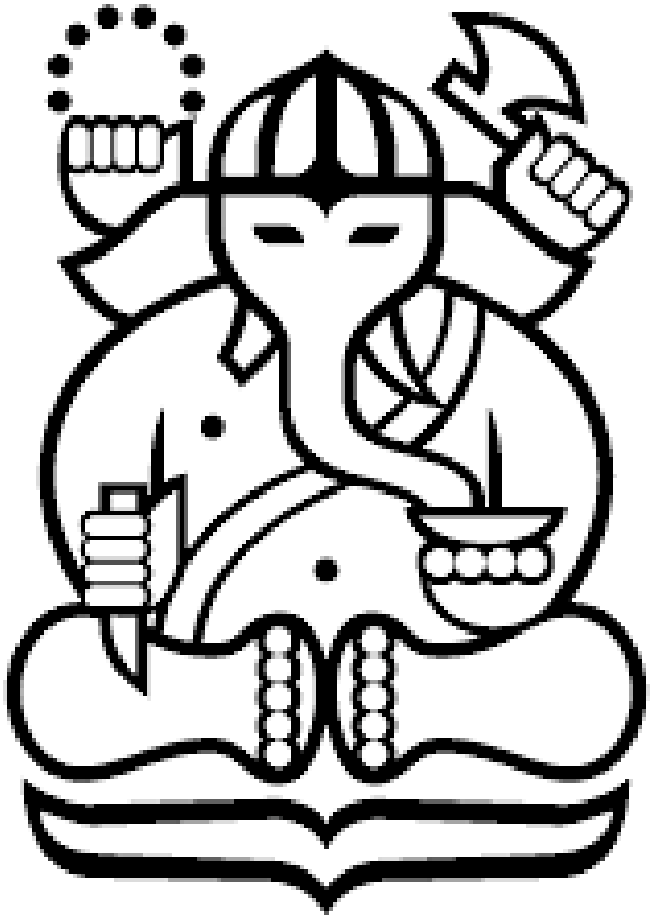
Laporan Tugas Besar 2
IF3170 Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin



Disusun oleh:

Maria Flora Renata Siringoringo	13522010
M Athaullah Daffa Kusuma Mulia	13522044
Dzaky Satrio Nugroho	13522059
Rafiki Prawira Harianto	13522065

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

2024

Daftar Isi

Bab 1: Preprocessing

Kami melakukan 5 langkah *preprocessing* dalam mempersiapkan dataset ini, yaitu

1.1. Handling Missing Data

Untuk meng-*handle missing data*, kami menggunakan metode imputasi. Untuk data numerik, kami menggunakan nilai median, sedangkan untuk data kategorikal kami menggunakan nilai modus/yang paling sering muncul.

1.2. Dealing with Outliers

Kami menentukan outlier dengan melihat nilai *interquartile range*. Outlier tinggi adalah data dengan nilai lebih besar dari nilai q3 + 1.5 * *interquartile range*, sedangkan outlier rendah adalah data dengan nilai lebih kecil dari nilai q1 - 1.5 * *interquartile range*. Kami hanya membersihkan outlier dari data numerikal.

1.3. Removing Duplicates

Kami men-*drop* semua data duplikat dengan menggunakan fungsi DataFrame.drop_duplicates().

1.4. Feature Engineering

Berikut adalah fitur fitur yang dibuat, beserta dengan rumusnya :

- df['byte_ratio'] = df['sbytes'] / (df['dbytes'] + 1)
- df['pkt_ratio'] = df['spkts'] / (df['dpkts'] + 1)
- df['load_ratio'] = df['sload'] / (df['dload'] + 1)
- df['jit_ratio'] = df['sjit'] / (df['djit'] + 1)
- df['inter_pkt_ratio'] = df['sinpkt'] / (df['dinpkt'] + 1)
- df['tcp_setup_ratio'] = df['tcprtt'] / (df['synack'] + df['ackdat'] + 1)
- df['total_bytes'] = df['sbytes'] + df['dbytes']
- df['total_pkts'] = df['spkts'] + df['dpkts']
- df['total_load'] = df['sload'] + df['dload']
- df['total_jitter'] = df['sjit'] + df['djit']
- df['total_inter_pkt'] = df['sinpkt'] + df['dinpkt']
- df['total_tcp_setup'] = df['tcprtt'] + df['synack'] + df['ackdat']
- df['byte_pkt_interaction_src'] = df['sbytes'] * df['spkts']
- df['byte_pkt_interaction_dst'] = df['dbytes'] * df['dpkts']
- df['load_jit_interaction_src'] = df['sload'] * df['sjit']
- df['load_jit_interaction_dst'] = df['dload'] * df['djit']
- df['pkt_jit_interaction_src'] = df['spkts'] * df['sjit']
- df['pkt_jit_interaction_dst'] = df['dpkts'] * df['djit']
- df['mean_pkt_size'] = df['smean'] + df['dmean']

- df['tcp_seq_diff'] = df['stcpb'] - df['dtcpb']

1.5. Feature Scaling

Disini digunakan Standardization (Z-score Scaling), dengan rumus

$$X' = \frac{X-\mu}{\sigma}$$

Dengan X adalah nilai awal, μ adalah rata-rata, dan σ adalah standar deviasi dari fiturnya.

1.6. Feature Encoding

Disini digunakan feature encoding berupa One-hot Encoding

1.7. Dimensionality Reduction

Disini digunakan Principal Component Analysis (PCA) dari *library* sklearn.decomposition..

Bab 2: Implementasi

2.1. KNN (K-Nearest Neighbors)

Algoritma KNN merupakan sebuah algoritma pembelajaran mesin yang cukup sederhana. Algoritma ini akan memprediksi hasil dari suatu parameter dengan mengurutkan seluruh titik yang ada di database dan mengambil K titik yang paling terdekat dari titik input. Setelah mengambil K titik yang terdekat, nantinya akan diambil modus dari hasil K titik terdekat tersebut sebagai return dari fungsi predict di algoritma ini.

Implementasi algoritma KNN dibuat dengan menggunakan sebuah class bernama KNearestNeighbors yang mempunyai beberapa atribut dan method. Berikut merupakan implementasi class KNearestNeighbors berupa atribut dan methodnya

```
from statistics import mode
import pickle
import numpy as np

class KNearestNeighbors:
    def __init__(self, jumlah_neighbor:int = 5, r:int = 2) -> None:
        """
        r = 1 => manhattan
        r = 2 => euclidian
        r >= 3 => minkowski
        """
        self.jumlah_neighbor = jumlah_neighbor
        self.r = r
        self.data_x = []
        self.data_y = []

    def hitung_jarak(self, titik1:list[int], titik2:list[int]) ->
int | Exception:
        if len(titik2) != len(titik1):
            return Exception("Tidak dapat menghitung jarak antara 2
titik yang berbeda dimensi")
        jaraknya = 0
        for i in range(len(titik1)):
            jaraknya += (abs(titik2[i] - titik1[i]))**(self.r)
        return (jaraknya**(1/self.r))

    def fit(self, x_train:list[list[int]], y_train:list[list[int]])
-> None:
        self.data_x.extend(x_train)
        self.data_y.extend(y_train)

    def predict_point(self, x_pred:list[int]) -> list[int]:
        list_jarak = []
```

```
        list_hasil = []
        for i in range(len(self.data_x)):
            jaraknya = self.hitung_jarak(x_pred, self.data_x[i])
            dimasukin = False
            i = 0
            while(not dimasukin and i < len(list_jarak)):
                if list_jarak[i] > jaraknya:
                    list_jarak.insert(i, jaraknya)
                    list_hasil.insert(i, tuple(self.data_y[i]))
                    dimasukin = True
                i+=1
            if not dimasukin:
                list_jarak.append(jaraknya)
                list_hasil.append(tuple(self.data_y[i]))
        toberet = mode(list_hasil[:(self.jumlah_neighbor)])
        return np.array(toberet)

    def predict(self, x_pred_list:list[list[int]]) ->
list[list[int]]:
        hasilnya = []
        for titik in x_pred_list:
            hasilnya.append(self.predict_point(titik))
        return np.array(hasilnya)

    def save(self, filename) -> None:
        try:
            with open(filename, "wb") as filenya:
                pickle.dump(self, filenya)
        except Exception as err:
            print(err)

    @classmethod
    def load(cls, filename):
        try:
            with open(filename, "rb") as filenya:
                return (pickle.load(filenya))
        except Exception as err:
            print(err)
```

Di class ini, KNearestNeighbors mempunyai beberapa fungsi, antara lain:

1. __init__
   Fungsi ini bertujuan untuk menginisialisasi class KNearestNeighbors. Fungsi ini menerima parameter input berupa jumlah neighbors dan algoritma yang digunakan untuk menghitung jarak antar titik.

2. hitung_jarak
   Fungsi ini bertujuan untuk menghitung jarak antara 2 titik. Rumus utama yang dipakai adalah

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^r\right)^{\frac{1}{r}}$$

Rumus di atas merupakan algoritma menghitung jarak antara 2 titik berdimensi n milik Minkowski. Algoritma Minkowski sendiri bisa diterapkan untuk algoritma Manhattan dan algoritma Euclidean, dimana jika $r = 1$ algoritma tersebut akan menjadi algoritma Manhattan dan jika $r = 2$ algoritma tersebut akan menjadi algoritma Euclidean.

3. fit
   Fungsi fit bertujuan untuk melatih mesin dengan data yang diberikan. Untuk algoritma K-Nearest Neighbor, tidak ada process pengolahan data sebelum mesin diuji. Algoritma ini cukup menerima data latihan dan menyimpannya. Saat ingin mempredict dengan KNN, nantinya titik akan langsung dihitung jaraknya dengan data yang dimasukkan di fit ini.

4. predict_point
   Fungsi ini bertujuan untuk mem-predict sebuah titik. Nantinya fungsi ini akan mengiterasi seluruh data yang telah disimpan di fungsi fit. Setiap iterasi, akan dihitung jarak antara titik yang ingin dipredict dan titik yang telah disimpan, lalu jarak tersebut akan disimpan di array yang terurut membesar. Setelah selesai mentraversal, nantinya array nya akan diambil hanya K elemen pertama (K telah dideklarasi saat menginitialize class KNN). Setelah diambil hanya K elemen pertama, akan dilakukan fungsi mode untuk diambil hasil mayoritas dari K Neighbors terdekat. Hasil fungsi mode tersebut akan direturn.

5. predict
   Fungsi ini bekerja seperti fungsi predict pada umumnya. Fungsi ini akan mengiterasi setiap elemen pada argumen dan akan memanggil fungsi predict_point. Fungsi ini juga akan mereturn seluruh hasil predict point yang dibungkus dengan array.

2.2. Naive-Bayes

Naive Bayes merupakan salah satu algoritma pembelajaran mesin yang umum. Algoritma ini digunakan untuk mengklasifikasikan data dengan menggunakan probabilitas apakah sampel masuk ke kelas tertentu. Naive Bayes akan menghitung seluruh kemungkinan suatu fitur untuk masuk ke kelas tersebut. Nantinya, Naive Bayes akan menghitung berapa kemungkinan sampel tersebut masuk ke kelas tertentu berdasarkan rumus berikut:

$$P(V_j \mid a_1, a_2, \ldots, a_n) = P(V_j) \cdot \prod_{i=1}^{n} P(a_i \mid V_j)$$

Setelah menghitung semua peluang sampel tersebut masuk ke kelas tertentu, akan diambil kelas yang memiliki probabilitas tertinggi sebagai hasil dari prediksi Naive Bayes.

Implementasi algoritma Naive-Bayes dibuat dengan menggunakan sebuah class bernama NaiveBayes yang mempunyai beberapa atribut dan metode. Berikut merupakan implementasi class NaiveBayes berupa atribut dan methodnya

```
import numpy as np
import pickle

class NaiveBayes:
    def __init__(self):
        self.data_x = []
        self.data_y = []
        self.freq_muncul = {}
        self.freq_muncul_hasil = {}
        self.y_unique = []

    def fit(self, x_train:list[list[int]],
y_train:list[list[int]]):
        self.data_x.extend(x_train)
        self.data_y.extend([tuple(y_list) for y_list in y_train])
        for i in range(len(self.data_x)):
            j = 1
            for elemen_column in self.data_x[i]:
                column_sekarang = "column" + str(j)
                if (column_sekarang) not in self.freq_muncul:
                    self.freq_muncul[column_sekarang] = {}
                if elemen_column not in
self.freq_muncul[column_sekarang]:

self.freq_muncul[column_sekarang][elemen_column] = {}
                if self.data_y[i] not in
self.freq_muncul[column_sekarang][elemen_column]:

self.freq_muncul[column_sekarang][elemen_column][self.data_y[i]] =
0

self.freq_muncul[column_sekarang][elemen_column][self.data_y[i]] +=
1
                j += 1
        for j in self.data_y:
            if j not in self.freq_muncul_hasil:
                self.freq_muncul_hasil[j] = 0
            self.freq_muncul_hasil[j] += 1
            if j not in self.y_unique:
                self.y_unique.append(j)

    def predict_point(self, x_pred:list[int]) -> list[int]:
        proba_each = []
        hasil_each = []
        for kemungkinan_hasil in self.y_unique:
            hasil_each.append(kemungkinan_hasil)
```

```
            j = 1
            proba_sekarang = 1
            for column in x_pred:
                column_sekarang = "column" + str(j)
                try:
                    proba_sekarang *=
(self.freq_muncul[column_sekarang][column][kemungkinan_hasil] /
self.freq_muncul_hasil[kemungkinan_hasil])
                except:
                    proba_sekarang *= 0
                    break
                j+=1
            proba_sekarang *=
(self.freq_muncul_hasil[kemungkinan_hasil])/len(self.data_y)
            proba_each.append(proba_sekarang)
        proba_terpilih = -1
        hasil_terpilih = self.y_unique[0]
        for i in range(len(hasil_each)):
            if proba_each[i] > proba_terpilih:
                proba_terpilih = proba_each[i]
                hasil_terpilih = hasil_each[i]
        return np.array(hasil_terpilih)

    def predict(self, x_pred_list:list[list[int]]) ->
list[list[int]]:
        hasilnya = []
        for titik in x_pred_list:
            hasilnya.append(self.predict_point(titik))
        return np.array(hasilnya)

    def save(self, filename) -> None:
        try:
            with open(filename, "wb") as filenya:
                pickle.dump(self, filenya)
        except Exception as err:
            print(err)

    @classmethod
    def load(cls, filename):
        try:
            with open(filename, "rb") as filenya:
                return (pickle.load(filenya))
        except Exception as err:
            print(err)
```

Naive-Bayes mempunyai beberapa fungsi, antara lain:

1. __init__
   Fungsi yang berguna untuk menginisialisasi class NaiveBayes. Fungsi ini akan mengset semua atribut yang dimiliki NaiveBayes menjadi kosong

.

2. fit
   Fungsi ini berguna untuk melatih class NaiveBayes di atas dengan data x_train dan y_train yang diberikan.Nantinya elemen-elemen dari x_train dan y_train akan disimpan ke data_x dan data_y di dalam class tersebut. elemen-elemen dari x_train tersebut juga akan disimpan ke sebuah dictionary yang merepresentasikan berapa kali kemunculan sebuah elemen di atribut-atribut tertentu.

3. predict_point
   Fungsi ini berguna untuk mempredict kelas/hasil dari input parameter. Nantinya seluruh kemungkinan hasil yang ada di database akan di iterasi, dan setiap kemungkinan hasil tersebut akan dihitung probabilitasnya input parameter untuk menjadi kemungkinan hasil tersebut dengan rumus yang telah dijelaskan di atas. Nantinya hasil probabilitas setiap iterasi akan disimpan ke sebuah array. Setelah iterasi seluruh kemungkinan selesai, akan dipilih probabilitas tertinggi di array tersebut dan akan direturn kemungkinan hasil yang memiliki probabilitas tertinggi di antara yang lain.

4. predict
   Fungsi ini sama seperti fungsi predict_point, tetapi fungsi ini menerima parameter berupa list of titik yang akan diprediksi. Fungsi ini akan mengiterasi seluruh elemen dari input dan memanggil fungsi predict_point untuk setiap elemennya. Nantinya setiap hasil dari predict_point tersebut akan disimpan ke sebuah array dan array tersebut direturn.

2.3. ID3 (Iterative Dichotomiser 3)

ID3 (Iterative Dichotomiser 3) merupakan algoritma machine learning yang cukup umum digunakan. ID3 merupakan salah satu jenis DTL (Decision Tree Learning) dimana output dari fitting nya berupa sebuah Pohon Keputusan. Algoritma ini menggunakan pendekatan greedy top-down untuk membangun Pohon Keputusannya. Algoritma ini akan memilih atribut-atribut yang menjadi percabangannya sesuai dengan information gain terbesar untuk memecah data-data pada setiap langkah-langkahnya. Sebelum menghitung information gain, perlu dihitung entropi awal dengan rumus:

$$Entropy(S) \equiv \sum_{i=1}^{c} - p_i \log_2 p_i$$

Setelah menghitung entropynya, information gain bisa diperoleh dengan rumus:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} Entropy(S_v)$$

Setelah menghitung masing-masing Information Gain, pilih atribut dengan Information Gain tertinggi sebagai pembagi pohon pada cabang saat ini. Ulangi proses di atas sampai semua contoh dalam subset memiliki label yang sama atau tidak ada lagi atribut yang tersisa untuk membagi lagi.

Implementasi algoritma ID3 dibuat dengan menggunakan sebuah class bernama ID3DecisionTree yang mempunyai beberapa atribut dan metode. Berikut merupakan implementasi class ID3DecisionTree berupa atribut dan methodnya

```
import numpy as np
import pickle

class ID3DecisionTree:
    def __init__(self, max_depth=None):
        self.tree = None
        self.max_depth = max_depth

    def entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        entropy = -np.sum(probabilities * np.log2(probabilities +
1e-10))
        return entropy

    def discretize(self, X, y):
        sorted_indices = np.argsort(X)
        X_sorted = X[sorted_indices]
        y_sorted = y[sorted_indices]

        candidates = []
        for i in range(len(X_sorted) - 1):
            if y_sorted[i] != y_sorted[i + 1]:
                midpoint = (X_sorted[i] + X_sorted[i + 1]) / 2
                candidates.append(midpoint)

        if not candidates:
            return X

        best_gain = -1
        best_threshold = None

        for threshold in candidates:
            binary_split = (X >= threshold).astype(int)
            gain = self.information_gain(binary_split, y)

            if gain > best_gain:
                best_gain = gain
                best_threshold = threshold

        return (X >= best_threshold).astype(int)

    def information_gain(self, X, y):
        parent_entropy = self.entropy(y)
        unique_values = np.unique(X)
```

```
        weighted_entropy = 0
        for value in unique_values:
            subset_mask = X == value
            subset_y = y[subset_mask]
            subset_prob = len(subset_y) / len(y)
            subset_entropy = self.entropy(subset_y)
            weighted_entropy += subset_prob * subset_entropy

        information_gain = parent_entropy - weighted_entropy
        return information_gain

    def _build_tree(self, X, y, depth=0):
        if (self.max_depth is not None and depth >= self.max_depth)
or (len(np.unique(y)) == 1):
            unique, counts = np.unique(y, return_counts=True)
            return unique[np.argmax(counts)] # Ambil kelas
mayoritas

        n_features = X.shape[1]
        best_gain = -1
        best_feature = None

        for feature in range(n_features):
            feature_values = X[:, feature] # Ambil semua row
feature

            # Cek kalau fitur continuous. Ambil batas kalau jumlah
row dalam fitur >10 itu continuous
            if len(np.unique(feature_values)) > 10:
                discretized = self.discretize(feature_values, y) #
Discretize
                gain = self.information_gain(discretized, y)
            else:
                gain = self.information_gain(feature_values, y)

            if gain > best_gain:
                best_gain = gain
                best_feature = feature

        if best_gain == 0:
            unique, counts = np.unique(y, return_counts=True)
            return unique[np.argmax(counts)]

        node = {}
        node['feature'] = best_feature
        unique_values = np.unique(X[:, best_feature])

        # Buat tree untuk setiap unique value
        for value in unique_values:
            mask = X[:, best_feature] == value
```

```
            sub_X = X[mask]
            sub_y = y[mask]

            subtree = self._build_tree(sub_X, sub_y, depth + 1)

            if 'branches' not in node:
                node['branches'] = {}
            node['branches'][value] = subtree

        return node

    def fit(self, X, y):
        # X = np.array(X)
        # y = np.array(y)

        self.tree = self._build_tree(X, y)
        return self

    def predict(self, X):
        # X = np.array(X)

        predictions = [self._predict_sample(sample) for sample in
X]
        return np.array(predictions)

    def _predict_sample(self, sample):
        node = self.tree

        while isinstance(node, dict):
            feature = node['feature']
            feature_value = sample[feature]

            if feature_value in node['branches']:
                node = node['branches'][feature_value]
            else:
                break

        # if isinstance(node, dict):
        #     unique, counts =
np.unique(list(node['branches'].values()), return_counts=True)
        #     node = unique[np.argmax(counts)]

        return node
```

ID3DecisionTree memiliki beberapa method antara lain:

1. __init__
   Fungsi dalam inisiasi class ID3DecisionTree, dengan Parameter max_depth menyatakan kedalaman maximal pohon keputusan (untuk membantu mengatasi

overfitting). max_depth memiliki default value None

2. entropy
   Menghitung entropi (ketidakpastian) dari suatu dataset. Ada penambahan 1e-10 dalam log2 untuk menghindari division by 0

3. discretize
   Mengubah fitur kontinu menjadi nilai biner berdasarkan informasi yang diberikan oleh target y. Mencari kandidat nilai threshold yang memisahkan data berdasarkan perubahan kelas pada sorted dataset

4. information_gain
   Menghitung information gain (pengurangan entropi) berdasarkan pembagian data menggunakan fitur X. Semakin besar nilai information gain, semakin baik fitur tersebut untuk membagi dataset.

5. _build_tree
   Fungsi rekursif untuk membangun pohon keputusan. Memilih fitur terbaik berdasarkan information_gain dan membaginya ke dalam cabang-cabang untuk setiap nilai unik fitur tersebut, hingga mencapai kedalaman maksimum atau nilai homogen

6. fit
   Fungsi ini digunakan untuk melatih model pohon keputusan. Memanggil fungsi _build_tree

7. predict
   Fungsi ini memprediksi kelas untuk setiap contoh pada dataset X menggunakan pohon keputusan yang sudah dibangun. Memanggil _predict_sample untuk setiap sampel dalam X.

8. _predict_sample
   Fungsi ini memprediksi kelas dari satu sampel berdasarkan pohon keputusan yang sudah dibangun. Fungsi ini berjalan mengikuti pohon keputusan sesuai dengan fitur pada sampel hingga mencapai daun pohon (prediksi akhir).

Bab 3: Hasil

Lorem ipsum

Pembagian tugas tiap anggota kelompok

No	NIM	Tugas
1	13522010	Naive Bayes
2	13522044	KNN
3	13522059	Preprocessing
4	13522065	ID3

Referensi

- https://www.ibm.com/topics/naive-bayes