TUGAS BESAR 2

IF3170 Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Disusun Oleh:

13522061	Maximilian Sulistiyo
13522075	Marvel Pangondian
13522083	Evelyn Yosiana
13522103	Steven Tjhia

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

DAFTAR ISI

DAFTAR ISI..... 2
Bab 1: Spesifikasi..... 3
Bab 2: Penjelasan Implementasi Algoritma..... 4
Bab 3: Cleaning dan Preprocessing..... 6
Bab 4: Perbandingan Hasil..... 13
Appendix..... 15

Bab 1: Spesifikasi

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, Anda diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah kalian pelajari di kuliah, yaitu **KNN, Gaussian Naive-Bayes, dan ID3** pada dataset **UNSW-NB15**. Rincian spesifikasi untuk tugas besar 2 dapat dilihat sebagai berikut:

1. Implementasi KNN *from scratch*.
    a. Minimal bisa menerima 2 input parameter
        i. Jumlah tetangga
        ii. Metrik jarak antar data point. Minimal dapat menerima 3 pilihan, yaitu Euclidean, Manhattan, dan Minkowski
2. Implementasi Gaussian Naive-Bayes *from scratch*.
3. Implementasi ID3 *from scratch*, termasuk pemrosesan data numerik sesuai materi yang dijelaskan dalam PPT kuliah.
4. Implementasi algoritma poin 1-3 menggunakan *scikit-learn*. Bandingkan hasil dari algoritma *from scratch* dan algoritma *scikit-learn*. Untuk ID3 di *scikit-learn*, gunakan DecisionTreeClassifier dengan parameter criterion='entropy' (memang tidak sama persis dengan ID3, tetapi cukup mendekati)
5. Model harus bisa di-save dan di-load. Implementasinya dibebaskan (misal menggunakan .txt, .pkl, dll).
6. [Bonus] Kaggle Submission pada link berikut.

Implementasi KNN, Gaussian Naive-Bayes, dan ID3 yang *from scratch* bisa dalam bentuk kelas-kelas (class KNN, dst.) yang nantinya akan di-import ke notebook pengerjaan. Untuk implementasi *from scratch*, *library* yang boleh digunakan adalah untuk perhitungan matematika saja seperti numpy dan sejenisnya.

Bab 2: Penjelasan Implementasi Algoritma

2.1. KNN (K-Nearest Neighbor)

Prinsip kerja algoritma KNN yaitu berdasarkan jarak suatu titik ke titik lainnya. Kedekatan ini bisa dihitung menggunakan beberapa metrik jarak yang umum digunakan dalam dunia matematika juga yaitu, *Euclidean*, *Manhattan*, atau *Minkowski*.

Proses train: untuk proses train, pada dasarnya class ini (dalam kode diberi nama KNN_Selfmade) hanya menyimpan data atribut-atribut (X_train) dan data target (y_train) dalam bentuk array numpy. Pada proses training, algoritma KNN tidak melakukan "pembelajaran" layaknya algoritma-algoritma lainnya. KNN hanya "menghafalkan" semua *data points* yang disimpan dalam array numpy tadi sebagai suatu basis pengetahuan (*knowledge base*) untuk dimanfaatkan saat proses prediksi.

Proses predict: untuk proses ini, algoritma KNN melakukan langkah-langkah berikut:

1. Tiap data yang akan diprediksi (X_test) dihitung jaraknya dengan tiap data yang ada pada X_train dengan menggunakan metrik yang dipilih.
2. Kemudian dicari sejumlah N data dengan jarak terdekat dengan data yang dites.
3. Kemudian dari N data terdekat tersebut, dicari modus labelnya yang kemudian akan menjadi hasil prediksi dari data yang dites tersebut.
4. Proses tersebut diulangi untuk seluruh data di X_test.

Parameter:

- neighbors: jumlah tetangga (jumlah titik data terdekat yang akan dicari modusnya, dalam penjelasan proses fit disebut N). By default = 3.
- metric: metrik perhitungan yang digunakan untuk mencari jarak antar dua data. Dalam kode saya terdapat 3 metrik yang dapat dipilih yaitu euclidean, manhattan, dan minkowski. By default = euclidean.
- p: parameter order yang digunakan untuk metrik *Minkowski*. By default = 3.

2.2. Naive Bayes

Prinsip kerja Naive Bayes yang diimplementasikan pada kode di tugas besar ini adalah algoritma berbasis probabilitas yang berdasarkan rumus probabilitas Naive Bayes, seperti yang diajarkan di kelas. Algoritma ini menghitung probabilitas setiap kelas untuk sebuah data baru berdasarkan probabilitas data train, kemudian hasil prediksi diambil dari label yang mendapat probabilitas tertinggi. Implementasi ini lebih cocok untuk data yang fiturnya kategorikal.

Proses train: untuk proses train, pada dasarnya class ini (Naive_Bayes_Selfmade) pertama-tama menghitung frekuensi nilai atribut. Jadi untuk setiap label (v_i), dihitung frekuensi setiap nilai unik pada setiap atribut (a_i). Frekuensi ini digunakan untuk menghitung probabilitas bersyarat $P(a_i \mid v_i)$. Setelah itu, dilakukan kalkulasi probabilitas

prior atau P(v_i) yaitu hasil pembagian jumlah data di label v_i dengan total jumlah data (total_samples = len(y) di kode). Hasil dari kedua langkah tadi disimpan dalam suatu tabel probabilitas yang dalam kode diimplementasi sebagai dictionary (self.class_probabilities dan self.attribute_probabilities)

Proses predict: untuk proses ini, algoritma Naive Bayes melakukan langkah-langkah berikut:

1. Menghitung probabilitas bersyarat P(a_i | v_i) untuk setiap kelas v_i, tapi sedikit berbeda dengan yang ada di kelas, di sini harus diterapkan suatu nilai probabilitas *default* yang sangat kecil (1e-6) untuk menghindari error apabila ada perkalian nol.
2. Probabilitas total untuk kelas vi dengan atribut yang diberikan dihitung dengan rumus:

   $$P(v_i \mid a_1, a_2, \ldots, a_n) = P(v_i) \cdot \prod_i P(a_i \mid v_i)$$

   Hasil ini akan memberikan probabilitas untuk setiap label bagi setiap instance dari data test.
3. Label yang menjadi hasil prediksi adalah label dengan probabilitas total tertinggi

Parameter: Untuk algoritma ini, tidak ada parameter yang perlu diberikan karena rumusnya sudah baku dan semua informasi yang dibutuhkan untuk menghitung rumusnya sudah diketahui oleh algoritma langsung dari data train dan test.

2.3. ID3

Konsep ID3 yaitu pembentukan tree dimana di tiap cabang akan ada 1 fitur untuk menentukan suatu data masuk ke cabang lebih dalam yang mana. Penentuan fitur yang digunakan dalam setiap cabang berdasarkan nilai information gainnya.

Proses train: untuk proses train, intinya yaitu pembentukan pohon secara rekursif. Untuk tiap percabangan, dicari fitur dengan information gain terbesar. Proses pembentukan pohon selesai ketika mencapai max depth, atau total data yang akan di-split kurang dari min samples.

Proses predict: untuk proses ini, intinya tiap data akan di-traverse dari satu cabang ke cabang lainnya yang lebih di bawahnya sampai mencapai leaf. Dari leaf tersebut, data yang akan di-predict mendapatkan labelnya. Hal tersebut dilakukan untuk tiap data yang ingin di-predict.

Bab 3: Cleaning dan Preprocessing

3.1. Menghapus Instance dengan Fitur Null > 90%

Instance dengan null > 90% dihapus karena dianggap tidak dapat memberikan insight dengan benar. Hal ini dikarenakan kebanyakan fiturnya akan di-*impute* dimana proses peng-*impute*-an belum tentu tepat.

```
threshold = len(train_set.columns) * 0.9
train_set_cleaned = train_set.dropna(thresh=threshold,
axis=0)
```

3.2. Mengisi Null Values

Instance dengan null > 90% dihapus karena dianggap tidak dapat memberikan insight dengan benar. Hal ini dikarenakan kebanyakan fiturnya akan di-*impute* dimana proses peng-*impute*-an belum tentu tepat.

Pengisian null values dengan **domain knowledge** menggunakan formula-formula tertentu untuk menghasilkan nilai yang lebih akurat berdasarkan fitur-fitur lainnya. Namun metode ini tidak dapat digunakan untuk seluruh fitur karena tidak semua fitur memiliki formulanya berdasarkan fitur lainnya.

```
if 'spkts' in df and 'dpkts' in df:
    df['dur'].fillna((df['spkts'] + df['dpkts']) /
(df['sload'] + df['dload']), inplace=True)
else:
    df['dur'].fillna(df['dur'].mean(), inplace=True)
—————————————————————————————————————
if 'spkts' in df and 'smean' in df:
    df['sbytes'].fillna(df['spkts'] * df['smean'],
inplace=True)
if 'dpkts' in df and 'dmean' in df:
    df['dbytes'].fillna(df['dpkts'] * df['dmean'],
inplace=True)
—————————————————————————————————————
if 'spkts' in df:
    df['sloss'].fillna(df['spkts'] * 0.01, inplace=True)
if 'dpkts' in df:
    df['dloss'].fillna(df['dpkts'] * 0.01, inplace=True)
—————————————————————————————————————
```

```
'dur' in df:
    df['sload'].fillna(df['sbytes'] / df['dur'],
inplace=True)
    df['dload'].fillna(df['dbytes'] / df['dur'],
inplace=True)
—————————————————————————————————
if 'spkts' in df:
    df['smean'].fillna(df['sbytes'] / df['spkts'],
inplace=True)
if 'dpkts' in df:
    df['dmean'].fillna(df['dbytes'] / df['dpkts'],
inplace=True)
—————————————————————————————————
if 'dur' in df:
    df['sjit'].fillna(df['dur'] * 0.01, inplace=True)
    df['djit'].fillna(df['dur'] * 0.01, inplace=True)
—————————————————————————————————
if 'spkts' in df:
    df['sinpkt'].fillna(df['dur'] / df['spkts'],
inplace=True)
if 'dpkts' in df:
    df['dinpkt'].fillna(df['dur'] / df['dpkts'],
inplace=True)
—————————————————————————————————
df['tcprtt'].fillna(df['synack'] + df['ackdat'],
inplace=True)
df['synack'].fillna(df['tcprtt'] - df['ackdat'],
inplace=True)
df['ackdat'].fillna(df['tcprtt'] - df['synack'],
inplace=True)
```

Pengisian null values pada *categorical feature* menggunakan **modus** karena modus dapat merepresentasikan kecenderungan kategori.

```
df['proto'].fillna(df['proto'].mode()[0], inplace=True)
df['state'].fillna(df['state'].mode()[0], inplace=True)
df['service'].fillna(df['service'].mode()[0],
inplace=True)
df['service'].fillna(df['service'].mode()[0],
inplace=True)
df['trans_depth'].fillna(df['trans_depth'].mode()[0],
inplace=True)
```

```
df['sttl'].fillna(df['sttl'].mean(), inplace=True)
df['dttl'].fillna(df['dttl'].mean(), inplace=True)
df['spkts'].fillna(df['spkts'].mean(), inplace=True)
df['dpkts'].fillna(df['dpkts'].mean(), inplace=True)
df['swin'].fillna(df['swin'].mean(), inplace=True)
df['dwin'].fillna(df['dwin'].mean(), inplace=True)
df['stcpb'].fillna(df['stcpb'].mean(), inplace=True)
df['dtcpb'].fillna(df['dtcpb'].mean(), inplace=True)
df['response_body_len'].fillna(df['response_body_len'].mea
n(), inplace=True)
count_features = [
    'ct_state_ttl', 'ct_flw_http_mthd', 'ct_ftp_cmd',
'ct_srv_src', 'ct_srv_dst',
    'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm',
'ct_dst_sport_ltm', 'ct_dst_src_ltm'
]
for feature in count_features:
    df[feature].fillna(df[feature].mean(), inplace=True)
```

Pengisian null values pada *categorical feature* menggunakan **mean** karena nilai mean dapat menginterpretasikan pusat data sehingga tidak mengganggu distribusi data. Pengisian dengan mean cocok digunakan untuk data yang berdistribusi normal.

```
df['sttl'].fillna(df['sttl'].mean(), inplace=True)
df['dttl'].fillna(df['dttl'].mean(), inplace=True)
df['spkts'].fillna(df['spkts'].mean(), inplace=True)
df['dpkts'].fillna(df['dpkts'].mean(), inplace=True)
df['swin'].fillna(df['swin'].mean(), inplace=True)
df['dwin'].fillna(df['dwin'].mean(), inplace=True)
df['stcpb'].fillna(df['stcpb'].mean(), inplace=True)
df['dtcpb'].fillna(df['dtcpb'].mean(), inplace=True)
df['response_body_len'].fillna(df['response_body_len'].mea
n(), inplace=True)
count_features = [
    'ct_state_ttl', 'ct_flw_http_mthd', 'ct_ftp_cmd',
'ct_srv_src', 'ct_srv_dst',
    'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm',
'ct_dst_sport_ltm', 'ct_dst_src_ltm'
]
for feature in count_features:
    df[feature].fillna(df[feature].mean(), inplace=True)
```

Pengisian dengan nilai 0 dilakukan dengan asumsi tidak adanya fitur.

```
df['is_sm_ips_ports'].fillna(0, inplace=True)
df['is_ftp_login'].fillna(0, inplace=True)
```

3.3. Menghapus Outlier dan Data yang Tidak Make Sense

Penghapusan ini bertujuan untuk menghapus noise dari data-data yang invalid.

```
def clean_outliers(df):

    valid_states = ['ACC', 'CLO', 'CON', 'ECO', 'ECR',
'FIN', 'INT', 'MAS', 'PAR', 'REQ', 'RST', 'TST', 'TXD',
'URH', 'URN', '-']
    df = df[df['state'].isin(valid_states)]

    df = df[df['dur'] >= 0]

    df = df[(df['sbytes'] >= 0) & (df['dbytes'] >= 0)]

    df = df[(df['sttl'].between(0, 255)) &
(df['dttl'].between(0, 255))]

    df = df[(df['spkts'] >= 0) & (df['dpkts'] >= 0)]

    df = df[(df['sjit'] >= 0) & (df['djit'] >= 0)]

    df = df[(df['sinpkt'] >= 0) & (df['dinpkt'] >= 0)]

    df = df[(df['tcprtt'] >= 0) & (df['synack'] >= 0) &
(df['ackdat'] >= 0)]

    binary_columns = ['is_sm_ips_ports', 'is_ftp_login']
    for col in binary_columns:
        df = df[df[col].isin([0, 1])]

    df = df[df['trans_depth'] >= 0]
    df = df[df['response_body_len'] >= 0]

    df = df[(df['smean'] >= 0) & (df['dmean'] >= 0)]

    count_fields = ['ct_state_ttl', 'ct_flw_http_mthd',
'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst',
                    'ct_dst_ltm', 'ct_src_ltm',
'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm']
    for field in count_fields:
        df = df[df[field] >= 0]
```

```
    return df
```

3.4. Menghapus Duplikat

```
train_set_cleaned = train_set_cleaned.drop_duplicates()
```

3.5. Encoding

```
one_hot_encode_column = ["proto", "state", "service"]
label_encode_column = ["is_sm_ips_ports", "is_ftp_login"]
target_encode_column = []

if one_hot_encode_column:
    df = one_hot_encode(df, columns=one_hot_encode_column)
if label_encode_column:
    df = label_encode(df, columns=label_encode_column)
if target_encode_column:
    for column in target_encode_column:
        df = target_encode(df, target_column=column)
```

3.6. Scaling

```
min_max_scale_column = []
standardize_column = [
    "dur", "sbytes", "dbytes", "sttl", "dttl", "sloss",
"dloss", "sload", "dload", "spkts", "dpkts", "swin",
"dwin",
    "stcpb", "dtcpb", "smean", "dmean", "trans_depth",
"response_body_len", "sjit", "djit", "sinpkt", "dinpkt",
    "tcprtt", "synack", "ackdat", "ct_state_ttl",
"ct_flw_http_mthd", "ct_ftp_cmd", "ct_srv_src",
"ct_srv_dst",
    "ct_dst_ltm", "ct_src_ltm", "ct_src_dport_ltm",
"ct_dst_sport_ltm", "ct_dst_src_ltm"
]
robust_scale_column = []
log_transform_column = ["sload", "dload",
"response_body_len", "sloss", "dloss"]
```

```
if standardize_column:
    df = standardize(df, columns=standardize_column)
if min_max_scale_column:
    df = min_max_scale(df, columns=min_max_scale_column)
if robust_scale_column:
    df = robust_scale(df, columns=robust_scale_column)
if log_transform_column:
    df = log_transform(df, columns=log_transform_column)
```

3.7. Undersampling

Undersampling dilakukan agar hasil tidak bias (dan juga untuk mengurangi training time).

```
def undersample_data(X, y, target_count=4000,
upsample=False):

    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)

    data = pd.concat([X, pd.Series(y, name='label')],
axis=1)
    resampled_data = []

    for label, group in data.groupby('label'):
        if len(group) > target_count:
            sampled_group = group.sample(n=target_count,
random_state=42)
        elif upsample and len(group) < target_count:
            sampled_group = group.sample(n=target_count,
replace=True, random_state=42)
        else:
            sampled_group = group
        resampled_data.append(sampled_group)

    resampled_data = pd.concat(resampled_data)
    resampled_X = resampled_data.drop(columns=['label'])
    resampled_y = resampled_data['label']

    print("Original class distribution:", Counter(y))
    print("Resampled class distribution:",
Counter(resampled_y))

    return resampled_X, resampled_y
```

Bab 4: Perbandingan Hasil

4.1. KNN

Cleaned	Undersampled	Scaled	Encoded	Scratch vs Library	Scratch vs Val Set
✔	✔	✔	✔	0.9095212295759788	0.5988194701873449
✔	✔		✔	0.8494681912800479	0.5153554421283755
	✔	✔	✔	0.9102911403233626	0.630899084661667
	✔		✔	0.8429382075337193	0.526733012061935

Dari hasil di atas diketahui bahwa hasil KNN dengan scaling relatif lebih bagus daripada yang tanpa scaling.

4.2. Naive Bayes

Cleaned	Undersampled	Scaled	Encoded	Scratch vs Library	Scratch vs Val Set
✔	✔	✔	✔	0.3014628304200291	0.5582708374918018
✔	✔		✔	0.4156947731614816	0.6834811371866891
	✔	✔	✔	0.26316689954090505	0.5468932675582423
	✔		✔	0.4150389232655622	0.6858478998545724
✔		✔	✔	0.13225355727280505	0.6672844962787647

✔			✔	0.44306937751290315	0.7356354615187203
		✔	✔	0.3009495565884399	0.6855912629387778
			✔	0.23368216943739484	0.7574495993612592

Dari hasil di atas diketahui bahwa hasil Naive Bayes tanpa di-scaling relatif lebih bagus daripada yang di-scaling. Selain itu, hasil yang tidak di-cleaning relatif lebih bagus daripada yang di-cleaning.

4.3. ID3

Cleaned	Undersampled	Scaled	Encoded	Scratch vs Library	Scratch vs Val Set
✔	✔	✔	✔	0.8504947389432262	0.3839858564544184
✔	✔		✔	0.9672645356297584	0.662950183923123
	✔	✔	✔	0.5333200262339959	0.3298354672217628
	✔		✔	0.9737945193760872	0.6991645042630243

Dari hasil di atas diketahui bahwa hasil ID3 tanpa di-scaling relatif lebih bagus daripada yang di-scaling.

Appendix

Kontribusi Anggota Kelompok

NIM	Nama	Kontribusi
13522061	Maximilian Sulistiyo	EDA, Split Training Set and Validation Set, Data Cleaning and Preprocessing, Pipelining, Modeling and Validation (KNN, Naive Bayes, and ID3).
13522075	Marvel Pangondian	EDA, Split Training Set and Validation Set, Data Cleaning and Preprocessing, Pipelining, Modeling and Validation (KNN, Naive Bayes, and ID3).
13522083	Evelyn Yosiana	EDA, Split Training Set and Validation Set, Data Cleaning and Preprocessing, Pipelining, Modeling and Validation (KNN, Naive Bayes, and ID3).
13522103	Steven Tjhia	EDA, Split Training Set and Validation Set, Data Cleaning and Preprocessing, Pipelining, Modeling and Validation (KNN, Naive Bayes, and ID3).