IMPLEMENTASI ALGORITMA KLASIFIKASI PEMBELAJARAN MESIN PADA DATA NETWORK INTRUSION DETECTION SYSTEM

Tugas Besar 2

Dikumpulkan sebagai salah satu tugas mata kuliah IF3170 Inteligensi Artifisial

Oleh:

Muhammad Fadli Alfarizi	13121140
Roby Pratama Sitepu	13121142
Muhammad Arviano Yuono	13621034
Hafizh Renanto Akhmad	13621060

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI i

BAB 1 ALGORITMA KLASIFIKASI PADA MACHINE LEARNING 1

1.1 Algoritma k-Nearest Neighbour 1

1.2 Algoritma Naive-Bayes 3

1.3 Algoritma Decision Tree 5

BAB 2 PERSIAPAN DATA 9

2.1 Cleaning 9

2.1.1 Handling Missing Data 9

2.1.2 Dealing With Outliers 10

2.1.3 Remove Duplicate 11

2.1.4 Feature Engineering 11

2.2 Preprocessing 12

2.2.1 Categorical Variable Encoding 13

2.2.2 Feature Scaling 13

BAB 3 HASIL PEMODELAN DAN ANALISIS PERBANDINGAN 16

3.1 Metode Validasi Kinerja Model 16

3.2 Algoritma k-Nearest Neighbor 17

3.2.1 Model scikit-learn 19

3.2.2 Model Scratch 21

3.3 Algoritma Naive Bayes 22

3.3.1 Model Scratch 23

3.3.2 Model scikit-learn 27

3.4 Algoritma Decision Tree 30

3.4.1 ID3 From Scratch 32

3.4.2 Model scikit-learn 33
3.5 Perbandingan..... 35
PEMBAGIAN TUGAS TIAP ANGGOTA KELOMPOK..... 37
REFERENSI 38

BAB 1
ALGORITMA KLASIFIKASI PADA MACHINE LEARNING

Algoritma klasifikasi merupakan salah satu metode dalam *machine learning,* yang termasuk ke dalam proses *supervised learning,* yang digunakan untuk mengklasifikasikan data ke dalam kategori atau kelas tertentu. Klasifikasi termasuk proses *supervised learning* karena memanfaatkan input yaitu berupa dataset yang berupa data dengan kelas dan labelnya masing-masing, proses tersebut lah yang membuat algoritma klasifikasi "belajar" dari dataset yang diberikan dan dapat mengambil kesimpulan untuk memprediksi label atau kategori yang sesuai dengan pola-pola hasil pembelajaran dalam data yang sudah ada tersebut.

Dimana secara lebih formal, proses atau *task* dari *supervised learning* dapat dinyatakan sebagai berikut:

Diberikan *training set* dengan N example pasangan input-output

$$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N),$$

Dimana pada setiap pasangan tersebut dibuat berdasarkan sebuah *unknown function* $y = f(x)$, kemudian dicari sebuah *function* h yang dapat mengaproksimasi *true function* f.

Fungsi H adalah sesuatu yang disebut juga hipotesis tentang dunia yang diamati. Yang digambarkan dari *hypothesis space* H dari fungsi yang mungkin muncul. Dimana sebagai contoh, *hypothesis space* tersebut dapat berupa fungsi polynomial pangkat 3, kumpulan fungsi Javascript, ataupun kumpulan Boolean logic. Dengan kata lain, bahwa h merupakan model dari data, yang didapat dari model kelas H, ataupun kita bisa katakan sebagai fungsi yang didapat dari kelas fungsi tersebut. Kemudian untuk keluaran (ouput) y_i sebagai *ground truth*, yang merupakan jawaban sebenarnya dari prediksi model kita seharusnya.

1.1 Algoritma k-Nearest Neighbour

Algoritma k-Nearest Neighbour (KNN) merupakan salah satu algoritma dalam machine learning yaitu pada permasalahan klasifikasi dalam *supervised learning* yang didasarkan pada "voting" pada nilai jarak yang didapat. Algoritma

ini sering juga disebut dengan algoritma machine learning *non-parametric*, dan juga "*lazy learning algorithm*", karena pada algoritma ini tidak adanya hipotesis awal atau tidak adanya asumsi yang kuat pada dataset yang diberikan, dan juga disebut "*lazy"*, karena algoritma ini tidak mempelajari pada model, tetapi hanya menyimpan/mengingat dataset training.

Proses pembentukan model algoritma KNN dan juga proses inferensinya dapat dijelaskan dalam beberapa langkah, yaitu proses input, pemilihan nilai k, perhitungan jarak, pemilihan neighbour terdekat, pemilihan dengan voting, dan proses pengeluaran output. Pada proses input diberikan data untuk training dengan data yang memiliki label. Pada proses pemilihan k ditentukan jumlah neighbour (k) yang akan dikonsiderasi dalam membuat model. Perhitungan jarak, dilakukan dengan pada setiap data poin baru (test poin) dihitung jarak semua poin dengan dibandingkan dengan dataset training, oleh sebab itu juga algoritma ini disebut algoritma "*lazy"* karena tidak adanya proses training, melainkan hanya proses penyimpanan semua dataset training pada model. Umumnya jarak dihitung menggunakan metric seperti *Euclidean distance*, *Manhattan,* atau *Minkowski distance* juga bisa digunakan. Misalkan terdapat dua *instance* $\mathbf{a} = (x_{a,1}, x_{a,2}, \dots, x_{a,n})$ dan $\mathbf{b} = (x_{b,1}, x_{b,2}, \dots, x_{b,n})$, dengan dimensi n, maka jarak Euclidean, Manhattan, dan Minkoswki dari kedua instance tersebut berturut-turut yakni:

$$d_{\mathrm{Euc}}(a,b) = \sqrt{\sum_{i=1}^{n} \left(x_{a,i} - x_{b,i}\right)^2}$$

$$d_{\mathrm{Man}}(a,b) = \sum_{i=1}^{n} \left|x_{a,i} - x_{b,i}\right|$$

$$d_{\mathrm{Min}}(a,b) = \sqrt[p]{\sum_{i=1}^{n} \left|x_{a,i} - x_{b,i}\right|^p}$$

Proses penentuan neighbour terdekat dilakukan dengan mengurutkan data pada test poin sesuai dengan nilai k nya dan ditentukan neighbour dengan nilai k terdekat. Lalu untuk mendapatkan hasil klasifikasi dilakukan proses voting, atau

pemilihan label kelas yang paling sering muncul diantara nilai K yang terdekat. Proses pengeluaran hasil (output) mengeluarkan hasil prediksi label untuk data test.

Algoritma ini sangat mudah dan memiliki kekompleksitasan yang sangat rendah jika dibandingkan dengan algoritma klasifikasi lainnya, karena pada prosesnya tidak terdapat proses training, dan juga proses inferensinya yang jauh lebih sederhana.

1.2 Algoritma Naive-Bayes

Algoritma Naive-Bayes merupakan algoritma klasifikasi dalam *supervised learning* yang didasarkan pada teorema Bayes dengan asumsi *conditional independence* antara pasangan tiap fitur diberikan nilai dari variabel kelas [1]. Teorema Bayes menyatakan bahwa untuk sebuah variabel kelas y dan fitur dependen x_1 hingga x_n, terdapat relasi:

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

dimana $P(y|x_1, \dots, x_n)$ merupakan probabilitas posterior, $P(y)$ merupakan probabilitas prior, dan $P(x_1, \dots, x_n|y)$ adalah *likelihood*. Dengan asumsi *conditional independence* "naif", didapatkan hubungan

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$$

untuk semua i, sehingga relasi tersebut tersimplifikasi menjadi:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\Pi_{i=1}^{n} P(x_i|y)}{P(x_1, \dots, x_n)}$$

Karena $P(x_1, \dots, x_n)$ konstan untuk setiap input, maka digunakan hubungan berikut dalam penyelesaian proses klasifikasi:

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Proses *training* atau pembuatan hipotesis dilakukan dengan menghitung nilai probabilitas prior $P(y)$ dan *likelihood* dari tiap fitur dependen $P(x_i|y)$ berdasarkan data latih yang diberikan. Proses inferensi atau prediksi pada data yang baru atau data uji dilakukan dengan mencari kelas dengan peluang posterior tertinggi, atau yang disebut dengan estimasi Maximum A Posteriori (MAP):

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Ada beberapa jenis algoritma Naive-Bayes yang mengasumsikan distribusi dari $P(x_i|y)$ yang berbeda-beda. Pada laporan ini, digunakan algoritma Gaussian Naive-Bayes. Algoritma Gaussian Naive-Bayes mengasumsikan distribusi dari *likelihood* tiap fitur sebagai distribusi normal:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

dimana μ_y dan σ_y^2 adalah nilai rata-rata dan varians dari fitur x_i untuk kelas y dari data latih. Proses pembuatan hipotesis dilakukan dengan menyimpan nilai μ_y dan σ_y^2 untuk setiap fitur untuk setiap kelas. Dengan asumsi ini, Gaussian Naive-Bayes sangat cocok digunakan pada dataset dengan fitur kontinu yang mengikuti distribusi normal atau mendekati normal.

Akan tetapi, algoritma Gaussian Naive-Bayes memiliki kekurangan, yakni ketika standar deviasi mendekati nol atau bahkan mencapai nol, sehingga nilai *likelihood* menjadi tidak terdefinisi. Hal ini dapat berakibat buruk dalam implementasi yang dapat mengakibatkan underflow. Untuk menangani hal tersebut, dalam implementasi laporan ini, digunakan *additive smoothing* mirip seperti yang digunakan dalam implementasi scikit-learn untuk algoritma Gaussian Naive Bayes, yakni dengan menambahkan sebuah parameter yang bernilai bagian kecil dari varians tertinggi dari fitur latih $X = [x_1 \quad \dots \quad x_n]^T$:

$$\sigma_y \leftarrow \sigma_y + \alpha \max(\operatorname{var} X)$$

dimana α disebut dengan *smoothing variable*. Selain itu, untuk menghindari *underflow* pula akibat *likelihood* yang sangat rendah jika varians sangat rendah, digunakan *log likelihood* untuk proses klasifikasi. Hal ini dapat dilakukan karena logaritma dari *likelihood* pasti akan meningkat jika *likelihood* meningkat.

$$\hat{y} = \arg\max_y \log\left[P(y) \prod_{i=1}^{n} P(x_i|y)\right] = \arg\max_y \left[\ln P(y) + \sum_{i=1}^{n} \ln P(x_i|y)\right]$$

dengan

$$\ln P(x_i|y) = -\frac{(x_i - \mu_y)^2}{2\sigma_y^2} - \frac{1}{2}\ln 2\pi\sigma_y^2$$

1.3 Algoritma Decision Tree

Algoritma *decision tree* atau pohon keputusan merupakan metode *supervised learning* yang dapat digunakan untuk proses klasifikasi dan regresi [2]. Sebuah *decision tree* merupakan representasi dari sebuah fungsi yang memetakan vektor dari nilai-nilai atribut atau fitur ke sebuah nilai output atau "decision" [3]. Pohon keputusan ini juga dapat dilihat sebagai representasi dari disjungsi aturan-aturan berbentuk konjungsi.

Proses inferensi dalam decision tree dilakukan melalui serangkaian pengujian berurutan, dimulai dari simpul akar, melewati cabang-cabang yang relevan, hingga mencapai simpul daun. Proses pelatihan atau pembentukan pohon melibatkan pemilihan fitur yang paling informatif secara iteratif sebagai pemisah, berdasarkan data latih, hingga pohon selesai terbentuk atau kriteria penghentian terpenuhi.

Algoritma *decision tree* memiliki berbagai aplikasi di dunia nyata, salah satunya di bidang medis untuk diagnosa penyakit. Keunggulan algoritma ini meliputi kemudahan interpretasi hasil serta efisiensi dalam pelatihan untuk dataset yang relatif kecil hingga menengah. Namun, algoritma ini juga memiliki beberapa kelemahan, seperti kecenderungan untuk overfitting pada dataset yang kompleks atau memiliki banyak atribut.

Ada berbagai algoritma decision tree yang telah dikembangkan, seperti ID3 (Iterative Dichotomiser 3), C4.5 (pengembangan dari ID3), dan CART (Classification and Regression Tree). Dalam laporan ini, algoritma ID3 digunakan sebagai fokus utama. Algoritma Iterative Dichotomiser 3 (ID3) dirancang oleh Ross Quinlan pada tahun 1986. Algoritma ini membangun sebuah pohon multi-cabang (multiway tree) di mana setiap node dipilih berdasarkan fitur kategorikal dengan pengaruh tertinggi terhadap variabel kelas.

Pengaruh tersebut diukur menggunakan *information gain*, yang merupakan metrik untuk menilai seberapa baik suatu fitur memisahkan data berdasarkan variabel kelas. Dalam menghitung *information gain*, digunakan satu parameter yang disebut dengan *entropi*. Entropi merepresentasikan tingkat ketidakpastian

pada sebuah variabel acak. Secarea umum, entropi dari sebuah variabel acak V dengan nilai v_k yang memiliki probabilitas $P(v_k)$ didefinisikan sebagai:

$$Entropy(V) = -\sum_{k} P(v_k) \log_2 P(v_k)$$

Dalam menghitung entropi kumpulan data latih S, entropy dapai dihitung dengan:

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

dimana c adalah banyak kelas pada data dan p_i merupakan proporsi kelas ke-i data pada S. Sebuah atribut/fitur A dengan d nilai yang berbeda membagi data latih E menjadi subset $S_1, \dots, S_d$. *information gain* dari atribut A merupakan ekspektasi pengurangan pada nilai entropi:

$$Gain(A) = Entropy(S) - \sum_{k=1}^{d} \frac{|S_k|}{|S|} Entropy(S_k)$$

Akan tetapi, algoritma ID3 memiliki beberapa kekurangan, salah satunya adalah tidak bisa menerima atribut atau fitur numerik. Dalam laporan ini, diimplementasikan adaptasi terhadap fitur numerik dari algoritma C4.5. Algoritma C4.5 menangani fitur numerik dengan tahapan berikut:

1. Urutkan data berdasarkan atribut numerik yang ditinjau.
2. Cari kumpulan pasangan *instance* yang bersebalahan dengan kelas yang berbeda dan nilai atribut numerik yang ditinjau yang berbeda.
3. Hitung nilai tengah dari atribut yang ditinjau untuk semua pasangan *instance* yang didapat.
4. Pisahkan data latih menjadi dua subset berdasarkan nilai tengah yang didapat dan hitung *information gain* berdasarkan pemisahan tersebut untuk tiap nilai tengah yang didapat.
5. Nilai *information gain* tertinggi dijadikan *information gain* untuk atribut tersebut, yang akan digunakan untuk menentukan atribut paling berpengaruh.

Dalam algoritma ID3 yang diimplementasikan, ditambahkan pula parameter kedalaman maksimum (maximum depth). Salah satu fungsinya adalah untuk mengurangi *overfitting*. Selain itu, jika dalam proses klasifikasi pada suatu *node*, ditemutkan nilai atribut kategori dari suatu instance yang ingin diprediksi tidak ada

dalam data latih, maka digunakan mayoritas kelas pada *node* tersebut. Untuk penjelasan algoritma ID3 yang diimplementasikan, dapat dilihat pada Algoritma 1 dan Algoritma 2. Fungsi Importance(*a, examples*) akan mengembalikan nilai *information gain* dari atribut kategorikal dan information gain maksimal yang bisa didapat dari atribut numerik. S adalah nilai *splitting value terbaik* yang didapat dan tidak akan didefnisikan jika atribut dengan information gain terbesar yang didapat adalah atribut kategotikal.

Algoritma 1. Proses Pelatihan ID3

```
function TrainingID3(examples, attributes, parent_examples, depth, max_depth) returns a tree

    if examples is empty then return MajorityVote(parent_examples)
    else if all examples have the same classification then return the classification
    else if attributes is empty or depth = max_depth then return MajorityVote(examples)
    else
        A, S ← argmax_{a ∈ attributes} Importance(a, examples)
        tree ← a new decision tree with root test A
        if A is categorical then
            for each value v of A do
                subset_examples ← {e : e ∈ examples and e.A = v}
                subtree ← TrainingID3(subset_examples, attributes – A, examples, depth + 1, max_depth)
                add a branch to tree with label (v) and subtree subtree, and majority MajorityVote(examples)
        else if A is numerical then
            subset_left ← {e : e ∈ examples and e.A ≤ S}
            subset_right ← {e : e ∈ examples and e.A > S}

            subtree_left ← TrainingID3(subset_left, attributes – A, examples, depth + 1, max_depth)
            subtree_right ← TrainingID3(subset_right, attributes – A, examples, depth + 1, max_depth)

            add a branch to tree with label (≤), split S, and subtree subtree_left
            add a branch to tree with label (>), split S, and subtree subtree_right

return tree
```

Algoritma 2. Proses Klasifikasi oleh ID3

```
function PredictID3(example, tree) returns a prediction result

if tree.subtree doesn't exist then
    return classification at tree

current_attribute ← tree.root
example_attribute ← value of current_attribute of example

if tree.split is not defined then
    if example_attribute not exist in all of tree.subtree.label then
        return tree.majority

    return PredictID3(example, tree.subtree where tree.subtree.label = example_attribute)
else then:
    if example_attrinute ≤ tree.split then
        return PredictID3(example, tree.subtree_left)
    else then
        return PredictID3(example, tree.subtree_right)
```

BAB 2
PERSIAPAN DATA

2.1 Cleaning

Pada proses persiapan data proses yang paling utama untuk dilakukan adalah proses pembersihan data (data cleaning), karena data mentah (raw data) dapat mengganggu dan juga merusak hasil dari prediksi yang dilakukan oleh model. Data mentah umumnya masih sangat acak dan mungkin memiliki nilai yang hilang (*missing values*), dan juga inkonsistensi. Pada proses pembuatan dan implementasi algoritma klasifikasi ini, digunakan beberapa langkah-langkah dan juga metode untuk melakukan proses data cleaning tersebut. Proses ini sangat penting karena dapat memastikan data dapat digunakan dan siap untuk dijadikan data training, kemudian dapat meningkatkan akurasi dan juga keandalan dari model klasifikasi *machine learning* yang digunakan. Langkah-langkah dan juga metode yang digunakan pada pengembangan dijelaskan di bawah.

2.1.1 Handling Missing Data

Tahapan pertama yang dilakukan pada data cleaning ialah mengatasi data yang hilang atau tidak memiliki nilai (missing data), dimana pada pengembangan ini dilakukan pemrosesan untuk missing data dengan melakukan pencarian data yang hilang atau missing, kemudian dilakukan pemrosesan pada nilai yang hilang tersebut, dapat dilakukan *drop* atau data yang hilang tersebut dihilangkan ataupun data yang hilang tersebut diisi dengan data tertentu. Metode *drop* atau menghilangkan data yang hilang tersebut umumnya dilakukan dimana data yang hilang tersebut tidak terlalu banyak atau relatif sedikit atau ketika data yang hilang tidak dianggap krusial untuk analisis atau prediksi. Dalam hal ini, penghapusan baris atau kolom yang mengandung nilai hilang dapat membantu mencegah distorsi dalam model tanpa mengurangi kualitas dataset secara signifikan. Namun, penggunaan metode ini harus lebih berhati-hati, karena penghapusan data yang terlalu banyak dapat menyebabkan hilangnya informasi yang berharga. Di sisi lain, metode imputasi (*imputation*) diterapkan ketika proporsi data yang hilang cukup

besar atau ketika informasi yang hilang dianggap penting untuk analisis. Dalam imputasi, nilai yang hilang digantikan dengan perkiraan yang dihitung berdasarkan nilai yang ada, misalnya dengan menggunakan rata-rata, median, modus, atau metode yang lebih kompleks seperti regresi atau algoritma berbasis pembelajaran mesin. Metode ini bertujuan untuk mempertahankan integritas dataset dan memastikan bahwa model tetap dapat dilatih dengan data yang lengkap. Pemilihan antara metode penghapusan (*drop*) dan imputasi sangat bergantung pada karakteristik data yang hilang, proporsi data yang hilang, serta tujuan analisis yang hendak dicapai. Pertimbangan utama adalah agar kualitas dan representativitas data tetap terjaga. Imputasi pada data tersebut dengan yaitu melakukan penggantian pada data yang hilang tersebut. Pada proses data cleaning kami, dilakukan proses imputation pada fitur numerik diisi dengan rata-rata nilai pada fitur tersebut dan pada fitur kategorikal diisi dengan modus atau data yang paling sering muncul pada fitur tersebut. Dengan pada aplikasinya kami gunakan kelas SimpleImputer, dan disesuaikan dengan strategi yang dipilih.

2.1.2 Dealing With Outliers

Outlier atau pencilan merupakan data yang secara signifikan menyimpang dari pola umum pada suatu himpunan data, baik berada jauh di atas nilai mayoritas maupun jauh di bawahnya, sehingga dapat mempengaruhi analisis statistika, pemodelan prediktif, dan inferensi yang diperoleh. Proses penentuan outlier dilakukan dengan terlebih dahulu menghitung kuartil (Q1 dan Q3) serta Inter Quartile Range (IQR) pada setiap fitur, untuk kemudian menetapkan batas bawah (Q1 - 1,5IQR) dan batas atas (Q3 + 1,5IQR) yang berfungsi sebagai ambang deteksi pencilan. Salah satu metode yang digunakan untuk menangani outlier adalah clipping, yaitu menggantikan nilai pencilan dengan nilai batas ambang terdekatnya (batas atas untuk outlier di atas ambang dan batas bawah untuk outlier di bawah ambang). Metode ini dipilih karena relatif sederhana, memastikan data tetap berada dalam rentang nilai yang wajar, serta mengurangi distorsi yang dapat ditimbulkan oleh nilai ekstrem terhadap analisis selanjutnya. Meskipun demikian, clipping bukan satu-satunya pendekatan yang dapat diterapkan; alternatif lain mencakup penghapusan outlier secara selektif, transformasi data (misalnya *log* atau *Box-Cox*),

atau penggunaan metode robust yang secara alami kurang sensitif terhadap nilai ekstrem. Dalam penelitian ini, proses penanganan outlier telah diimplementasikan menggunakan kelas BoundOutlierHandlerStrategy, yang mengotomatiskan deteksi dan koreksi nilai pencilan sesuai prosedur clipping yang telah ditetapkan..

2.1.3 Remove Duplicate

Proses selanjutnya ialah mengatasi data yang terduplikasi atau memiliki nilai yang sama pada satu fitur yang sama. Data yang terduplikat tersebut dapat memengaruhi hasil pada model, karena dapat membuat adanya overfitting dan juga model menjadi kurang general. Data tersebut juga dapat memengaruhi proses komputasional, seperti waktu yang diperlukan dan juga costnya. Sehingga pada proses data cleaning kami dilakukan pencarian pada data yang terduplikasi dan kemudian data tersebut dihapus dari dataset. Pada aplikasinya digunakan kelas yaitu DropDuplicateHandlerStrategy.

2.1.4 Feature Engineering

Feature engineering merupakan salah satu proses dan juga metode pada data cleaning, dimana dilakukan beberapa hal untuk meningkatkan efisiensi dan juga efektivitas dari model yang dikembangkan nantinya. Beberapa hal yang dilakukan pada feature engineering meliputi, feature selection, pembuatan fitur baru, diskritisasi data kontinu dan sebagainya.

2.1.4.1 Feature Selection

Pada tahap feature engineering dalam penelitian ini, digunakan metode feature selection berbasis Random Forest untuk mengidentifikasi fitur-fitur dengan pengaruh paling tinggi terhadap performa model, yang diimplementasikan melalui kelas *RandomForestSelectionHandlerStrategy*. Metode ini dipilih karena Random Forest, sebagai salah satu metode ensembel berbasis pohon keputusan, memiliki ketahanan terhadap overfitting dan mampu menangani data dengan dimensi yang tinggi serta beragam tipe variabel. Random Forest membangun sejumlah pohon keputusan pada sub-sampel data pelatihan yang dipilih secara acak (*bootstrap sampling*), kemudian menggabungkan prediksi dari seluruh pohon tersebut dengan

cara pemungutan suara mayoritas atau perata-rataan nilai, sehingga menghasilkan model yang lebih stabil dan robust. Untuk penentuan fitur penting, setiap pohon dalam Random Forest menggunakan metrik impuritas seperti *Gini Impurity* untuk mengevaluasi kontribusi setiap fitur dalam memecah node secara optimal; secara matematis, *Gini Impurity* pada sebuah node dengan kelas-kelas {1, 2,...,K} dapat diformulasikan sebagai:

$$I_{Gini} = \sum_{k=1}^{K} p_k(1 - p_k)$$

dengan p_k merupakan proporsi pengamatan yang tergolong dalam kelas k pada node tersebut. Nilai impuritas ini dihitung sebelum dan sesudah pemecahan (splitting) node oleh sebuah fitur, lalu perbedaan nilai tersebut dirata-ratakan pada seluruh pohon untuk menghasilkan ukuran kepentingan fitur (feature importance). Dengan demikian, fitur yang secara konsisten memberikan pengurangan impuritas yang besar dianggap memiliki peran lebih signifikan, dan fitur-fitur tersebut diprioritaskan. Hasil akhirnya adalah rangkaian fitur dengan nilai kepentingan tertinggi yang dipilih sebagai masukan model, sehingga dapat meningkatkan kinerja model baik dari segi akurasi, interpretabilitas, maupun efisiensi komputasi.

2.2 Preprocessing

Tahapan preprocessing merupakan tahapan lanjutan dalam proses persiapan data dalam pembuatan model klasifikasi. Proses atau metode yang dilakukan meliputi feature scaling, encoding categorical variables, handling imbalance classes, dimensionality reduction, normalization. Pada model yang kami kembangkan dilakukan metode yaitu feature scaling, dan encoding categorical variables. Encoding dilakukan menggunakan metode One-Hot Encoding, yang bekerja dengan membuat kolom binary (0 atau 1) pada setiap kategori di dalam fitur nominal. Untuk metode feature scaling yang digunakan ialah scaling dengan Z-score scaling, dengan menjadikan distribusi data memiliki nilai rata-rata (mean) di nol dan juga dengan standar deviasi sama dengan satu.

2.2.1 Categorical Variable Encoding

Salah satu teknik yang paling umum digunakan dalam mengatasi variabel kategorial dalam machine learning adalah One-Hot Encoding, di mana suatu variabel kategorial dengan n kemungkinan kategori direpresentasikan menjadi n kolom biner yang masing-masing mewakili kategori tertentu, dan setiap instance akan diwakili oleh vektor biner e yang memenuhi $e \in \{0,1\}^1$, dengan tepat satu elemen bernilai 1 dan seluruh elemen lainnya bernilai 0, sehingga proses transformasi kategori $C = \{c_1, c_2, \dots, c_n\}$ yang direpresentasikan menjadi $e_i = [0, \dots, 0, 1, 0, \dots, 0]$ untuk kategori c_i dapat dituliskan sebagai $f(c_i) = e_i$, di mana e_i adalah vektor satu-hot (one-hot) dengan dimensi n. Pendekatan ini memiliki keunggulan karena bersifat non-ordinal, sehingga model tidak akan secara implisit mempersepsikan adanya hierarki atau jarak antar kategori yang bersifat palsu seperti pada label encoding atau ordinal encoding, di mana kategori dapat saja dianggap memiliki urutan nilai (misalnya kategori 'A' < 'B' < 'C') yang tidak sesuai dengan sifat sebenarnya dari data nominal yang digunakan, dan metode-metode lain seperti embedding (misalnya word embedding) umumnya digunakan untuk data dengan dimensi sangat besar atau membutuhkan representasi yang lebih kompleks; dengan One-Hot Encoding, setiap kategori berdiri sendiri sehingga menghindari bias akibat hubungan ordinal yang tidak diinginkan, serta memudahkan proses selanjutnya seperti *feature engineering* dan pemilihan fitur penting dengan metode seperti random forest, yang memang memerlukan input numerik, dan karena alasan inilah pada program kami proses encoding dilakukan terlebih dahulu sebelum tahap feature engineering untuk memastikan model dapat memanfaatkan data dengan optimal.

2.2.2 Feature Scaling

Salah satu proses/tahapan yang dilakukan pada data preprocessing adalah melakukan feature scaling, yaitu yang digunakan pada program kami ialah Z-score feature scaling. Metode ini dikenal juga dengan standarisasi, dimana teknik ini digunakan untuk mengubah fitur menjadi dalam bentuk skala standar dengan mean/rata-rata pada 0 dan standar deviasi bernilai 1. Dengan formulanya sebagai berikut:

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

Dengan X adalah nilai original dari fitur, μ adalah rata-rata dari fitur, dan σ adalah standar deviasi dari fitur.

Metode ini sangat diperlukan terutama pada algoritma machine learning yang sensitif terhadap skala input dari fitur-fiturnya. Z-score scaling dilakukan dengan menghitung jarak (mengurangi) setiap nilai dari rata-rata setiap fiturnya, dan dibagi dengan dengan standar deviasi fiturnya. Metode ini membuat distribusi dari data fitur tersebut memiliki nilai rata-rata 0 dan standar deviasi bernilai 1, sehingga secara efektif menormalisasi data. Standarisasi memastikan bahwa fitur dengan satuan ataupun rentang (jarak) yang berbeda tidak mempengaruhi proses learning, sehingga mencegah beberapa fitur mengubah proporsional dan mengurangi performa model yang dibuat. Pada program yang kami buat digunakan kelas yaitu ZScoreHandlerStrategy untuk aplikasinya.

Selain metode standarisasi (Z-score feature scaling), pendekatan lain yang umum digunakan adalah normalisasi (normalization). Normalisasi merupakan teknik yang mengubah nilai fitur menjadi berada dalam rentang tertentu, umumnya [0,1], tanpa memodifikasi sifat keproporisonalan antar nilai-nilai fitur. Tujuan utama dari normalisasi adalah memastikan tidak adanya fitur yang memiliki dominasi skala yang berlebihan terhadap fitur lainnya. Dengan demikian, setiap fitur akan memiliki bobot pengaruh yang lebih seimbang pada proses pembelajaran model. Adapun salah satu formulasi normalisasi yang umum digunakan adalah metode min-max normalization, sebagaimana ditunjukkan dalam persamaan berikut:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Dengan X' adalah nilai fitur yang telah dinormalisasi, X adalah nilai fitur asli, $\min(X)$ adalah nilai minimum dari fitur tersebut, dan $max(X)$ adalah nilai maksimum dari fitur yang sama. Normalisasi ini akan memetakan semua nilai fitur ke dalam rentang [0,1], sehingga setiap fitur memiliki skala yang sebanding. Hal ini sangat bermanfaat terutama pada algoritma machine learning yang sensitif terhadap variasi skala input, seperti algoritma berbasis jarak (misalnya k-Nearest Neighbors) atau algoritma yang memanfaatkan gradient-based optimization

(misalnya *Neural Networks* dan *Logistic Regression*). Dengan menggunakan normalisasi, proses iterasi dalam menemukan parameter optimal menjadi lebih stabil, konvergen lebih cepat, dan dapat meningkatkan kinerja model secara keseluruhan.

BAB 3
HASIL PEMODELAN DAN ANALISIS PERBANDINGAN

3.1 Metode Validasi Kinerja Model

Pada laporan ini, performa model implementasi yang kami buat akan dibandingkan dengan model dari library scikit-learn. Sesuai dengan spesifikasi tugas ini, akan digunakan metrik penilaian untuk menilai performa model, yaitu dengan *F1-Score Macro Average*. Untuk mendapatkan nilai *F1-Score Macro Average*, pertama kita perlu menghitung nilai *F1* dari masing-masing kelas dengan,

$$F1 = \frac{2 \times precision \times recall}{precision \times recall}$$

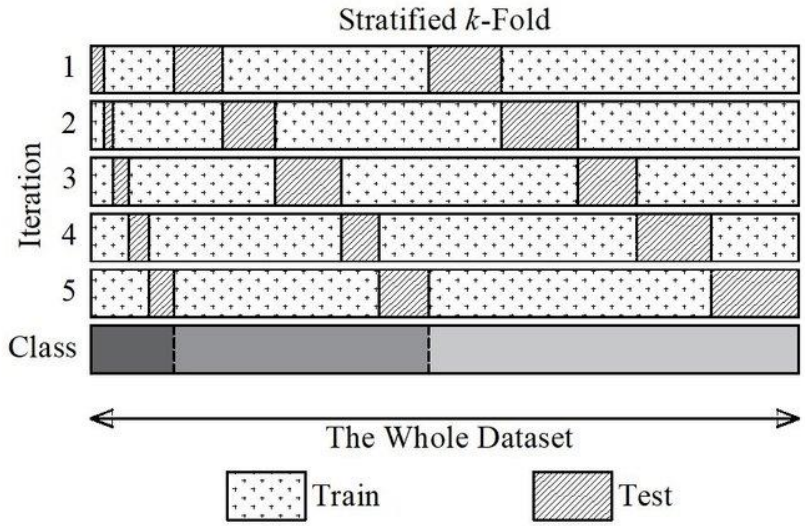
dimana,

$$precision = \frac{TP}{TP+FP} \text{ dan } recall = \frac{TP}{TP+FN}$$

Kemudian, nilai *F1-Score Macro Average* dapat kita cari dengan mencari nilai rata-rata dari nilai F1 tiap kelas dengan,

$$F1\ Macro\ Average = \frac{1}{n}\sum_{i=1}^{n} F1_i$$

Selain itu, metode validasi model yang kami gunakan pada laporan ini adalah *stratified k-fold* seperti yang diilustrasikan pada Gambar 1. Kami memutuskan menggunakan metode validasi ini karena terdapat *imbalance* pada dataset yang kami gunakan. Untuk menghindari hasil yang "kebetulan" jika hanya dilakukan pada satu distribusi *train-val splitting*, digunakan metode *cross validation.* Untuk menjaga distribusi kelas ketika membentuk *training split* pada dataset kita, digunakan metode *stratified k-fold*.



Gambar 1. Ilustrasi metode cross validation stratified k-Fold

3.2 Algoritma k-Nearest Neighbor

Untuk algoritma k-Nearest Neighbor, input yang dibutuhkan harus berupa data numerik. Oleh karena itu, tahapan proses pre-processing untuk model ini yaitu:

1. Handling Missing Data
   a. Fitur numerik: menggunakan nilai rata-rata
   b. Fitur kategorikal: menggunakan metode *most frequent*
2. Remove Duplicate
   Data duplikat akan dihilangkan
3. Dealing With Outliers
   Data numerik yang termasuk *outliers* akan diubah menjadi batas maksimum atau minimum yang didapat dari *IQR* dataset.
4. Feature Engineering
   a. Feature selection: menggunakan *importance value* dari model *RandomForestClassifier dari sklean* dengan parameter *percentage* sebesar 25% untuk fitur numerik
   b. Feature scaling: menggunakan metode Z-score untuk data numerik
5. Encoder

Untuk fitur kategorikal, dilakukan *encoding* dengan menggunakan metode *one-hot encoder* dari *sklearn*

Dengan menggunakan hasil pre-processing yang sama untuk kedua model, didapat performa masing-masing model dengan menggunakan metode Stratified k-fold sebagai berikut. Model scikit-learn yang diuji dengan tiga strategi perhitungan jarak, yaitu Euclidean, Manhattan, dan Minkowski (dengan $p = 3$), menunjukkan bahwa pendekatan dengan jarak Manhattan pada n = 5 (yaitu jumlah tetangga pada algoritma k-Nearest Neighbor) memberikan nilai F1 score paling tinggi dibandingkan dengan penggunaan jarak Euclidean dan Minkowski pada nilai n yang sama. Hal ini mengindikasikan bahwa secara konseptual, jarak Manhattan dengan jumlah tetangga yang cukup (dalam hal ini n=5) lebih sesuai untuk pola penyebaran data yang digunakan, sehingga dapat meningkatkan akurasi klasifikasi sekaligus memperbaiki keseimbangan antara presisi dan recall yang tercermin pada metrik f1 score.

Di sisi lain, dalam implementasi algoritma yang dibuat sendiri (dari scratch), pilihan untuk menggunakan jarak Manhattan dengan n=5 akan memerlukan waktu komputasional yang sangat lama ataupun lebih tinggi. Hal ini dikarenakan perhitungan jarak Manhattan untuk setiap titik data terhadap seluruh titik dalam himpunan tetangga terdekat membutuhkan proses iteratif yang meningkat secara signifikan ketika skala data menjadi besar. Oleh karena itu, pilihan jatuh pada penggunaan jarak Euclidean dengan n=5 ketika membangun algoritma secara manual, mengingat Euclidean distance umumnya lebih sederhana secara komputasional dan dapat lebih mudah dioptimasi dalam implementasi awal.

Selain itu, terdapat tren peningkatan F1 score seiring dengan meningkatnya jumlah n. Secara teoritis, peningkatan n dapat memberikan keputusan klasifikasi yang lebih stabil karena didasari oleh lebih banyak tetangga (data poin pembanding), sehingga mengurangi sensitivitas terhadap noise atau outlier. Akan tetapi, peningkatan n yang berlebihan juga dapat mengurangi spesifisitas model dengan membuatnya menjadi terlalu general dan tidak sensitif terhadap variasi lokal. Dalam konteks kasus ini, peningkatan n pada penggunaan jarak Manhattan dari scikit-learn mungkin menemukan keseimbangan optimal pada n=5, sementara

hal yang sama belum tentu tercapai dengan mudah dalam implementasi manual akibat keterbatasan waktu dan sumber daya komputasi.

Kemudian didapatkan juga hasil dengan menggunakan metode perhitungan jarak Euclidean distance, algoritma yang dibuat sendiri (dari scratch) memiliki hasil dan performa yang kurang lebih sama. Seperti dijelaskan pada paragraf-paragraf sebelumnya, trend F1 score yang didapat menggunakan algoritma yang dikembangkan sendiri dan juga menggunakan library scikit-learn, memiliki hasil yang cukup mirip, sehingga dapat diartikan model algoritma k-Nearest Neighbour yang dikembangkan sendiri cukup valid untuk menjadi model klasifikasi permasalahan network intrusion detection system.

Dengan menggunakan hasil pre-processing yang sama untuk kedua model, didapat performa masing-masing model dengan menggunakan metode *Stratified k-fold* sebagai berikut.

3.2.1 Model scikit-learn

Pada implementasi k-Nearest Neighbor, akan dilakukan perbandingan performa dari beberapa variasi konfigurasi pada model scikit-learn terlebih dahulu. Kemudian, model *scratch* akan ditrain menggunakan parameter yang menghasilkan performa terbaik dengan tambhanan konsiderasi waktu komputasional.

3.2.1.1 Strategi Euclidean Distance

A. n_neighbor = 2

```
Maximum F1 Score That can be obtained from this model is: 53.21745441700762 %

Minimum F1 Score: 49.77887092965984 %

Overall F1 Score: 51.36209576539091 %

Standard Deviation is: 1.3511587404602299 %
```

B. n_neighbor = 3

```
Maximum F1 Score That can be obtained from this model is: 55.195598870690944 %

Minimum F1 Score: 50.52512155321035 %

Overall F1 Score: 52.825086237017715 %

Standard Deviation is: 1.5439959183033078 %
```

C. *n_neighbor = 5*

```
Maximum F1 Score That can be obtained from this model is: 54.91969988048056 %

Minimum F1 Score: 52.625892465115044 %

Overall F1 Score: 53.688199437558445 %

Standard Deviation is: 0.869093299605011 %
```

3.2.1.2 Strategi Manhattan Distance

A. *n_neighbor = 2*

```
Maximum F1 Score That can be obtained from this model is: 53.241774330745905 %

Minimum F1 Score: 50.17948390847853 %

Overall F1 Score: 51.703784577404264 %

Standard Deviation is: 1.196607075648477 %
```

B. *n_neighbor = 3*

```
Maximum F1 Score That can be obtained from this model is: 55.4329207056251 %

Minimum F1 Score: 50.951964322998435 %

Overall F1 Score: 53.27654823563866 %

Standard Deviation is: 1.518918810124087 %
```

C. *n_neighbor = 5*

```
Maximum F1 Score That can be obtained from this model is: 54.993277223623124 %

Minimum F1 Score: 53.00504568269009 %

Overall F1 Score: 54.026483727094046 %

Standard Deviation is: 0.7628163007843567 %
```

3.2.1.3 Strategi Minkowski Distance

A. *n_neighbor = 2*

```
Maximum F1 Score That can be obtained from this model is: 52.96514410402244 %

Minimum F1 Score: 49.66588702424623 %

Overall F1 Score: 51.30911639076569 %

Standard Deviation is: 1.3050637578008526 %
```

B. n_neighbor = 3

```
Maximum F1 Score That can be obtained from this model is: 54.91969988048056 %

Minimum F1 Score: 52.625892465115044 %

Overall F1 Score: 53.688199437558445 %

Standard Deviation is: 0.869093299605011 %
```

C. n_neighbor = 5

```
Maximum F1 Score That can be obtained from this model is: 54.81168014544402 %

Minimum F1 Score: 52.315550509259786 %

Overall F1 Score: 53.42706432548789 %

Standard Deviation is: 0.996198749356133 %
```

3.2.2 Model Scratch

Dapat kita lihat dari hasil performa model KNN scikite-learn bahwa 2 model dengan performa tertinggi didapatkan dengan menggunakan konfigurasi pada 3.2.1.2.C dan 3.2.1.1.C. Namun dengan konsiderasi tingginya waktu komputasi dari konfigurasi 3.2.1.2.C, diputuskan bahwa model *scratch* dari KNN akan menggunakan konfigurasi yang sama dengan 3.2.1.1.C dengan hasil performa sebagai berikut,

```
Maximum F1 Score That can be obtained from this model is: 55.351535683566325 %

Minimum F1 Score: 52.44446583987059 %

Overall F1 Score: 53.567023024434825 %

Standard Deviation is: 1.0675191511177902 %
```

3.3 Algoritma Naive Bayes

Untuk algoritma Naive Bayes, input yang dibutuhkan harus berupa data numerik. Oleh karena itu, tahapan proses pre-processing untuk model ini yaitu:

1. Handling Missing Data
    c. Fitur numerik: menggunakan nilai rata-rata
    d. Fitur kategorikal: menggunakan metode *most frequent*
2. Remove Duplicate
    Data duplikat akan dihilangkan
3. Dealing With Outliers
    Data numerik yang termasuk *outliers* akan diubah menjadi batas maksimum atau minimum yang didapat dari *IQR* dataset.
4. Feature Engineering
    c. Feature selection: menggunakan metode *Random Forest* dengan parameter *percentage* sebesar 25% untuk fitur numerik
    d. Feature scaling: menggunakan metode Z-score untuk data numerik
5. Encoder
    Untuk fitur kategorikal, dilakukan *encoding* dengan menggunakan metode *one-hot encoder* dari *sklearn*

Dengan menggunakan hasil pre-processing yang sama untuk kedua model, didapat performa masing-masing model dengan menggunakan metode Stratified k-fold sebagai berikut. Pada perbandingan tersebut, algoritma Gaussian Naive Bayes (GNB) yang dikembangkan secara manual (dari awal/scratch) menunjukkan kemiripan yang sangat signifikan dalam skor F1 dengan model Gaussian Naive Bayes yang diimplementasikan melalui pustaka scikit-learn. Hal ini mengindikasikan bahwa prosedur komputasi, estimasi parameter, dan pendekatan probabilitas yang dikembangkan dalam implementasi mandiri tersebut telah dilakukan secara akurat dan konsisten, sehingga tidak terjadi perbedaan berarti dalam kinerja model.

Selanjutnya, pada eksperimen variasi nilai parameter var_smoothing, diperoleh gambaran bahwa performa F1 score mengalami peningkatan seiring dengan penyesuaian nilai var_smoothing dalam rentang tertentu. Sebagaimana terilustrasi pada Gambar 2 dan 3, F1 score meningkat secara bertahap hingga

mencapai titik optimum pada sekitar var_smoothing = 10^{-4} (0.0001) sebelum kembali menurun. Puncak kinerja ini menunjukkan bahwa pemilihan parameter var_smoothing memiliki peran penting dalam mengendalikan tingkat regularisasi pada estimasi varians fitur, yang pada akhirnya mempengaruhi stabilitas prediksi GNB. Dengan demikian, titik optimal var_smoothing = 0.0001 untuk model GNB yang dikembangkan dari awal dapat dipandang sebagai bukti empiris bahwa tuning parameter yang tepat dapat memperbaiki generalisasi model, dan pada saat yang sama, hasil yang serupa dari implementasi pustaka scikit-learn mengafirmasi bahwa metode yang telah dirancang memang sesuai dengan prinsip-prinsip dasar Gaussian Naive Bayes.

.

3.3.1 Model Scratch

A. *var_smoothing = 0.1*

```
Maximum F1 Score That can be obtained from this model is: 6.032992879844562 %

Minimum F1 Score: 5.728219765739176 %

Overall F1 Score: 5.894385758453359 %

Standard Deviation is: 0.09756053603067119 %
```

B. *var_smoothing = 0.01*

```
Maximum F1 Score That can be obtained from this model is: 25.99097147617885 %

Minimum F1 Score: 25.80964222322839 %

Overall F1 Score: 25.91619587045108 %

Standard Deviation is: 0.07325507265528161 %
```

C. *var_smoothing = 0.001*

```
Maximum F1 Score That can be obtained from this model is: 31.923694685016002 %

Minimum F1 Score: 31.0747718677277 %

Overall F1 Score: 31.453311589503556 %

Standard Deviation is: 0.299650140787475 %
```

D. *var_smoothing = 0.0001*

```
Maximum F1 Score That can be obtained from this model is: 31.809169988796103 %

Minimum F1 Score: 31.318179207011248 %

Overall F1 Score: 31.547789624494392 %

Standard Deviation is: 0.1974741489857664 %
```

E. var_smoothing = 10^{-5}

```
Maximum F1 Score That can be obtained from this model is: 31.98426443763426 %

Minimum F1 Score: 31.049605703640736 %

Overall F1 Score: 31.64350392104105 %

Standard Deviation is: 0.35967953382623596 %
```

F. var_smoothing = 10^{-6}

```
Maximum F1 Score That can be obtained from this model is: 31.02837066765176 %

Minimum F1 Score: 29.49210238811687 %

Overall F1 Score: 30.341565070382675 %

Standard Deviation is: 0.528975942727846 %
```

G. var_smoothing = 10^{-7}

```
Maximum F1 Score That can be obtained from this model is: 27.793571796802656 %

Minimum F1 Score: 27.03034419717817 %

Overall F1 Score: 27.47675849054384 %

Standard Deviation is: 0.24828347972456116 %
```

H. var_smoothing = 10^{-8}

```
Maximum F1 Score That can be obtained from this model is: 26.002284421056398 %

Minimum F1 Score: 25.449210196978463 %

Overall F1 Score: 25.7729142574317 %

Standard Deviation is: 0.20633388196136193 %
```

I. var_smoothing = 10^{-9}

```
Maximum F1 Score That can be obtained from this model is: 24.172039611168422 %

Minimum F1 Score: 23.296003142952358 %

Overall F1 Score: 23.864307442831535 %

Standard Deviation is: 0.30431065323408213 %
```

J. *var_smoothing* = 10^{-10}

```
Maximum F1 Score That can be obtained from this model is: 22.301504592151485 %

Minimum F1 Score: 21.456908662799197 %

Overall F1 Score: 21.894075306547006 %

Standard Deviation is: 0.3185328437802964 %
```

K. *var_smoothing* = 10^{-11}

```
Maximum F1 Score That can be obtained from this model is: 20.580303893340584 %

Minimum F1 Score: 19.508075173497392 %

Overall F1 Score: 20.06255059042638 %

Standard Deviation is: 0.3691667512616795 %
```

L. *var_smoothing* = 10^{-12}

```
Maximum F1 Score That can be obtained from this model is: 19.228652303168296 %

Minimum F1 Score: 16.974352811085936 %

Overall F1 Score: 17.85254050224761 %

Standard Deviation is: 0.8017796784311422 %
```

M. *var_smoothing* = 10^{-13}

```
Maximum F1 Score That can be obtained from this model is: 17.161542930764266 %

Minimum F1 Score: 15.146911337689131 %

Overall F1 Score: 16.244187793385187 %

Standard Deviation is: 0.7794073823795116 %
```

N. *var_smoothing* = 10^{-14}

```
Maximum F1 Score That can be obtained from this model is: 16.01530072666853 %

Minimum F1 Score: 14.32307159754761 %

Overall F1 Score: 15.256226191838584 %

Standard Deviation is: 0.6722506896939223 %
```

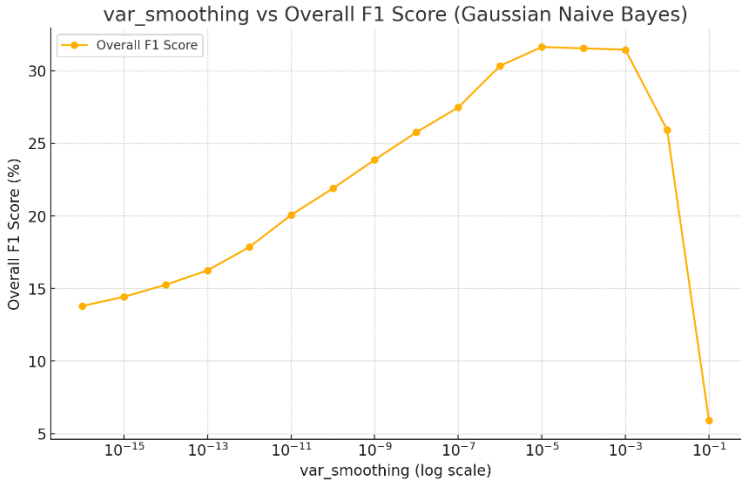
O. var_smoothing = 10^{-15}

```
Maximum F1 Score That can be obtained from this model is: 15.063578820609155 %

Minimum F1 Score: 13.740556306298151 %

Overall F1 Score: 14.425254976786128 %

Standard Deviation is: 0.5016969126049992 %
```

P. var_smoothing = 10^{-16}

```
Maximum F1 Score That can be obtained from this model is: 13.995188219796944 %

Minimum F1 Score: 13.536838542099384 %

Overall F1 Score: 13.78796621331057 %

Standard Deviation is: 0.1622897988526078 %
```

Dari hasil tersebut didapat plotting F1 score (macro) terhadap parameter var_smoothing seperti yang ditunjukkan pada Gambar 2.



Gambar 2. Grafik F1 (macro) score vs parameter var_smoothing dari model Gaussian Naïve Bayes scratch

3.3.2 Model scikit-learn

```
Maximum F1 Score That can be obtained from this model is: 24.17453811265781 %

Minimum F1 Score: 23.301920932345826 %

Overall F1 Score: 23.86679247242983 %

Standard Deviation is: 0.30316535568651026 %
```

A. var_smoothing = 0.1

```
Maximum F1 Score That can be obtained from this model is: 6.032992879844562 %

Minimum F1 Score: 5.728219765739176 %

Overall F1 Score: 5.894385758453359 %

Standard Deviation is: 0.09756053603067119 %
```

B. var_smoothing = 0.01

```
Maximum F1 Score That can be obtained from this model is: 25.991089839315563 %

Minimum F1 Score: 25.80964222322839 %

Overall F1 Score: 25.916030340428343 %

Standard Deviation is: 0.07345804349851436 %
```

C. var_smoothing = 0.001

```
Maximum F1 Score That can be obtained from this model is: 31.8509262818079 %

Minimum F1 Score: 31.0536034757088 %

Overall F1 Score: 31.431950350836534 %

Standard Deviation is: 0.2832076496298289 %
```

D. var_smoothing = 0.0001

```
Maximum F1 Score That can be obtained from this model is: 31.80768472359904 %

Minimum F1 Score: 31.31384034009844 %

Overall F1 Score: 31.545627802567523 %

Standard Deviation is: 0.19958790089803516 %
```

E. *var_smoothing* = 10^{-5}

```
Maximum F1 Score That can be obtained from this model is: 31.97058962102656 %

Minimum F1 Score: 31.084333013787113 %

Overall F1 Score: 31.649347336985556 %

Standard Deviation is: 0.34531847589588266 %
```

F. *var_smoothing* = 10^{-6}

```
Maximum F1 Score That can be obtained from this model is: 31.037686213628245 %

Minimum F1 Score: 29.488232582564898 %

Overall F1 Score: 30.34337906460845 %

Standard Deviation is: 0.5372436016801619 %
```

G. *var_smoothing* = 10^{-7}

```
Maximum F1 Score That can be obtained from this model is: 27.800515049739833 %

Minimum F1 Score: 27.0306409074199 %

Overall F1 Score: 27.480465100837893 %

Standard Deviation is: 0.2504189749224885 %
```

H. *var_smoothing* = 10^{-8}

```
Maximum F1 Score That can be obtained from this model is: 26.00162317118221 %

Minimum F1 Score: 25.44901043568406 %

Overall F1 Score: 25.77502506656532 %

Standard Deviation is: 0.2069480173978005 %
```

I. *var_smoothing* = 10^{-9}

```
Maximum F1 Score That can be obtained from this model is: 24.17453811265781 %

Minimum F1 Score: 23.301920932345826 %

Overall F1 Score: 23.86679247242983 %

Standard Deviation is: 0.30316535568651026 %
```

J. *var_smoothing* = 10^{-10}

```
Maximum F1 Score That can be obtained from this model is: 22.30380706334383 %

Minimum F1 Score: 21.460437052856555 %

Overall F1 Score: 21.899274449213628 %

Standard Deviation is: 0.31983982828820484 %
```

K. var_smoothing = 10^{-11}

```
Maximum F1 Score That can be obtained from this model is: 20.581349264750983 %

Minimum F1 Score: 19.53674944671628 %

Overall F1 Score: 20.080980020938462 %

Standard Deviation is: 0.3613115762619097 %
```

L. var_smoothing = 10^{-12}

```
Maximum F1 Score That can be obtained from this model is: 19.229833178753324 %

Minimum F1 Score: 16.9798581647008 %

Overall F1 Score: 17.883998275705277 %

Standard Deviation is: 0.779713979889837 %
```

M. var_smoothing = 10^{-13}

```
Maximum F1 Score That can be obtained from this model is: 17.17440888771063 %

Minimum F1 Score: 15.154874699408136 %

Overall F1 Score: 16.253248964063566 %

Standard Deviation is: 0.7806762976262588 %
```

N. var_smoothing = 10^{-14}

```
Maximum F1 Score That can be obtained from this model is: 16.016070028171043 %

Minimum F1 Score: 14.323209926920827 %

Overall F1 Score: 15.261476406152982 %

Standard Deviation is: 0.6724447380069949 %
```

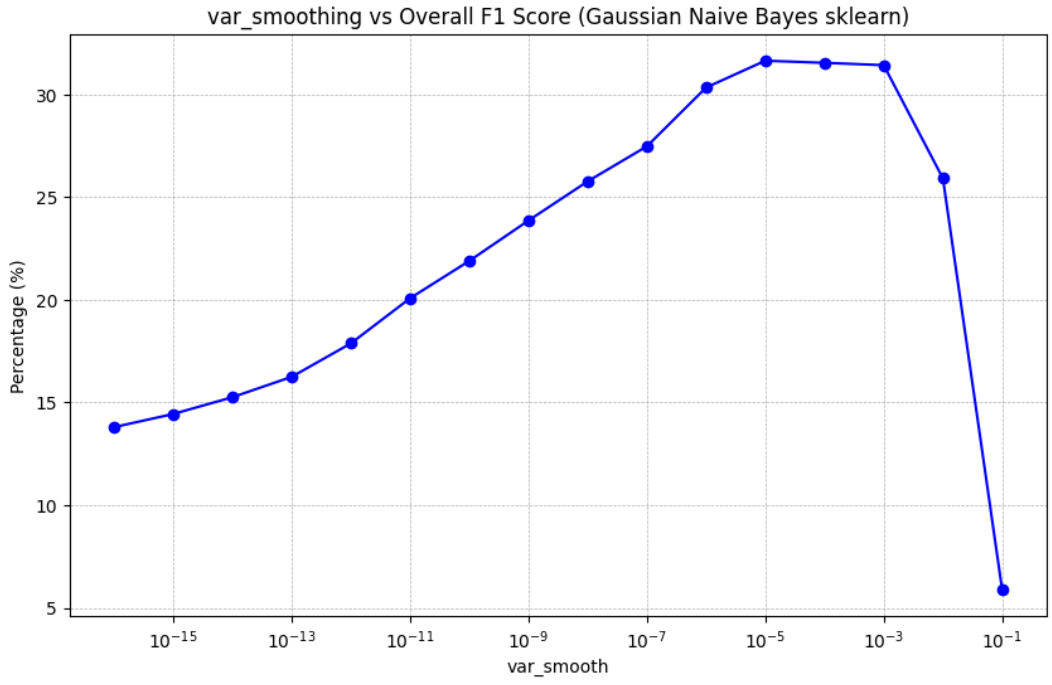
O. var_smoothing = 10^{-15}

```
Maximum F1 Score That can be obtained from this model is: 15.096404626361585 %

Minimum F1 Score: 13.74392943807706 %

Overall F1 Score: 14.443815477909705 %

Standard Deviation is: 0.5101840425616687 %
```

P. *var_smoothing* = 10^{-16}

```
Maximum F1 Score That can be obtained from this model is: 14.028519566363313 %

Minimum F1 Score: 13.536650281132594 %

Overall F1 Score: 13.803575178201626 %

Standard Deviation is: 0.17713526465393123 %
```

Dari hasil tersebut didapat plotting F1 score (macro) terhadap parameter var_smoothing untuk model scikit-learn seperti yang ditunjukkan pada Gambar 3.



Gambar 3. Grafik F1 (macro) score vs parameter var_smoothing dari model Gaussian Naïve Bayes dari scikit-learn

3.4 Algoritma Decision Tree

Untuk algoritma ID3 *from scratch*, input yang dibutuhkan dapat berupa data numerik atau kategorikal. Namun, untuk model DecisionTreeClassifier (dengan

criteria entropy) dari scikit-learn, diperlukan data numerik. Oleh karena itu, tahapan proses pre-processing untuk model ini yaitu:

1. Handling Missing Data
    e. Fitur numerik: menggunakan nilai rata-rata
    f. Fitur kategorikal: menggunakan metode *most frequent*
2. Remove Duplicate
    Data duplikat akan dihilangkan
3. Dealing With Outliers
    Fitur numerik yang termasuk *outliers* akan diubah menjadi batas maksimum atau minimum yang didapat dari *IQR* dataset.
4. Feature Engineering
    e. Feature selection: menggunakan metode *Random Forest* dengan parameter *percentage* sebesar 25% untuk fitur numerik
    f. Feature scaling: menggunakan metode Z-score untuk data numerik
5. Encoder
    Untuk fitur kategorikal dari model *DecisionTreeClassifier* dari *scikit-learn*, dilakukan *encoding* dengan menggunakan metode *one-hot encoder* dari *sklearn.* Sementara untuk model yang kami buat *from scratch*, tidak dibutuhkan encoder dikarenakan model kami dapat menerima input berupa fitur numerik maupun kategorikal.

Dengan menggunakan hasil pre-processing seperti yang dijelaskan pada masing-masing model, didapat performa masing-masing model dengan menggunakan metode Stratified k-fold sebagai berikut. Pada perbandingan performa tersebut, implementasi algoritma ID3 yang dibangun dari awal (dari scratch) menunjukkan hasil yang sangat kompetitif ataupun cukup dekat dengan yang diimplementasikan melalui pustaka scikit-learn. Hal ini mengindikasikan bahwa proses pengambilan keputusan berbasis entropi dan pemilihan atribut terbaik yang dilakukan secara manual telah disusun dengan baik, sehingga menghasilkan model yang memiliki tingkat generalisasi yang serupa dengan solusi standar yang tersedia secara luas.

Selanjutnya, berdasarkan evaluasi yang dilakukan dengan memvariasikan nilai kedalaman maksimum pohon (max_depth), terlihat adanya tren peningkatan

skor F1 seiring bertambahnya kedalaman pohon hingga mencapai kondisi terbaik tanpa batasan kedalaman (max_depth = None), seperti terilustrasi pada Gambar 4 dan 5. Pada saat kedalaman pohon dibatasi pada nilai yang relatif kecil (misalnya 5), performa model masih berada pada tingkat yang lebih rendah, diduga karena terbatasnya kompleksitas pohon dalam mempelajari pola data yang lebih rumit. Namun, ketika max_depth ditingkatkan menjadi 10, skor F1 mengalami peningkatan yang signifikan, menunjukkan bahwa penambahan kedalaman pohon memberi fleksibilitas yang lebih besar dalam memodelkan keragaman kelas. Ketika batasan ini dihilangkan sama sekali (max_depth = None), skor F1 mencapai tingkat yang mendekati atau bahkan setara dengan performa model Naive Bayes sklearn, serta mengindikasikan bahwa pohon keputusan ID3 yang dikembangkan secara manual telah mampu memaksimalkan pemanfaatan informasi pada dataset. Dengan demikian, pemilihan parameter max_depth yang tepat terbukti krusial dalam meningkatkan kualitas pembelajaran ID3 dari awal, serta mengafirmasi kesesuaian pendekatan yang telah diambil dalam implementasi manual tersebut.

3.4.1 ID3 From Scratch

Untuk model scratch, akan dilakukan implementasi pada 3 konfigurasi *max_depth*

A. Kedalaman Maksimum 5

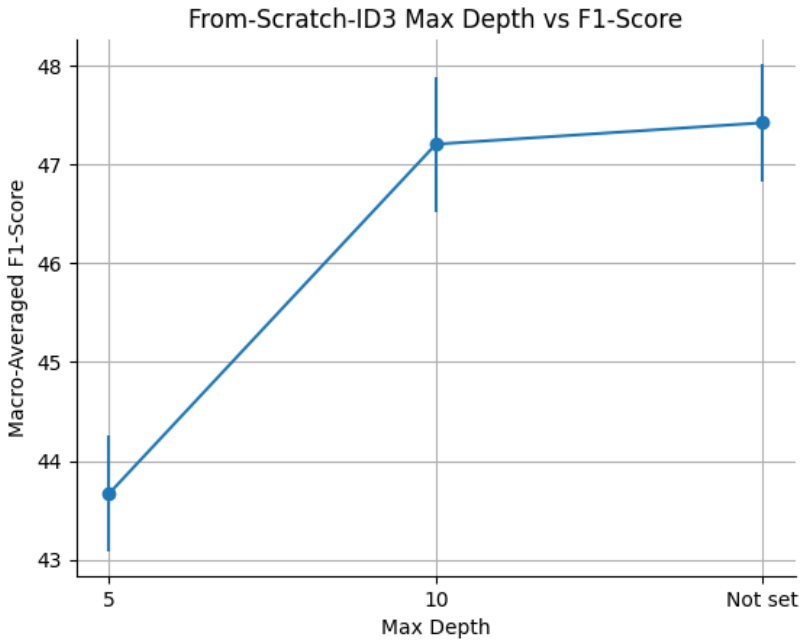
```
Maximum F1 Score That can be obtained from this model is: 44.464231243900265 %

Minimum F1 Score: 42.87626906088809 %

Overall F1 Score: 43.66989720189527 %

Standard Deviation is: 0.5859063638592655 %
```

B. Kedalaman Maksimum 10

```
Maximum F1 Score That can be obtained from this model is: 48.327164708438865 %

Minimum F1 Score: 46.57004166151138 %

Overall F1 Score: 47.20575298844602 %

Standard Deviation is: 0.6855433643471853 %
```

C. Tanpa Kedalaman Maksimum

```
Maximum F1 Score That can be obtained from this model is: 48.53849993286253 %

Minimum F1 Score: 46.73076820987706 %

Overall F1 Score: 47.42207589479325 %

Standard Deviation is: 0.6963242270267179 %
```



Gambar 4. Grafik F1 (macro) score terhadap variable parameter max_depth pada model Decision Tree dari scikit-learn

3.4.2 Model scikit-learn

A. Kedalaman Maksimum 5

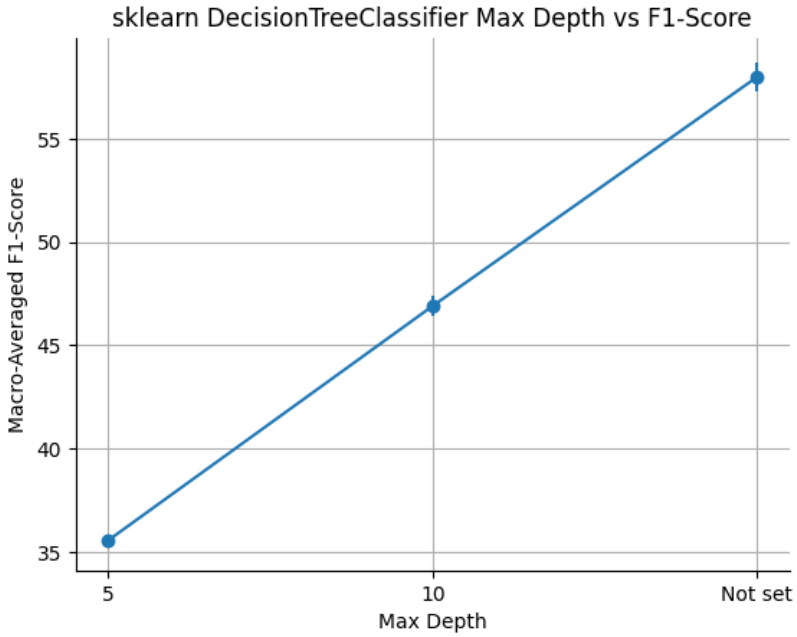
```
Maximum F1 Score That can be obtained from this model is: 36.02348775971844 %

Minimum F1 Score: 35.25728445427488 %

Overall F1 Score: 35.548714809603744 %

Standard Deviation is: 0.2756695781287563 %
```

B. Kedalaman Maksimum 10

```
Maximum F1 Score That can be obtained from this model is: 47.71742941087459 %

Minimum F1 Score: 46.34966172073449 %

Overall F1 Score: 46.89952355688014 %

Standard Deviation is: 0.49094321828781945 %
```

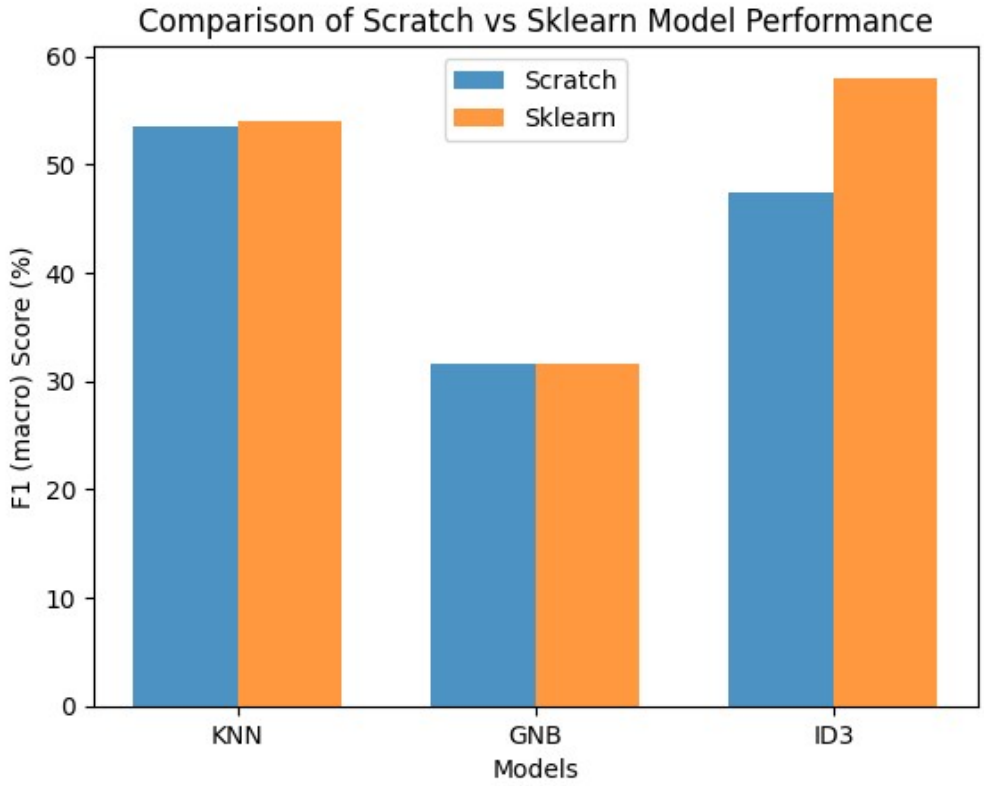
C. Tanpa Kedalaman Maksimum

```
Maximum F1 Score That can be obtained from this model is: 58.685004859816445 %

Minimum F1 Score: 56.67109670752115 %

Overall F1 Score: 57.98750129904408 %

Standard Deviation is: 0.7096917061238148 %
```



Gambar 5. Grafik F1 (macro) score terhadap variable parameter max_depth pada model Decision Tree scratch

3.5 Perbandingan



Gambar 6. Grafik F1 (macro) score dari tiap model terbaik di setiap algoritma

Dari hasil implementasi model scikit-learn dan model buatan kami, dapat dilihat perbandingan performa dari model terbaik di setiap algoritma dari model scikit-learn dan juga model buatan kami seperti yang ditunjukkan di Gambar 6.

Dapat dilihat bahwa pada algoritma Naive Gaussian Bayes, performa dari model scikite-learn dan model *scratch* sangatlah dekat. Perbedaan kecil ini disebabkan oleh perbedaan implementasi dari *smoothing algorithm* pada kedua algorithm tersebut.

Kemudian, dapat kita lihat perbedaan performa pada algoritma Decision Tree antara model dari scikite-learn dengan model *scratch* kami. Perbedaan ini disebabkan oleh beberapa hal, pertama seperti yag sudah dijelaskan pada Bab 3.4 bahwa terdapat perdebaan proses *pre-processing* yang dilakukan antara kedua model tersebut, yaitu pada bagian *encoder* dimana pada model *scratch* kami tidak membutuhkan adanya *encoding*. Hal tersebut memiliki kemungkinan untuk memengaruhi hasil performa dari model. Selain itu, perbedaan yang paling signifikan adalah perbedaan algoritma yang dipakai, dimana model dari scikite-

learn menggunakan algoritma CART untuk model Decision Tree mereka sementara model *scratch* kami, menggunakan algoritma ID3 untuk model Decision Tree nya.

Terakhir, dapat kita lihat bahwa perbedaan performa antara model dari scikit-learn dan model buatan kami pada algoritma k-Nearest Neighbor sangatlah minim. Hal ini menunjukkan bahwa implementasi yang telah kami buat dapat divalidasi karena hasilnya sudah cukup dekat dengan model referensi dari sci-kit learn.

PEMBAGIAN TUGAS TIAP ANGGOTA KELOMPOK

Tabel 1: Kontribusi Angota

NIM	Nama	Kontribusi
13121140	Muhammad Fadli Alfarizi	• Membuat algoritma k-nearest neighbors *from scratch* • Membuat kode data cleaning dan preprocessing (termasuk Pipiline *from scratch)*
13121142	Roby Pratama Sitepu	• Membuat laporan bagian algoritma k-nearest neighbors • Membuat laporan bagian data cleaning, dan preprocessing • Membantu pembuatan algoritma k-nearest neighbors from scratch
13621034	Muhammad Arviano Yuono	• Melakukan validasi tiap model • Membuat *submission* • Membuat laporan bagian validasi model
13621060	Hafizh Renanto Akhmad	• Membuat algoritma Gaussian naive Bayes *from scratch* • Membuat laporan bagian algoritma Gaussian naive Bayes • Membuat algoritma iterative dichotomizer (ID3) *from scratch* yang bisa menerima fitur numerik • Membuat laporan bagian iterative dichotomizer (ID3)

REFERENSI

[1] scikit-learn, "1.9. Naive Bayes," scikit-learn, [Online]. Available: https://scikit-learn.org/1.5/modules/naive_bayes.html.

[2] scikit-learn, "1.10. Decision Trees," scikit-learn, [Online]. Available: https://scikit-learn.org/1.5/modules/tree.html.

[3] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Harlow, Essex: Pearson, 2009.