Laporan Tugas Besar 2
IF3170 Intelegensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:
Kelompok 15
13522022 - Renaldy Arief Susanto
13522084 - Dhafin Fawwaz Ikramullah
13522189 - Abdul Rafi Radityo Hutomo
13522109 - Azmi Mahmud Bazeid

Institut Teknologi Bandung

Daftar Isi

I. Deskripsi Persoalan

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**. Yang akan menjadi pembahasan dalam laporan ini antara lain sebagai berikut:

1. Implementasi KNN *from scratch* yang dapat menerima 2 input parameter yaitu:
    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..
4. Implementasi algoritma poin 1-3 menggunakan *scikit-learn* serta perbandingan hasil dari algoritma *from scratch* dan algoritma *scikit-learn*.
5. Save & Load Model.
6. 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.

II. Pembahasan

A. Penejelasan Algoritma

1. K-Nearest Neighbor (KNN)

KNN merupakan salah satu algoritma *Supervised Learning* yang dilakukan untuk mengklasifikasi data baru dengan menghitung jarak antara data baru dengan seluruh data dalam dataset. Jarak tersebut dapat dihitung dengan algoritma seperti Euclidean Distance untuk data numerik dan Manhattan Distance untuk data categorical. Lalu dipilih k data terdekat berdasarkan jarak yang sudah dihitung tadi.

Kelebihan:
- Lebih mudah diimplementasikan dibandingkan dengan algoritma lain.
- Tidak membutuhkan proses training.

Kekurangan:
- Membutuhkan memori yang cukup besar karena seluruh dataset harus disimpan saat ingin melakukan prediksi.
- Processing akan membutuhkan waktu yang cukup lama untuk jumlah dataset yang besar.
- Bergantung pada pilihan nilai dari K dan algoritma untuk menentukan jarak.

Berikut ini implementasi dari KNN.

```
from typing import Any, List, Callable, Tuple
from heapq import heappush, heappop

MatrixLike = List[List[Any]]
ArrayLike = List[Any]

class KNeighborsClassifierFromScratch():
    n_neighbors: int
    distance_func: Callable
    X: MatrixLike
    y: MatrixLike | ArrayLike
    _categorical_column_indexes: List[int]
    _is_categorical_list: List[bool]
    algorithm: str

    def __init__(
        self,
        n_neighbors: int = 5,
        categorical_column_indexes: List[int] = [],
        algorithm: str = "euclidean" # euclidean, manhattan, minkowski
    ) -> None:
        self.n_neighbors = n_neighbors
        self._categorical_column_indexes = categorical_column_indexes
        self._is_categorical_list: List[bool] = []
        self.algorithm = algorithm

    def fit(self, X: MatrixLike, y: MatrixLike | ArrayLike) ->
'KNeighborsClassifierFromScratch':
        self.X = X
        self.y = y
```

```
        self._is_categorical_list = [i in self._categorical_column_indexes for i
in range(len(X[0]))] # instead of just [2, 4], we do [False, False, True, False,
True, False, False] (assuming len(X[0]) == 7)
        return self

    def euclidean_distance(self, feature: ArrayLike, group: ArrayLike) -> float:
        distance = 0
        for i in range(len(feature)):
            # if isinstance(feature[i], str) or isinstance(group[i], str):
continue # no need because we're using one hot
            distance += (feature[i] - group[i]) ** 2
        # distance = distance ** 0.5
        return distance

    def manhattan_distance(self, feature: ArrayLike, group: ArrayLike) -> float:
        distance = 0
        for i in range(len(feature)):
            # if isinstance(feature[i], str) or isinstance(group[i], str):
continue # no need because we're using one hot
            distance += abs(feature[i] - group[i])
        return distance

    def minkowski_distance(self, feature: ArrayLike, group: ArrayLike, p: float
= 3) -> float:
        distance = 0
        for i in range(len(feature)):
            # if isinstance(feature[i], str) or isinstance(group[i], str):
continue # no need because we're using one hot
            distance += abs(feature[i] - group[i]) ** p
        # distance = distance ** 1/p
        return distance

    def predict(self, X: MatrixLike) -> ArrayLike:
        predictions: List[any] = []

        distance_func = self.euclidean_distance
        if self.algorithm == "manhattan":
            distance_func = self.manhattan_distance
        elif self.algorithm == "minkowski":
            distance_func = self.minkowski_distance
        self.distance_func = distance_func

        progress = 0
        max_progress = len(X)
        for new_row in X:
            progress += 1
            print(f"\rProgress:
{progress}/{max_progress}|{progress/max_progress}", end="")


            distance_list = []
            for i, dataset_row in enumerate(self.X):
                distance = distance_func(new_row, dataset_row)
                heappush(distance_list, (distance, dataset_row, self.y[i]))
                if len(distance_list) > self.n_neighbors:
                    heappop(distance_list)
            group_by = {}
            for distance, row, classification in distance_list:
```

```
                if classification not in group_by:
                    group_by[classification] = 0
                group_by[classification] += 1

            max_group = None
            max_count = 0
            for classification, count in group_by.items():
                if count > max_count:
                    max_count = count
                    max_group = classification

            predictions.append(max_group)

        print()

        return predictions
```

2. Naive-Bayes

Naive Bayes adalah salah satu algoritma machine learning berbasis probabilistik yang digunakan untuk klasifikasi. Algoritma ini didasarkan pada Teorema Bayes dengan asumsi "naif" bahwa setiap fitur saling independen terhadap variabel conditional kelas-kelasnya (conditional independence). Ketika x1, x2, …, xn saling independen terhadap variabel target Ci, maka P(Ci | x1,x2,..,xn) = P(Ci) * P(x1,..,xn|Ci) / P(x1,..,xn). Karena P(x1,..,xn) tidak bergantung pada kelas Ci maka P(Ci | x1,x2,..,xn) proporsional terhadap P(Ci) * P(x1,..,xn|Ci). Karena kondisi conditional independence, maka P(Ci | x1,x2,..,xn) proporsional terhadap P(Ci) * P(x1|Ci) * P(x2|Ci) * … * P(xn|Ci).

Dengan Gaussian naive Bayes, diasumsikan bahwa distribusi setiap fitur adalah kontinu dan terdistribusi normal atau mendekati normal sehingga dapat dihitung pdf untuk setiap nilai untuk setiap kelasnya:

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

(wikipedia.org)

Lalu dapat dipilih kelas dengan peluang terbesar.

1. Inisialisasi Model

    - Parameter model seperti rata-rata (mean), variansi, prior probabilitas kelas, dan parameter smoothing (variance smoothing) diinisialisasi.

- Scaling fitur: Sebelum melatih model, semua fitur diskalakan menggunakan StandardScaler untuk memastikan setiap fitur memiliki nilai yang terpusat pada 0 dengan variansi 1.

2. Pelatihan Model (fit Method)

Ketika model dilatih menggunakan data train X dan label y, langkah berikut dilakukan:

- Menghitung prior probabilitas kelas:
  Model menghitung probabilitas awal dari setiap kelas ($P(Y=c)$) berdasarkan proporsi jumlah data untuk masing-masing kelas. Ini dikenal sebagai prior probability.

- Menghitung rata-rata dan variansi untuk setiap fitur per kelas:
  a. Untuk setiap kelas, model menghitung rata-rata (μ) dan variansi ($\sigma 2$) dari setiap fitur. Ini membantu memodelkan distribusi normal Gaussian untuk setiap fitur, yang diasumsikan sebagai distribusi data.

- Menambahkan smoothing ke variansi:
  a. Variance smoothing diterapkan untuk memastikan nilai variansi tidak menjadi nol. Ini mencegah pembagian dengan nol ketika menghitung probabilitas bersyarat.

3. Perhitungan Log-Likelihood Bersama (_joint_log_likelihood Method)

Setelah model dilatih, saat memprediksi data baru (X), model menghitung log-likelihood bersama ($P(Y \mid X)$) untuk setiap kelas berdasarkan:

a. Log prior: Menggunakan prior probabilitas kelas ($\log P(Y)$).

b. Log-likelihood fitur: Untuk setiap fitur, model menggunakan distribusi Gaussian. Kemudian menghitung log-likelihood total dengan menjumlahkan kontribusi dari semua fitur.

4. Prediksi Probabilitas (predict_proba Method)

Setelah log-likelihood dihitung, model mengonversinya menjadi probabilitas dengan cara:

1. Normalisasi log-likelihood: Log-likelihood dari semua kelas dinormalisasi menggunakan fungsi stabil (log-sum-exp) untuk mencegah angka terlalu besar/kecil.
2. Eksponensiasi log-likelihood: Log-likelihood kemudian diubah menjadi probabilitas yang sebenarnya menggunakan fungsi eksponensial.

5. Prediksi Kelas (predict Method)

Untuk memprediksi kelas data baru:

1. Memilih kelas dengan probabilitas maksimum: Model menghitung probabilitas setiap kelas untuk data input. Kelas dengan probabilitas tertinggi dipilih sebagai hasil prediksi (argmax $P(Y \mid X)$)

Kelebihan:
- Processing cukup cepat dan efisien untuk jumlah data yang besar.
- Cukup mudah diimplementasikan untuk dataset dengan fitur yang banyak.
- Tidak terlalu rentan terhadap overfit jika datanya sedikit.

Kekurangan:
- Adanya asumsi bahwa masing-masing fitur saling independen terhadap variabel target yang bisa jadi tidak sesuai.
- Jika antar fitur memiliki korelasi yang cukup besar, performa prediksi akan memburuk.
- Keberhasilan sangat bergantung pada pengetahuan awal atau pengetahuan mengenai masa sebelumnya untuk melakukan prediksi.

3. Iterative Dichotomiser 3 (ID3)

ID3 merupakan salah satu algoritma Supervised Learning yang dilakukan untuk mengklasifikasi data baru dengan membuat *decision tree* dan memanfaatkan konsep *information gain* dan *entropy* untuk memilih atribut yang akan dijadikan cabang berikutnya. Algoritma ini diimplementasikan mengikuti salindia kuliah.

Decision Tree Learning (Russel & Norvig, 2021)

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
  tree ← a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v_k\}$
    subtree ← DECISION-TREE-LEARNING(exs, $attributes - A$, *examples*)
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

Information gain
Gain ratio

|A|=2: binary tree
|A| min 2: n-ary tree

Pohon diimplementasikan sebagai *recursive dictionary* dalam Python.

```
from collections import defaultdict
T = lambda: defaultdict(T)
```

Dan berikut adalah implementasi fungsi ID3.

```
def plurality(examples: DataFrame, targetCol) :
	return examples[targetCol].mode()[0]

def make_id3_tree(cur: DataFrame, attr: list[str], prev: DataFrame, targetCol, maxdepth
= 10, depth = 0) :


	root = T()
	print(".", end = '')
	if cur.empty:
	root[result] = plurality(prev, targetCol)
	elif len(cur[targetCol].unique()) == 1 :
	root[result] = cur[targetCol].unique()[0]
	elif not attr or maxdepth == depth :
	root[result] = plurality(cur, targetCol)

	else :
	A = max(attr, key = lambda at: calcgain(dataframe=cur, gainCol=at,
targetCol=targetCol))

	for vk in cur[A].unique() :
		nxt = cur[cur[A] == vk]
		nxt_attr = attr.copy()
		nxt_attr.remove(A)
		subtree = make_id3_tree(nxt, nxt_attr, cur, targetCol, maxdepth, depth +
1)
		root[(A, vk)] = subtree

	if prev is None : print("Done!")
	return root
```

Objek pohon yang dikembalikan akan digunakan dalam fungsi *classify*.

```
def id3classify(tree: T, data: dict) :

	while result not in tree :
	found = 0
	for key, subtree in tree.items() :

		attr, attr_val = key
		if data[attr] == attr_val :
		    tree = subtree
		    found = 1
		    break

	if not found :
		tree = tree[next(iter(tree.keys()))]

	return tree[result]
```

Kelebihan:
- Mudah diimplementasikan.
- Dapat menangani data kategorikal dan numerik.

Kekurangan:
- Rentan terhadap overfit.
- Tidak efisien untuk dataset dengan jumlah besar.
- Akan ada bias terhadap atribut yang memiliki banyak nilai unik.

B. Cleaning dan Preprocessing

1. Handling missing data

Untuk menangani data yang tidak ada, dilakukan proses impute. Impute dilakukan dengan strategi mengganti dengan mean/median untuk data numerik berdasarkan dengan nilai skewnya. Untuk data kategorikal dilakukan impute dengan mengganti datanya dengan modusnya. Hal ini dilakukan untuk data training dan test/validasi.

2. Dealing with outlliers

Untuk mendeteksi outlier kita menggunakan strategi mencari nilai yang di luar range [q1 - 1.5 IQR, q3 + 1.5 IQR] dengan q1 adalah kuartil 1, q3 adalah kuartil 3 dan IQR adalah interquartil range. Kemudian, untuk menangani outlier kita mengganti nilainya dengan nilai median. Proses ini dilakukan untuk data training dan test/validasi

3. Remove duplicates

Dalam melakukan training model, dilakukan penghapusan untuk data duplikat hal ini dilakukan agar data tidak memiliki bias terhadap data yang terduplikat tersebut. Proses ini dilakukan hanya untuk data training.

4. Feature engineering

Pada tahap ini dilakukan pemilihan dan manipulasi fitur-fitur yang ada, tetapi kami belum sempat untuk mengeksplorasi berbagai kemungkinan yang ada sehingga pada implementasinya, hanya diterapkan feature selection, yaitu penghapusan atribut id dan label dalam pembuatan model.

Preprocessing

1. Feature scaling

Feature Scaling adalah proses merubah skala dari nilai pada fitur numerik untuk memenuhi rentang atau distribusi tertentu. Pada pengerjaan tugas besar ini, kami mencoba standardisasi dan normalisasi. Berdasarkan hasil percobaan normalisasi lebih efektif sehingga pada pekerjaan akhir kami menggunakan normalisasi.

2. Feature encoding

Feature encoding adalah proses merubah data menjadi bentuk yang dapat dipahami oleh algoritma machine learning. Pada dataset yang diberikan, atribut yang perlu diubah adalah

atribut yang memiliki nilai nominal. Strategi encoding yang kami gunakan adalah OneHotEncoding, yaitu membagi setiap nilai nominal yang mungkin menjadi nilai binary.

3. Handling imbalanced datasets

Menangani dataset yang memiliki bias dapat dilakukan dengan metode oversampling, tetapi implementasi kami belum diintegrasikan ke keseluruhan program.

4. Data normalization

Normalisasi data adalah proses yang serupa dengan Feature Scaling dan diimplementasikan dengan MinMaxScaler.

5. Dimensionality Reduction

Dimensionality Reduction adalah proses mengurangi dimensi data dengan menggabungkan nilai dari satu atau lebih atribut. Salah satu opsi dimensionality reduction adalah dengan Principal Component Analysis (PCA) yang kami implementasikan, tetapi belum dilakukan tuning untuk parameternya sehingga belum diintegrasikan ke flow program.

C. Perbandingan Hasil Prediksi dari Berbagai Algoritma

1. K-Nearest Neighbor (KNN)

Hasil yang didapatkan melalui implementasi from scratch dengan sklearn adalah sama persis. Tidak ada perbedaan 1 elemen pun. Saat dibandingkan antara *from scratch* dengan validation set dan *sklearn* dengan validation set, keduanya memiliki f1 score yaitu 0.0107. Rendahnya skor ini terjadi karena bisa jadi pemilihan nilai k yang kurang optimal, dan bisa juga karena kelemahan dari KNN itu sendiri yang kurang baik jika harus mencari jarak dengan fitur antar kategori. KNN Juga bergantung dengan scaling dari fitur sehingga walaupun dilakukan normalisasi, bisa saja fitur yang seharusnya lebih berperngaruh malah kurang berpengaruh.

From Scratch VS Sklearn

```
differences, percentage = count_differences(predictions, sklearn_prediction)
print("Difference count:", differences)
print("Percentage:", percentage * 100, "%")
score = get_score(predictions, sklearn_prediction)
print("Score:", score)
```

```
Difference count: 0
Percentage: 100.0 %
Score: 1.0
```

From Scratch VS Validation Set

```
differences, percentage = count_differences(predictions, val_df_attack_cat.values.tolist())
print("Difference count:", differences)
print("Percentage:", percentage * 100, "%")
score = get_score(predictions, val_df_attack_cat.values.tolist())
print("Score:", score)
```

```
Difference count: 34186
Percentage: 2.517893296073459 %
Score: 0.01073949346313678
```

From Sklearn VS Validation Set

```
differences, percentage = count_differences(sklearn_prediction, val_df_attack_cat.values.tolist())
print("Difference count:", differences)
print("Percentage:", percentage * 100, "%")
score = get_score(sklearn_prediction, val_df_attack_cat.values.tolist())
print("Score:", score)
```

```
Difference count: 34186
Percentage: 2.517893296073459 %
Score: 0.01073949346313678
```

2. Naive-Bayes

```
Custom Implementation Results:
                precision    recall  f1-score   support

      Analysis       0.02      0.06      0.03       384
      Backdoor       0.03      0.14      0.05       367
           DoS       0.26      0.00      0.01      2459
      Exploits       0.94      0.16      0.27      6636
       Fuzzers       0.19      0.06      0.09      3637
       Generic       0.76      0.93      0.84      7975
        Normal       1.00      1.00      1.00     11254
Reconnaissance       0.11      0.01      0.01      2070
     Shellcode       0.05      1.00      0.10       259
         Worms       0.01      0.86      0.01        28

      accuracy                           0.58     35069
     macro avg       0.34      0.42      0.24     35069
  weighted avg       0.72      0.58      0.57     35069

Custom Model Accuracy: 0.5801

 Scikit-learn Implementation Results:
                precision    recall  f1-score   support

      Analysis       0.00      0.00      0.00       384
      Backdoor       0.00      0.00      0.00       367
           DoS       0.00      0.00      0.00      2459
      Exploits       0.00      0.00      0.00      6636
       Fuzzers       0.00      0.00      0.00      3637
       Generic       0.45      0.99      0.62      7975
        Normal       0.53      0.83      0.65     11254
Reconnaissance       0.00      0.00      0.00      2070
     Shellcode       0.00      0.00      0.00       259
         Worms       0.00      0.00      0.00        28

      accuracy                           0.49     35069
     macro avg       0.10      0.18      0.13     35069
  weighted avg       0.27      0.49      0.35     35069

 Scikit-learn Model Accuracy: 0.4925
```

Walaupun Gaussian Naive Bayes cukup sederhana, ada beberapa aspek yang dapat memengaruhi keakurasian implementasi Gaussian Naive Bayes yaitu diantara:

1. Numerical Stability: Perhitungan dengan log agar tidak overflow dan underflow
2. Smoothing: Penggunaan angka konstanta kecil yang variatif
3. Handling kasus khusus: Ketika nilai probabilitas yang ekstrem atau bernilai 0.

3. Iterative Dichotomiser 3 (ID3)

Hasil prediksi model kami menggunakan ID3 ada dua yang sekiranya perlu dibahas, yaitu sebagai berikut.

a. Hasil ID3 ketika digunakan untuk data yang **belum** di-*preprocess*.
b. Hasil ID3 ketika digunakan untuk data yang **sudah** di-*preprocess*.

Berdasarkan hasil yang kami temukan, data yang belum di-*preprocess* mempunyai hasil yang lebih baik dari data yang sudah di-*preprocess*. Hasil untuk data yang sudah di-*preprocess* biasanya mempunyai akurasi sekitar **0.3** ketika diuji pada *validation set*, tetapi hanya berupa satu atau dua jenis klasifikasi berbeda sehingga **tidak bisa** dikatakan sebagai hasil yang baik.

Hasil untuk data yang belum di-*preprocess* mempunyai akurasi hanya sekitar **0.25**, tetapi hasil tersebut konsisten dengan submisi di *kaggle*. Kemudian hasil akurasi untuk model scikit-learn adalah **0.7**.

```
from sklearn.tree import DecisionTreeClassifier

obj = DecisionTreeClassifier(criterion='entropy')

obj.fit(train_df_no_attack_cat.values.tolist(), attack_cat_df.values.tolist())
res = obj.predict(val_df_no_attack_cat.values.tolist())

hit = 0
for i in range(len(val_df_attack_cat)) :
    if val_df_attack_cat[i] == res[i] :
        hit += 1

acc = hit / len(val_df_attack_cat)
acc
```

```
0.7798910718868516
```

III. Kontribusi Tiap Anggota Kelompok

NIM	Nama	Kontribusi
13522022	Renaldy Arief Susanto	Algoritma ID3, menyusun laporan
13522084	Dhafin Fawwaz Ikramullah	Algoritma KNN, menyusun laporan
13522189	Abdul Rafi Radityo Hutomo	Data Preprocessing, menyusun laporan
13522109	Azmi Mahmud Bazeid	Algoritma Naive Bayes, menyusun laporan

IV. Referensi

- The UNSW-NB15 Dataset | UNSW Research
- UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set) | IEEE Conference Publication | IEEE Xplore
- K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks
- What Are Naïve Bayes Classifiers? | IBM
- Decision Trees: ID3 Algorithm Explained | Towards Data Science
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier