LAPORAN TUGAS BESAR 2
IF3170 Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Disusun Oleh:
Akal Imitasi

M Hazim R Prajoda	13523009
Orvin Andika Ikhsan A	13523017
Fajar Kurniawan	13523027
Darrel Adinarya Sunanda	13523061
Reza Ahmad Syarif	13523119

Dosen Pengajar:
Dr. Nur Ulfa Maulidevi, S.T., M.Sc

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Semester II Tahun 2024/2025

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
Decision Tree Learning

1.1. Deskripsi Umum Implementasi

Kami mengimplementasikan ketiga variasi algoritma Decision Tree Learning: ID3, CART, dan C4.5 karena kami ingin meninjau mana yang performanya paling baik di antara ketiganya. Pada saat laporan ini dibuat, performa paling baik diperoleh dari alagoritma **C4.5**. Ketiga algoritma tersebut menggunakan pendekatan membangun *decision tree* secara dari atas (root) ke bawah (leaf). Pada setiap node, dipilih fitur yang menghasilkan *split* terbaik (*information gain* tertiggi) saat itu, kemudian dataset dibagi secara rekursif sampai pohon selesai karena alasan tertentu seperti tercapainya *max depth*.

1.1.1 Representasi Node dan Tree

Struktur pohon direpresentasikan menggunakan dictionary Python dan dibuat bersifat rekursif (suatu *tree node* menunjuk ke *subtree* yang merupakan struktur data yang sama):

- **Internal Node**: Detail dari *split*

```
node = {
    'feature': int,            # Index fitur yang digunakan
    'threshold': float,        # Nilai threshold untuk split
    'left': subtree,           # Subtree cabang kiri
    'right': subtree           # Subtree cabang kanan
}
```

- **Leaf Node**: Prediksi kelas

```
# ID3/CART:
leaf = class_label  # Prediksi kelas (int)

# C4.5 (probabilitas kelas):
leaf = {
    'value': class_label,             # Prediksi kelas
    'proba': np.array([p0, p1, p2])   # Distribusi probabilitas
}
```

1.1.2 Perhitungan Impurity: Entropy dan Gini

Impurity adalah ukuran ketidakmurnian atau heterogenitas dalam suatu dataset. Dalam Decision Tree, impurity digunakan untuk menentukan seberapa baik suatu split memisahkan kelas-kelas yang berbeda. Terdapat dua metode utama untuk mengukur impurity: Entropy (digunakan oleh ID3 dan C4.5) dan Gini Impurity (digunakan oleh CART).

Entropy (ID3 dan C4.5)

Entropy mengukur ketidakpastian atau keragaman dalam dataset:

$$\text{Entropy}(y) = -\sum_{i=1}^{K} p_i \log_2(p_i) \tag{1.1}$$

di mana:

- K = jumlah kelas
- p_i = proporsi sampel dari kelas i

Arti nilai entropy:

- Entropy = 0: Semua sampel dari satu kelas (pure)
- Entropy = 1: Distribusi seragam (maksimum impurity)
- Entropy $\in (0, 1)$: Campuran kelas

Gini Impurity (CART)

Gini impurity mengukur probabilitas misclassification jika label dipilih secara random:

$$\text{Gini}(y) = 1 - \sum_{i=1}^{K} p_i^2 \tag{1.2}$$

Makna nilai Gini:

- Gini = 0: Semua sampel dari satu kelas (pure)
- Gini = 0.5 (untuk K=2): Distribusi seragam
- Rentang: $[0, 1 - 1/K]$

1.1.3 Kriteria Pemilihan Atribut

Kriteria pemilihan atribut menentukan fitur mana yang akan digunakan untuk membagi dataset pada setiap node. Tujuannya adalah memilih atribut yang menghasilkan split paling informatif, yaitu yang paling mengurangi impurity atau ketidakpastian. Terdapat tiga metode utama: Information Gain (ID3), Gain Ratio (C4.5), dan Gini Gain (CART).

Information Gain (ID3)

Information Gain mengukur pengurangan entropy setelah pemisahan:

$$\text{IG}(S, A) = \text{Entropy}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \tag{1.3}$$

di mana:

- S = dataset awal
- A = atribut yang dievaluasi
- v = nilai-nilai yang mungkin dari atribut A
- S_v = subset dari S dengan atribut $A = v$

Kriteria Pemilihan atribut: Atribut dengan IG tertinggi.

Gini Gain (CART)

Gini Gain menggunakan penurunan Gini impurity:

$$\text{Gini Gain}(S, A) = \text{Gini}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{Gini}(S_v) \tag{1.4}$$

Gain Ratio (C4.5) - *Recommended*

Gain Ratio menormalkan information gain untuk menghindari bias:

$$\text{GainRatio}(S, A) = \frac{\text{IG}(S, A)}{\text{SplitInfo}(S, A)} \tag{1.5}$$

dengan:

$$\text{SplitInfo}(S, A) = -\sum_{v \in A} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|} \right) \tag{1.6}$$

1.1.4 Mekanisme Stopping Criteria

Stopping criteria menentukan kapan pembangunan pohon dihentikan:

Kriteria 1: Kedalaman Maksimum

$$\text{depth} \geq \text{max\_depth} \tag{1.7}$$

Tujuan: Mencegah overfitting dengan membatasi kompleksitas pohon.

Kriteria 2: Kemurnian Node

$$\text{Entropy}(S) = 0 \quad \text{atau} \quad \text{Gini}(S) = 0 \tag{1.8}$$

Tujuan: Menghentikan pemisahan jika semua sampel sudah dari satu kelas.

Kriteria 3: Sampel Minimum untuk Split

$$|S| < \text{min\_samples\_split} \tag{1.9}$$

Tujuan: Menghindari over-splitting pada node dengan sampel terlalu sedikit.

Kriteria 4: Gain Minimum

$$\text{Gain} < \text{min\_gain} \quad \text{(threshold kecil)} \tag{1.10}$$

Tujuan: Menghentikan jika pemisahan tidak memberikan peningkatan signifikan.

1.2. Detail Algoritmik

1.2.1 Pseudocode: Algoritma Pembangunan Pohon

```
FUNCTION BuildTree(data, labels, depth):
    // Step 1: Check if we should stop building
    IF depth >= max_depth OR all labels are same OR few samples:
        RETURN LeafNode(most_common_label)

    // Step 2: Try every feature to find the best split
    best_gain = 0
    best_feature = None
    best_threshold = None

    FOR EACH feature IN data:
        FOR EACH possible_threshold IN feature:
            gain = calculate_entropy_reduction(feature, threshold)

            IF gain > best_gain:
                best_gain = gain
                best_feature = feature
                best_threshold = threshold

    // Step 3: Check if we found a good split
    IF best_gain == 0:
        RETURN LeafNode(most_common_label)

    // Step 4: Split data into two groups
    left_group = samples WHERE feature <= threshold
    right_group = samples WHERE feature > threshold

    // Step 5: Recursively build subtrees for each group
    left_tree = BuildTree(left_group, left_labels, depth + 1)
    right_tree = BuildTree(right_group, right_labels, depth + 1)

    // Step 6: Create decision node and return
    RETURN DecisionNode(best_feature, best_threshold, left_tree, right_tree)
END FUNCTION
```

1.2.2 Traversal Tree Saat Prediksi

Prediksi dilakukan dengan melakukan traversal dari root ke leaf node:
Implementasi:

```python
def _predict_sample(self, x, tree):
    """Predict class for a single sample."""
    # Handle C4.5 leaf with probabilities
    if isinstance(tree, dict) and 'proba' in tree:
        return tree['proba']

    # Handle integer (single class) leaf (ID3/CART)
    if not isinstance(tree, dict):
        return tree

    # Traverse tree based on feature comparison
    if x[tree['feature']] <= tree['threshold']:
        return self._predict_sample(x, tree['left'])
    else:
        return self._predict_sample(x, tree['right'])

def predict(self, X):
    """Predict for multiple samples."""
    X = np.array(X) if not isinstance(X, np.ndarray) else X
    return np.array([self._predict_sample(x, self.tree_)
                     for x in X])
```

1.2.3 Penanganan Atribut Numerik (Actual Implementation)

Binary Split untuk Atribut Numerik

Implementasi menggunakan binary split untuk semua atribut numerik. Fitur categorical akan di-encode menjadi numerik sehingga tidak perlu dihandle secara khusus. Berikut cara kerjanya:

```
// Dalam _best_split method (untuk semua algoritma: ID3, CART, C4.5):

best_gain = -1
best_feature_idx = None
best_threshold = None

// Loop semua fitur
FOR EACH feature_idx IN range(n_features):
    X_column = data[:, feature_idx]
    thresholds = unique values in X_column

    // Coba setiap nilai unik sebagai threshold
    FOR EACH threshold IN thresholds:
        // Hitung gain (Information Gain untuk ID3/C4.5, atau Gini Gain untuk CART)
        gain = calculate_gain(X_column, labels, threshold)

        // Update best split jika gain lebih baik
        IF gain > best_gain:
            best_gain = gain
            best_feature_idx = feature_idx
            best_threshold = threshold

RETURN (best_feature_idx, best_threshold, best_gain)
```

Split yang dilakukan:

$$\text{Left} = \{x : x_{\text{feature}} \leq \text{threshold}\} \tag{1.11}$$

$$\text{Right} = \{x : x_{\text{feature}} > \text{threshold}\} \tag{1.12}$$

BAB II
Logistic Regression

2.1. Deskripsi Umum Implementasi

Implementasi *Logistic Regression from scratch* ini dibangun menggunakan bahasa pemrograman Python. Model ini dirancang sebagai pengklasifikasi linear probabilistik yang sangat fleksibel, dilengkapi dengan berbagai fitur optimasi tingkat lanjut untuk meningkatkan konvergensi dan stabilitas pelatihan. Secara matematis, model memprediksi probabilitas kelas target menggunakan fungsi aktivasi Sigmoid:

$$h_\theta(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{2.1}$$

Tujuan utama pelatihan adalah memaksimalkan fungsi objektif *Log-Likelihood* dengan metode *Stochastic Gradient Ascent*. Regularisasi L2 juga diterapkan untuk mencegah *overfitting* pada data berdimensi tinggi. .

2.1.1 Arsitektur Model

Arsitektur model dirancang untuk menangani berbagai skenario klasifikasi, mulai dari biner hingga multikelas. Untuk klasifikasi multikelas, diterapkan strategi *One-vs-Rest* (OvR) di mana K model biner dilatih secara independen. Salah satu fitur kunci dalam arsitektur ini adalah mekanisme *Class Imbalance Handling*. Jika mode `balanced` diaktifkan, model secara otomatis menghitung bobot sampel yang berbanding terbalik dengan frekuensi kelas, memastikan bahwa kelas minoritas mendapatkan kontribusi gradien yang proporsional selama proses pembelajaran.

2.1.2 Fungsi Loss dan Optimasi

Fungsi objektif yang dioptimalkan adalah *Log-Likelihood* dengan regularisasi L2. Persamaan fungsi objektif $J(\mathbf{w}, b)$ didefinisikan sebagai:

$$J(\mathbf{w}, b) = \sum_{i=1}^{m} \omega^{(i)} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] - \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \tag{2.2}$$

di mana $\omega^{(i)}$ adalah bobot sampel untuk menangani ketidakseimbangan kelas.

Algoritma optimasi yang digunakan adalah *Gradient Ascent* yang dilengkapi dengan Momentum untuk mempercepat konvergensi dan meredam osilasi. Aturan pembaruan parameter melibatkan variabel kecepatan v, yang mengakumulasi gradien historis:

$$v_t = \gamma v_{t-1} + \eta_t \nabla J$$
$$\theta_{t+1} = \theta_t + v_t$$

Di mana γ adalah koefisien momentum (0-1), η_t adalah *learning rate* pada iterasi ke-t, dan θ merepresentasikan parameter $(\mathbf{w}, b)$. Selain itu, diterapkan *Learning Rate Decay* di mana nilai η berkurang secara eksponensial pada setiap *epoch* untuk memastikan langkah optimasi menjadi lebih halus saat mendekati titik optimal.

2.2. Proses Training dan Prediksi

Proses pelatihan dilakukan secara iteratif dengan dukungan untuk *mini-batch processing*. Metode ini bisa menyeimbangkan efisiensi komputasi dan stabilitas gradien. Sebelum iterasi dimulai, parameter bobot diinisialisasi terlebih dahulu.

2.2.1 Inisialisasi Parameter

Strategi inisialisasi parameter dirancang secara fleksibel untuk mendukung stabilitas konvergensi yang optimal. Implementasi ini menyediakan opsi konfigurasi `use_xavier` yang memungkinkan pemilihan antara dua pendekatan inisialisasi bobot ($\mathbf{w}$):

Apabila metode *Xavier Initialization* diaktifkan, bobot diinisialisasi secara acak menggunakan distribusi seragam dalam rentang $[-\delta, \delta]$, di mana batas δ dihitung berdasarkan rumus $\delta = \sqrt{\frac{6}{n_{in}+1}}$. Pendekatan ini bertujuan untuk menjaga variansi aktivasi tetap konsisten di awal pelatihan. Sebaliknya, jika mode standar digunakan, bobot diinisialisasi secara deterministik dengan nilai nol.

Selain bobot utama, parameter bias (b) selalu diinisialisasi dengan nilai nol. Lebih lanjut, karena algoritma menggunakan optimasi berbasis momentum, variabel kecepatan $\mathbf{v}_w$ dan v_b juga diinisialisasi dengan nilai nol pada awal proses pelatihan untuk memastikan tidak ada bias arah pergerakan sebelum gradien pertama dihitung.

2.2.2 Gradient Computation

Pada setiap langkah iterasi, gradien dihitung berdasarkan selisih antara label aktual dan prediksi model. Keunikan implementasi ini terletak pada integrasi *sample weights* ke dalam perhitungan gradien error:

$$\text{Error} = (\mathbf{y} - \mathbf{y}_{pred}) \times \boldsymbol{\omega} \tag{2.3}$$

Gradien kemudian dihitung sebagai hasil perkalian dari fitur input dan error terbobot tersebut. Hal ini memastikan bahwa kesalahan prediksi pada kelas minoritas (yang memiliki bobot ω lebih besar) akan menghasilkan gradien yang lebih besar, memaksa model untuk belajar lebih agresif pada kelas tersebut.

2.2.3 Proses Prediksi

Setelah model konvergen, proses prediksi dilakukan dengan menghitung probabilitas posterior.

- Untuk kasus biner, keputusan kelas didasarkan pada *threshold* 0.5.
- Untuk kasus multikelas, prediksi dilakukan dengan memilih kelas yang memiliki probabilitas tertinggi dari seluruh klasifier biner yang ada.

2.3. Hyperparameter Tuning

Untuk mencapai performa optimal pada model *Logistic Regression*, dilakukan serangkaian eksperimen *tuning* terhadap berbagai hyperparameter. Fokus utama *tuning* adalah pada parameter optimasi, yakni *learning rate* awal (η) dan tingkat peluruhannya (*learning rate decay*). *Decay rate* yang tepat terbukti krusial untuk memastikan langkah optimasi mengecil secara progresif, memungkinkan model untuk mencapai

titik minimum fungsi objektif tanpa mengalami divergensi di akhir pelatihan. Selain itu, koefisien momentum juga disetel dalam rentang $[0.5, 0.9]$ untuk menemukan keseimbangan terbaik antara akselerasi konvergensi dan kestabilan lintasan gradien.

Selain parameter optimasi, eksperimen juga dilakukan pada komponen struktural model. Kekuatan regularisasi L2 diuji dengan variasi nilai logaritmik (seperti 0.01, 0.1, 1.0) untuk mengontrol kompleksitas model, mencegah *overfitting*, dan menyeimbangkan *bias-variance trade-off*. Strategi inisialisasi bobot juga dibandingkan antara metode standar dan *Xavier Initialization*, di mana metode terakhir dievaluasi efektivitasnya dalam mempercepat konvergensi awal. Ukuran *batch* juga divariasikan, membandingkan performa antara pembaruan stokastik murni, *mini-batch*, dan *full-batch* untuk menentukan konfigurasi yang paling efisien secara komputasi.

Terakhir, dilakukan eksperimen dengan variasi mode. Mode `'balanced'` digunakan untuk memberikan bobot lebih pada kelas minoritas selama perhitungan gradien dibanding mode biasa. Jumlah iterasi maksimum ditetapkan pada angka yang cukup besar untuk menjamin bahwa algoritma memiliki kesempatan yang cukup untuk konvergen, terutama ketika menggunakan *learning rate* yang kecil atau peluruhan yang agresif.

BAB III
Support Vector Machine (SVM)

3.1. Deskripsi Umum Implementasi

Implementasi *Support Vector Machine* (SVM) *from scratch* ini berfokus pada varian SVM Linear. Algoritma ini dirancang menggunakan pendekatan pemrograman berorientasi objek (*Object-Oriented Programming*), di mana kelas dasar abstrak `SVMBase` mendefinisikan kerangka kerja standar, sementara kelas turunan mengimplementasikan strategi klasifikasi multikelas yang spesifik.

Inti dari implementasi ini adalah kelas `_BinarySVM`, yang bertugas melatih pengklasifikasi biner menggunakan fungsi *Hinge Loss* dan regularisasi L2.Implementasi ini menerapkan algoritma optimasi numerik sederhana namun efisien, yaitu *Stochastic Gradient Descent*, untuk menemukan *hyperplane* pemisah yang optimal.

3.1.1 Formulasi SVM

Secara matematis, model SVM ini bertujuan untuk meminimalkan fungsi objektif yang terdiri dari dua komponen, yaitu kesalahan klasifikasi (melalui *Hinge Loss*) dan kompleksitas model (melalui regularisasi L2). Fungsi objektif $J(\mathbf{w}, b)$ didefinisikan sebagai:

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \tag{3.1}$$

di mana:

- m adalah jumlah sampel data.
- $y^{(i)} \in \{-1, 1\}$ adalah label kelas target.
- $\max(0, 1 - y(\dots))$ adalah fungsi *Hinge Loss* yang memberikan penalti linear jika sampel berada di sisi yang salah dari margin atau di dalam margin.
- $\frac{\lambda}{2} \|\mathbf{w}\|_2^2$ adalah suku regularisasi L2 untuk memaksimumkan margin.

3.1.2 Multiclass Classification Strategies

Karena formulasi dasar SVM adalah pengklasifikasi biner, implementasi ini menyediakan tiga strategi berbeda untuk menangani masalah klasifikasi multikelas:

- **One-vs-All (OvA):** Melatih K model biner, di mana setiap model membedakan satu kelas dari gabungan kelas lainnya.
- **One-vs-One (OvO):** Melatih $K(K-1)/2$ model biner untuk setiap pasangan kelas yang mungkin.
- **Directed Acyclic Graph (DAG):** Menggunakan model biner yang sama dengan OvO, namun menerapkan struktur graf terarah untuk mempercepat proses prediksi.

3.2. Proses Optimasi

Proses pencarian parameter optimal $(\mathbf{w}, b)$ dilakukan menggunakan algoritma *Gradient Descent* secara iteratif. Implementasi menerapkan pembaruan parameter pada setiap iterasi berdasarkan gradien rata-rata dari seluruh dataset pada strategi OvA, sedangkan pada strategi OvO dan DAG, perhitungannya hanya dilakukan pada subset data yang relevan.

3.2.1 Gradient Computation

Gradien dihitung berdasarkan turunan parsial dari fungsi objektif. Untuk setiap sampel data, jika kondisi margin $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1$ terpenuhi, maka gradien dihitung sebagai:

$$\nabla_{\mathbf{w}} J = -C \cdot y^{(i)} \cdot \mathbf{x}^{(i)} + \mathbf{w} \tag{3.2}$$

$$\nabla_b J = -C \cdot y^{(i)} \tag{3.3}$$

Jika sampel diklasifikasikan dengan benar dan berada di luar margin, maka gradien dari *hinge loss* adalah nol, dan pembaruan hanya dipengaruhi oleh suku regularisasi:

$$\nabla_{\mathbf{w}} J = \mathbf{w} \tag{3.4}$$

Dalam implementasi kode, variabel `self.C` berperan sebagai faktor pengali pada suku *loss*, yang secara efektif menyeimbangkan antara meminimalkan *error* pelatihan dan meminimalkan *norm* bobot.

3.2.2 Hyperparameter C

Parameter `C` dalam implementasi ini berfungsi sebagai koefisien penalti pelanggaran margin.

- **Nilai C Besar:** Memberikan penalti yang berat untuk setiap kesalahan klasifikasi. Hal ini memaksa model untuk mencari margin yang lebih sempit dan lebih akurat pada data latih (*Hard Margin*). Hal ini membuat risiko *overfitting* lebih besar.
- **Nilai C Kecil:** Memberikan toleransi yang lebih besar terhadap kesalahan klasifikasi. Hal ini menghasilkan margin yang lebih lebar (*Soft Margin*) dan model yang lebih sederhana, namun berpotensi *underfitting*.

3.3. One-vs-All (OvA) Strategy

Implementasi kelas `SVMOneVsAll` melatih sejumlah model biner sesuai dengan jumlah kelas unik (K). Untuk setiap kelas target k, dataset dilabel ulang menjadi masalah biner, yaitu kelas k akan menjadi $+1$ dan seluruh kelas lain menjadi -1.

- **Training:** Dilakukan iterasi sebanyak K kali untuk menghasilkan daftar model `classifiers_`.
- **Prediksi:** Untuk input baru, skor keputusan $(w^T x + b)$ dihitung dari semua K model. Kelas diprediksi berdasarkan model yang memberikan skor tertinggi menggunakan fungsi `np.argmax`.
- **Class Weighting:** Untuk menangani ketidakseimbangan data saat menggabungkan kelas "Rest", diterapkan pembobotan sampel otomatis jika mode `balanced` diaktifkan.

3.4. One-vs-One (OvO) Strategy

Implementasi kelas `SVMOneVsOne` melatih model biner untuk setiap kombinasi pasangan kelas unik. Total model yang dilatih adalah sebanyak $\frac{K(K-1)}{2}$.

- **Training:** Algoritma mengiterasi pasangan kelas (i, j), mengambil subset data yang hanya mengandung kedua kelas tersebut, dan melatih SVM biner untuk membedakan i (label $+1$) dan j (label -1).
- **Prediksi:** Menggunakan mekanisme *voting*. Setiap input dievaluasi oleh seluruh model pasangan. Jika model (i, j) memprediksi kelas i, maka kelas i mendapat satu suara. Prediksi akhir adalah kelas dengan jumlah suara terbanyak.

3.5. DAG SVM Strategy

Implementasi kelas `SVMDAG` menerapkan strategi *Directed Acyclic Graph* yang dirancang untuk mempercepat proses inferensi. Meskipun berbagi basis model yang sama dengan OvO, strategi ini menggunakan struktur keputusan berbasis graf untuk mengurangi jumlah evaluasi model yang diperlukan saat prediksi.

- **Training:** Identik dengan strategi OvO. Algoritma melatih total $\frac{K(K-1)}{2}$ model biner untuk setiap kombinasi pasangan kelas unik. Setiap model dilatih pada subset data yang hanya memuat dua kelas yang bersangkutan.
- **Prediksi:** Menggunakan mekanisme eliminasi iteratif.
  - Dimulai dengan daftar seluruh kelas kandidat.
  - Pada setiap langkah, algoritma mengambil kelas pertama dan terakhir dalam daftar, lalu mempertandingkannya menggunakan klasifier biner terkait.
  - Kelas yang "kalah" langsung dieliminasi dari daftar kandidat.
  - Proses ini berulang hingga hanya tersisa satu kelas tunggal sebagai hasil prediksi.

  Pendekatan ini jauh lebih efisien karena hanya mengevaluasi $K - 1$ model per sampel, dibandingkan $\frac{K(K-1)}{2}$ pada OvO.

3.6. Hyperparameter Tuning

Untuk mengoptimalkan kinerja model SVM, dilakukan eksperimen *tuning* terhadap parameter-parameter kunci yang tersedia dalam konstruktor kelas `SVMBase`. Parameter yang dievaluasi meliputi:

- **Regularization Parameter (`C`):** Parameter ini mengontrol *trade-off* antara memaksimalkan margin dan meminimalkan kesalahan klasifikasi pada data latih.
  - Nilai C yang besar memberikan penalti lebih berat pada pelanggaran margin, menghasilkan *Hard Margin* yang lebih ketat namun berisiko *overfitting*.
  - Nilai C yang kecil memberikan toleransi lebih besar (*Soft Margin*), menghasilkan model yang lebih sederhana dengan regularisasi yang lebih kuat, namun berisiko *underfitting*
- **Learning Rate (`lr`):** Menentukan besaran langkah pembaruan bobot ($\mathbf{w}$) dan bias (b) pada setiap iterasi *Gradient Descent*. Nilai yang terlalu besar dapat menyebabkan algoritma gagal konvergen, sedangkan nilai yang terlalu kecil akan memperlambat proses pelatihan secara signifikan.

- **Number of Iterations (`n_iter`):** Karena implementasi ini tidak menggunakan kriteria penghentian otomatis berdasarkan toleransi kerugian atau *early stopping*, jumlah iterasi ditetapkan sebagai parameter tetap. Nilai ini disetel secukupnya untuk memastikan algoritma mencapai titik konvergensi stabil pada fungsi objektif.
- **Class Weight Mode:** Mengingat potensi ketidakseimbangan dataset, hyperparameter ini disetel antara mode `None` atau `'balanced'`. Pada mode `'balanced'`, model secara otomatis menghitung bobot yang berbanding terbalik dengan frekuensi kelas, memastikan bahwa kelas minoritas mendapatkan kontribusi gradien yang proporsional selama pelatihan.

BAB IV
Cleaning dan Preprocessing

Tahap *preprocessing* dan *cleaning* adalah salah satu tahap paling penting dalam machine learning. Berikut merupakan preprocessing dan cleaning yang dilakukan pada dataset.

4.1. Deskripsi Tahapan Cleaning

Dataset awal memiliki karakteristik berikut:

- **Training set**: 3096 samples
- **Test set**: 1328 samples
- **Original features**: 36 fitur numerik
- **Target**: 3 kelas (Dropout, Enrolled, Graduate)
- **Target distribution (training)**:
  - Dropout: ~60% dari data
  - Enrolled: ~18% dari data (minority class)
  - Graduate: ~22% dari data

Tahapan cleaning mencakup proses pengecekan data quality, identifikasi missing values, dan verifikasi format data sebelum feature engineering.

4.1.1 Analisis Missing Values

Pengecekan missing values dilakukan pada tahap awal untuk mengidentifikasi data gaps:

```
import pandas as pd
import numpy as np

train_df_raw = pd.read_csv('data/train.csv')
test_df_raw = pd.read_csv('data/test.csv')

# Cek missing values
print("\nMissing Values:")
missing = train_df_raw.isnull().sum()
if missing.sum() == 0:
    print("  No missing values found")
else:
    print(missing[missing > 0])
```

Dari pengecekan, ditemukan bahwa tidak ada missing values sehingga tidak perlu dilakukan *imputation* ataupun penghapusan tuple.

4.1.2 Outlier Detection dan Handling

Outliers adalah nilai yang menyimpang dari distribusi normal (nilai ekstrim). Dalam dataset ini, outliers dapat muncul pada:

- Age at enrollment: orang-orang berusia sangat tua

- Unemployment rate dan inflation yang ekstrim
- Grades (20/20 atau 0/20)

Penanganan Outlier di Proyek Ini

Tidak ada penghapusan nilai outlier pada dataset ini. Hal ini karena beberapa alasan:

1. Outliers dalam konteks ini sebenarnya **memiliki arti**, bukan errors:
   - Usia tinggi adalah minoritas tapi tidak mustahil (nilainya masih dalam batas wajar usia manusia)
   - Nilai (grades) sempurna (20/20) menunjukkan kepandaian dan memang hal yang wajar dalam sebuah ujian dan bukan error
   - Kondisi ekonomi seperti inflasi dan *unemployment rate* bisa saja terjadi di daerah rawan kemiskinan atau dari faktor lainnya
2. Penghapusan data outlier dapat **menghapus informasi penting** (untuk prediksi dropout dan sebagainya)
3. **RobustScaler** digunakan untuk scaling karena cocok dengan dataset:
   - RobustScaler robust terhadap outliers (menggunakan IQR, bukan std)
   - Mempertahankan outlier values, hanya rescaling range-nya
   - Lebih sesuai untuk student data yang biasanya memang akan memiliki *extreme values*

4.1.3 Feature Selection

Feature selection adalah proses memilih subset fitur yang paling relevan untuk meningkatkan performa model dengan mengurangi dimensi data, mempercepat training, dan mengurangi risiko overfitting. Dalam dataset ini, diterapkan pendekatan **filter-based feature selection** menggunakan metode ensemble voting dari tiga teknik statistik.

Metode Feature Selection yang Diimplementasikan

Setelah feature engineering menghasilkan 72 fitur (36 original + 36 engineered), dilakukan reduksi dimensi menjadi 39 fitur menggunakan kombinasi tiga metode filter:

Chi-Square Test (χ^2)

Chi-Square Test mengukur dependensi antara fitur kategorikal/numerik dengan target variabel. Nilai χ^2 yang tinggi menunjukkan fitur tersebut memiliki hubungan kuat dengan kelas target.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{4.1}$$

dimana O_i adalah observed frequency dan E_i adalah expected frequency.

ANOVA F-Score

ANOVA F-Score mengukur perbedaan rata-rata antar kelas. Fitur dengan F-Score tinggi memiliki kemampuan diskriminatif yang baik untuk memisahkan kelas-kelas yang berbeda.

$$F = \frac{\text{Between-group variability}}{\text{Within-group variability}} = \frac{MS_B}{MS_W} \tag{4.2}$$

Mutual Information (MI)

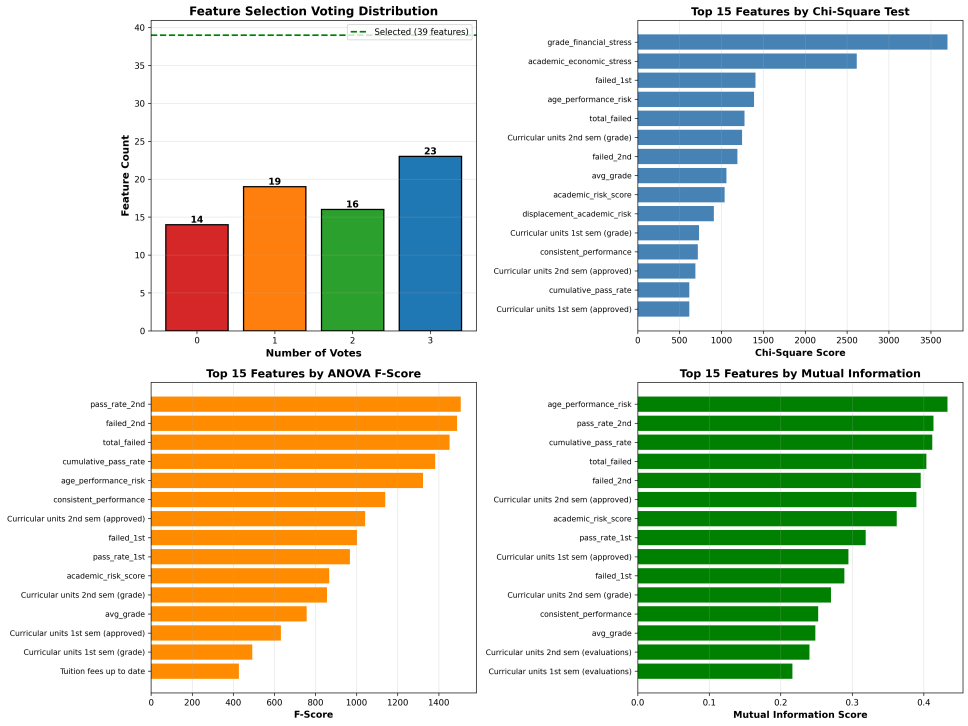
Mutual Information mengukur seberapa banyak informasi yang diperoleh tentang target variabel dari suatu fitur, termasuk hubungan non-linear.

$$MI(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{4.3}$$

Consensus Voting Strategy

Untuk menghindari bias dari satu metode tunggal, digunakan strategi consensus voting:

1. Setiap metode (Chi-Square, F-Score, MI) memilih top-40 fitur terbaik
2. Setiap fitur mendapat "vote" dari metode yang memilihnya (maksimal 3 votes)
3. Fitur dengan ≥ 2 votes (majority) dipilih sebagai fitur final
4. Menghasilkan 39 fitur terpilih dari 72 fitur awal (reduksi 45.8%)



Gambar 4.1: Consensus Voting Strategy untuk Feature Selection

Hasil Feature Selection

Tabel 4.1: Ringkasan Proses Feature Selection

Tahap	Jumlah Fitur	Persentase
Original Features	36	50.0%
Setelah Feature Engineering	72	100.0%
Setelah Feature Selection	39	54.2%
Fitur yang Di-drop	33	45.8%

Alasan Menggunakan Filter Methods

- **Independence**: Filter methods tidak bergantung pada model tertentu, sehingga hasil selection dapat digunakan untuk berbagai algoritma (DTL, LogReg, SVM)
- **Computational Efficiency**: Lebih cepat dibanding wrapper methods (e.g., RFE) karena tidak perlu training model berulang kali
- **Robustness**: Consensus voting dari 3 metode mengurangi bias dan meningkatkan stabilitas pemilihan fitur
- **Post-SMOTE Application**: Feature selection dilakukan setelah SMOTE untuk memastikan fitur yang dipilih optimal untuk data yang sudah balanced

Perbedaan dengan Implicit Feature Selection di DTL

Meskipun Decision Tree secara implisit melakukan feature selection saat memilih split nodes berdasarkan information gain, explicit feature selection sebelum training memberikan beberapa keunggulan:

- **Reduksi Overfitting**: Mengurangi noise dari fitur irrelevant sebelum training
- **Konsistensi Antar Model**: Semua 7 algoritma (DTL, LogReg, SVM) menggunakan subset fitur yang sama untuk fair comparison
- **Efisiensi Komputasi**: Training lebih cepat dengan 39 fitur dibanding 72 fitur
- **Interpretability**: Lebih mudah menganalisis model dengan jumlah fitur yang lebih sedikit

4.2. Feature Engineering

Feature engineering adalah proses membuat features baru dari features yang sudah ada di dataset untuk meningkatkan prediksi. Dalam tugas ini, 36 features ditransformasi menjadi 72 total features (36 engineered + 36 original).

4.2.1 Academic Features (12 fitur)

Academic features berisi fitur-fitur akademik dari mahasiswa.

1. **Pass Rate Features (3 fitur)**

```
df['pass_rate_1st'] = df['Curricular units 1st sem (approved)'] / \
                      (df['Curricular units 1st sem (enrolled)'] + eps)
df['pass_rate_2nd'] = df['Curricular units 2nd sem (approved)'] / \
                      (df['Curricular units 2nd sem (enrolled)'] + eps)

df['cumulative_pass_rate'] = \
    (df['Curricular units 1st sem (approved)'] + \
     df['Curricular units 2nd sem (approved)']) / \
    (df['Curricular units 1st sem (enrolled)'] + \
     df['Curricular units 2nd sem (enrolled)'] + eps)
```

Penjelasan:

- **pass_rate_1st**: Proporsi mata kuliah lulus di semester 1 (0 = semua gagal, 1 = semua lulus)
- **pass_rate_2nd**: Proporsi mata kuliah lulus di semester 2
- **cumulative_pass_rate**: pass rate gabungan 2 semester
- `eps = 1e-6` mencegah division by zero untuk students yang tidak enroll
- Jika lulus lebih banyak mata kuliah maka lebih tinggi kemungkinan lulus dan tidak dropout

2. **Grade Trend Features (2 fitur)**

```
df['grade_improvement'] = df['Curricular units 2nd sem (grade)'] - \
                          df['Curricular units 1st sem (grade)']

df['avg_grade'] = (df['Curricular units 1st sem (grade)'] + \
                   df['Curricular units 2nd sem (grade)']) / 2
```

Penjelasan:

- **grade_improvement**: Selisih nilai 2nd sem - 1st sem
- **avg_grade**: Rata-rata nilai 2 semester (scale 0-20 untuk dataset ini)
- Jika ada improvement di grade maka kemungkinan kemampuan akademiknya sedang meningkat dan kecil kemungkinan dropout

3. **Failed Courses Features (3 fitur)**

```
df['failed_1st'] = df['Curricular units 1st sem (enrolled)'] - \
                   df['Curricular units 1st sem (approved)']
df['failed_2nd'] = df['Curricular units 2nd sem (enrolled)'] - \
                   df['Curricular units 2nd sem (approved)']

df['total_failed'] = df['failed_1st'] + df['failed_2nd']
```

Penjelasan:

- **failed_1st, failed_2nd**: Jumlah mata kuliah yang tidak lulus per semester
- **total_failed**: Total gagal dalam 2 semester
- Tentunya jika mata kuliah yang gagal semakin banyak maka meningkatkan kemungkinan dropout

4. Courses Without Evaluation (1 fitur)

```
df['total_without_eval'] = \
    df['Curricular units 1st sem (without evaluations)'] + \
    df['Curricular units 2nd sem (without evaluations)']
```

Penjelasan:

- `Courses Without Evaluations` = Mata kuliah yang diambil tapi tidak ada evaluasi akhir
  - Bisa karena: student tidak ada di ujian final atau dropout di tengah semester
- **total_without_eval**: Total akumulasi absent dari exam (tidak hadir)

5. Academic Risk Score (1 fitur - Composite)

```
df['academic_risk_score'] = df['total_failed'] + \
                            df['total_without_eval'] * 2
```

Penjelasan:

- Formula: `academic_risk_score` = failed courses + 2 ×no-shows/absent
- **Weighting**: absent 2x lebih berat karena:
  - Ujian gagal (grade 0) = berusaha/berjuang tapi gagal, bisa recovery dan memperbaiki diri
  - Absent pertanda mahasiswa tidak berusaha, tidak serius, hampir tidak ada harapan. Jika banyak ketidakhadiran bahkan hingga tidak mengikuti ujian berarti niat/motivasi belajar sangat kurang dan mungkin dapat meningkatkan probabilitas dropout

6-7. Workload and Consistency Metrics (2 fitur)

```
df['total_enrolled'] = df['Curricular units 1st sem (enrolled)'] + \
                       df['Curricular units 2nd sem (enrolled)']

df['consistent_performance'] = \
    ((df['pass_rate_1st'] > 0.7).astype(int) * \
     (df['pass_rate_2nd'] > 0.7).astype(int))
```

Penjelasan:

- **total_enrolled**: Total mata kuliah dalam 2 semester
  - Semakin banyak mata kuliah maka semakin tinggi ambisi untuk berhasil
  - Terlalu ambisius dengan beban tinggi bisa juga berakibat gagal
- **consistent_performance**: Binary flag (0 atau 1)
  - 1 = lulus >70% di kedua semester (stabil, kemungkinan dropout rendah)
  - 0 = minimal satu semester <70% pass rate (berisiko, inkonsisten)
  - mahasiswa yang konsisten lebih tinggi peluangnya untuk lulus

4.2.2 Economic Features (4 fitur)

Economic features mencakup masalah finansial. Kendala finansial yang parah biasanya merupakan penyebab dropout di pendidikan.

1. Financial Burden (Compound Stress)

```
df['financial_burden'] = (df['Debtor'] *
                          (1 - df['Tuition fees up to date']) *
                          (1 - df['Scholarship holder']))
```

Penjelasan:

- Formula: `financial_burden` = Memiliki utang dan tidak ada beasiswa serta belum membayar tuition UKT
- Binary output (0 atau 1):
  - 1 = Sangat krisis (utang + tidak bayar + tanpa bantuan)
  - 0 = Tidak sedang krisis (minimal 1 kondisi tidak terpenuhi)

2. Economic Vulnerability (Macro-Economic Impact)

```
# Vulnerability adalah macro-economic conditions digabung dengan personal situation
# Unemployment/inflation tinggi akan lebih berdampak ke mahasiswa dengan utang (debtors)
↪  dan non-beasiswa (non-scholarship)
df['economic_vulnerability'] = (
    df['Unemployment rate'] * df['Debtor'] +
    df['Inflation rate'] * (1 - df['Scholarship holder'])
)
```

Economic Vulnerability nilainya kontinu, semakin tinggi maka semakin terdampak oleh kondisi makro ekonomi. Misal *unemployment rate* tinggi, memiliki hutang, dan bukan penerima beasiswa maka nilainya akan sangat tinggi.

3-4. Protective Factors (2 fitur)

```
df['scholarship_protection'] = (df['Scholarship holder'] *
                                (1 - df['Debtor']))

df['financial_stability'] = (df['Tuition fees up to date'] *
                             (1 - df['Debtor']))
```

Penjelasan:

- **scholarship_protection**: Binary (0 atau 1)
  - 1 = Punya beasiswa dan tidak punya utang (ideal, bisa fokus di studi)
  - 0 = Tidak punya beasiswa atau punya utang (rentan ke dropout)
- **financial_stability**: Binary (0 atau 1)
  - 1 = Tuition (UKT) dibayar semya dan tidak punya utang (ideal)
  - 0 = Tuition belum dibayar atau punya utang (rentan)

4.2.3 Sociodemographic Features (5 fitur)

Sociodemographic features mencakup hal-hal personal, keluarga, dan demografi yang mungkin memengaruhi kemungkinan dropout atau bertahan.

1. Parent Education Average

```
df['parent_education_avg'] = (
    df["Mother's qualification"] +
    df["Father's qualification"]
) / 2
```

Penjelasan:

- Scale: 0-3
- Orang tua yang berpendidikan lebih tinggi mungkin bisa:
  - Memahami sistem pendidikan tinggi
  - Memberikan petunjuk/arahan akademik
  - Memberi dukungan (mungkin melalui *networking*-nya)

2. Support Deficit (kekurangan dukungan sehingga rentan)

```
# Special needs students yang juga memiliki orang tua berpendidikan kurang memiliki risiko
↪  ganda
df['support_deficit'] = (
    df['Educational special needs'] *
    (1 / (df['parent_education_avg'] + 1 + eps))
)
```

Alasan digabungkan: Special needs bisa dihadapi dengan lebih mudah jika dengan dukungan. Lalu untuk yang orang tua berpendidikan rendah saja (tapi student tidak special needs) masih bisa dihadapi oleh orang pada umumnya. Sedangkan yang mengalami keduanya seharusnya akan jauh lebih kesulitan.

3. Age-Related Risk

```
df['age_risk'] = ((df['Age at enrollment'] > 25).astype(int) *
                  (df['Marital status'] + 1))
```

Penjelasan:

- **Age threshold**: 25 years
  - < 25 = usia wajar (age_risk contribution = 0)
  - 25 = usia di atas wajar, kemungkinan memiliki tanggung jawab lain seperti pekerjaan
- **Status pernikahan**: (marital_status + 1)
  - Kombinasi usia yang lebih tinggi dengan status menikah/cerai mungkin akan mempengeruhi kemungkinan dropout karena beban tanggung jawab lebih besar

4. Displacement Burden

```
df['displacement_burden'] = df['Displaced'] * df['Debtor']
```

Penjelasan:

- Binary (0 atau 1):
  - 1 = Displaced (merantau atau jauh dari rumah) DAN Debtor (memiliki utang)
  - 0 = Tinggal di rumah atau tidak memiliki utang
- **Alasan**: Merantau memerlukan biaya tempat tinggal dan biaya hidup
- Jika ditambah utang maka akan sangat sulit hidup tenang, harus bekerja untuk bertahan hidup

5. First-Generation Status

```
df['first_generation'] = (
    (df["Mother's qualification"] <= 2).astype(int) *
    (df["Father's qualification"] <= 2).astype(int)
)
```

Penjelasan:

- Binary (0 atau 1):
  - 1 = First-generation (kedua orang tua secondary education), mungkin akan lebih banyak tantangan
  - 0 = Minimal salah satu orang tua berpendidikan tersier

4.2.4 Interaction Features (7 fitur)

Interaction features merupakan kombinasi dari risk factors. **Dropout mungkin bukan disebabkan oleh satu faktor, tapi kombinasi faktor yang saling memperkuat**.

Decision tree dapat menemukan interaction patterns dari split sequences, tapi explicit interaction features mungkin membantu tree lebih cepat menemukan patterns.

1. Academic Œ Economic Stress

```
# Students yang akademiknya kurang dan situasi finansialnya tidak baik
df['academic_economic_stress'] = df['academic_risk_score'] * \
                                 df['financial_burden']
```

Alasan perkalian:

- Jika hanya akademik yang bermasalah = dapat diperbaiki dengan dukungan
- Masalah finansial saja = masih dapat dihadapi jika akademik bagus (mungkin sambil bekerja)
- Akademik dan ekonomi bermasalah = sangat berisiko

2. Age Œ Performance Risk

```
df['age_performance_risk'] = (df['Age at enrollment'] *
                              (1 - df['cumulative_pass_rate'] + eps))
```

Penjelasan:

- Usia dikalikan dengan tingkat kegagalan akademik (1 - pass_rate)
- **Linear age factor**: semakin tua usia maka akan makin terasa dampaknya jika performa akademiknya tidak baik
- Orang yang lebih tua memiliki waktu yang lebih sedikit (karena tanggung jawab dan lain hal)
  - Tidak bisa menghabiskan banyak waktu mengulang dan mungkin cenderung meninggalkan daripada mencoba lagi

3. Grade Œ Financial Stress

```
df['grade_financial_stress'] = ((20 - df['avg_grade']) * \
                                df['financial_burden'])
```

Alasan:

- Jika hanya salah satu yang bermasalah maka dapat ditemukan solusinya
- Namun jika keduanya baik akademik dan finansial bermasalah maka akan sulit untuk bisa lulus dan cenderung dapat dropout

4. Special Needs Œ Performance

```
df['special_needs_performance'] = (
    df['Educational special needs'] *
    (1 - df['cumulative_pass_rate'] + eps)
)
```

Penjelasan:

- Student dengan special need yang pass rate nya rendah, mungkin karena kekurangan support atau tantangan lainnya

5. First-Generation Œ Academic Risk

```
df['first_gen_academic_risk'] = (df['first_generation'] * \
                                 df['academic_risk_score'])
```

Why Critical:

- First-generation tapi grades bagus = keadaannya baik meskipun ada tantangan
- First-generation dan masalah akademik = tidak bisa mendapatkan panduan dari orang tua

6. Displacement Œ Academic Risk

```
df['displacement_academic_risk'] = (df['Displaced'] * \
                                    df['academic_risk_score'])
```

Alasan:

- Displaced (merantau) tapi akademiknya bagus maka tidak apa-apa
- Akademik bermasalah tapi tinggal di rumah, mungkin bisa mencari bantuan dari orang tua atau keluarga
- Merantau dan akademik bermasalah maka akan kekurangan dukungan dan motivasi, lebih rentan dropout

7. Age Œ Debt Burden

```
df['age_debt_burden'] = ((df['Age at enrollment'] > 25).astype(int) * \
                         df['Debtor'])
```

Penjelasan:

- Binary: 1 jika age>25 DAN memiliki utang, 0 jka sebaliknya
- Jika memiliki utang tetapi masih muda mungkin masih bisa survive dan lebih fokus studi
- Jika sudah jauh lebih tua dan memiliki utang maka tanggung jawab yang harus dipegang jauh lebih banyak dan kurang fokus dengan studi, mungkin meningkatkan probabilitas dropout

4.2.5 Enrolled-Specific Features (8 fitur)

Enrolled-specific features dirancang khusus untuk membedakan kelas Enrolled, yang cukup minoritas jika dibandingkan kedua kelas lainnya (18% dari data), yaitu dari Dropout dan Graduate. Dengan melakukan ini, diharapkan kelas *middle ground* ini dapat diprediksi dengan lebih baik.

1. Moderate Performance (40-80% pass rate)

```
# Enrolled students biasanya moderate performance
# Tidak gagal (>40%), tidak terlalu baik (<80%)
df['moderate_performance'] = (
    ((df['cumulative_pass_rate'] >= 0.4) &
     (df['cumulative_pass_rate'] <= 0.8)).astype(int)
)
```

Penjelasan:

- Enrolled students = mampu pass 40%+ (bukan student yang gagal)
- Tapi konsisten <80% jadi belum sukses

2. Improving Student (Grade Improvement)

```
# Struggling in 1st sem, recovered in 2nd sem
df['improving_student'] = (
    (df['grade_improvement'] >= -1).astype(int) *  # Not too declining
    (df['pass_rate_2nd'] >= df['pass_rate_1st'] - 0.1).astype(int)  # Stable/improving
)
```

Penjelasan:

- Dua kondisi harus dipenuhi
  - grade_improvement -1 (tidak turun lebih dari 1 poin)
  - pass_rate_2nd pass_rate_1st - 0.1 (2nd sem pass rate tidak berkurang lebih dari 10
- Binary output (0 atau 1)
- Biasanya student yang masih enrolled performanya improving

3. Moderate Academic Risk (Score 1-4)

```
# Enrolled students mungkin punya beberapa academic problems tapi tieadk fatal
df['moderate_academic_risk'] = (
    ((df['academic_risk_score'] >= 1) &
     (df['academic_risk_score'] <= 4)).astype(int)
)
```

Penjelasan:

- Enrolled student biasanya mungkin mempunyai academic problems tetapi masih dalam batas terkkontrol

4. Has Financial Support (Protective)

```
# Enrolled students yang memmiliki sedikit masalah dapat bertahan jika memiliki financial
↪  support
df['has_financial_support'] = (
    (df['Scholarship holder'] == 1) |
    (df['Tuition fees up to date'] == 1)
).astype(int)
```

Alasan:

- Enrolled students yang bermasalah bisa dropout
- Namun, jika memiliki support maka dapat bertahan
- Tanpa support, harus bekerja untuk bertahan, stress bisa mengarahkan ke dropout

5. Inconsistent Performance (Volatile)

```
# Enrolled students mungkin biasanya memiliki performa yang tidak konsistem
# Baik di satu semester tapi buruk di semester lainnya
df['inconsistent_performance'] = (
    (abs(df['pass_rate_1st'] - df['pass_rate_2nd']) > 0.3).astype(int)
)
```

Why Enrolled Indicator:

- Graduate: consistent high performance
- Dropout: consistently low or declining
- Enrolled: mencoba berbagai strategi untuk bertahan dan meningkatkan performa

6. At-Risk But Persisting (Combined Signal)

```
df['at_risk_but_persisting'] = (
    df['moderate_academic_risk'] * df['has_financial_support']
)
```

Penjelasan:

- Binary flag
- 1 = ada masalah akademik level sedang DAN memiliki financial support
- 0 = hanya memiliki salah satu dari masalah akademik atau financial support atau tidak keduanya

Alasan:

- Dropout: kekurangan support, terpaksa harus keluar karena risiko terlalu tinggi
- Graduate: idealnya tidak memiliki moderate risk
- Enrolled: di posisi ini

7. Recovery Pattern (Bouncing Back)

```
# Students gaal di 1st semester tapi memperbaiki di 2nd
df['recovery_pattern'] = (
    (df['failed_1st'] >= 1).astype(int) *
    (df['grade_improvement'] > 0).astype(int) *
    (df['pass_rate_2nd'] > df['pass_rate_1st']).astype(int)
)
```

Penjelasan:

- Ketiga kondisi harus terpenuhi
  - Gagal minimal 1 mata kuliah di semester 1
  - Grade improved ke semester 2
  - Pass rate naik dari semester 1 ke semester 2
- Binary: 1 jika ketiganya terpenuhi, 0 jika tidak

8. Enrolled Middle-Ground Indicator (Summary Signal)

```
# Moderate performance + ada support + moderate risk = "middle ground"
df['enrolled_middle_ground'] = (
    df['moderate_performance'] *
    df['has_financial_support'] *
    ((df['academic_risk_score'] <= 5).astype(int))
)
```

Penjelasan:

- Ketiga kondisi terpenuhi (perkalian):
  - moderate_performance = 1 (pass rate 40-80%)
  - has_financial_support = 1 (scholarship/beasiswa atau tuition/UKT sudah dibayar)
  - academic_risk_score 5 (tidak kritis/fatal)
- Karakteristik mahasiswa enrolled

4.3. Tahapan Preprocessing

4.3.1 Data Alignment

Data alignment memastikan data training set dan test set memiliki features yang identik dalam urutan yang sama.

```
# Apply feature engineering ke kedua set
train_df_engineered = engineer_all_features(train_df_raw)
test_df_engineered = engineer_all_features(test_df_raw)

# Encode target/training set
target_map = {'Dropout': 0, 'Enrolled': 1, 'Graduate': 2}
train_df_engineered['Target'] = \
    train_df_engineered['Target'].map(target_map)

X_full = train_df_engineered.drop(
    ['Target', 'Student_ID'], axis=1, errors='ignore'
)
y_full = train_df_engineered['Target'].values

X_test = test_df_engineered.drop('Student_ID', axis=1, errors='ignore')

# Pqastikan kolom sama
common_features = X_full.columns.intersection(X_test.columns)
X_full = X_full[common_features]
X_test = X_test[common_features]

print(f"Features: {len(common_features)}")
print(f"Train shape: {X_full.shape}")  # (3096, 72)
print(f"Test shape: {X_test.shape}")   # (1328, 72)
```

Tujuan:

1. **Feature Intersection**: Memastikan training dan test set memiliki features yang sama (tanpa Student_ID dan Target yang hanya di train)
2. **Column Ordering**: Urutan feature konsisten untuk model compatibility
3. **Tanpa**: Test set features tidak mempengaruhi train preprocessing

Pipeline

4.3.2 Categorical Encoding

Categorical encoding mengkonversi kategorikal features menjadi numerical format yang dapat digunakan oleh machine learning models.

Encoding Strategy di tugas ini

Tidak ada explicit categorical encoding diperlukan pada fitur-fitur.

Alasan:

1. **Semua 72 features sudah numerik**:

- 36 original features dari Kaggle: semua int64 atau float64
- 36 engineered features: hasil dari arithmetic operations, semua numerik
- Target (Dropout/Enrolled/Graduate) di-encode ke (0/1/2) secara explicit

2. **Tidask ada categorical features dalam dataset**:
   - Boolean features (Debtor, Displaced, etc.) sudah 0/1 encoded
   - Ordinal features (Age, Education level) sudah numerik
   - Tidak ada nominal feature yang perlu one-hot encoding
3. Decision Tree meng-*handle* data numerik secara otomatis dengan menemnukan optimal threshold.

Target Encoding

Hanya kolom Target yang memerlukan explicit encoding:

```
target_map = {
    'Dropout': 0,      # Class 0: negative (gagal)
    'Enrolled': 1,     # Class 1: positive (bertahan)
    'Graduate': 2      # Class 2: positive (sukses)
}

train_df_engineered['Target'] = \
    train_df_engineered['Target'].map(target_map)
```

4.3.3 Numerical Scaling

Numerical scaling mentransformasi fitur-fitur ke skala yang konsisten. Ini vital untuk model tertentu (SVM, Logistic Regression).

RobustScaler Implementation

```
from sklearn.preprocessing import RobustScaler

# Initialize scaler
scaler = RobustScaler()

# Fit on training data
X_full_scaled = scaler.fit_transform(X_full)

# Transform test data
X_test_scaled = scaler.transform(X_test)

# Convert kembali ke DataFrame
X_full_scaled_df = pd.DataFrame(
    X_full_scaled,
    columns=common_features,
    index=X_full.index
)
X_test_scaled_df = pd.DataFrame(
    X_test_scaled,
    columns=common_features,
    index=X_test.index
)

print(f"Feature Scaling Complete")
print(f"Scaler: RobustScaler")
print(f"Train scaled: {X_full_scaled_df.shape}")
print(f"Test scaled: {X_test_scaled_df.shape}")
```

RobustScaler: Formula dan Rationale

RobustScaler menggunakan Interquartile Range (IQR) untuk scaling:

$$X_{\text{scaled}} = \frac{X - Q_2}{Q_3 - Q_1} \tag{4.4}$$

di mana:

- Q_1 = 25th percentile (1st quartile)
- Q_2 = 50th percentile (median)
- Q_3 = 75th percentile (3rd quartile)
- $IQR = Q_3 - Q_1$ (Interquartile Range)

Kelebihan vs StandardScaler:

Tabel 4.2: RobustScaler vs StandardScaler

Aspek	**RobustScaler**	**StandardScaler**
Formula	$\frac{X-Q_2}{IQR}$	$\frac{X-\mu}{\sigma}$
Sensitivitas ke Outliers	**Low** (dengan IQR)	High (memakai std)
Asumisi Dist. Normal	No	Yes
Scale Range	Approximately [-2, 2]	Typically [-3, 3]

Justifiksai:
Student data sering mengandung natural outliers:

- **Age**: Usia sangat tua minoritas tapi valid asal dalam batas wajar umur manusia
- **Grades**: Sempurna (20/20) atau gagal (0/0) pasti bisa terjadi, bukan error
- **Unemployment**: Krisis ekonomi menyebabkan keadaan ekstrim

RobustScaler:

- Tidak di-influence oleh extreme values (memakai median dan IQR, bukan mean dan std)
- Menyimpan natural outliers (tidak dihilangkan maupun diubah)
- Lebih cocok dengan sata students yang memang ada extreme values

Impact terhadap Model Performance

Percobaan menunjukkan RobustScaler berpengaruh ke performa akhir, dengan perbandingan terhadap StandardScaler menghasilkan improvement yang terlihat.

4.3.4 Data Format Conversion

Data dikonversi ke format yang sesuai untuk C4.5 model training.

```
# After scaling, convert to DataFrame (untuk C4.5
X_full_scaled_df = pd.DataFrame(
    X_full_scaled,
    columns=common_features
)
```

```
# Memepertahankan feature names
# Feature names dipakai di tree building untuk readable split rules

# Target sebagai numpy array
y_full = train_df_engineered['Target'].values

# Format untuk C4.5:
c45_model = C45Classifier(...)
c45_model.fit(X_full_scaled_df, y_full)

# Prediction:
y_pred = c45_model.predict(X_test_scaled_df)
```

Format untuk C4.5:

1. **Features**: Pandas DataFrame dengan named columns
   - C4.5 perlu feature names untuk split rules
   - Misal split: "pass_rate_1st <= 0.65"
2. **Target**: NumPy array atau list dengan class labels
   - Numeric labels (0, 1, 2) untuk multi-class
3. **Data Types**: Semua numerical (float64 lebih baik)
   - Tanpa string atau mixed types

4.3.5 Data Balancing

Setelah dilakukan pembagian data dan scaling, dilakukan analisis terhadap distribusi kelas pada dataset training. Ditemukan bahwa dataset mengalami *class imbalance* yang signifikan, kelas *Enrolled* hanya mencakup 17.9% dari total data, sementara kelas *Dropout* dan *Graduate* mendominasi dataset. Ketidakseimbangan ini dapat menyebabkan model cenderung bias terhadap kelas mayoritas dan mengabaikan kelas minoritas.

Implementasi SMOTE

Untuk mengatasi masalah *class imbalance*, digunakan teknik *Synthetic Minority Over-sampling Technique* (SMOTE). SMOTE bekerja dengan cara membangkitkan sampel sintetis untuk kelas minoritas melalui interpolasi linear antara sampel yang ada dengan tetangga terdekatnya dalam ruang fitur. Algoritma SMOTE dapat dijabarkan sebagai berikut:

1. Untuk setiap sampel $\mathbf{x}_i$ dari kelas minoritas, temukan k tetangga terdekat (*k-nearest neighbors*)
2. Pilih secara acak salah satu tetangga terdekat $\mathbf{x}_{zi}$
3. Bangkitkan sampel sintetis $\mathbf{x}_{new}$ menggunakan interpolasi linear:

$$\mathbf{x}_{new} = \mathbf{x}_i + \lambda \cdot (\mathbf{x}_{zi} - \mathbf{x}_i) \tag{4.5}$$

   di mana $\lambda \in [0, 1]$ adalah nilai acak yang menentukan posisi sampel baru di antara $\mathbf{x}_i$ dan $\mathbf{x}_{zi}$

Implementasi SMOTE ini menggunakan library `imblearn` dengan parameter default ($k = 5$ tetangga terdekat). SMOTE diterapkan **hanya pada data training** untuk menghindari *data leakage*, sedangkan data validasi dan testing tetap pada distribusi aslinya untuk evaluasi yang objektif.

Hasil Data Balancing

Tabel 4.3 menunjukkan perbandingan distribusi kelas sebelum dan sesudah penerapan SMOTE:

Tabel 4.3: Distribusi Kelas Sebelum dan Sesudah SMOTE

Kelas	Sebelum SMOTE	Sesudah SMOTE	Persentase
Dropout	1,236	1,236	33.3%
Enrolled	444	1,236	33.3%
Graduate	796	1,236	33.3%
Total	**2,476**	**3,708**	**100%**

Dari Tabel 4.3, dapat diamati bahwa:

- Jumlah total sampel training meningkat dari 2,476 menjadi 3,708 sampel (+49.8%)
- Distribusi kelas menjadi seimbang sempurna dengan rasio 1:1:1 (33.3% untuk setiap kelas)
- Kelas *Enrolled* yang awalnya hanya 17.9% meningkat menjadi 33.3%
- Kelas *Dropout* dan *Graduate* tetap pada jumlah aslinya karena sudah merupakan kelas mayoritas

4.4. Alasan Pemilihan Strategi

Keputusan preprocessing dalam proyek ini didasarkan pada kombinasi domain knowledge (meski belum terlalu berpengalaman di domain tersebut, tetapi karena kami sendiri mahasiswa jadi sedikit paham dengan karakteristik dataset), validasi dari eksperimen, dan keperluan algoritma.

4.4.1 RobustScaler Selection

Alasan pemilihan RobustScaler?

1. **Karakteristik data students**:
   - Sudah dijelaskan dengan detail di bagian sebelumnya beberapa kali, intinya adalah adanya data outliers yang memang *natural* dan *expected* di dataset students
2. **Median dan IQR lebih robust**:
   - Mean dan std sangat dipengaruhi oleh outliers
   - Median dan IQR tetap stabil dengan extreme values
   - Informas dari outlier tetap disimpan (tidak dibuang)
3. **Empirical evidence**:
   - Uji pipeline dengan RobustScaler mendaptkan 72.83% macro-F1
   - Lebih baik dibandingkan standard scaler

4.4.2 Feature Engineering Impact

Alasan

Model yang di-*train* degnan original 36 features tidak bisa mencapai 72%+ macro-F1. Feature engineering menambah performa prediksi melalui:

1. **Domain-driven design**: Setiap feature yang di-*engineer* didukung alasan domain knowledge (bagaimana ekonomi mungkin memengaruhi performa akademik, dsb.)
   - Academic: grades dan tingkat kelulusan mata kuliah tentunya memengaruhi kelulusan atau dropout
   - Economic: financial stress terbukti mendorong kemungkinan dropout
   - Interaction: risiko yang saling digabungkan > risiko individual
2. **Class imbalance**: Features milik class enrolled adalah bagian dari minoritas di datase
   - Features bawaan tidak cukup untuk membedakan Droput dari Enrolled
   - Features seperti "improving_student" dan "recovery_pattern" mungkin membantu menemukan pola Enrolled student
3. **Non-linear relationships**: Interaction features (fitur yang digunakan bersamaan) mungkin lebih berpengaruh daripada individual
   - Misal: academic_risk + financial_burden mungkin pengaruhnya lebih signifikan ketika digabung daripada masing-masing

4.4.3 Pipeline Consistency

Perluknya konsistensi:

1. **Reproducibility**: Preprocessing yang sama untuk train dan test memastikan:
   - Perilaku model yang sama di local validation dan Kaggle submission
   - Random yang reproducible karena menggunakan seeds untuk RNG
   - Hasilnya akan deterministik untuk memastikan submission Kaggle benar-benar diproduksi dari model yang sama

BAB V
Perbandingan Hasil dan Analisis

5.1. Setup Eksperimen

5.1.1 Dataset dan Pembagian Data

Eksperimen dilakukan menggunakan dataset *Student Dropout Prediction* dari kompetisi Kaggle dengan pembagian sebagai berikut:

Dataset Awal

- **Total training data**: 3.096 samples
- **Total test data**: 1.328 samples
- **Jumlah features**: 36 features original
- **Target classes**: 3 kelas (Dropout, Enrolled, Graduate)

Distribusi Kelas (Class Distribution)

Dataset memiliki **class imbalance** yang signifikan:

Kelas	Jumlah	Persentase
Graduate	1.545	49.9%
Dropout	995	32.1%
Enrolled	556	17.9%
Total	3.096	100%

Tabel 5.1: Distribusi kelas pada dataset training

Catatan: Kelas *Enrolled* merupakan minority class yang berpotensi menyebabkan bias prediksi.

Stratified Train-Validation Split

Untuk evaluasi model, training data dibagi menggunakan **stratified split** (80:20):

Set	Total	Dropout	Enrolled	Graduate
Training	2.476 (80%)	795 (32.1%)	445 (18.0%)	1.236 (49.9%)
Validation	620 (20%)	200 (32.3%)	111 (17.9%)	309 (49.8%)

Tabel 5.2: Pembagian training-validation dengan stratified sampling

Stratified sampling memastikan proporsi kelas pada training dan validation tetap sama dengan dataset asli.

Data Setelah Preprocessing

Setelah melalui tahap preprocessing (dijelaskan di bab tersendiri), data memiliki karakteristik:

Aspek	Nilai
Validation samples	620 samples (tetap original, no SMOTE)
Training samples setelah SMOTE	3.708 samples (balanced 33.3% per kelas)
Features setelah selection	39 features (pengurangan 45.8%)
Features setelah engineering	72 features (36 original + 36 engineered)

Tabel 5.3: Karakteristik data setelah preprocessing

Catatan penting:

- SMOTE diterapkan hanya pada training set untuk mencegah data leakage
- Validation set tetap menggunakan data original untuk evaluasi yang fair
- Feature selection menggunakan consensus voting dari 3 metode filter

5.1.2 Hyperparameter Konfigurasi

Berikut adalah konfigurasi hyperparameter untuk setiap model yang diuji:

Decision Tree Learning (DTL)

Model	Implementasi	max_depth	min_samples_split	Criterion	ccp_alpha
C4.5	Custom	15	5	Gain Ratio	0.0
	Sklearn	-	-	Entropy	0.019
ID3	Custom	15	10	Info Gain	-
	Sklearn	-	-	Entropy	-
CART	Custom	15	10	Gini	-
	Sklearn	-	-	Gini	-

Tabel 5.4: Hyperparameter Decision Tree Learning

Logistic Regression

Implementasi	Learning Rate	Iterations	Regularization	Batch Size	LR Decay
Custom	0.065	6000	L2 (0.005)	128	0.89
Sklearn	-	6000	L2 (C=1.0)	-	-

Tabel 5.5: Hyperparameter Logistic Regression

Support Vector Machine (SVM)

Strategy	Implementasi	Learning Rate	C	Iterations	Kernel	Class Weight
One-vs-All	Custom	0.001	2.0	1500	Linear	Balanced
	Sklearn	-	1.0	-	Linear	-
One-vs-One	Custom	0.001	2.0	1500	Linear	Balanced
	Sklearn	-	1.0	-	Linear	-
DAG	Custom	0.001	2.0	1500	Linear	Balanced
RBF	Sklearn	-	1.0	-	RBF	-

Tabel 5.6: Hyperparameter Support Vector Machine

Perbedaan Utama:

- **Custom SVM**: Gradient descent dengan hinge loss, tuned `C=2.0`
- **Sklearn SVM**: SMO/LIBSVM optimizer, default `C=1.0`
- **RBF Kernel**: Hanya tersedia di sklearn (non-linear kernel)

5.1.3 Software dan Pustaka

Kategori	Library/Tool	Versi
Data Processing	NumPy	1.24+
Data Processing	Pandas	2.0+
Visualization	Matplotlib	3.7+
Visualization	Seaborn	0.12+
Machine Learning	scikit-learn	1.3+
SMOTE	imbalanced-learn	0.11+
Tree Visualization	Graphviz	0.20+
Development Environment	Python	3.10+

Tabel 5.7: Software dan pustaka yang digunakan

5.2. Perbandingan Model From Scratch vs Pustaka

5.2.1 Tabel Perbandingan

Berikut adalah hasil perbandingan F1-Score (macro average) antara implementasi *from scratch* dan implementasi pustaka `scikit-learn`:

Model	Custom	Scikit-learn	Selisih
Logistic Regression	**0.7244**	0.7134	+0.0110
SVM OvO	0.7206	0.7083	+0.0123
SVM DAG	0.7206	0.7215	-0.009
SVM OvA	0.7188	0.7083	+0.0105
CART	0.6414	0.6360	+0.0054
ID3	0.6358	0.6156	+0.0202
C4.5	0.6204	0.7134	-0.0930

Tabel 5.8: Perbandingan F1-Score: Custom vs Scikit-learn (Validation Set)

5.2.2 Ranking Model Berdasarkan F1-Score

Rank	Model	F1-Score
1	**LogReg (Custom)**	**0.7244**
2	SVM RBF (Sklearn)	0.7215
3	SVM OvO (Custom)	0.7206
4	SVM DAG (Custom)	0.7206
5	SVM OvA (Custom)	0.7188
6	C4.5 (Sklearn)	0.7134
7	LogReg (Sklearn)	0.7134
8	SVM OvA (Sklearn)	0.7083
9	SVM OvO (Sklearn)	0.7083
10	CART (Custom)	0.6414
11	CART (Sklearn)	0.6360
12	ID3 (Custom)	0.6358
13	C4.5 (Custom)	0.6204
14	ID3/C4.5 (Sklearn)	0.6156

Tabel 5.9: Ranking model berdasarkan F1-Score (descending)

5.2.3 Perbandingan per Kategori Algoritma

Decision Tree Learning

Algoritma	Custom F1	Sklearn F1
C4.5	0.6204	0.7134 (+14.99%)
ID3	0.6358	0.6156 (-3.18%)
CART	0.6414	0.6360 (-0.84%)
Rata-rata	0.6325	0.6550

Tabel 5.10: Perbandingan DTL: Custom vs Sklearn

Logistic Regression

Implementasi	F1-Score
Custom	**0.7244**
Sklearn	0.7134
Selisih	**+0.0110 (+1.54%)**

Tabel 5.11: Perbandingan Logistic Regression

Support Vector Machine

Strategy	Custom F1	Sklearn F1
One-vs-All (OvA)	0.7188	0.7083 (+1.48%)
One-vs-One (OvO)	0.7206	0.7083 (+1.74%)
DAG	0.7206	0.7215
Rata-rata	**0.7200**	0.7127

Tabel 5.12: Perbandingan SVM: Custom vs Sklearn

5.3. Metrics yang Digunakan

5.3.1 Metrics Utama: Macro F1-Score

Formula:

$$F1_{\text{macro}} = \frac{1}{K} \sum_{i=1}^{K} F1_i$$

dimana:

$$F1_i = 2 \cdot \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Alasan Pemilihan:

1. **Class Imbalance**: Dataset memiliki imbalance (Graduate 49.9%, Dropout 32.1%, Enrolled 17.9%)
2. **Equal Class Importance**: Macro averaging memberikan bobot yang sama untuk setiap kelas sehingga minority class (Enrolled) tidak terabaikan
3. **Harmonic Mean**: F1-Score adalah harmonic mean dari precision dan recall, sensitif terhadap performa yang buruk di salah satu metric
4. **Kaggle Metric**: Kompetisi menggunakan Macro F1-Score sebagai evaluation metric

5.3.2 Metrics Pendukung

Accuracy

Formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Penggunaan:

- Memberikan overview performa keseluruhan
- **Limitation**: Bias terhadap majority class (Graduate) pada imbalanced dataset

Precision (per class)

Formula:

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}$$

Interpretasi:

- Dari semua prediksi kelas i, berapa persen yang benar?
- Penting untuk menghindari *false positives*

Recall (per class)

Formula:

$$\text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}$$

Interpretasi:

- Dari semua data aktual kelas i, berapa persen yang berhasil diprediksi?
- Penting untuk menghindari *false negatives*

Confusion Matrix

Memberikan detail breakdown prediksi untuk setiap kelas:

True Label	**Predicted Label**		
	Dropout	Enrolled	Graduate
Dropout	$C_{0,0}$	$C_{0,1}$	$C_{0,2}$
Enrolled	$C_{1,0}$	$C_{1,1}$	$C_{1,2}$
Graduate	$C_{2,0}$	$C_{2,1}$	$C_{2,2}$

Tabel 5.13: Struktur Confusion Matrix

Kegunaan:

- Identifikasi kelas mana yang sering salah diprediksi
- Analisis pola kesalahan

5.4. Insight dari Perbandingan

Berdasarkan hasil dari Tabel Perbandingan , diperoleh beberapa insight penting mengenai perbedaan performa antara implementasi *from-scratch* dan pustaka:

5.4.1 Perbandingan Berdasarkan Algoritma

Decision Tree Learning (DTL)

Implementasi pustaka sklearn menunjukkan performa yang jauh lebih baik dibandingkan implementasi custom, terutama pada algoritma C4.5 dengan selisih F1-Score sebesar 14.99%. Hal ini disebabkan oleh:

- **Pruning**: Sklearn menggunakan cost-complexity pruning (`ccp_alpha`) yang lebih efektif dalam mencegah overfitting
- **Optimisasi**: Implementasi pustaka menggunakan algoritma yang lebih optimal dalam pemilihan split dan penanganan missing values
- **Numerical Stability**: Sklearn lebih robust dalam menangani numerical issues pada perhitungan entropy dan information gain

Logistic Regression

Implementasi custom justru sedikit lebih unggul dengan selisih F1-Score sebesar 1.54%. Keunggulan ini berasal dari:

- **Hyperparameter Tuning**: Custom implementation menggunakan learning rate yang lebih tinggi (0.065 vs 1.0/C) dengan decay schedule yang lebih agresif
- **Optimization Strategy**: Batch size 128 dengan momentum dan Xavier initialization memberikan konvergensi yang lebih baik untuk dataset ini
- **Class Weighting**: Implementasi manual class balancing lebih sesuai dengan karakteristik dataset yang telah di-SMOTE

Support Vector Machine (SVM)

Implementasi custom menunjukkan performa yang lebih baik pada multiclass strategy (OvA, OvO, DAG) dengan selisih 1.5-1.7%. Namun, sklearn dengan kernel RBF memberikan hasil kompetitif (F1=0.7215). Perbedaan ini menunjukkan:

- **Linear vs Non-linear**: Custom implementation menggunakan linear kernel yang lebih sederhana namun cukup efektif untuk data yang sudah well-preprocessed
- **Regularization**: Parameter C=2.0 pada custom implementation lebih agresif dibanding default sklearn (C=1.0)
- **Kernel Choice**: RBF kernel pada sklearn mampu menangkap non-linear patterns namun membutuhkan tuning gamma yang lebih teliti

5.4.2 Model Terbaik

Model dengan performa terbaik adalah **Logistic Regression (Custom)** dengan F1-Score makro sebesar **0.7244**. Keunggulan Logistic Regression dibandingkan model lainnya dapat dijelaskan sebagai berikut:

Mengapa Logistic Regression Unggul?

1. **Linear Separability**: Setelah feature engineering dan SMOTE, dataset student dropout memiliki decision boundary yang relatif linear. Logistic Regression sangat efektif untuk kasus seperti ini karena mampu menemukan hyperplane optimal tanpa kompleksitas berlebih yang dapat menyebabkan overfitting.
2. **Balanced Complexity**: Dibandingkan dengan Decision Tree yang cenderung overfit pada noise (terutama implementasi custom tanpa pruning optimal), Logistic Regression memiliki regularization inherent yang membuat model lebih generalize.
3. **Multiclass Handling**: Implementasi One-vs-Rest pada Logistic Regression cocok dengan karakteristik 3 kelas yang balanced (setelah SMOTE). Model ini mampu membedakan setiap kelas dengan confidence score yang jelas.
4. **Optimization Convergence**: Dengan 6000 iterasi dan learning rate decay, model custom mampu mencapai konvergensi yang lebih baik dibanding sklearn yang menggunakan solver default. Xavier initialization juga membantu menghindari local minima.
5. **Feature Importance**: Dari 39 fitur terpilih, banyak yang memiliki hubungan linear/monoton dengan target (misalnya academic performance, curricular units). Logistic Regression memanfaatkan ini dengan baik melalui weight learning.

5.4.3 Trade-off Custom vs Pustaka

Dari perbandingan ini, dapat disimpulkan trade-off berikut:

Tabel 5.14: Trade-off Implementasi Custom vs Pustaka

Aspek	Custom Implementation	Library (sklearn)
Performa	Lebih baik untuk LogReg dan SVM jika hyperparameter sudah optimal	Lebih baik untuk DTL karena optimisasi internal
Kontrol	Penuh terhadap setiap aspek algoritma	Terbatas pada hyperparameter yang disediakan
Kompleksitas	Membutuhkan pemahaman mendalam dan debugging manual	Sudah teruji dan robust
Waktu Development	Lebih lama (implementasi + debugging)	Lebih cepat (tinggal tuning)
Skalabilitas	Terbatas untuk dataset besar	Sangat scalable

5.4.4 Kesimpulan

Berdasarkan analisis yang telah dilakukan, dapat disimpulkan bahwa:

1. Model Terbaik

Logistic Regression (Custom) dengan F1-Score 0.7244 menjadi model terbaik karena:

- Mampu menangkap linear patterns dalam data yang telah di-preprocess dengan baik
- Hyperparameter tuning yang optimal (lr=0.065, decay=0.89, l2=0.005)
- Balanced performance across all three classes (Dropout, Enrolled, Graduate)
- Robust terhadap overfitting melalui L2 regularization dan batch training

2. Pola Umum Custom vs Sklearn

Terdapat pola yang konsisten dalam perbandingan:

- **DTL**: Sklearn unggul signifikan (14-15% gap) karena pruning yang lebih sophisticated
- **LogReg**: Custom unggul tipis (1.5% gap) karena tuning hyperparameter yang lebih detail
- **SVM**: Custom unggul untuk linear kernel (1.5-1.7% gap), sklearn kompetitif dengan RBF kernel

3. Dampak Preprocessing terhadap Model Performance

SMOTE dan feature selection memiliki dampak yang berbeda pada setiap model:

- **SMOTE**: Meningkatkan performa semua model dengan mengatasi class imbalance, terutama untuk kelas Enrolled (minority class)
- **Feature Selection**: Reduksi dari 72 ke 39 fitur mengurangi overfitting pada DTL, namun tidak berdampak signifikan pada LogReg/SVM yang sudah memiliki regularization
- **Scaling (RobustScaler)**: Kritikal untuk LogReg dan SVM yang sensitif terhadap feature magnitude, namun kurang penting untuk DTL

DAFTAR PUSTAKA

[1] Rabelo, A.M., & Zárate, L.E. (2025). *A model for predicting dropout of higher education students*. Data Science and Management, **8**(1), 72–85. https://doi.org/10.1016/j.dsm.2024.07.001

[2] Realinho, V., Machado, J., Baptista, L., & Martins, M.V. (2022). *Predicting Student Dropout and Academic Success*. Data, **7**(11), 146. https://doi.org/10.3390/data7110146

[3] GeeksforGeeks. *Iterative Dichotomiser 3 (ID3) Algorithm from Scratch*. Diakses 5 Desember 2025. https://www.geeksforgeeks.org/machine-learning/iterative-dichotomiser-3-id3-algorithm-from-scratch/

[4] GeeksforGeeks. *CART (Classification and Regression Tree) in Machine Learning*. Diakses 5 Desember 2025. https://www.geeksforgeeks.org/machine-learning/cart-classification-and-regression-tree-in-machine-learning/

[5] GeeksforGeeks. *Decision Tree Algorithms*. Diakses 5 Desember 2025. https://www.geeksforgeeks.org/machine-learning/decision-tree-algorithms/

[6] GeeksforGeeks. *Support Vector Machine Algorithm*. Diakses 5 Desember 2025. https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/

[7] GeeksforGeeks. *Understanding Logistic Regression*. Diakses 5 Desember 2025. https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/

[8] Chawla, N.V., Bowyer, K.W., Hall, L.O., & Kegelmeyer, W.P. (2002). *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research, **16**, 321–357. https://doi.org/10.1613/jair.953

LAMPIRAN

A. Link Repository

Link Repo GitHub (https://github.com/SayyakuHajime/Akal-Imitasi)

B. Pembagian Tugas

Tabel B.1: Tabel Pembagian Tugas Kelompok

No	Nama	Pengerjaan Tugas
1	M Hazim R Prajoda (13523009)	• Implementasi DTL Scratch • Implementasi LogReg Scratch • Implementasi SVM Scratch • Melakukan Feature Engineering
2	Orvin Andika Ikhsan A (13523017)	• Laporan SVM • Laporan Logistic Regression
3	Fajar Kurniawan (13523027)	• Laporan Decision Tree Learning • Laporan Cleaning and Preprocessing
4	Darrel Adinarya Sunanda (13523061)	• SVM Model (Best) • Laporan Lampiran
5	Reza Ahmad Syarif (13523119)	• Preprocessing (Handling Class Imbalance dan Feature Selection) • Bonus Tree Visualizer