preprocess_data(X)
        n_samples = len(X)
        total_batches = (n_samples + self.batch_size - 1) //
```

```
self.batch_size
        y_pred = []
        print("Starting prediction...")

        for i in range(0, n_samples, self.batch_size):
            batch_num = i // self.batch_size + 1
            X_batch = X.values[i : i + self.batch_size]
            print(f"Processing batch {batch_num}/{total_batches}...")
            y_pred.extend(self._batch_predict(X_batch))

        print("Prediction completed.")
        return np.array(y_pred)

    def evaluate(self, X_test, y_test):
        print("\nStarting evaluation...")
        y_pred = self.predict(X_test)
        print("Evaluation completed.\n")
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='macro')

        return accuracy, f1
```

1.2. Implementasi Algoritma Gaussian Naive - Bayes

Gaussian Naive Bayes (GNB) adalah varian dari algoritma Naive Bayes yang dirancang khusus untuk menangani data numerik kontinu. Dalam banyak aplikasi klasifikasi, data fitur seringkali bersifat kontinu, seperti tinggi, berat, suhu, atau atribut lain yang berdistribusi kontinu. GNB memanfaatkan asumsi bahwa setiap fitur mengikuti distribusi Gaussian (*normal distribution*), sehingga cocok untuk menangani masalah ini.

Berikut beberapa Teori yang digunakan oleh algoritma ini:

1. **Teorema Bayes**: GNB mengandalkan Teorema Bayes untuk menghitung probabilitas dari suatu kelas berdasarkan beberapa kelas yang diberikan.

$$P(C_k | X) = \frac{P(X|C_k).P(C_k)}{P(X)}$$

2. **Distribusi Gaussian:** Distribusi Gaussian, yang juga dikenal sebagai distribusi Normal, adalah distribusi probabilitas yang sering digunakan untuk menggambarkan fenomena alami seperti distribusi tinggi badan atau berat badan dalam populasi, kesalahan pengukuran, dan berbagai fenomena lainnya. Distribusi ini secara alami muncul dalam banyak situasi di dunia nyata. Distribusi Gaussian atau normal adalah salah satu distribusi statistik yang paling umum digunakan. Untuk fitur kontinu, distribusi probabilitasnya dinyatakan sebagai:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(X-\mu)^2}{2\sigma^2}}$$

Implementasi Gaussian Naive-Bayes pada Tugas Besar ini dapat dijelaskan melalui langkah-langkah kode di bawah:

1. Inisiasi (__init__)

   Pada tahap ini, konstruktor dapat menerima dua buah parameter:

   - Priors : Probabilitas prior dari masing-masing kelas. (Optional)
   - Var_smoothing: Konstanta untuk mencegah varians terlalu kecil yang dapat menyebabkan perhitungan numerik tidak stabil. (Optional)

2. Training (fit)

Pada tahap ini akan ditentukan kelas-kelas unik dalam target y. Jika prior tidak diberikan pada saat inisiasi, maka akan dilakukan perhitungan prior secara otomatis sebagai proporsi masing-masing kelas dalam dataset. Kemudian akan dilakukan perhitungan parameter distribusi Gaussian(mean dan varians) untuk setiap fitur dalam masing-masing kelas, dan kemudian menambahkan smoothing ke varians untuk stabilitas numerik.

3. Menghitung Probabilitas Gaussian (calculate_probability)

   Fungsi ini digunakan untuk menghitung nilai probabilitas untuk setiap fitur berdasarkan distribusi Gaussian dengan parameter yang telah dihitung di tahap sebelumnya (mean dan varians).

4. Prediksi (predict)

   Pada tahap ini akan dilakukan prediksi label untuk data baru yang akan diuji. Untuk setiap data akan dihitung probabilitas posterior menggunakan fungsi ***calculate_probability*** terhadap semua kelas. Probabilitas posterior dihitung sebagai log prior ditambah dengan log likelihood setiap fitur. Hal ini dilakukan untuk stabilitas numerik). Kemudian, akan dikembalikan kelas dengan probabilitas terbesar.

5. Evaluasi (score dan macro_fi_score)

   Metode **score** menghitung akurasi klasifikasi secara sederhana, sedangkan metode **macro F1 score** menghitung rata-rata F1 score untuk semua kelas secara seimbang, sehingga lebih cocok untuk data dengan ketidakseimbangan kelas.

```
import numpy as np
from sklearn.metrics import f1_score

class GaussianNaiveBayesScratch:
    def __init__(self, priors=None, var_smoothing=1e-9):
        self.priors = priors
        self.var_smoothing = var_smoothing
        self.class_summaries = {}
        self.classes = None

    def fit(self, X, y):
        self.classes = np.unique(y)
        if self.priors is not None:
            if len(self.priors) != len(self.classes):
                raise ValueError("Jumlah priors harus sama dengan jumlah
kelas.")
            self.priors = np.array(self.priors)
            if not np.isclose(self.priors.sum(), 1):
                self.priors = self.priors / self.priors.sum()
        else:
            self.priors = np.array([np.mean(y == c) for c in
self.classes])

        var_max = np.var(X, axis=0).max()

        self.class_summaries = {}
        for idx, c in enumerate(self.classes):
            X_c = X[y == c]
            summaries = []
            for feature in zip(*X_c):
                mean = np.mean(feature)
                var = np.var(feature)
                # Tambahkan var_smoothing
                var += self.var_smoothing * var_max
                summaries.append((mean, var))
            self.class_summaries[c] = summaries

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

    def predict(self, X):
        predictions = []
        for x in X:
            posteriors = []
            for idx, c in enumerate(self.classes):
                prior = self.priors[idx]
                posterior = np.log(prior)
                for i in range(len(x)):
                    mean, var = self.class_summaries[c][i]
                    prob = self.calculate_probability(x[i], mean, var)
                    # Untuk stabilitas numerik, tambahkan log(prob)
                    if prob > 0:
                        posterior += np.log(prob)
                    else:
                        posterior += -np.inf
```

```
                posteriors.append(posterior)
            predictions.append(self.classes[np.argmax(posteriors)])
        return predictions

    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)

    def macro_f1_score(self, X, y):
        predictions = self.predict(X)
        return f1_score(y, predictions, average='macro')
```

1.3. Implementasi Algoritma ID3

Algoritma ID3 (Iterative Dichotomiser 3) adalah metode pembentukan pohon keputusan yang digunakan untuk klasifikasi. ID3 bekerja dengan memilih atribut yang memberikan "gain informasi" (information gain) tertinggi sebagai akar atau simpul pohon pada setiap iterasi. Proses ini diulang secara rekursif untuk setiap cabang hingga semua data dalam subset tersebut memiliki kelas yang sama, atau kriteria pemberhentian lainnya terpenuhi. Information gain dihitung berdasarkan entropi, yaitu ukuran ketidakpastian atau keacakan dalam data. Algoritma ini secara sistematis membagi dataset menjadi subset yang semakin homogen pada setiap langkah, sehingga membentuk pohon yang dapat digunakan untuk prediksi. Algoritma ini sangat cocok untuk data yang bersifat kategorikal tetapi dapat diadaptasi untuk data numerik dengan menggunakan nilai ambang (threshold).

Dalam implementasi algoritma ID3 pada kode ini, proses pemilihan threshold dilakukan secara mendalam dan sistematis. Pada setiap fitur/kolum, algoritma mengurutkan nilai-nilai fitur tersebut secara ascending dengan target variabelnya. Threshold dihitung di titik-titik perubahan target variabel, yaitu pada setiap perubahan kelas dalam target yang terkait dengan nilai fitur. Nilai threshold ini ditentukan dengan mengambil rata-rata dua nilai berturut-turut sebelum dan sesudah perubahan. Dengan cara ini, threshold hanya dipertimbangkan di lokasi yang secara potensial relevan untuk membedakan kelas target, sehingga mempercepat dan memfokuskan proses pemilihan.

Setelah semua threshold dihitung untuk suatu fitur, dihitung distribusi setiap kelas di subset kiri dan kanan untuk setiap threshold. Ini dilakukan dengan menggunakan kumulatif jumlah sampel untuk setiap kelas sebelum dan sesudah threshold. Dengan data ini, probabilitas

kelas di subset kiri dan kanan dihitung, lalu digunakan untuk mengukur entropi subset tersebut. Entropi ini kemudian digabungkan untuk menghitung entropi total setelah pembagian, yang selanjutnya digunakan untuk menghitung gain informasi threshold tersebut. Algoritma mengevaluasi setiap threshold dengan menghitung gain informasi (information gain). Information gain adalah perbedaan antara entropi sebelum dan sesudah pembagian data berdasarkan threshold tersebut. Entropi setelah pembagian dihitung dengan membagi data menjadi dua subset, yaitu di bawah dan di atas threshold, lalu menghitung rata-rata bobot entropi dari kedua subset tersebut. Untuk setiap fitur, threshold dengan gain informasi tertinggi dipilih sebagai kandidat terbaik untuk fitur tersebut. Setelah iterasi selesai untuk semua fitur, fitur dengan gain informasi terbaik di antara semua fitur dipilih sebagai fitur terbaik untuk membagi data pada simpul tersebut.

Setelah fitur terbaik dipilih, data dibagi menjadi dua subset berdasarkan threshold fitur tersebut: subset kiri (di bawah threshold) dan subset kanan (di atas atau sama dengan threshold). Algoritma kemudian memanggil meng-build tree secara rekursif untuk membangun pohon keputusan pada masing-masing subset. Proses ini terus diulang hingga mencapai kondisi pemberhentian, seperti semua data dalam subset memiliki kelas yang sama, jumlah sampel dalam subset terlalu kecil min_samples_split atau kedalaman pohon mencapai batas maksimum.

Proses ini memastikan bahwa setiap fitur dievaluasi secara menyeluruh untuk menemukan pembagian terbaik berdasarkan perubahan informasi yang dihasilkan. Setelah satu fitur terbaik digunakan untuk membagi data, fitur-fitur lainnya akan kembali dievaluasi secara lokal di subset masing-masing selama iterasi rekursif berikutnya. Dengan cara ini, ID3 membangun pohon keputusan yang secara progresif memecah data menjadi subset yang semakin homogen berdasarkan fitur dan threshold yang paling informatif.

```
import numpy as np
import pandas as pd

class ID3Classifier:
    def __init__(self, max_depth=None, min_samples_split=10,
min_information_gain=1e-7):
        self.tree = None
        self.label_encoder = LabelEncoder()
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_information_gain = min_information_gain

    def fit(self, X_train, y_train):
        self.features = np.array(X_train.columns)
        X = X_train.values
        y = self.label_encoder.fit_transform(y_train)
        self.tree = self._build_tree(X, y)

    def _entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        probs = counts / counts.sum()
        entropy = -np.sum(probs * np.log2(probs + 1e-9))  # Adding
epsilon to prevent log2(0)
        return entropy

    def _best_split(self, X, y):
        best_feature = None
        best_threshold = None
        best_info_gain = -1
        parent_entropy = self._entropy(y)
        n_samples, n_features = X.shape

        classes = np.unique(y)
        n_classes = len(classes)

        for feature_idx in range(n_features):
            X_column = X[:, feature_idx]
            sorted_indices = np.argsort(X_column)
            X_sorted = X_column[sorted_indices]
            y_sorted = y[sorted_indices]

            # Find unique split points
            unique_values, unique_indices = np.unique(X_sorted,
return_index=True)
            unique_values = unique_values[1:]  # Exclude the first
unique value
            unique_indices = unique_indices[1:]  # Corresponding indices

            if len(unique_values) == 0:
                continue

            thresholds = (X_sorted[unique_indices - 1] +
X_sorted[unique_indices]) / 2

            # Number of samples left and right for each threshold
            num_left = unique_indices
```

```
            num_right = n_samples - num_left

            # Initialize entropies
            entropies_left = np.zeros(len(thresholds))
            entropies_right = np.zeros(len(thresholds))

            for cls in classes:
                cls_mask = y_sorted == cls
                cls_cumsum = np.cumsum(cls_mask).astype(int)
                cls_total = cls_cumsum[-1]

                cls_left = cls_cumsum[unique_indices - 1]
                cls_right = cls_total - cls_left

                # Avoid division by zero
                probs_left = np.divide(cls_left, num_left,
out=np.zeros_like(cls_left, dtype=float), where=num_left != 0)
                probs_right = np.divide(cls_right, num_right,
out=np.zeros_like(cls_right, dtype=float), where=num_right != 0)

                # Calculate entropy components
                entropies_left -= probs_left * np.log2(probs_left +
1e-9)
                entropies_right -= probs_right * np.log2(probs_right +
1e-9)

            # Compute weighted entropy
            weighted_entropy = (num_left / n_samples) * entropies_left +
(num_right / n_samples) * entropies_right
            info_gains = parent_entropy - weighted_entropy

            # Find the best split for this feature
            max_info_gain_idx = np.argmax(info_gains)
            max_info_gain = info_gains[max_info_gain_idx]

            # Update best split if the current one is better
            if max_info_gain > best_info_gain:
                best_info_gain = max_info_gain
                best_feature = feature_idx
                best_threshold = thresholds[max_info_gain_idx]

        return best_feature, best_threshold, best_info_gain

    def _build_tree(self, X, y, depth=0):
        num_samples, num_features = X.shape
        num_labels = len(np.unique(y))

        # Stopping criteria
        if num_labels == 1:
            return y[0]
        if num_samples < self.min_samples_split:
            return self._majority_class(y)
        if self.max_depth is not None and depth >= self.max_depth:
            return self._majority_class(y)

        # Find the best split
```

```
        best_feature, best_threshold, best_info_gain =
self._best_split(X, y)

        # If no information gain is achieved, return the majority class
        if best_info_gain < self.min_information_gain:
            return self._majority_class(y)

        # Split the data
        left_indices = X[:, best_feature] < best_threshold
        right_indices = X[:, best_feature] >= best_threshold

        # If no split is possible, return the majority class
        if np.sum(left_indices) == 0 or np.sum(right_indices) == 0:
            return self._majority_class(y)

        left_X, left_y = X[left_indices], y[left_indices]
        right_X, right_y = X[right_indices], y[right_indices]

        # Recursively build the left and right subtrees
        left_subtree = self._build_tree(left_X, left_y, depth + 1)
        right_subtree = self._build_tree(right_X, right_y, depth + 1)

        # Construct the current node
        tree = {
            'feature': best_feature,
            'threshold': best_threshold,
            'info_gain': best_info_gain,
            'left': left_subtree,
            'right': right_subtree
        }

        return tree

    def _majority_class(self, y):
        return np.bincount(y).argmax()

    def predict(self, X):
        X = X.values  # Convert to NumPy array
        predictions = np.array([self._predict_instance(instance) for
instance in X])
        return self.label_encoder.inverse_transform(predictions)

    def _predict_instance(self, instance):
        node = self.tree
        while isinstance(node, dict):
            feature = node['feature']
            threshold = node['threshold']
            if instance[feature] < threshold:
                node = node['left']
            else:
                node = node['right']
        return node

    def extract_feature_importances(self,tree, n_features):
        feature_importances = np.zeros(n_features)
```

```
        def traverse_tree(node, feature_importances):
            # Base case: if the node is a leaf, return
            if not isinstance(node, dict):
                return

            # Update the importance of the current feature
            feature = node['feature']
            info_gain = node['info_gain']
            feature_importances[feature] += info_gain

            # Recursively traverse left and right subtrees
            traverse_tree(node['left'], feature_importances)
            traverse_tree(node['right'], feature_importances)

        traverse_tree(tree, feature_importances)
        return feature_importances
```

BAB III
PEMROSESAN DATA

3.1 Data Cleaning

Pada tahapan pembersihan data, dilakukan berbagai tahapan memastikan data konsisten untuk semua *fold* dan juga untuk menjaga kebenaran data terhadap model.

3.1.1 Penanganan data duplikat

Penanganan data duplikat dilakukan dengan fungsi *handle_duplicates*. Dari hasil fungsi tersebut, didapat bahwa fungsi tersebut tidak melakukan apapun, dan menandakan bahwa data tersebut bebas dari data duplikat.

3.1.2 Penanganan data kosong

Penanganan data kosong dilakukan dengan menggunakan *pipeline*. Pipeline menggunakan *BaseEstimator* dan *TransformerMixin.* Penanganan fungsi data kosong dilakukan dengan menggunakan *Feature Imputer*. Metode *imputing* dilakukan dengan menggunakan *Iterative Imputer* pada data numerik dan data paling banyak input untuk data kategorikal. Metode ini memerlukan waktu yang cukup lama, tergantung dari waktu yang diperlukan. Alasan penggunaan metode ini yaitu karena untuk pengisian data secara *mean* (*simple imputer)* terasa tidak cocok karena tidak adanya basis, namun lebih cocok untuk *regression*.

3.2 Data Preprocessing

Pada bagian ini, digunakan *pipeline* juga untuk dilakukan memproses data. Seperti yang sudah dijelaskan diatas, digunakan *BaseEstimator* dan *TransformerMixin.* Terdapat berbagai tahapan dari *data preprocessing*.

3.2.1 Feature Encoder

Bagian ini mengubah data kategorik menjadi tipe data yang lebih mudah diterima untuk model. Untuk setiap nilai unik dari sebuah data kategorikal, akan dipisah secara horizontal, menjadi tiap nilai unik dari atribut tersebut dan diisi dengan nilai *binary*, 0 atau 1. Hal ini dilakukan karena model dapat bekerja dengan lebih baik jika memasukkan data dalam bentuk angka.

3.2.1 Feature Scaler

Bagian ini menggunakan *library Standard Scaler* untuk melakukan *scaling*. Selain itu, juga dilakukan normalisasi logaritmik karena dari analisis didapat bahwa data sebenarnya cukup *skewed*. Kelas ini batal untuk diimplementasikan pada *pipeline* karena memberikan hasil yang kurang optimal. Alasan pembatalan akan dijelaskan di bagian evaluasi di bawah.

3.2.3 Feature Creator

Bagian ini membuat fitur tambahan yang dapat menambah akurasi model. Dalam bagian ini, dilakukan berbagai penambahan fitur:

1. Atribut proto_transformed

   Merupakan atribut yang hanya berisi top 5 data paling atas dari hasil EDA dan melabel sisanya dengan *others*.

2. Atribut byte_ratio

   Merupakan rasio antara jumlah byte yang dikirim dari sumber (sbytes) terhadap jumlah byte yang diterima oleh tujuan (dbytes + 1), digunakan untuk memahami proporsi data yang dikirim dibandingkan yang diterima.

3. Atribut pkt_ratio

   Merupakan rasio antara jumlah paket yang dikirim dari sumber (spkts) terhadap jumlah paket yang diterima oleh tujuan (dpkts + 1), digunakan untuk melihat keseimbangan pengiriman dan penerimaan paket.

4. Atribut load_ratio

   Merupakan rasio antara beban lalu lintas jaringan yang dikirim (sload) dengan beban yang diterima (dload + 1), untuk mengukur proporsi beban data sumber dan tujuan.

5. Atribut jit_ratio

   Merupakan rasio antara jitter sumber (sjit) terhadap jitter tujuan (sjit + 1), menunjukkan stabilitas lalu lintas jaringan.

6. Atribut inter_pkt_ratio

   Merupakan rasio antara interval antar paket yang dikirim (sinpkt) terhadap interval antar paket yang diterima (dinpkt + 1), digunakan untuk menganalisis pola waktu pengiriman data.

7. Atribut tcp_setup_ratio

Merupakan rasio waktu koneksi TCP (tcprtt) terhadap total waktu synack dan ackdat, untuk mengukur efisiensi proses setup koneksi TCP.

8. Atribut total_bytes

   Merupakan jumlah total byte dari sumber (sbytes) dan tujuan (dbytes), untuk menggambarkan total volume data yang ditransfer.

9. Atribut total_pkts

   Merupakan jumlah total paket dari sumber (spkts) dan tujuan (dpkts), menggambarkan total paket dalam sesi jaringan.

10. Atribut total_load

    Merupakan jumlah total beban dari sumber dan tujuan (sload + dload), menggambarkan total lalu lintas data yang ditangani.

11. Atribut total_jitter

    Merupakan jumlah total jitter dari sumber (sjit) dan tujuan (djit), digunakan untuk menganalisis stabilitas jaringan secara keseluruhan.

12. Atribut total_inter_pkt

    Merupakan jumlah total interval antar paket dari sumber (sinpkt) dan tujuan (dinpkt), memberikan gambaran tentang pola waktu pengiriman data.

13. Atribut total_tcp_setup

    Merupakan jumlah total waktu yang digunakan untuk proses koneksi TCP (tcprtt, synack, dan ackdat), menggambarkan waktu total setup koneksi.

14. Atribut byte_pkt_interaction_src

    Merupakan interaksi antara byte sumber (sbytes) dengan jumlah paket sumber (spkts), untuk melihat pengaruh ukuran data terhadap jumlah paket.

15. Atribut byte_pkt_interaction_dst

    Merupakan interaksi antara byte tujuan (dbytes) dengan jumlah paket tujuan (dpkts), menunjukkan hubungan ukuran data terhadap jumlah paket yang diterima.

16. Atribut load_jit_interaction_src

    Merupakan interaksi antara beban sumber (sload) dengan jitter sumber (sjit), untuk menganalisis pengaruh beban data terhadap kestabilan lalu lintas.

17. Atribut load_jit_interaction_dst

Merupakan interaksi antara beban tujuan (dload) dengan jitter tujuan (djit), untuk memahami hubungan antara beban data dengan kestabilan jaringan.

18. Atribut pkt_jit_interaction_src

    Merupakan interaksi antara jumlah paket sumber (spkts) dengan jitter sumber (sjit), menunjukkan pengaruh jumlah paket terhadap kestabilan.

19. Atribut pkt_jit_interaction_dst

    Merupakan interaksi antara jumlah paket tujuan (dpkts) dengan jitter tujuan (djit), untuk menganalisis pengaruh jumlah paket yang diterima terhadap kestabilan.

20. Atribut mean_pkt_size

    Merupakan ukuran rata-rata paket yang dikirimkan dari sumber (smean) ditambah dengan rata-rata paket dari tujuan (dmean), menggambarkan ukuran rata-rata data dalam sesi jaringan.

21. Atribut tcp_seq_diff

    Merupakan selisih antara urutan segmen TCP sumber (stcpb) dan tujuan (dtcpb), digunakan untuk menganalisis sinkronisasi antara sumber dan tujuan.

Penambahan fitur ini dengan mempertimbangkan referensi dari berbagai pihak, hasil dari *business understanding*, serta percobaan untuk memastikan memberikan hasil terbaik.

3.2.4 Feature Dropper

Kelas ini berfungsi untuk menghilangkan berbagai atribut yang dianggap tidak berguna. Pertama dilakukan pengecekan untuk mengecek apakah pada data terdapat atribut yang memang berkorelasi secara langsung satu sama lain. Untuk atribut yang memang berkorelasi, dibuang untuk atribut yang memang sudah memiliki korelasi tinggi untuk memastikan bahwa datanya tetap terasa tidak redundan. Selain itu, dilakukan pemangkasan untuk data yang dianggap memang tidak berguna, seperti beberapa atribut proto yang sudah di *encode.* Perilaku ini dilakukan untuk mencegah *multicollinearity*. *Multicollinearity* disebabkan oleh 2 data yang berkorelasi , sehingga menyebabkan redudansi, sehingga harus dibuang untuk mencegah redudansi.

3.2.5 PCA (Tidak diimplementasikan)

Diberikan *pipeline* khusus untuk melakukan PCA, namun tidak dilakukan pemrosesan terhadap *pipeline* ini karena dianggap datanya memiliki varian yang sudah sesuai. Oleh karena

itu, bahkan jika memang harus dilakukan PCA, tetap tidak akan terjadi perubahan signifikan, justru malah hanya akan menghasilkan 2 atribut PCA.

BAB IV
EVALUASI

4.1 Evaluasi KNN

Penerapan algoritma KNN *from scratch* dapat menyaingi hasil dari algoritma KNN. Algoritma ini memiliki hasil yang dapat dibandingkan dengan algoritma *built-in*, hanya dengan kekurangan yaitu penggunaan RAM dan waktu yang dibutuhkan. KNN merupakan sebuah algoritma *lazy learner*, dimana algoritma ini tidak membangun model, melainkan hanya menyimpan data pelatihan. Jika dilakukan komparasi secara besar-besaran, tentu akan sangat memakan RAM. Kami berhasil mengakali hal ini dengan melakukan pembagian per *batch*, namun hal ini tidak menyelesaikan permasalahan waktu.

Untuk perbandingan performansi, dari perbandingan metrik *euclidean*, terdapat selisih yang cukup sedikit. Dari algoritma *from scratch*, didapat nilai sebesar 0.3572, sedangkan untuk algoritma *built-in*, didapat nilai sebesar 0.3626. Adanya perbedaan mungkin dapat disebabkan oleh berbagai hal seperti pembulatan. Namun, secara keseluruhan, algoritma kami sudah dapat menyaingi algoritma *built-in* dari sisi waktu performansi.

Sangat disayangkan bahwa kami tidak menyadari bahwa terdapat lebih dari 3 implementasi metrik KNN, dimana kami hanya mengetes sampai *euclidean, manhattan,* dan, *Minkowski*. Setelah mencapai jumlah pengumpulan maksimal, ternyata metrik *hamming* dapat menyaingi nilai dari algoritma lainnya. Implementasi *hamming* dapat menyebabkan nilai F1 *macro* sebesar 0.55, yang dapat menyaingi algoritma pengumpulan kami, yakni ID3.

4.2 Evaluasi Naive Bayes

Dalam evaluasi komparasi Naive Bayes *from scratch* dengan *built-in*, kami mendapati hasil yang serupa. Dari percobaan ketiga algoritma, baik *from scratch* maupun *built-in*, algoritma ini memiliki akurasi yang paling kecil.

Kami berhasil mendapati alasan dari lemahnya algoritma ini. Algoritma ini tidak bekerja dengan baik pada dataset yang memiliki jumlah kolom yang sangat besar. Sebagai contoh, dalam implementasi datasets dengan jumlah kolom 21 dibandingkan dengan jumlah kolom sebanyak 36, terdapat perbedaan yang cukup signifikan, yaitu dari 0.14 menjadi 0.09 untuk F1-Score macro.

Hal ini juga mungkin disebabkan oleh isi dari data yang sangat tidak berkorelasi, ataupun terjadinya korelasi berlebihan, dimana untuk percobaan ketika terjadi PCA, dengan jumlah kolom sebanyak 20, juga memberikan hasil yang baik yaitu 0.17.

Secara keseluruhan ini, algoritma ini kurang bekerja dengan baik pada datasets yang besar karena sulitnya untuk mencari korelasi antar fitur. Namun, untuk performa perbandingan antara dari *built-in* dan *from scratch*, terdapat perubahan yang kurang signifikan, sehingga memastikan bahwa algoritma kami sudah bekerja dengan sangat baik. Keduanya memberikan nilai f1 *macro* yang sama yaitu 0.909.

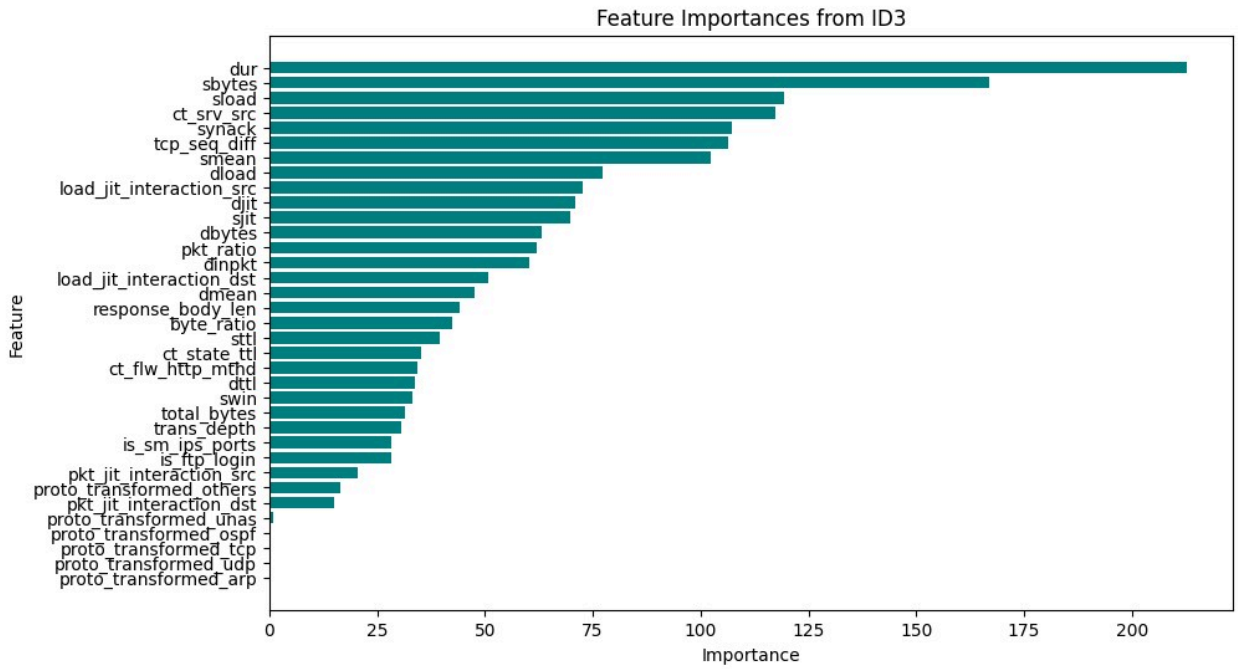
4.3 Evaluasi ID3

Dalam evaluasi komparasi ID3 *from scratch* dibandingkan dengan *built-in*, didapati bahwa keduanya memiliki perbedaan dari 0.5459 dengan 0.542. Perbandingan ini disebabkan oleh karena adanya perlakukan *tuning* pada ID3 *from scratch*. Namun, secara keseluruhan, bahkan ketika dilakukan pengecekan kepada datasets untuk berbagai fold, memberikan hasil yang cukup konsisten.

Adapun konsiderasi yang diberikan, yakni yaitu bahwa implementasi *from scratch* kami memiliki optimasi waktu yang lebih buruk dibandingkan dengan *built-in*. Hal ini terjadi karena pembangunan *tree* yang dilakukan secara kurang optimal sehingga menyebabkan algoritma menjadi lebih lama.

Selain itu, untuk kebutuhan *hyperparameter tuning*, terdapat limitasi dari sisi kami, dimana kami hanya terbatas untuk *max_depth*, *max_splits,* dan *max_info_gain*. Namun, dari parameter ini, per pembuatan laporan ini, dengan mengandalkan *max_depth = None*, *max_splits* = 10, dan *max_info_gain* sebesar 1*10^-7, kami berhasil mendapatkan nilai makro F1 score sebesar 0.489. Tentu saja , hal ini mengingat bahwa data disajikan pada 80% dari data *test*, namun kami yakin terhadap 20% proporsi lainnya. Hal ini disebabkan karena kami telah memastikan untuk mengambil parameter yang memiliki nilai terbaik jika dikombinasikan dengan cross validation yang dilakukan dengan *fold* = 5.

Evaluasi penggunaan fitur:



Diketahui bahwa algoritma kami dapat menerima hampir semua kolom, kecuali hasil *encoding* dari atribut *proto*. Namun, kami percaya bahwa hasil ini disebabkan oleh kebiasaan kami yang tidak melakukan *scaling*. Namun, jika dilakukan *scaling*, justru akan menurunkan skor dari F1 macro. Hal ini mungkin disebabkan untuk algoritma yang memerlukan perhitungan, detail dari datanya sendiri dapat berperan penting. Dengan memotong *range* dari atribut, membuat fitur menjadi kurang signifikan, sehingga menyebabkan model tidak memberikan hasil yang optimal.

BAB V
KESIMPULAN

5.1 Kesimpulan

Secara umum, implementasi 3 algoritma kami dari *scratch* telah dapat berfungsi dengan baik. Parameter keberhasilan kami ada pada perbandingan dengan hasil dari *library* , yang dimana algoritma buatan kami hanya memiliki kelemahan minor dalam hal waktu dan *resource* yang dibutuhkan

Kami terakhir mengumpulkan hasil dari algoritma ID3 karena kami merasa lebih cocok, dari hal fleksibilitas, dan juga performa. Adapun saingan yang mungkin dapat menghasilkan hasil serupa yakni berupa KNN dengan metrik *hamming*. Kami berhasil mendapatkan skor f1 macro skor sebesar 0.489 untuk *test case* 80% di kaggle.

Pemilihan model ini disebabkan karena kami merasa bahwa kedua model yang disebutkan sebelumnya memang cocok untuk *high dimensional* data, artinya data yang memiliki kolom yang banyak. Algoritma *Naive Bayes* kurang cocok untuk permasalahan ini karena algoritma ini lebih lemah untuk data dengan dimensi tinggi.

5.2 Saran

Menurut kami, sebaiknya sebelum melakukan pengumpulan, diperhatikan apakah hasilnya merupakan hasil unik. Hal ini dikarenakan kami sering kali mengirimkan hasil yang sama pada *kaggle*, sehingga menyebabkan kami kehabisan berbagai kesempatan.

Selain itu, juga disarankan untuk lebih berfokus pada pemilihan model, karena kami lebih sering menghabiskan waktu dalam *tuning* model, dan itu menyebabkan kami tidak mengeksplorasi opsi lain, seperti KNN with hamming, dimana kami hanya mengimplementasikan hal tersebut setelah jatah pengumpulan habis.

BAB VI
LAMPIRAN

5.1 Appendix

5.2 Pembagian Tugas

Nama/NIM	Kontribusi
Wilson Yusda (13522019)	KNN, Data Cleaning
Filbert (13522021)	ID3, Debugging
Farel Winalda (13522047)	ID3, ReadMe
Benardo (13522055)	Naive Bayes, Debugging

BAB VII
REFERENSI

1. P. Patidar, "Classification Using Gaussian Naive Bayes From Scratch," *Level Up Coding*, Mar. 1, 2023. [Online]. Available: https://levelup.gitconnected.com/classification-using-gaussian-naive-bayes-from-scratch-6b8ebe830266. [Accessed: Dec. 14, 2024].
2. J. Brownlee, "Naive Bayes Classifier From Scratch in Python," *Machine Learning Mastery*, Oct. 25, 2019. [Online]. Available: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/. [Accessed: Dec. 14, 2024].
3. "Iterative Dichotomiser 3 (ID3) Algorithm From Scratch," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/iterative-dichotomiser-3-id3-algorithm-from-scratch/. [Accessed: Dec. 14, 2024].
4. T. R. Joy, "Step by Step Decision Tree ID3 Algorithm From Scratch in Python (No Fancy Library)," *Medium*, Mar. 28, 2021. [Online]. Available: https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f. [Accessed: Dec. 14, 2024].
5. A. Bhattacharjee, "Implementing the k-Nearest Neighbors (kNN) Algorithm From Scratch in Python," *Medium*, Sept. 24, 2023. [Online]. Available: https://medium.com/@avijit.bhattacharjee1996/implementing-the-k-nearest-neighbors-knn-algorithm-from-scratch-in-python-3b83a4fe8. [Accessed: Dec. 13, 2024].
6. "Introduction to DTL," Institut Teknologi Bandung, *Artificial Intelligence Parent Class - Modeling Decision Tree Learning (DTL)*, [PDF]. Available: https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/298936-Modeling-Decision-Tree-Learning-DTL/120485-Modul-Introduction-to-DTL/1731401412073_IF3170_SupervisedLearning_DTL.pdf. [Accessed: Dec. 14, 2024].