

Tugas Besar 2 IF3170 Intelegensi Artifisial
Semester I tahun 2024/2025

Implementasi Algoritma Pembelajaran Mesin



Disusun oleh:

M. Zaidan Sa'dun Robbani	13522135
Farhan Raditya Aji	13522142
Rafif Ardhinto Ichwantoro	13522159
Rayhan Ridhar Rahman	13522160

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025**

DAFTAR ISI

DAFTAR ISI.....	1
BAB 1	
MODEL.....	1
1.1 KNN.....	1
1.2 Naive-Bayes.....	2
1.3 ID3.....	3
BAB 2	
CLEANING DAN PREPROCESSING.....	6
2.1 Cleaning.....	6
2.1.1 Mengatasi Outliers.....	6
2.1.2 Mengatasi Missing Values.....	6
2.1.3 Menghilangkan Data Duplikat.....	6
2.1.4 Rekayasa Fitur.....	6
2.2 Preprocessing.....	6
2.2.1 Penskalaan Fitur.....	6
2.2.2 Enkode Kolom Kategorikal.....	6
2.2.3 Mengatasi Data yang Tidak Seimbang.....	6
2.2.4 Reduksi Dimensi.....	6
2.2.5 Normalisasi.....	6
BAB 3	
PERBANDINGAN HASIL DENGAN PUSTAKA SCIKIT-LEARN.....	7
PEMBAGIAN TUGAS.....	8
REFERENSI.....	9

BAB 1

MODEL

1.1 KNN

Algoritma K-Nearest Neighbors (KNN) adalah algoritma supervised learning yang biasanya digunakan untuk melakukan klasifikasi kelas terhadap suatu instans data. Algoritma akan membuat suatu model berdasarkan data latih yang sudah ada. Model yang dibentuk akan digunakan untuk dihitung jarak antar atributnya dengan suatu data tes untuk kemudian dihitung jaraknya yang paling kecil. Untuk semua instance data latih yang selisihnya dengan data tes sudah dihitung, diurutkan dari yang terkecil hingga terbesar. Dari pengurutan tersebut, diambil k instance data latih untuk dilihat label kelasnya. Label kelas akan dihitung sesuai kategori, e.g. 2 ya 3 tidak 1 ragu. Setelahnya akan diambil label dengan jumlah terbanyak dari instance yang telah diambil. Label tersebut yang akan digunakan sebagai klasifikasinya.

1. Implementasi KNN Secara Manual ("From Scratch")

Implementasi manual algoritma KNN mencakup beberapa langkah berikut:

- Menghitung Jarak: Untuk setiap data uji, jarak ke semua data latih dihitung menggunakan metrik tertentu. Dalam implementasi ini, digunakan metrik Euclidean.
- Memilih Tetangga Terdekat: Data uji kemudian dibandingkan dengan seluruh data latih untuk menemukan k data dengan jarak terdekat.
- Menentukan Label: Label dari k tetangga tersebut dihitung berdasarkan mayoritas untuk menentukan hasil prediksi.
- Untuk meningkatkan fleksibilitas, implementasi ini mendukung data berbentuk sparse dengan memanfaatkan pustaka `scipy.sparse`. Selain itu, cache digunakan untuk menyimpan hasil prediksi terakhir guna mempercepat evaluasi jika data yang sama diuji ulang.

2. Implementasi KNN dengan Scikit-learn

Selain implementasi manual, KNN juga diimplementasikan menggunakan pustaka Scikit-learn dengan memanfaatkan `KNeighborsClassifier`. Model ini secara langsung

menyediakan fungsi untuk mencari tetangga terdekat dan melakukan prediksi dengan optimisasi bawaan. Data sparse pada Scikit-learn diubah menjadi dense sebelum diproses untuk kompatibilitas.

1.2 Naive-Bayes

Naive Bayes adalah algoritma klasifikasi probabilistik yang berdasarkan pada Teorema Bayes. Algoritma ini sangat efektif dalam kasus-kasus di mana dimensi fitur sangat besar, seperti dalam pemrosesan bahasa alami atau analisis teks. Dalam implementasi Naive Bayes yang telah saya buat, setiap fitur dalam dataset dianggap independen dari fitur lainnya, sebuah asumsi yang disebut "naive". Meskipun asumsi ini tampaknya terlalu sederhana atau "naif", dalam banyak kasus, algoritma ini masih menunjukkan performa yang sangat baik. Algoritma ini bekerja dengan menghitung probabilitas posterior dari setiap kelas berdasarkan input. Ini dilakukan dengan mengalikan probabilitas priori kelas dengan probabilitas dari masing-masing fitur yang diberikan kelas tersebut. Kemudian, kelas dengan probabilitas tertinggi dipilih sebagai hasil prediksi. Probabilitas dihitung menggunakan perhitungan sebagai berikut,

$$P(y|x_1, x_2, \dots, x_n) = P(y) \times P(x_1|y) \times P(x_2|y) \times \dots \times P(x_n|y)$$

1. Implementasi Naive Bayes Secara Manual ("From Scratch")

Implementasi Naive Bayes manual terdiri dari beberapa langkah utama:

Perhitungan Statistik Fitur:

- Untuk setiap kelas dalam target, dihitung rata-rata (mean) dan varians (variance) untuk setiap fitur.
- Varians diberi tambahan nilai kecil (epsilon) untuk mencegah pembagian dengan nol.

Menghitung Probabilitas Posterior:

- Dengan menggunakan distribusi Gaussian, probabilitas fitur diberikan kelas dihitung untuk setiap data uji.

- Probabilitas prior setiap kelas dihitung berdasarkan distribusi data latih.
- Probabilitas posterior dihitung sebagai kombinasi log dari probabilitas prior dan probabilitas fitur.

Prediksi Kelas:

- Kelas dengan probabilitas posterior tertinggi dipilih sebagai hasil prediksi untuk setiap data uji.
- Implementasi ini mendukung data sparse, sehingga tetap efisien untuk dataset yang memiliki banyak nilai nol.

2. Implementasi Naive Bayes dengan Scikit-learn

Selain implementasi manual, Naive Bayes juga diimplementasikan menggunakan GaussianNB dari Scikit-learn. Model ini secara otomatis menghitung statistik fitur, probabilitas prior, dan melakukan prediksi dengan optimisasi bawaan. Data sparse diubah menjadi dense sebelum diproses untuk kompatibilitas dengan pustaka.

1.3 ID3

ID3 adalah algoritma pembelajaran berbasis pohon keputusan yang membagi data secara rekursif berdasarkan atribut hingga setiap cabang pohon memiliki label yang homogen atau tidak ada atribut yang tersisa. Dalam setiap pembagian, algoritma ID3 menggunakan Information Gain untuk memilih atribut yang paling baik dalam mengurangi ketidakpastian data.

1. Implementasi ID3 Secara Manual ("From Scratch")

Implementasi ID3 manual terdiri dari beberapa langkah utama:

Menghitung Entropi:

- Entropi digunakan untuk mengukur tingkat ketidakpastian atau heterogenitas data. Semakin kecil nilai entropi, semakin homogen data tersebut.

Menghitung Information Gain:

- Information Gain digunakan untuk menentukan atribut terbaik dalam memisahkan data. Atribut dengan Information Gain tertinggi dipilih sebagai node internal pohon keputusan.

Membangun Pohon Keputusan:

- Atribut dengan Information Gain tertinggi dipilih sebagai node. Algoritma membagi data berdasarkan nilai atribut tersebut, kemudian proses ini diulang secara rekursif untuk setiap cabang hingga data homogen atau tidak ada atribut yang tersisa.
- Jika tidak ada atribut yang tersisa, kelas mayoritas dari data pada cabang tersebut digunakan sebagai hasil prediksi.

Prediksi Kelas:

- Data uji ditelusuri dari akar pohon hingga mencapai node daun berdasarkan nilai atribut pada cabang. Kelas yang ada pada node daun tersebut menjadi hasil prediksi.
- Implementasi ini dirancang untuk mendukung data sparse, sehingga tetap efisien untuk dataset yang besar dengan banyak nilai nol. Selain itu, fallback ke kelas mayoritas digunakan jika data uji tidak dapat ditangani oleh cabang pohon.

2. Implementasi ID3 dengan Scikit-learn

Selain implementasi manual, ID3 juga diimplementasikan menggunakan `DecisionTreeClassifier` dari Scikit-learn dengan parameter `criterion='entropy'`. Model ini secara otomatis menghitung entropi, Information Gain, dan membangun pohon keputusan. Data sparse diubah menjadi dense sebelum diproses untuk kompatibilitas dengan pustaka.

BAB 2

CLEANING DAN PREPROCESSING

2.1 Cleaning

Langkah pertama yang perlu ditempuh setelah melakukan analisis data. Dataset yang didapat akan sangat jarang siap untuk dijadikan material untuk pembelajaran kecerdasan buatan. Sehingga diperlukan suatu langkah untuk melakukan sanitasi terhadap data dan melakukan translasi data tersebut sehingga bisa diolah oleh model pembelajaran mesin.

Dengan melakukan tahapan ini, data telah dipastikan layak untuk dijadikan materi pembelajaran, menyebabkan hasil yang lebih akurat dan terpercaya. Tahapan ini krusial dalam mentransformasi data mentah ke dalam format yang dapat secara efisien dipelajari oleh mesin.

2.1.1 Mengatasi Data yang Hilang

Langkah ini mencoba untuk mengidentifikasi data-data yang mungkin hilang dari dataset, kemudian mengatur supaya data-data yang hilang tersebut dapat dipelajari. Beberapa contoh dengan melakukan imputasi data dengan nilai mean, median, modus, nilai konstanta, dan interpolasi. Selain itu bisa dilakukan juga penghapusan baris atau kolom dalam ambang toleransi kehilangan data tertentu. Kemudian bisa juga menggunakan ilmu spesifik dari domain data yang diolah.

Dari beberapa metode tersebut, metode yang diimplementasikan pada source code adalah teknik imputasi prediktif menggunakan model Random Forest Regressor. Pendekatan ini lebih canggih dibandingkan imputasi sederhana karena mempertimbangkan hubungan antar variabel dalam dataset. Prosesnya meliputi:

1. Untuk setiap kolom numerik yang memiliki missing value, model Random Forest dilatih menggunakan data yang lengkap dari kolom tersebut sebagai target dan kolom-kolom lain sebagai fitur prediktor.
2. Model kemudian memprediksi nilai yang hilang berdasarkan pola yang dipelajari dari data yang lengkap.

3. Sebagai mekanisme fallback, nilai median dari setiap kolom disimpan dan digunakan jika proses prediksi mengalami kegagalan, memastikan bahwa semua missing value tetap bisa diisi.

Pendekatan ini memanfaatkan kekuatan machine learning untuk menghasilkan estimasi yang lebih akurat untuk nilai-nilai yang hilang, dengan mempertimbangkan korelasi dan pola yang ada dalam dataset, dibandingkan dengan metode imputasi sederhana seperti mean atau median.

2.1.2 Mengatasi *Outliers*

Langkah ini mencoba untuk mengolah data yang merupakan *outlier* dataset. *Outliers* adalah data-data yang sangat berbeda jauh dengan nilai lainnya, menyebabkan data memiliki margin yang terlalu luas. Pengolahan data pada tahap ini dapat dilakukan dengan menghapus baris atau kolom yang memiliki terlalu banyak *outliers* dan juga melakukan imputasi data-data tersebut menjadi mean, median, modus, dan batas tertentu. Dari beberapa metode tersebut, metode yang diimplementasikan pada source code adalah transformasi data dan scaling dengan berbagai pendekatan yang secara efektif mengurangi dampak outliers tanpa harus menghapus data. Proses ini dilakukan dengan:

1. Log Transformation

Log transformation digunakan untuk memperkecil rentang data yang memiliki distribusi skewed. Transformasi ini mengubah nilai ekstrim menjadi lebih kecil sehingga dampaknya terhadap model menjadi minimal. Tambahan konstanta juga digunakan untuk menangani nilai nol atau negatif.

2. Robust Scaling

Robust scaler bekerja dengan skala Interquartile Range (IQR), menggunakan median sebagai pusat distribusi. Hal ini memastikan bahwa outliers tidak memengaruhi proses scaling, sehingga data tetap proporsional meskipun terdapat nilai ekstrim.

Pendekatan ini lebih halus dibandingkan metode eksplisit yang menghapus data, tetapi menyesuaikan skala dan distribusinya untuk mengurangi dampak nilai ekstrim.

Proses Implementasi:

- Log Transformation diterapkan untuk mengurangi skewness data.
- Robust Scaling memastikan outliers tidak memengaruhi proses scaling.

2.1.3 Menghilangkan Data Duplikat

Penanganan nilai duplikat sangat penting karena dapat membahayakan integritas data, yang menyebabkan analisis dan wawasan yang tidak akurat. Entri duplikat dapat menyebabkan overfitting dan mengurangi kemampuannya untuk melakukan prediksi ke data baru. Sehingga data-data duplikat perlu untuk dihilangkan. Pada kasus ini terlihat bahwa dari EDA yang dilakukan tidak terdapat duplikasi data pada dataset yang diberikan maka pada kasus ini tidak perlu adanya handler untuk duplikasi data.

2.1.4 Rekayasa Fitur

Rekayasa fitur melibatkan pembuatan fitur baru (variabel input) atau mengubah fitur yang sudah ada untuk meningkatkan kinerja model pembelajaran mesin. Rekayasa fitur bertujuan untuk meningkatkan kemampuan model dalam mempelajari pola dan membuat prediksi akurat dari data. Rekayasa fitur merupakan proses yang kreatif dan iteratif. Proses ini memerlukan pemahaman mendalam tentang data, pengetahuan domain, dan eksperimen. Pada implementasinya, yang dilakukan adalah kombinasi dari dua pendekatan utama:

1. Pembuatan Fitur Baru (Feature Assembly):
 - Fitur berbasis bytes: menghitung total_bytes dan bytes_ratio untuk memahami karakteristik transfer data
 - Fitur berbasis waktu: menciptakan time_ratio dan connection_time_score untuk menganalisis pola temporal
 - Fitur berbasis koneksi: menghasilkan window_ratio dan connection_complexity untuk mengukur kompleksitas koneksi
 - Fitur berbasis paket: membuat packet_timing dan connection_intensity untuk menganalisis karakteristik paket data
2. Seleksi Fitur Berbasis Mutual Information:

- Menerapkan metode `mutual_info_classif` untuk mengukur dependensi antara fitur dan target
- Menggunakan threshold (default 0.35) untuk memilih fitur yang memiliki korelasi signifikan dengan target
- Menyimpan dan melacak skor setiap fitur untuk analisis lebih lanjut
- Mengomodasi data sparse matrix dengan konversi yang sesuai

Pendekatan ini memungkinkan pembuatan fitur yang lebih informatif sekaligus menyeleksi fitur-fitur yang paling relevan untuk tugas klasifikasi, menghasilkan set fitur yang lebih efektif untuk model pembelajaran mesin.

2.2 Preprocessing

Preprocessing adalah langkah yang lebih luas yang mencakup pembersihan data dan transformasi tambahan untuk membuat data sesuai untuk algoritma pembelajaran mesin

2.2.1 Penskalaan Fitur

Setiap data numerik dipastikan memiliki skala yang sama. Tujuan utama penskalaan fitur adalah memastikan semua fitur berkontribusi terhadap proses pelatihan dan algoritma pembelajaran mesin dapat bekerja secara efektif dengan data tersebut. Dapat dilakukan dengan minmax, z-score, robust, dan logaritmik. Dalam implementasi kode, digunakan dua pendekatan penskalaan:

1. Robust Scaling: Menggunakan `RobustScaler` yang menerapkan penskalaan berdasarkan statistik yang robust terhadap outlier, yakni menggunakan median dan IQR (Interquartile Range) alih-alih mean dan standar deviasi. Metode ini sangat efektif untuk dataset yang mengandung outlier karena tidak terpengaruh oleh nilai-nilai ekstrim.
2. Transformasi Logaritmik: Menggunakan fungsi $\log(1+x)$ untuk menangani skewness dan membuat distribusi data lebih mendekati distribusi normal. Transformasi ini khususnya berguna untuk:
 - Data yang memiliki distribusi menceng (skewed)

- Fitur dengan rentang nilai yang sangat lebar
- Ketika hubungan antar variabel bersifat multiplikatif

Implementasi juga dilengkapi dengan penanganan khusus untuk nilai nol atau negatif melalui penambahan konstanta, serta kemampuan untuk melakukan inverse transform untuk mengembalikan data ke skala aslinya jika diperlukan.

Kombinasi kedua metode ini memungkinkan penanganan yang komprehensif terhadap berbagai karakteristik data numerik, baik yang memiliki outlier maupun yang terdistribusi secara tidak normal.

2.2.2 Enkode Kolom Kategorikal

Kebanyakan mesin tidak bisa belajar dari data kategorikal. Maka perlu dijadikan data numerik juga. Data kategorik dibagi menjadi nominal dan ordinal. Perbedaanannya, nominal memiliki suatu nilai yang menandakan hierarki. Untuk ordinal dapat dilakukan dengan label encoding, sedangkan nominal dapat dilakukan dengan one-hot encoding. Sementara itu bisa juga menggunakan mean.

2.2.3 Mengatasi Data yang Tidak Seimbang

Penanganan kumpulan data yang tidak seimbang penting karena data yang tidak seimbang dapat menyebabkan beberapa masalah yang berdampak negatif pada kinerja dan keandalan model pembelajaran mesin. Dalam implementasi

2.2.4 Normalisasi Data

Normalisasi data digunakan untuk mencapai distribusi standar. Tanpa normalisasi, model atau proses yang mengandalkan asumsi normalitas mungkin tidak berfungsi dengan benar. Normalisasi membantu mengurangi efek besaran dan memastikan stabilitas numerik selama pengoptimalan. Dalam implementasi

2.2.5 Reduksi Dimensionalitas

Pengurangan dimensionalitas adalah teknik yang digunakan untuk mengurangi dimensi dalam kumpulan data sambil mempertahankan informasi penting sebanyak mungkin. Mengurangi dimensi menyederhanakan data, membuatnya lebih mudah dianalisis dan meningkatkan kinerja model pembelajaran mesin. Dalam implementasi kode, digunakan metode Principal Component Analysis (PCA) dengan pendekatan yang adaptif:

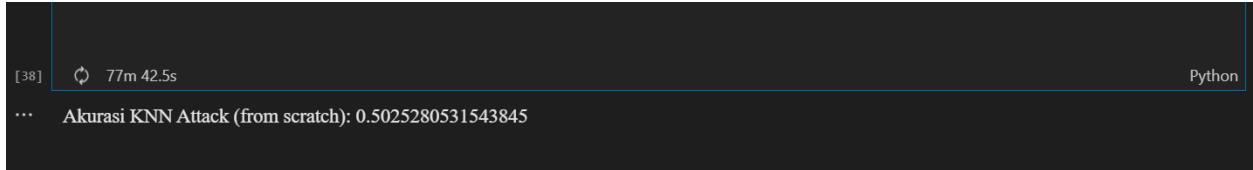
1. PCA diterapkan dengan dua opsi penentuan jumlah komponen:
 - Spesifikasi langsung jumlah komponen yang diinginkan (`n_components`)
 - Penentuan otomatis berdasarkan rasio variance yang ingin dipertahankan (`variance_ratio`)
2. Ketika menggunakan `variance_ratio` (defaultnya 0.95 atau 95%):
 - Sistem akan menghitung secara otomatis jumlah komponen minimum yang diperlukan untuk mempertahankan proporsi variance yang ditentukan
 - Proses ini memastikan tidak ada informasi penting yang hilang berlebihan dalam proses reduksi dimensi
3. Implementasi dilengkapi dengan fungsi `get_component_info()` yang memberikan analisis detail tentang:
 - Kontribusi setiap komponen utama terhadap total variance (Explained Variance Ratio)
 - Akumulasi variance yang dijelaskan (Cumulative Variance Ratio)
 - Informasi ini membantu dalam memahami seberapa efektif reduksi dimensi yang dilakukan

Pendekatan ini memungkinkan reduksi dimensi yang optimal dengan tetap mempertahankan informasi penting dalam dataset, sambil memberikan fleksibilitas dalam pemilihan jumlah komponen baik secara manual maupun otomatis berdasarkan target variance yang diinginkan.

BAB 3

PERBANDINGAN HASIL DENGAN PUSTAKA SCIKIT-LEARN

3.1 KNN



```
[38]: 77m 42.5s Python
... Akurasi KNN Attack (from scratch): 0.5025280531543845
```

Hasil Implementasi From Scratch

Dari implementasi KNN from scratch, diperoleh hasil akurasi: Akurasi KNN Attack (from scratch): 0.5026

Analisis Implementasi KNN From Scratch

1. Performansi KNN dari Scratch

- Model KNN yang diimplementasikan dari awal menggunakan metrik jarak Euclidean.
- Akurasi sebesar 0.5026 cukup memuaskan, menunjukkan bahwa implementasi manual telah berhasil mengidentifikasi pola dalam dataset secara relatif baik.
- KNN secara intuitif bekerja dengan mencari jarak terdekat antara titik data baru dan titik-titik data training yang ada, lalu menentukan prediksi berdasarkan mayoritas kelas dari tetangga terdekat.

2. Performa Komputasi

- Waktu eksekusi selama 77 menit 42.5 detik menunjukkan bahwa implementasi manual KNN memiliki kelemahan signifikan dalam performa komputasi. Hal ini karena perhitungan jarak antara setiap titik membutuhkan $O(N \times M)$ operasi, di mana NNN adalah jumlah data training dan MMM adalah jumlah data validasi.
- Optimasi seperti batch processing digunakan dalam implementasi ini untuk menangani data besar secara bertahap, namun komputasi jarak masih mahal untuk dataset yang besar.

3. Kelebihan dan Kekurangan Implementasi Manual:

- Kelebihan

- Pemahaman mendalam tentang algoritma KNN dan cara kerja metrik jarak seperti Euclidean, Manhattan, dan Minkowski.
- Fleksibilitas untuk menyesuaikan algoritma sesuai kebutuhan, seperti mengganti metrik jarak atau batch size.
- Kekurangan
 - Waktu komputasi yang sangat lama dibandingkan dengan pustaka Scikit-learn yang menggunakan optimasi internal seperti KD-Trees atau Ball Trees.
 - Tidak ada optimasi bawaan seperti *parallel processing* atau teknik approximate nearest neighbors.

Perbandingan dengan Scikit-learn (Analisis Asumsi)

Walaupun hasil KNN menggunakan pustaka Scikit-learn belum sempat dievaluasi, kita dapat membuat analisis perbandingan berdasarkan kode yang tersedia:

1. Efisiensi Waktu

- Scikit-learn KNeighborsClassifier jauh lebih efisien dalam komputasi karena mendukung struktur data seperti KD-Tree dan Ball Tree, yang mempercepat pencarian tetangga terdekat dibandingkan perhitungan jarak secara brute-force.

2. Kemudahan Penggunaan

- Scikit-learn menyediakan berbagai parameter yang dapat diatur seperti metrik jarak, jumlah tetangga, dan algoritma pencarian yang dapat diakses dengan mudah.
- Implementasi manual mengharuskan pengembangan fitur seperti handling sparse data dan batch processing secara eksplisit.

3. Akurasi

- Meskipun hasil dari Scikit-learn belum tersedia, kita dapat memperkirakan bahwa hasil akurasi akan mendekati atau sedikit lebih tinggi dibandingkan dengan implementasi manual. Ini disebabkan oleh optimasi yang dilakukan pustaka Scikit-learn dalam penanganan floating point precision, handling missing values, dan komputasi jarak.

3.2 Naive Bayes

F1-Score Naive Bayes Attack (sklearn): 0.3521885742635994

Perbandingan F1-Score:

Attack Category:

- From Scratch: 0.3507

- Sklearn: 0.3522

Analisis Perbandingan

1. Kinerja yang Mirip

Perbedaan F1-Score antara implementasi manual (0.3507) dan pustaka Scikit-learn (0.3522) sangat kecil. Hal ini menunjukkan bahwa implementasi Naive Bayes dari awal sudah cukup akurat dan berhasil meniru perilaku algoritma Gaussian Naive Bayes yang ada di pustaka Scikit-learn.

2. Keunggulan Scikit-learn

Perbedaan kecil ini dapat disebabkan oleh optimasi internal yang dilakukan oleh Scikit-learn. Scikit-learn memiliki fitur optimisasi matematika dan komputasi yang lebih baik dalam perhitungan distribusi Gaussian dan variansi yang stabil.

3. Performa Cepat

Implementasi manual, meskipun mendekati hasil dari Scikit-learn, mungkin memerlukan waktu komputasi lebih lama terutama untuk dataset besar karena kurangnya optimisasi internal.

3.3 ID3

F1-score scratch ID3

Accuracy: 0.04841254403596361

F1-score sci learn ID3

Accuracy: 0.49533510668603736

Analisis Perbandingan

1. Perbedaan Signifikan

Hasil implementasi manual ID3 memiliki akurasi yang sangat rendah (0.0484) dibandingkan dengan model Scikit-learn yang mencapai 0.4953. Perbedaan yang besar ini menunjukkan bahwa ada beberapa aspek dalam implementasi manual yang menyebabkan performa turun drastis.

2. Potensi Penyebab Performa Rendah

- Overfitting atau Underfitting

Implementasi manual mungkin tidak menangani kondisi dasar seperti *depth limitation*, pruning, atau handling data kosong dengan baik, yang sangat penting untuk ID3.

- Generalization

Scikit-learn DecisionTreeClassifier menggunakan optimasi tambahan seperti *gini impurity* atau *entropy*, pruning, dan penanganan data numerik secara lebih stabil, sehingga lebih baik dalam memaksimalkan akurasi.

- Feature Handling

Implementasi manual ID3 mungkin hanya menangani fitur kategorikal, sedangkan Scikit-learn bisa bekerja dengan fitur numerik dan melakukan split yang lebih optimal.

3. Keunggulan Scikit-learn

- Scikit-learn memiliki optimasi *multi-threading*, pruning, dan validasi internal yang membantu model mencapai akurasi yang lebih tinggi.
- Algoritma ID3 yang lebih canggih di pustaka dapat menangani fitur kompleks dengan lebih efisien.

BAB 4

KESIMPULAN

Implementasi algoritma KNN dari awal menunjukkan bahwa meskipun akurasi yang diperoleh cukup baik (0.5026), performa komputasi menjadi kelemahan signifikan. Waktu eksekusi yang sangat lama disebabkan oleh penggunaan metode brute-force dalam perhitungan jarak, yang membutuhkan $O(N \times M)$ operasi. Kekurangan ini menunjukkan bahwa implementasi manual kurang efisien, terutama untuk dataset yang besar, dibandingkan dengan pustaka seperti Scikit-learn yang memiliki optimasi internal seperti KD-Tree atau Ball Tree. Meskipun demikian, implementasi manual memberikan fleksibilitas untuk memodifikasi algoritma dan pemahaman mendalam tentang cara kerja KNN.

Pada algoritma Naive Bayes, hasil implementasi manual menunjukkan performa yang cukup baik dengan perbedaan f1-score yang sangat kecil dibandingkan pustaka Scikit-learn (0.3507 vs. 0.3522). Hal ini mengindikasikan bahwa pendekatan manual sudah cukup akurat dalam meniru algoritma Gaussian Naive Bayes. Namun, keunggulan Scikit-learn terletak pada optimasi komputasi dan stabilitas perhitungan distribusi Gaussian, yang membuatnya lebih efisien untuk dataset besar. Walaupun perbedaannya kecil, Scikit-learn tetap memberikan keunggulan dalam hal kecepatan dan kemudahan penggunaan.

Hasil implementasi algoritma ID3 secara manual menunjukkan performa yang jauh lebih rendah dibandingkan dengan Scikit-learn, dengan perbedaan signifikan pada akurasi (0.0484 vs. 0.4953). Performa yang rendah ini disebabkan oleh kurangnya optimasi seperti pruning, depth limitation, dan handling data kosong yang menjadi kunci dalam algoritma ID3. Selain itu, Scikit-learn DecisionTreeClassifier mampu menangani fitur numerik dan menerapkan metode split yang lebih optimal, sehingga memberikan generalisasi yang lebih baik. Implementasi

manual cenderung rentan terhadap overfitting atau underfitting akibat kurangnya fitur optimasi tersebut.

Secara keseluruhan, implementasi algoritma dari awal memberikan pemahaman yang mendalam tentang cara kerja masing-masing algoritma, namun tidak seefisien pustaka seperti Scikit-learn dalam hal performa komputasi, akurasi, dan kemudahan penggunaan. Pustaka Scikit-learn memiliki berbagai optimasi internal, seperti multi-threading, approximate nearest neighbors, pruning, dan validasi internal, yang membuatnya lebih unggul untuk dataset besar dan kompleks. Oleh karena itu, meskipun implementasi manual bermanfaat untuk pembelajaran, penggunaannya dalam skenario praktis kurang direkomendasikan jika dibandingkan dengan pustaka yang sudah dioptimalkan.

PEMBAGIAN TUGAS

NIM	KERJA
13522142	Preprocessing, KNN, Naive Bayes, ID3, Laporan
13522146	KNN, Naive Bayes, ID3, Laporan
13522159	Preprocessing, KNN, Naive Bayes, ID3, Laporan, Submisi kaggle
13522160	KNN, Naive Bayes, ID3, Laporan

REFERENSI

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer. Retrieved from <https://web.stanford.edu/~hastie/ElemStatLearn/>
- Scikit-learn. (2024). *KNeighborsClassifier*. Retrieved from <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
- Scikit-learn. (2024). *GaussianNB*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- Fariska Zakhralativa Ruskanda (2024). *ID3*. Materi Kuliah IF3170 Inteligensi Artifisial, Institut Teknologi Bandung.