INFORMATICS ENGINEERING STUDY PROGRAM
SCHOOL OF ELECTRICAL ENGINEERING DAN INFORMATICS
INSTITUT TEKNOLOGI BANDUNG

TUGAS BESAR 2

Implementasi Algoritma Pembelajaran Mesin

Dibuat oleh:
Group 22 - *terakhir aja isi nama dulu*

Anggota:

1. 13522013 - Denise Felicia Tiowanni
2. 13522074 - Muhammad Naufal Aulia
3. 13522101 - Abdullah Mubarak

IF3170 - Inteligensi Artifisial

2024

Daftar Isi

Bab I. Implementasi Algoritma Pembelajaran Mesin

1.1. Implementasi Algoritma KNN

K-Nearest Neighbors (KNN) adalah algoritma berbasis *lazy learning* dan *instance-based learning* yang digunakan untuk klasifikasi dan regresi. Berikut adalah **langkah utama** dari algoritma KNN:

1. Menghitung jarak antara titik data yang akan diprediksi (test data) dan seluruh titik data dalam data latih (training data).
2. Memilih k titik tetangga terdekat berdasarkan jarak.
3. Menentukan hasil prediksi berdasarkan mayoritas label (klasifikasi) atau rata-rata nilai label (regresi) dari tetangga terdekat.

Berikut adalah metode-metode utama yang dibuat untuk mendukung fungsionalitas kelas KNN:

➜ **euclidean_distance(self, X, Y)**: merupakan method untuk menghitung jarak Euclidean antara dua himpunan data X dan Y. Jarak Euclidean dihitung dengan rumus sebagai berikut.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

➜ **manhattan_distance(self, X, Y)**: merupakan method untuk menghitung jarak Manhattan antara dua himpunan data X dan Y. Jarak Manhattan dihitung dengan rumus sebagai berikut.

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

➜ **minkowski_distance(self, X, Y)**: merupakan method untuk menghitung jarak Minkowski antara dua himpunan data X dan Y. Jarak Minkowski dihitung dengan rumus sebagai berikut.

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

Parameter p lah yang digunakan untuk menyesuaikan jarak Minkowski, dimana jika p = 2 maka jarak ini sama dengan jarak Euclidean, dan jika p = 1 maka jarak ini sama dengan jarak Manhattan.

- ➜ **get_distance(self, X, Y)**: merupakan method memilih metode perhitungan jarak sesuai parameter distance.
- ➜ **get_neighbors(self, test_points)**: merupakan method untuk menghitung jarak antara titik uji (test_points) dan seluruh titik latih serta menentukan k tetangga terdekat berdasarkan jarak terkecil.
- ➜ **fit(self, X_train, y_train)**: merupakan method untuk menyimpan data latih X_train dan label y_train ke dalam objek KNN.
- ➜ **predict(self, X_test)**: merupakan method untuk memprediksi output untuk data uji X_test berdasarkan tetangga terdekat. Untuk regresi, menghitung rata-rata label dari tetangga terdekat.

1.2. Implementasi Algoritma Gaussian Naive-Bayes

Algoritma ini diimplementasikan dalam kelas GaussianNaiveBayes, dimana nantinya akan digunakan untuk klasifikasi data berdasarkan pendekatan probabilitas Bayes. Berikut adalah **langkah utama** dari algoritma Gaussian Naive-Bayes:

1. Identifikasi kelas unik yang ada dalam target.
2. Untuk setiap kelas, hitung rata-rata dan variansi untuk setiap fitur.
3. Hitung prior probabilities.

Terdapat sejumlah method yang dibuat untuk mendukung fungsionalitas algoritma ini, yaitu sebagai berikut.

- ➜ **fit(self, X, y)**: merupakan method untuk training berdasarkan data input (X) dan label target (y).
- ➜ **_gaussian_pdf(self, x, mean, variance)**: method sebagai fungsi untuk menghitung probabilitas densitas pada setiap nilai fitur menggunakan distribusi Gaussian. Perhitungan ini melibatkan rata-rata, varians, dan nilai fitur berdasarkan formula distribusi normal. Nilai epsilon juga diterapkan untuk

menghindari kesalahan matematika seperti pembagian dengan nol. Berikut adalah rumus yang digunakan untuk perhitungan rata-rata dan varians.

$$\mu_j = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{ij}, \quad \sigma_j^2 = \frac{1}{N_k} \sum_{i=1}^{N_k} (x_{ij} - \mu_j)^2$$

➜ **_calculate_likelihood(self, X, cls)**: method untuk menghitung Likelihood, yakni probabilitas data muncul untuk kelas tertentu. Untuk setiap sampel, likelihood dihitung dengan mengalikan probabilitas Gaussian dari setiap fitur dalam sampel tersebut. Perhitungan ini dilakukan dengan rumus sebagai berikut.

$$P(x_j|C_k) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

➜ **predict(self, X)**: merupakan method untuk tahap prediksi. Menggunakan istilah posterior, prior. Posterior untuk setiap kelas dihitung sebagai hasil perkalian antara prior dan likelihood. Posterior terbesar menentukan kelas prediksi untuk setiap sampel. Berikut adalah rumus yang digunakan untuk menghitung posterior.

$$P(C_k|X) = P(C_k) \times \prod_{j=1}^{n} P(x_j|C_k)$$

Posterior dihitung untuk semua kelas, lalu model memprediksi kelas dengan nilai posterior tertinggi.

1.3. Implementasi Algoritma ID3

ID3 adalah algoritma yang digunakan untuk membangun pohon keputusan. ID3 bekerja dengan memilih fitur yang memberikan informasi terbesar (Information Gain) untuk membagi dataset pada setiap node. Proses ini berulang hingga kondisi berhenti dipenuhi. Berikut adalah **<u>langkah utama</u>** dari algoritma ID3:

1. Hitung Entropy dari dataset untuk mengukur ketidakpastian.
2. Untuk setiap fitur, hitung Information Gain.

3. Pilih fitur dengan Information Gain terbesar untuk membuat split.
4. Rekursif membagi dataset hingga mencapai salah satu kondisi berhenti, yaitu antara semua data dalam node memiliki label yang sama (pure node), tidak ada lagi fitur yang tersisa untuk dipilih, atau kedalaman maksimum pohon telah tercapai.

Kelas ID3 yang diimplementasikan bertanggung jawab untuk melatih model dengan membangun pohon keputusan sampai melakukan prediksi berdasarkan pohon yang sudah dibangun. Kelas ini menerima tiga parameter, yaitu:

1. max_depth: Kedalaman maksimum pohon (untuk mencegah overfitting).
2. min_samples_split: Jumlah minimum sampel untuk melakukan split.
3. min_impurity_decrease: Pengurangan minimum impurity untuk melakukan split.

Berikut adalah penjelasan untuk fungsi-fungsi utama pada kelas ID3.

➜ **entropy(y)**: merupakan method untuk menghitung entropy dari dataset y, dimana entropy adalah ukuran ketidakpastian atau impurity dalam data.

$$\text{Entropy} = -\sum_{i=1}^{n} P_i \log_2(P_i)$$

Semakin rendah entropy, semakin “pure” data tersebut.

➜ **information_gain(self, X_column, y)**: merupakan method untuk menghitung Information Gain untuk satu fitur, dimana rumus Information Gain adalah entropy total - entropy rata-rata dari subset. Information Gain digunakan untuk memilih fitur terbaik untuk membagi data pada setiap node.

➜ **best_split(self, X, y, feature_names)**: merupakan method untuk menentukan fitur terbaik untuk membuat split. Untuk fitur numerik, menghitung split pada setiap threshold, sementara untuk fitur kategorikal, langsung membagi berdasarkan nilai unik.

➜ **build_tree(self, X, y, feature_names, depth=0)**: merupakan method untuk secara rekursif membangun pohon keputusan, jika label pure (semua sama), return label. Jika tidak ada fitur lagi, return label mayoritas. Method ini juga

akan memilih fitur terbaik (menggunakan best_split). Pada proses ini, akan di cek apakah iterasi sudah mencapai kondisi berhenti, yaitu:

- Maksimal kedalaman tercapai.
- Sampel terlalu sedikit untuk dibagi lebih lanjut.
- Semua data memiliki label yang sama.

Untuk fitur kontinu, data akan dibagi ke kiri dan kanan (tergantung threshold), sementara untuk fitur kategorikal, akan dibuat cabang untuk setiap fitur yang unik.

➜ **fit(self, X, y, feature_names)**: merupakan method untuk memulai proses training dengan memanggil build_tree.

➜ **predict_instance(self, instance, tree)**: merupakan method untuk memprediksi satu instance data berdasarkan nilai fitur instance.

➜ **predict(self, X, feature_names)**: merupakan method untuk memprediksi banyak instance.

Bab II. Tahap Cleaning dan Preprocessing

> *Penjelasan tahap cleaning dan preprocessing yang dilakukan beserta dengan alasannya.*

2.1. Tahap Cleaning

2.1.1. Missing Values

Untuk menangani missing value, pertama-tama dataset dibagi terlebih dahulu berdasarkan atributnya, yaitu numerik dan kategorikal.

1. **Numeric Column**

   Atribut dengan nilai skew < 1, missing value-nya akan diisi dengan mean, sedangkan atribut yang nilai skew tinggi (> 1), missing value-nya diisi dengan median.

2. **Categorical Column**

   Terdapat tiga pendekatan pengisian missing value pada atribut kategori:

   - Diisi dengan modus masing-masing atribut.
   - Diisi menggunakan KNN dengan jumlah tetangga 50.
   - Missing value dijadikan sebagai kategori baru pada atribut tersebut.

   Setelah pengujian empiris, didapatkan bahwa pendekatan yang menghasilkan hasil terbaik ialah modus.

2.1.2. Outlier

Untuk menangani outlier, kami mengimplementasikan kelas khusus bernama OutlierHandler yang digunakan untuk mendeteksi dan menangani nilai-nilai outlier pada fitur numerik dalam dataset. Kelas ini mendukung dua metode utama dalam menangani outlier, yaitu Interquartile Range (IQR) dan metode clipping berdasarkan ambang tertentu.

1. **Metode Interquartile Range (IQR)**

   Metode ini digunakan untuk mendeteksi dan memotong nilai-nilai outlier berdasarkan IQR, yaitu rentang antara kuartil pertama (Q1) dan kuartil ketiga (Q3). Langkah-langkahnya adalah sebagai berikut.

i. Hitung Q1 (persentil ke-25) dan Q3 (persentil ke-75) dari setiap kolom fitur numerik.

ii. Hitung IQR sebagai:

$$\text{IQR} = Q3 - Q1$$

iii. Tentukan lower bound dan upper bound sebagai:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

iv. Nilai di luar rentang tersebut akan di-clip (dipotong) ke lower bound atau upper bound.

2. **Metode Clipping**

   Metode ini memotong nilai-nilai ekstrim berdasarkan kondisi tertentu, yaitu jika nilai maksimum suatu kolom 10 kali lebih besar dari nilai median dan lebih besar dari 10, maka nilai tersebut dipotong di persentil ke-95. Tujuannya adalah untuk mengurangi pengaruh nilai-nilai ekstrem yang terlalu jauh dari distribusi umum data.

2.1.3. Removing Duplicates

Kami mengecek adanya nilai duplikat pada dataset train dan test menggunakan method duplicated() dari Pandas, dimana:

- df_train.duplicated() akan mengembalikan True untuk baris duplikat dalam dataset train.
- df_test.duplicated() akan mengembalikan True untuk baris duplikat dalam dataset test.

Setelah pengecekan, kami menemukan bahwa tidak ada nilai duplikat baik pada dataset train maupun test. Meskipun demikian, proses pemeriksaan ini tetap dilakukan untuk memastikan kualitas dan integritas data sebelum tahap analisis lebih lanjut.

2.1.4. Feature Engineering

1. **Remove Useless Attribute**

   Atribut yang dianggap tidak berguna adalah atribut yang tidak memberikan kontribusi informasi bagi model, seperti ID. Untuk melakukan hal tersebut, kami membuat sebuah kelas khusus bernama ColumnDropper untuk menghapus atribut yang tidak relevan.

2. **Remove Similar Attribute**

   Atribut dengan korelasi tinggi dapat menyebabkan redundansi informasi dan mempengaruhi performa model. Kami mengimplementasikan metode ini dalam dua tahap:

   i. Identifikasi Kolom Konstan: kolom konstan adalah atribut yang memiliki **standar deviasi nol** (semua nilai sama). Kolom ini dihapus karena tidak memberikan variasi informasi.

   ii. Identifikasi Kolom dengan Korelasi Tinggi: kami menggunakan korelasi **Pearson** pada kolom numerik yang sudah diskalakan untuk mendeteksi atribut yang memiliki korelasi tinggi. Atribut dengan korelasi lebih dari 0.9 dianggap mirip dan salah satunya dihapus.

2.2. Tahap Preprocessing

2.2.1. Feature Scaling: *Robust Scaler*

Robust Scaler adalah teknik normalisasi data yang menggunakan median dan interquartile range (IQR) untuk mengubah skala fitur dalam dataset. Teknik ni berbeda dari teknik normalisasi lainnya, seperti Min-Max Scaler atau Standard Scaler, yang lebih sensitif terhadap outliers. Karena Robust Scaler menggunakan median dan IQR, teknik ini lebih tahan terhadap pengaruh nilai ekstrim (outlier), yang sering ditemukan pada dataset yang memiliki distribusi yang tidak teratur atau data yang sangat bervariasi, termasuk dataset UNSW nb15 ini.

2.2.2. Feature Encoding: *Target Encoding*

Untuk tahap ini, kami menggunakan target encoding. Target encoding adalah metode encoding untuk fitur kategorikal, di mana setiap kategori pada kolom diganti dengan rata-rata target yang sesuai dengan kategori

tersebut. Target encoding juga tidak menambah dimensi seperti One-Hot Encoding, sehingga lebih hemat memori.

2.2.3. Handling Imbalanced Dataset

Diketahui ternyata dataset ini memiliki imbalance data dimana sebagian target label memiliki distribusi lebih besar dibanding sebagian lainnya. Oleh karena itu kami mencoba menggunakan teknik handling yakni oversampling data. Sebelumnya kami sudah mencoba menggunakan teknik undersampling namun hasilnya tidak baik karena daa yang dihasilkan menjadi sangat sedikit (berkisar 1000 baris saja). Lalu dengan oversampling, digunakan SMOTE dan SMOTEENN. Hasil dari preprocessing yang menggunakan SMOTEENN ternyata lebih baik dibanding SMOTE. Hasil ini kemungkinan disebabkan oleh kombinasi teknik SMOTE (Synthetic Minority Over-sampling Technique) dan ENN (Edited Nearest Neighbors). SMOTE secara umum menghasilkan sampel sintetis untuk meningkatkan jumlah data kelas minoritas, namun dapat memperkenalkan noise atau data yang tidak representatif jika ada overlap dengan kelas mayoritas. Di sisi lain, SMOTEENN tidak hanya melakukan oversampling dengan SMOTE, tetapi juga menerapkan ENN untuk menghapus sampel yang berada di dekat batas kelas atau sampel yang salah klasifikasi, sehingga meningkatkan kualitas data minoritas yang dihasilkan. Dengan mengurangi noise dan meningkatkan keakuratan distribusi kelas minoritas, SMOTEENN sering kali menghasilkan model yang lebih baik dan lebih generalisasi daripada SMOTE saja.

Namun begitu, secara keseluruhan tetap saja hasil prediksi dengan bantuan handling imbalance dan tanpa handling, ternyata masih lebih baik yang tanpa menggunakan handling imbalance apapun. Dengan begitu untuk dataset ini kami memutuskan untuk tidak menggunakan teknik imbalance handling.

2.2.4. **Data Normalization:** *Power Transformer Yeo-Johnson*

Untuk tahap ini, kami menggunakan Power Transformer Yeo-Johnson untuk menormalisasikan data. Power Transformer dengan metode Yeo-Johnson adalah teknik transformasi data yang bertujuan untuk meningkatkan distribusi data agar mendekati distribusi normal (Gaussian). Metode ini berguna terutama ketika data memiliki skewness yang tinggi, seperti halnya data di dataset UNSW nb15 ini. Tujuan utama dari penggunaan Power Transformer Yeo-Johnson adalah:

- Mengurangi Skewness: Dengan membuat data lebih simetris.
- Menurunkan Standar Deviasi: Mengurangi penyebaran data untuk meningkatkan stabilitas analisis.
- Meningkatkan Kinerja Model: Model machine learning (seperti KNN) seringkali bekerja lebih baik dengan data yang mendekati distribusi normal.

2.2.5. **Dimensionality Reduction:** *PCA*

Untuk tahap ini, kami menggunakan Principal Component Analysis (PCA). PCA adalah teknik reduksi dimensi yang digunakan untuk mengurangi jumlah fitur dalam dataset sambil mempertahankan sebanyak mungkin informasi atau variansi dari data aslinya. Implementasi PCA dalam kode di atas menggunakan Scikit-learn untuk mereduksi dimensi dataset. Berikut adalah tahapan yang kami gunakan dalam implementasi PCA.

i. Label Encoding: bertujuan untuk mengubah fitur kategorikal menjadi nilai numerik yang akan memungkinkan algoritma PCA (yang hanya beroperasi pada data numerik) untuk digunakan.

ii. Scaling Data Menggunakan Min-Max Scaler

```
from sklearn.preprocessing import MinMaxScaler

mm_scaler = MinMaxScaler()

fmo_X_train_df[LABELS] =
mm_scaler.fit_transform(fmo_X_train_df[LABELS])
```

Ini bertujuan untuk Normalisasi data ke dalam rentang [0, 1]. Hal ini karena PCA sensitif terhadap skala fitur, sehingga MinMaxScaler akan mereskalakan setiap fitur ke dalam rentang antara 0 dan 1 untuk memastikan semua fitur memiliki skala yang seimbang.

iii. Reduksi Dimensi Menggunakan PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=36)
pca.fit(fmo_X_train_df)

var = np.cumsum(np.round(pca.explained_variance_ratio_,
decimals=4) * 100)
print("Variance from sklearn model: ", var)
```

- PCA(n_components=36): PCA menghitung hingga 36 komponen utama (jumlah maksimal).
- explained_variance_ratio_: Proporsi variansi data yang dijelaskan oleh setiap komponen utama.
- Variansi kumulatif dihitung dengan np.cumsum, menunjukkan sejauh mana informasi dari dataset asli telah dipertahankan.

Darisana, diperoleh hasil:

```
Variance from sklearn model:  [39.87 53.   63.72 69.65 74.02 77.89 81.35 83.41 85.28 87.09 88.43 89.74
 90.84 91.87 92.8  93.67 94.39 95.08 95.68 96.23 96.73 97.22 97.67 98.07
 98.44 98.78 99.09 99.37 99.57 99.76 99.87 99.94 99.96 99.97 99.97 99.97]
```

yang mempunyai arti yaitu komponen pertama menjelaskan 39.87% variansi data dan komponen ke-18, misalnya, sudah mencakup sekitar 99% informasi dari dataset asli. Dengan demikian, kita dapat mereduksi ke 18 komponen utama saja.

Bab III. Pembahasan

> *Perbandingan hasil prediksi dari algoritma yang diimplementasikan dengan hasil yang didapatkan dengan menggunakan pustaka. Jelaskan insight yang kalian dapatkan dari perbandingan tersebut. Perbandingan hasil dapat menggunakan metrics yang sesuai dengan permasalahan yang ada. Model yang telah dibuat divalidasi menggunakan metode seperti **train-test split** atau **k-fold cross-validation** untuk memastikan kinerja yang optimal.*

3.1. Perbandingan Hasil Algoritma

3.1.1. KNN

Menggunakan algoritma KNN yang kami buat dan algoritma Scikit-learn, diperoleh hasil sebagai berikut.

Test Data	
Custom KNN	Scikit-learn KNN
F1 Score: 0.51	F1 Score: 0.50

Dari tabel di atas, dapat dilihat bahwa hasil score yang dihasilkan oleh model KNN kami tidak berbeda jauh dari model KNN oleh Scikit-learn. Hal ini karena baik algoritma KNN yang kami buat sendiri maupun Scikit-learn KNN menggunakan prinsip dasar algoritma KNN yang sama, yaitu sama-sama menghitung jarak antara data test dan data training menggunakan jarak Euclidean.

3.1.2. Gaussian Naive-Bayes

Menggunakan algoritma Gaussian Naive-Bayes yang kami buat dan metode validasi **k-fold cross-validation** dengan jumlah fold sebanyak 5, diperoleh hasil sebagai berikut.

Training Data	
Fold	F1 Macro Score
1	0.5403
2	0.5385

3	0.5406
4	0.5404
5	0.5356
Average F1 Macro Score: 0.5391 ± 0.0019	
Test Data	
F1 Macro Score: 0.5360	

Sementara itu, apabila menggunakan algoritma Gaussian Naive-Bayes yang diimpor dari library Scikit-learn, diperoleh hasil sebagai berikut.

Training Data	
Fold	F1 Macro Score
1	0.5403
2	0.5385
3	0.5406
4	0.5404
5	0.5356
Average F1 Macro Score: 0.5391 ± 0.0019	
Test Data	
F1 Macro Score: 0.5360	

Dari hasil yang diperoleh, diperoleh **insight** sebagai berikut.

1. **Kinerja pada Training Data**
   - Gaussian Naive-Bayes Implementasi Sendiri
     - Rata-rata F1 Macro Score: 0.5391 ± 0.0019
     - Rentang skor pada 5 fold: 0.5356 - 0.5406
   - Gaussian Naive-Bayes Scikit-learn
     - Rata-rata F1 Macro Score: 0.5391 ± 0.0019
     - Rentang skor pada 5 fold: 0.5356 - 0.5406

   Hasil pada training data menunjukkan bahwa algoritma Gaussian Naive-Bayes dari Scikit-learn dan implementasi sendiri memberikan

hasil yang identik (sama). Hal ini wajar terjadi karena Gaussian Naive-Bayes memiliki perhitungan yang sederhana, dengan asumsi bahwa setiap fitur mengikuti distribusi Gaussian (normal), sehingga perbedaan implementasi menjadi minimal.

2. **Kinerja pada Test Data**
   - Gaussian Naive-Bayes Implementasi Sendiri
     - F1 Macro Score: 0.5360
   - Gaussian Naive-Bayes Scikit-learn
     - F1 Macro Score: 0.5360

   Pada data test, performa kedua model juga memberikan hasil yang sama. Hal ini menunjukkan bahwa kedua implementasi memiliki kemampuan yang setara dalam melakukan generalisasi terhadap data uji.

3.1.3. ID3

Menggunakan algoritma ID3 yang kami buat dan metode validasi **k-fold cross-validation** dengan jumlah fold sebanyak 5, diperoleh hasil sebagai berikut.

Training Data	
Fold	F1 Macro Score
1	0.4575
2	0.4539
3	0.4547
4	0.4524
5	0.4616
Average F1 Macro Score: 0.4560 ± 0.0033	
Test Data	
F1 Macro Score: 0.4643	

Sementara itu, apabila menggunakan algoritma ID3 yang diimpor dari library Scikit-learn (DecisionTreeClassifier), diperoleh hasil sebagai berikut.

Training Data	
Fold	F1 Macro Score
1	0.4706
2	0.4547
3	0.4630
4	0.4569
5	0.4653
Average F1 Macro Score: 0.4621 ± 0.0057	
Test Data	
F1 Macro Score: 0.4586	

Perbandingan ini dilakukan dengan metriks yang sama, yaitu max_depth = 20 dan min_samples_split = 20. Pemilihan metriks ini dilakukan berdasarkan hasil eksperimen kami, dimana ketika nilai max_depth dibuat lebih dari 20, performa model yang diukur menggunakan F1 score mengalami penurunan akibat overfitting. Sebaliknya, ketika max_depth dibuat kurang dari 20, performa model juga menurun karena pohon menjadi terlalu dangkal dan tidak mampu menangkap pola yang lebih kompleks dalam data (underfitting). Hal serupa terjadi pada min_samples_split. Ketika nilai ini dinaikkan lebih dari 20, performa model menurun karena pohon menjadi kurang fleksibel dalam membagi node, sehingga kehilangan kemampuan untuk mempelajari pola secara optimal. Namun, ketika nilai min_samples_split dikurangi dari 20, pohon terlalu sering membagi node meskipun jumlah sampel kecil, yang akhirnya meningkatkan risiko overfitting. Pada model ID3 yang diimpor dari library, yaitu DecisionTreeClassifier, digunakan

criterion='entropy' yang juga sama dengan perhitungan information gain pada model ID3 yang kami buat, yaitu menggunakan entropy.

Dari hasil yang diperoleh, diperoleh **insight** sebagai berikut.

3. **Kinerja pada Training Data**
   - ID3 Implementasi Sendiri
     - Rata-rata F1 Macro Score: 0.4560 ± 0.0033
     - Rentang skor pada 5 fold: 0.4524 - 0.4616
   - ID3 Scikit-learn
     - Rata-rata F1 Macro Score: 0.4621 ± 0.0057
     - Rentang skor pada 5 fold: 0.4547 - 0.4706

   Algoritma ID3 dari scikit-learn memiliki rata-rata F1 Macro Score yang lebih tinggi pada training data dibandingkan dengan implementasi ID3 sendiri. Selain itu, performa algoritma scikit-learn terlihat lebih konsisten (rentang nilai lebih kecil) dibandingkan implementasi sendiri. Hal ini menunjukkan bahwa implementasi dari scikit-learn lebih optimal dalam mempelajari pola pada data training.
4. **Kinerja pada Test Data**
   - ID3 Implementasi Sendiri
     - F1 Macro Score: 0.4365
   - ID3 Scikit-learn
     - F1 Macro Score: 0.4134

   Pada test data, implementasi ID3 yang dibuat sendiri justru memiliki performa yang lebih baik dibandingkan algoritma ID3 dari scikit-learn. Skor F1 Macro pada test data menunjukkan bahwa implementasi sendiri memiliki generalisasi yang sedikit lebih baik daripada scikit-learn, meskipun pada training data performanya lebih rendah. Perbedaan ini bisa disebabkan oleh strategi regularisasi yang digunakan pada algoritma scikit-learn atau cara pemrosesan data yang berbeda.

Perbedaan performa antara implementasi ID3 sendiri dan algoritma ID3 dari Scikit-learn dapat disebabkan oleh beberapa faktor, salah satunya adalah banyaknya parameter lain yang dimiliki oleh algoritma pohon keputusan di Scikit-learn. Dimana DecisionTreeClassifier sebenarnya tidak murni hanya menggunakan ID3 (parameter entropy) saja, tetapi seringkali memiliki mekanisme tambahan seperti pilihan criterion='gini', min_samples_leaf, max_features, class_weight, splitter, dan lain-lain yang **tidak digunakan** dalam percobaan perbandingan ini.

Bab IV. Penutup

4.1. Github Repository

Repository untuk tugas besar ini dapat diakses pada tautan berikut: https://github.com/b33rk/Tubes2-AI.

4.2. Kesimpulan

Dalam tugas ini, telah diimplementasikan tiga algoritma pembelajaran mesin, yaitu K-Nearest Neighbors (KNN), Gaussian Naive-Bayes, dan ID3, serta dibandingkan hasilnya dengan implementasi dari library Scikit-learn.

Selain itu, didapat bahwa tahap data cleaning dan preprocessing memainkan peran penting dalam meningkatkan performa model. Metode seperti penanganan missing values, scaling, normalisasi, dan reduksi dimensi terbukti membantu model bekerja lebih optimal. Perbandingan performa algoritma ini menegaskan pentingnya pemilihan parameter yang tepat, validasi menggunakan metode cross-validation, dan strategi preprocessing yang baik untuk mencapai hasil prediksi yang akurat dan mampu melakukan generalisasi dengan baik.

4.3. Saran

- Bila mempunyai cukup waktu, lakukan hyperparameter tuning yang lebih sistematis menggunakan metode seperti Grid Search atau Random Search untuk mendapatkan kombinasi parameter terbaik pada algoritma.
- Peningkatan model ID3
  - Tambahkan fitur **pruning** pada implementasi ID3 untuk mengurangi overfitting dan meningkatkan performa model pada data test.
- Lakukan eksplorasi dan seleksi fitur yang lebih mendalam menggunakan metode seperti Feature Importance atau Recursive Feature Elimination (RFE) untuk memastikan fitur yang digunakan benar-benar relevan terhadap target.

Pembagian Tugas

Kegiatan	Nama (NIM)
Implementasi Model KNN	Abdullah Mubarak (13522101)
Implementasi Model Gaussian Naive-Bayes	Muhammad Naufal Aulia (13522074)
Implementasi Model ID3	Denise Felicia Tiowanni (13522013)
Data Cleaning and Preprocessing, Pipelines, Modeling	Denise Felicia Tiowanni (13522013) Muhammad Naufal Aulia (13522074) Abdullah Mubarak (13522101)
Laporan	Denise Felicia Tiowanni (13522013) Muhammad Naufal Aulia (13522074) Abdullah Mubarak (13522101)

Referensi

UNSW Research. *The UNSW-NB15 Dataset*. Diakses dari https://research.unsw.edu.au/projects/unsw-nb15-dataset

IEEE Xplore. *UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)*. Publikasi Konferensi IEEE. Diakses dari https://ieeexplore.ieee.org/abstract/document/7348942

GeeksforGeeks. *K-Nearest Neighbor (KNN) Algorithm*. Diakses dari https://www.geeksforgeeks.org/k-nearest-neighbours/

IBM. *What Are Naïve Bayes Classifiers?*. Diakses dari https://www.ibm.com/topics/naive-bayes

Towards Data Science. *Decision Trees: ID3 Algorithm Explained*. Diakses dari https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1