Tugas Besar 2 IF3170 Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Kelompok 34 Egiluylzinnn 🙏

Nicholas Andhika Lucas	13523014
Shanice Feodora Tjahjono	13523097
Jonathan Kenan Budianto	13523139
Mahesa Fadhillah Andre	13523140
Muhammad Farrel Wibowo	13523153

BANDUNG

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR ISI **2**
1. Implementasi dan Penjelasan **3**
1.1. Penjelasan Tahap Cleaning dan Preprocessing yang Dilakukan dan Alasan 3
1.1.1. Exploratory Data Analysis 3
1.1.2. Split Training Set and Validation Set 3
1.1.3. Data Cleaning: Dealing with Outliers 4
1.1.4. Data Cleaning: Feature Engineering 5
1.1.5. Data Preprocessing: Feature Scaling 8
1.1.6. Data Preprocessing: Feature Encoding 9
1.1.7. Data Preprocessing: Handling Imbalanced Dataset 10
1.1.8. Data Preprocessing: Data Normalization 10
1.1.9. Data Preprocessing: Dimensionality Reduction 10
1.1.10. Data Preprocessing Pipeline 10
1.2. Penjelasan Implementasi Decision Tree Learning 11
1.3. Penjelasan Implementasi Logistic Regression 19
1.4. Penjelasan Implementasi SVM 24
Alur Proses: 30
1. Memulai dengan daftar kandidat yang berisi seluruh kelas 30
2. Mengambil kelas "Paling Kiri" dan "Paling Kanan" dari daftar kandidat 30
3. Menggunakan model SVM spesifik yang terlatih untuk pasangan tersebut guna melakukan prediksi 30
4. Berdasarkan hasil prediksi (siapa yang menang), kelas yang kalah dieliminasi dari daftar kandidat 30
5. Proses berulang hingga hanya tersisa satu kelas, yang kemudian dikembalikan sebagai hasil prediksi 30
2. Hasil Prediksi Algoritma dan Perbandingan dengan Pustaka **31**
2.1. Hasil Pengujian Decision Tree Learning 32
2.2. Hasil Pengujian Logistic Regression 33
2.3. Hasil Pengujian SVM 34
2.4. Pembahasan dan Insight yang Didapatkan 35
3. Kesimpulan dan Saran **36**
3.1. Kesimpulan 36
3.2. Saran 36
4. Pembagian Tugas **37**
5. Referensi **38**

1. Implementasi dan Penjelasan

1.1. Penjelasan Tahap Cleaning dan Preprocessing yang Dilakukan dan Alasan

Data cleaning dilakukan untuk mempersiapkan dataset agar dapat digunakan untuk data modelling. Dataset sering masih dalam bentuk yang belum sempurna, baik itu karena ada missing values, kolom-kolom yang tidak relevan untuk prediksi target, ataupun data-data yang tidak masuk akal atau tidak mungkin. Dalam implementasi pada tugas besar ini, data cleaning melalui proses handling untuk outliers dan feature engineering untuk mengekstraksi informasi yang penting dari dataset.

Data preprocessing dilakukan untuk …

1.1.1. Exploratory Data Analysis

Sebelum memulai *data cleansing*, tahap ini dilakukan untuk menganalisis dan menemukan pola-pola dari dataset yang diberikan. Harapan yang ingin dicapai pada Exploratory Data Analysis ini adalah mencari tahu deskripsi dan statistik dari dataset, menemukan anomali pada dataset, serta menemukan insight baru atau hubungan dari visualisasi data. Hal ini diimplementasikan dengan kode seperti:

```
df_train.info()
print(df_train.describe())
print(df_train.isnull().sum())
df_train.duplicated().sum()
(df_train['Target'].value_counts(normalize=True) * 100
```

Berdasarkan exploratory data analysis yang dilakukan, ditemukan bahwa tidak ada anomali pada data (missing values, duplicated values, NaN value), sehingga **tahap handling missing data dan duplicates pada data cleaning dapat dilewati.** Kemudian, ditemukan juga insight dari distribusi kolom target, yaitu: Graduate sebanyak 1546 (49,93%); Dropout sebanyak 994 (32,1%); dan Enrolled sebanyak 556 (17,95%).

Untuk visualisasi data, … #### TO DO ####

1.1.2. Split Training Set and Validation Set

Kemudian, sebelum diproses pada data cleaning, dataset akan dipisahkan terlebih dahulu menjadi training set dan validation set. Model machine learning akan

dilatih menggunakan training set, kemudian hasil model tersebut akan divalidasi atau diuji menggunakan validation set. Untuk tahap ini, dilakukan splitting data dengan distribusi sebesar 80% untuk training set dan 20% untuk validation set. Splitting dilakukan dengan stratifikasi pada kolom Target, yaitu menjaga rasio distribusi dari kategori-kategori pada Target ketika dilakukan splitting pada dataset. Alasannya adalah untuk mempertahankan distribusi Target yang nyata pada dataset sebelum di-split.

Alasan pemilihan angka 80/20 tersebut adalah itu merupakan rasio distribusi yang standar dilakukan untuk dataset yang lebih kecil pada umumnya, yaitu sekitar 3000 rows pada dataset ini. Rasio yang umum digunakan lainnya adalah 70/30 dan dapat menjadi alternatif. Sedangkan, dataset dengan jumlah rows yang lebih banyak, misalnya mencapai jutaan data akan memberikan rasio yang jauh lebih kecil untuk validation set.

```
train_set, val_set = train_test_split(
    df_train,
    test_size=0.2,
    random_state=42,
    stratify=df_train['Target']
)
```

1.1.3. Data Cleaning: Dealing with Outliers

Outliers adalah value data-data yang berbeda jauh dari mayoritas/median dari dataset, yang dapat menandakan value tersebut tidak mungkin tercapai, atau sekedar tidak sesuai dengan pola dari dataset dan sebaiknya diproses – meskipun value tersebut mungkin ada – untuk mempertahankan pola dataset.

Awalnya, dilakukan analisis deteksi outlier menggunakan metode IQR yang menentukan outlier berdasarkan upper bound dan lower bound menggunakan nilai rentang antar kuartil Q1 dan Q3. Hasilnya adalah sebagai berikut:

```
Column Previous qualification (grade): 95 outliers (3.84%)
Column Admission grade: 46 outliers (1.86%)
Column Age at enrollment: 251 outliers (10.14%)
Column Curricular units 1st sem (credited): 329 outliers (13.29%)
Column Curricular units 1st sem (enrolled): 246 outliers (9.94%)
Column Curricular units 1st sem (evaluations): 87 outliers (3.51%)
Column Curricular units 1st sem (approved): 99 outliers (4.00%)
Column Curricular units 1st sem (grade): 406 outliers (16.40%)
Column Curricular units 1st sem (without evaluations): 174 outliers (7.03%)
Column Curricular units 2nd sem (credited): 301 outliers (12.16%)
Column Curricular units 2nd sem (enrolled): 215 outliers (8.68%)
Column Curricular units 2nd sem (evaluations): 59 outliers (2.38%)
Column Curricular units 2nd sem (approved): 27 outliers (1.09%)
Column Curricular units 2nd sem (grade): 494 outliers (19.95%)
Column Curricular units 2nd sem (without evaluations): 153 outliers (6.18%)
```

Dengan rentang persentase outlier sebesar 1-20%, ada beberapa kolom yang memiliki outlier yang sedikit; menandakan kemungkinan besar value tersebut merupakan anomali, serta kolom dengan outlier yang lebih banyak; menandakan kemungkinan besar outlier tersebut bukan sekedar anomali, tetapi nilai yang wajar didapatkan. Outlier handling perlu dilakukan untuk menstabilkan nilai varians dan mendekatkan data ke distribusi normal, agar data lebih stabil untuk diproses pada tahap modelling. Oleh karena itu, dilakukan implementasi metode handling outlier sebagai berikut:

1. **Melakukan capping/clipping pada outlier menjadi nilai upper atau lower bound untuk kolom dengan frekuensi outlier di bawah 10%.**
   Outlier kemungkinan besar merupakan anomali yang jarang terjadi, sehingga dapat di-cap saja untuk mempertahankan pola dari dataset.
2. **Melakukan log transformation pada kolom dengan frekuensi outlier di atas 10%.**
   Nilai dari outlier sebaiknya dipertahankan karena bukan sekedar anomali. Oleh karena itu, untuk mempertahankan pola dari dataset, outlier akan ditransformasi agar nilainya lebih mendekati median dari value kolom tersebut.

Catatan: tidak dilakukan outlier handling pada grade karena nilai ujian memang wajar untuk memiliki nilai yang ekstrem. Outlier handling pada nilai ujian justru akan merusak skala nilai.

1.1.4. Data Cleaning: Feature Engineering

Feature engineering membuat feature-feature baru atau melakukan transformasi pada feature yang sudah ada dengan tujuan meningkatkan kualitas dari feature untuk kebutuhan pelatihan model machine learning. Diimplementasikan feature engineering sebagai berikut:

1. **Membuat Interaction Feature**
   Interaction feature menciptakan feature baru berdasarkan gabungan dari feature yang sudah ada. Hal ini dilakukan karena ada beberapa feature yang memiliki keterkaitan dan dapat dikembangkan, yaitu kolom “units” yang terdiri dari semester 1 dan semester 2. Diciptakan sebanyak 8 feature baru yang merupakan gabungan dari kolom semester 1 dengan semester 2:
   - Total_enrolled
   - Total_approved
   - Total_credited
   - Average_grade
   - Approval_rate_sem1 (Approved/Enrolled)

- Approval_rate_sem2
- Grade_improvement (grade semester 2 - grade semester 1)

2. Feature Selection

Feature selection memilih features yang lebih berpengaruh dan melakukan drop pada features yang redundan atau tidak berpengaruh untuk menyederhanakan model. Metode yang digunakan adalah melakukan dropping pada kolom-kolom yang redundan, sedangkan pertimbangan tidak melakukan seleksi feature yang paling berpengaruh adalah akan dilakukan dimension reductionality di tahap preprocessing.

Pemilihan kolom yang redundan dilakukan dengan **menghitung korelasi antar kolom**. Jika ada pasangan kolom yang memiliki korelasi tinggi, artinya salah satu kolom tersebut dapat di-drop. Hal ini lebih masuk akal jika melihat pada dataset, ada kolom-kolom dari semester 1 dan semester 2 yang memiliki pola nilai yang sama, sehingga ada kemungkinan besar redundancy dari memiliki 2 jenis kolom yang sama. Oleh karena itu, dapat dilakukan feature selection untuk mengurangi redundancy ini. Dalam implementasi, digunakan threshold nilai korelasi antar kolom sebesar 0,9 untuk menentukan apakah suatu pasangan kolom redundan.

Kemudian, pemilihan kolom mana yang akan di-drop dilakukan dengan menghitung korelasi kolom tersebut terhadap target dan melakukan drop pada kolom dengan korelasi yang lebih rendah.

```
# Redundancy Filtering (Correlation)
# We look for features correlated > 0.90 with each other.
# We drop the one that is LESS correlated with the Target.

# 1. Calculate Feature-Feature Correlation
feature_corr = analysis_df.drop(columns=['Target_Encoded']).corr().abs()

# 2. Calculate Feature-Target Correlation
target_corr = analysis_df.drop(columns=['Target_Encoded']).apply(
    lambda x: x.corr(analysis_df['Target_Encoded'])
).abs()

# 3. Identify pairs and select the one to drop
upper = feature_corr.where(np.triu(np.ones(feature_corr.shape),
k=1).astype(bool))
features_to_drop = []

for column in upper.columns:
    if any(upper[column] > 0.90): # threshold of correlation: 0.9
        # Find all features this column conflicts with
        conflicting_features = upper.index[upper[column] > 0.90].tolist()
        for feature in conflicting_features:
```

```
            # Compare their correlation with the Target
            score_col = target_corr[column]
            score_feat = target_corr[feature]

            # Drop the weaker predictor
            if score_col < score_feat:
                features_to_drop.append(column)
            else:
                features_to_drop.append(feature)

features_to_drop = list(set(features_to_drop))
train_set_eng = train_set_eng.drop(columns=features_to_drop)
val_set_eng = val_set_eng.drop(columns=features_to_drop)
```

```
# Hasil:
Dropping 10 redundant features: ['Total_enrolled', 'Total_credited',
'Nacionality', 'Curricular units 1st sem (approved)', 'Curricular units 1st sem
(enrolled)', 'Curricular units 2nd sem (approved)', 'Curricular units 1st sem
(credited)', 'Total_approved', 'Curricular units 1st sem (grade)',
'Average_grade']
```

3. Binning/Discretization

Tahap ini memisahkan fitur-fitur numerik kontinu, misalnya seperti age group, menjadi kategorikal dengan mengelompokkan age values pada rentang tertentu menjadi kategori. Selain itu, binning juga dilakukan untuk menyederhanakan kategori-kategori suatu feature yang jarang muncul, terutama bagi feature yang memiliki banyak kategori.

Pada dataset ini, ada 2 metode binning yang diimplementasikan, yaitu **domain-based** dan **frequency-based** binning. Domain-based binning melakukan pengelompokkan value numerik kontinu ke dalam suatu grup dan ini diimplementasikan pada kolom ‘Age at enrollment’, sehingga kategorinya menjadi ‘Young’, ‘Traditional’, dan ‘Mature’.

```
def bin_age(age_log):
    if age_log < np.log(20):
        return 0  # Young
    elif age_log <= np.log(25):
        return 1  # Traditional
    else:
        return 2  # Mature

train_set_eng['Age at enrollment'] = train_set_eng['Age at
enrollment'].apply(bin_age)
val_set_eng['Age at enrollment'] = val_set_eng['Age at
enrollment'].apply(bin_age)

# Note: menggunakan log-scale pada threshold umur karena kolom ini ditransformasi
log saat outlier handling.
```

Kemudian, frequency-based binning dilakukan pada kolom-kolom kategorikal lain pada dataset yang memiliki kategori yang banyak dengan persebaran yang tidak merata (tail-ended). Berdasarkan analisis dan visualisasi dengan histogram, kolom-kolom yang cocok diimplementasikan metode ini adalah: ["Mother's occupation", "Father's occupation", "Previous qualification"]. Sebagai contoh aplikasi frequency-based binning, berikut analisis data pada kolom “Mother’s occupation”:

```
Total unique categories: 30
Total samples: 2476

Top 10 most frequent categories:
                      Count  Percentage
Mother's occupation
9                       858       34.65
4                       474       19.14
5                       300       12.12
3                       184        7.43
2                       179        7.23
7                       166        6.70
0                        85        3.43
6                        57        2.30
1                        55        2.22
90                       40        1.62
Categories with < 5% frequency: 24/30
```

Terdapat sebanyak 24 kolom yang memiliki frekuensi kemunculan di bawah 5% (angka ini digunakan sebagai threshold kategori yang redundan). Untuk menyederhanakan kategori pada kolom ini yang banyak, diaplikasikan binning. Ke-24 kolom redundan tersebut dikelompokkan ke kategori baru ‘Other’ yang dalam numerik direpresentasikan dengan nilai “-1”. Hasilnya adalah sebagai berikut:

```
1. Mother's occupation Binning
   Binned 24/30 categories to 'Other'
   'Other' category (-1) count: 315

                     Count  Percentage
Mother's occupation
-1                     315       12.72
 2                     179        7.23
 3                     184        7.43
 4                     474       19.14
 5                     300       12.12
 7                     166        6.70
 9                     858       34.65
```

1.1.5. Data Preprocessing: Feature Scaling

Feature scaling bertujuan untuk menyamakan skala feature numerik ke rentang yang sama. Dengan demikian, seluruh feature setara dalam berkontribusi pada

proses pelatihan model, tanpa menyebabkan ketidakseimbangan akibat feature yang memiliki skala yang lebih besar atau lebih kecil.

Implementasi feature scaling menggunakan Standardization (Z-score Scaling) yang mentransformasi feature sehingga memiliki nilai mean 0 dan standar deviasi 1. Metode ini dipilih karena lebih fleksibel dalam menstandarkan data yang memiliki outlier-outlier tanpa membatasinya pada suatu rentang (seperti Min-Max Scaling). Dengan demikian, transformasi ini lebih tepat untuk pelatihan model yang digunakan pada tugas besar ini.

```
scaler_transformer = StandardScalerTransformer(numerical_cols_to_scale)
scaler_transformer.fit(X_train)
X_train = scaler_transformer.transform(X_train)
X_val   = scaler_transformer.transform(X_val)
print(f"Std after scaling: {X_train[numerical_cols_to_scale].std().mean():.6f}")
```

1.1.6. Data Preprocessing: Feature Encoding

Feature encoding bertujuan untuk mengonversi kolom kategorikal menjadi kolom numerik agar dapat diproses untuk tahap modelling. Terdapat dua jenis data kategorikal, yaitu data nominal – data tanpa urutan dan hanya untuk pengelompokkan saja, dan data ordinal – data yang memiliki urutan atau makna pada nilainya. Sesuai dengan jenis data ini, akan diaplikasikan label encoding untuk data ordinal, kemudian one-hot encoding untuk data nominal yang bukan binary (beberapa kolom sudah binary dengan kategori value 1/0, sehingga tidak perlu di-encoding lagi).

Alasan pemilihan jenis encoding ini, pertama-tama untuk label encoding adalah agar kategori value pada kolom-kolom disederhanakan untuk modelling namun tetap mempertahankan tingkatan/order-nya dengan label terurut, seperti 0, 1, dan 2. Sedangkan, one-hot encoding mengubah data nominal yang memiliki banyak kategori menjadi binary 1/0 untuk tiap kategori value dari kolom tersebut. Kemudian, metode lain yang tidak digunakan adalah target/mean encoding yang mengubah tiap kategori dengan mean dari variabel target per kategori. Alasan tidak digunakannya metode ini adalah metode ini lebih cocok digunakan untuk dataset dan kategori yang lebih banyak. Apabila diimplementasikan pada dataset ini, ada risiko *oversimplifying* data yang relatif sudah sedikit.

Berdasarkan analisis jenis kolom dan kategorinya dari training set yang sudah di cleaning, pengelompokkan kolom kategorikal adalah sebagai berikut:

```
nominal_features_not_binary = ['Marital status', 'Application mode', 'Course',
"Mother's occupation", "Father's occupation"]

ordinal_features = [‘Application order', Previous qualification', "Mother's
qualification", "Father's qualification", 'Age at enrollment']
```

```
# Proses encoding menggunakan fungsi ini
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore', dtype=int)
oe = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1,
dtype=int)

# Kode untuk fit dan transform pada kolom-kolom train_set dan val_set
```

1.1.7. Data Preprocessing: Handling Imbalanced Dataset

Imbalanced dataset handling bertujuan untuk mengurangi bias kepada kelas mayoritas yang dapat menyebabkan masalah pada pelatihan model nantinya. Oleh karena itu, beberapa metode handling ini bertujuan untuk meningkatkan jumlah atau bobot dari kelas minoritas agar dataset lebih seimbang.

Handling ini diimplementasikan menggunakan resampling method yang meningkatkan instance pada kelas minoritas (oversampling). Digunakan metode SMOTETomek. Strateginya adalah untuk menyeimbangkan kelas minoritas menuju 70% dari kelas mayoritas. Dengan target sampling ratio ini, metode SMOTE akan melakukan penambahan sampel baru yang sintetis dan realistis, kemudian Tomek berfungsi untuk menghapus pasangan yang berada pada threshold rasio 0.7 untuk mencegah overfitting.

```
imbalance_handler = ImbalanceHandler(sampling_ratio=0.7, random_state=42)
X_train_resampled, y_train_resampled = imbalance_handler.fit_resample(X_train,
y_train)
X_train = pd.DataFrame(X_train_resampled, columns=X_train.columns)
y_train = pd.Series(y_train_resampled, name='Target')
```

Contohnya, pada dataset kelas Target, setelah diaplikasikan handling imbalanced dataset ini, perubahan rasio kelas menjadi seperti berikut:

```
Class proportions:
Target
Graduate    49.919225
Dropout     32.108239
Enrolled    17.972536
```

```
>>> Applying SMOTETomek (SMOTE + Tomek Links) <<<

New class proportions:
Target
Graduate    41.596045
Enrolled    29.696328
Dropout     28.707627
```

1.1.8. Data Preprocessing: Data Normalization

Normalisasi data dilakukan untuk menormalkan feature agar memiliki distribusi yang standar, terutama untuk feature-feature yang memiliki skewness yang tinggi dan tidak simetris. Dengan demikian, data akan lebih stabil untuk pelatihan dan optimisasi model.

Implementasi normalisasi data dilakukan dengan menggunakan metode Transformasi Yeo-Johnson dengan PowerTransformer. Transformasi ini akan mentransformasi nilai yang positif dan negatif sehingga lebih fleksibel digunakan pada dataset, terutama jika ada outlier-outlier data. Di sini, transformasi akan dilakukan untuk kolom-kolom yang memiliki nilai skewness di atas 1.0 (terkecuali feature biner). Dengan demikian, pengaruh outlier yang tersisa dari tahap data cleaning dapat dikurangi.

```
normalizer = NormalizationTransformer(skewness_threshold=1.0)
normalizer.fit(X_train)
skewness_before = X_train[normalizer.highly_skewed_cols].skew()
X_train = normalizer.transform(X_train)
X_val   = normalizer.transform(X_val)
skewness_after = X_train[normalizer.highly_skewed_cols].skew()
```

1.1.9. Data Preprocessing: Dimensionality Reduction

Setelah melakukan perubahan, transformasi, dan penambahan feature-feature, dataset akan memiliki sejumlah features, kadangkala terlalu banyak yang dapat menyebabkan masalah untuk pelatihan model. Oleh karena itu, dimensionality reduction bertujuan untuk mengurangi jumlah features yang ada dengan mempertahankan informasi feature sebanyak mungkin.

Implementasi ini dilakukan menggunakan Principal Component Analysis (PCA) yang mengonversikan gabungan feature-feature yang saling berhubungan menjadi feature baru principal component yang mempertahankan inti informasi dari gabungan feature tersebut. Implementasi ini melalui tahapan membuang feature yang kurang informatif dan tidak perlu dipertahankan apabila nilai variansi < 0.01, kemudian melakukan feature selection berdasarkan important yang dihitung dari

nilai MI, dan terakhir melakukan PCA untuk mempertahankan 95% variansi apabila belum tercapai.

```
dim_reducer = DimensionalityReducer(
    variance_threshold=0.01,
    feature_selection_method='adaptive',
    min_features=15,
    max_features=40,
    apply_pca=True,
    pca_variance=0.95
)
dim_reducer.fit(X_train, y_train)
X_train = dim_reducer.transform(X_train)
X_val   = dim_reducer.transform(X_val)
```

Contohnya di sini, implementasi dimensionality reduction berhasil mereduksi feature sebagai berikut:

```
======================================================================
DIMENSIONALITY REDUCTION SUMMARY
======================================================================
Final feature count: 14
Original features: 83
Reduction: 83.1%
```

1.1.10. Data Preprocessing: Function dan Pipeline

Untuk mengimplementasikan tahapan data preprocessing ini, dilakukan *hybrid approach* untuk mentransformasi dataset, yaitu menciptakan fungsi-fungsi per tahap data preprocessing yang akan dijalankan pada dataset secara manual, atau menciptakan pipeline secara keseluruhan untuk langsung mentransformasi dataset.

Sedangkan, metode pipeline dapat dilakukan karena implementasi preprocessing menggunakan library sklearn. Pipeline diimplementasikan sebagai fungsi yang dimanfaatkan untuk mentransformasi dataset test untuk kebutuhan submission. Fungsinya berbentuk sebagai berikut:

```
def apply_preprocessing_pipeline(df, is_test=False,
feature_engineering_done=False):
    """
    Apply the complete preprocessing pipeline to new data

    Parameters:
    -----------
    df : DataFrame
        Input dataframe
    is_test : bool
        If True, expects no 'Target' column and skips SMOTE
    feature_engineering_done : bool
        If True, assumes feature engineering already applied
```

```
    Returns:
    --------
    X_processed : DataFrame/array
        Preprocessed features
    y : Series (only if is_test=False)
        Target variable
    """
    # Kode data preprocessing dari langkah-langkah sebelumnya
```

1.2. Penjelasan Implementasi Decision Tree Learning

Algoritma Decision Tree yang diimplementasikan adalah Iterative Dichotomiser 3 (ID3) from scratch. Implementasi ini dirancang untuk menangani data bertipe numerik (continuous) maupun kategorikal, serta memiliki mekanisme penanganan nilai kosong (null values) secara otomatis. Selain itu, implementasi ini dilengkapi dengan parameter pre-pruning (max_depth dan min_samples_split) untuk mencegah overfitting.

Kelas Node

Kelas ini merepresentasikan sebuah simpul (node) dalam pohon keputusan. Setiap objek Node dapat berperan sebagai:

1. Leaf Node (Daun): Menyimpan nilai prediksi akhir (prediction) jika simpul ini adalah ujung pohon.
2. Decision Node (Cabang): Menyimpan informasi fitur yang diuji (feature_idx, feature_name), tipe fitur (is_continuous), nilai ambang batas untuk fitur numerik (threshold), dan pointer ke simpul anak (children).

```
class Node:
   def __init__(self, prediction=None):
       self.prediction = prediction
       self.feature_idx = None
       self.feature_name = None
       self.is_continuous = False
       self.threshold = None
       self.children = {}
```

Kelas ID3DecisionTree

Kelas utama yang membungkus logika algoritma ID3. Kelas ini mewarisi BaseClassifier dan memiliki metode utama fit untuk pelatihan dan predict untuk prediksi. Konstruktor menerima hyperparameter min_samples_split (jumlah sampel minimum untuk membelah simpul) dan max_depth (kedalaman maksimum pohon) untuk mengontrol kompleksitas model.

```
class ID3DecisionTree(BaseClassifier):
   def __init__(self, min_samples_split=2, max_depth=None):
       self.min_samples_split = min_samples_split
       self.max_depth = max_depth
       self.root = None
       self.feature_types = []
       self.feature_names = []
```

Fungsi fit

Fungsi ini bertugas mempersiapkan data sebelum proses pembentukan pohon dimulai. Langkah-langkah yang dilakukan:

1. Menangani Missing Values: Mengisi nilai kosong (NaN) dengan mean untuk data numerik dan mode untuk data kategorikal .
2. Deteksi Tipe Fitur: Mengidentifikasi secara otomatis apakah setiap fitur bersifat continuous (angka) atau categorical agar strategi percabangan yang digunakan tepat .
3. Memanggil fungsi rekursif _id3 untuk mulai membangun pohon dari akar (root).

```
   def fit(self, X, y, feature_names=None):

       if isinstance(X, pd.DataFrame):
           if feature_names is None: feature_names = X.columns.tolist()
           X_df = X.copy()
       else:
           X_df = pd.DataFrame(X)

       y = np.array(y)

       # Menangani nilai yang hilang
       for col in X_df.columns:
           if X_df[col].isnull().any():
                       if pd.api.types.is_numeric_dtype(X_df[col]): X_df[col] =
X_df[col].fillna(X_df[col].mean())
                else: X_df[col] = X_df[col].fillna(X_df[col].mode()[0])

       X_vals = X_df.values

       # Mendifinisikan tipe fitur
       self.feature_types = []
       n_features = X_vals.shape[1]
       for i in range(n_features):
           val = X_vals[0, i]
           self.feature_types.append('continuous' if isinstance(val, (int, float,
np.number)) and not isinstance(val, str) else 'categorical')

       # Menyimpan nama fitur
        self.feature_names = feature_names if feature_names else [f"feat_{i}" for
i in range(n_features)]

       self.root = self._id3(X_vals, y, list(range(n_features)), depth=0)
```

Cara Kerja:

1. Validasi Input: Mengonversi data latih menjadi format Pandas DataFrame untuk memudahkan pengolahan.
2. Imputasi Data: Mengisi nilai kosong (null) pada kolom numerik dengan mean dan kolom kategorikal dengan mode.

3. Identifikasi Fitur: Mengecek setiap kolom untuk menentukan apakah tipe datanya continuous (angka) atau categorical (kategori) agar algoritma tahu cara memprosesnya.
4. Inisialisasi Pohon: Memanggil fungsi inti _id3 untuk mulai membangun struktur pohon dari simpul akar (root).

Fungsi predict

Deskripsi: Fungsi ini merupakan pintu gerbang utama untuk melakukan klasifikasi pada sekumpulan data baru (dataset). Fungsi ini menerima matriks fitur (X) dan mengembalikan array berisi label prediksi untuk setiap baris data tersebut.

```
def predict(self, X: np.ndarray) -> np.ndarray:
    X = np.array(X, dtype=object)
    return np.array([self._traverse(x, self.root) for x in X])
```

Cara Kerja:

1. Standardisasi Input: Mengonversi data input X menjadi numpy array dengan tipe data object. Hal ini dilakukan untuk menjaga konsistensi dan mencegah error jika data input mengandung campuran tipe data (misal: angka dan string) atau masih dalam format DataFrame.
2. Iterasi Prediksi: Melakukan perulangan (looping) menggunakan list comprehension untuk setiap sampel (x) dalam dataset X.
3. Delegasi ke _traverse: Untuk setiap sampel, fungsi ini memanggil fungsi helper _traverse yang akan menelusuri pohon dari akar (self.root) hingga menemukan label prediksi.
4. Agregasi Hasil: Mengumpulkan semua hasil prediksi dari _traverse dan mengembalikannya dalam bentuk array numpy satu dimensi.

Fungsi _id3 (Inti Algoritma)

Fungsi rekursif ini adalah otak dari algoritma ID3. Logika utamanya:

1. Base Cases (Kondisi Berhenti): Berhenti dan membuat Leaf Node jika:
    - Semua label di simpul seragam (murni).
    - Atribut habis, kedalaman maksimum tercapai, atau jumlah sampel kurang dari min_samples_split .
2. Best Split: Memanggil _get_best_attribute untuk mencari fitur dengan Information Gain tertinggi.
3. Recursive Splitting:

- Continuous: Membelah menjadi 2 cabang (Binary Split: <= dan >) berdasarkan threshold terbaik .
- Categorical: Membelah menjadi banyak cabang (Multi-way Split) sesuai jumlah nilai unik pada fitur tersebut .

```
   def _id3(self, X, y, attribute_indices, depth):
       n_samples = X.shape[0]
       n_labels = len(np.unique(y))

       # Ketika semua label sama
       if n_labels == 1: return Node(prediction=y[0])

       # Ketika tidak ada atribut tersisa atau mencapai kedalaman maksimum
       hit_max = (self.max_depth is not None and depth >= self.max_depth)
                if len(attribute_indices) == 0 or hit_max or n_samples <
self.min_samples_split:
           return Node(prediction=self._most_common_label(y))

       # Memilih atribut terbaik untuk split
             best_feat, best_gain, best_thr = self._get_best_attribute(X, y,
attribute_indices)
       if best_feat is None: return Node(prediction=self._most_common_label(y))

       # Membuat node
       node = Node()
       node.feature_idx = best_feat
       node.feature_name = self.feature_names[best_feat]
       node.is_continuous = (self.feature_types[best_feat] == 'continuous')

       # Membagi dataset dan merekursi
       if node.is_continuous:
           node.threshold = best_thr
           left_mask = X[:, best_feat] <= best_thr
           X_l, y_l = X[left_mask], y[left_mask]
           X_r, y_r = X[~left_mask], y[~left_mask]

                           if len(X_l) == 0 or len(X_r) == 0: return
Node(prediction=self._most_common_label(y))
            node.children['<='] = self._id3(X_l, y_l, attribute_indices, depth +
1)
           node.children['>'] = self._id3(X_r, y_r, attribute_indices, depth + 1)
       else:
           new_attrs = [a for a in attribute_indices if a != best_feat]
           for val in np.unique(X[:, best_feat]):
               mask = X[:, best_feat] == val
               X_sub, y_sub = X[mask], y[mask]
                child = Node(prediction=self._most_common_label(y)) if len(X_sub)
== 0 else self._id3(X_sub, y_sub, new_attrs, depth + 1)
               node.children[val] = child
       return node
```

Cara Kerja:

1. Cek Kondisi Berhenti: Algoritma akan berhenti membelah dan membuat simpul daun (Leaf Node) jika:
    - Label data sudah seragam (murni).

- Kedalaman pohon mencapai max_depth.
- Jumlah data kurang dari min_samples_split.

2. Seleksi Fitur Terbaik: Jika belum berhenti, algoritma mencari fitur dengan Information Gain tertinggi menggunakan fungsi _get_best_attribute.
3. Pembentukan Cabang:
    - Jika fitur numerik, data dibelah menjadi dua jalur (<= threshold dan > threshold).
    - Jika fitur kategorikal, data dibelah sebanyak jumlah kategori unik yang ada.
4. Rekursi: Melakukan pemanggilan fungsi _id3 kembali untuk setiap cabang anak yang terbentuk hingga seluruh pohon selesai dibangun.

Fungsi _get_best_attribute

Fungsi ini bertugas menyeleksi fitur mana yang paling optimal untuk dijadikan pemecah data (split) pada node saat ini. Kriteria "terbaik" didasarkan pada nilai Information Gain tertinggi.

```
   def _get_best_attribute(self, X, y, attribute_indices):
       best_gain, best_feat, best_thr = -1, None, None
       for idx in attribute_indices:
           X_col = X[:, idx]
           if self.feature_types[idx] == 'continuous':
               thresholds = np.unique(X_col)
                  if len(thresholds) > 50: thresholds = np.percentile(thresholds,
np.linspace(0, 100, 10))
               for thr in thresholds:
                   gain = self._calc_gain_cont(y, X_col, thr)
                      if gain > best_gain: best_gain, best_feat, best_thr = gain,
idx, thr
           else:
               gain = self._calc_gain_cat(y, X_col)
                 if gain > best_gain: best_gain, best_feat, best_thr = gain, idx,
None
       return best_feat, best_gain, best_thr
```

Cara Kerja:

1. Melakukan iterasi untuk setiap fitur yang tersedia (attribute_indices).
2. Mengecek tipe fitur:
    - Continuous (Angka): Mencari nilai ambang batas (threshold) terbaik. Untuk efisiensi komputasi, jika variasi nilai sangat banyak (>50), algoritma hanya mengecek 10 titik sampel (persentil) alih-alih mengecek semua nilai unik .
    - Categorical (Kategori): Menghitung gain berdasarkan pengelompokan nilai unik kategori tersebut.

3. Menyimpan dan mengembalikan fitur dengan Gain terbesar.

Fungsi _calc_gain_cat

Menghitung Information Gain untuk fitur bertipe kategorikal (nominal).

```
    def _calc_gain_cat(self, y, X_col):
        parent_ent = self._entropy(y)

        # Siapkan variabel
        n, child_ent_sum = len(y), 0

        # Cari nilai unik anak
        vals, counts = np.unique(X_col, return_counts=True)

        # Hitung entropi anak
        for val, count in zip(vals, counts):
            child_ent_sum += (count / n) * self._entropy(y[X_col == val])
        return parent_ent - child_ent_sum
```

Cara Kerja:

1. Menghitung entropi awal (parent entropy) sebelum pemisahan.
2. Memecah data menjadi beberapa cabang sesuai jumlah nilai unik pada fitur tersebut (Multi-way split).
3. Menghitung rata-rata entropi terbobot dari anak-anak cabang (weighted child entropy).
4. Mengembalikan nilai Gain = Entropi Awal - Entropi Anak .

Fungsi _calc_gain_cont

Menghitung Information Gain untuk fitur bertipe kontinu (numerik) berdasarkan threshold tertentu.

```
    def _calc_gain_cont(self, y, X_col, thr):
        parent_ent = self._entropy(y)
        n = len(y)

        #  Tentukan kubu kiri
        left = X_col <= thr

        # Kalau hasil pemisahannya kosong
        if np.sum(left) == 0 or np.sum(~left) == 0: return 0
        # Ambil label untuk masing-masing kubu
        y_l, y_r = y[left], y[~left]
                        return    parent_ent    -    ((len(y_l)/n)*self._entropy(y_l)    +
(len(y_r)/n)*self._entropy(y_r))
```

Cara Kerja:

1. Membagi data menjadi dua kelompok biner: Kiri (nilai <= threshold) dan Kanan (nilai > threshold).
2. Melakukan validasi: Jika pembagian menghasilkan satu sisi kosong, maka gain dianggap 0 (tidak valid).
3. Menghitung Gain dengan mengurangi entropi awal dengan rata-rata entropi dari dua kelompok tersebut.

Fungsi _entropy

Menghitung nilai Shannon Entropy, yaitu ukuran ketidakmurnian (impurity) atau keacakan dalam sekumpulan data label.

```
   def _entropy(self, y):
         ps = (np.bincount(y) if np.issubdtype(y.dtype, np.integer) else np.unique(y,
return_counts=True)[1]) / len(y)
       return -np.sum([p * np.log2(p) for p in ps if p > 0])
```

Cara Kerja:

1. Menghitung probabilitas (p) kemunculan setiap kelas target dalam data y.
2. Menerapkan rumus $-\sum(p \times \log_2(p))$. Semakin seragam datanya (misal semua labelnya 'Lulus'), entropinya mendekati 0. Semakin acak (50:50), entropinya mendekati 1.

Fungsi _most_common_label

Menentukan label mayoritas (modus) dari sekumpulan target. Fungsi ini digunakan saat membuat Leaf Node atau saat mekanisme voting diperlukan.

```
def _most_common_label(self, y):
   return None if len(y) == 0 else Counter(y).most_common(1)[0][0]
```

Cara Kerja: Menggunakan Counter untuk menghitung frekuensi setiap label dan mengembalikan label dengan jumlah terbanyak.

Fungsi _traverse

Fungsi rekursif untuk melakukan prediksi pada satu sampel data dengan menelusuri pohon dari akar hingga daun.

```
   def _traverse(self, x, node):
       # Jika mencapai daun
```

```
        if node.prediction is not None: return node.prediction

        # Ambil nilai fitur untuk node saat ini
        val = x[node.feature_idx]

        if node.is_continuous:
            next_node = node.children.get('<=' if val <= node.threshold else '>')
        else:
            next_node = node.children.get(val)

        # Jika nilai fitur tidak ada di cabang, ambil cabang pertama yang ada
                        if   next_node   is   None   and   node.children:   next_node   =
list(node.children.values())[0]
        return self._traverse(x, next_node) if next_node else node.prediction
```

Cara Kerja:

1. Cek apakah node saat ini adalah Daun (punya prediction). Jika ya, kembalikan prediksi.
2. Jika node percabangan, cek nilai fitur sampel data.
3. Tentukan arah:
    - Jika kontinu: Pilih cabang <= atau > berdasarkan threshold.
    - Jika kategorikal: Pilih cabang yang sesuai dengan nilai kategori.
4. Lanjut ke node anak berikutnya secara rekursif.

Fungsi print_tree

Memvisualisasikan struktur pohon keputusan dalam format teks agar mudah dibaca dan dianalisis.

```
     def  print_tree(self,  node=None,  prefix="",  is_last=True,  is_root=True,  branch="",
max_depth=None, cur_depth=0, file=None):
        if node is None: node = self.root
        if max_depth is not None and cur_depth > max_depth:
            if is_last and not is_root: print(f"{prefix}└── ...", file=file)
            return

             txt  =  f"Output:  {node.prediction}"  if  node.prediction  is  not  None  else
f"[{node.feature_name}]"
        if is_root: print(txt, file=file); new_prefix = ""
        else:
            branch_txt = f"({branch})"
                  print(f"{prefix}{'└── ' if is_last else '├── '}{branch_txt} {txt}",
file=file)
            new_prefix = prefix + ("    " if is_last else "│   ")

        if node.prediction is None:
            keys = list(node.children.keys())
            for i, k in enumerate(keys):
                  self.print_tree(node.children[k], new_prefix, i == len(keys)-1, False,
str(k), max_depth, cur_depth + 1, file)
```

Cara Kerja: Menggunakan penelusuran Depth First Search (DFS) rekursif untuk mencetak setiap node dan menggambarkan hierarki pohon.

1.3. Penjelasan Implementasi Logistic Regression

Pada dataset yang diberikan, variabel target memiliki beberapa kelas yang lebih dari dua (*multi class*). Sehingga, tidak dapat digunakan algoritma regresi logistik pada umumnya yang hanya dapat memprediksi target yang bersifat biner. Oleh karena itu, model regresi logistik diimplementasikan menggunakan algoritma Softmax (disebut juga Multinomial Logistic Regression).

Algoritma Softmax merupakan sebuah variasi dari algoritma model regresi logistik yang biasa digunakan, namun dapat memprediksi target yang memiliki kelas lebih dari dua. Model mempelajari bobot untuk setiap kelas target dan menggunakan fungsi aktivasi Softmax untuk menghasilkan distribusi probabilitas di seluruh kelas target. Optimasi digunakan menggunakan pendekatan *Gradient-based optimization* terhadap *Categorical Cross-Entropy Loss*.

Kelas Softmax Regression

Kelas ini merupakan inti dari model regresi logistik multinomial. Kelas ini mewarisi BaseClassifier dan menyimpan hyperparameter seperti *lr* (learning rate) dan *max_iter* (jumlah epoch). Atribut utama yang dipelajari selama proses training adalah self.weights, yaitu matriks berukuran (*n_classes x n_features + 1*) yang merepresentasikan koefisien model untuk setiap kelas terhadap setiap fitur (termasuk bias).

```
class SoftmaxRegression(BaseClassifier):
    def __init__(self, lr=0.1, max_iter=100, n_classes=None, verbose=False):
        self.lr = lr
        self.max_iter = max_iter
        self.n_classes = n_classes
        self.verbose = verbose
        self.weights = None
        self.mean = None
        self.std = None
        self.loss_history = []
        self.classes_ = None
```

Fungsi _add_bias

Deskripsi fungsi: Menambahkan kolom bernilai 1 di awal matriks fitur input X. Ini memungkinkan model untuk mempelajari intercept (bias) sebagai bagian dari operasi perkalian matriks dot product.

```
def _add_bias(self, X: np.ndarray) -> np.ndarray:
        bias = np.full((X.shape[0], 1), 1)
```

```
        return np.hstack((bias, X))
```

Cara kerja:

1. Membuat vektor kolom yang berisi angka 1 sebanyak jumlah sampel data.
2. Menggabungkan vektor tersebut ke sisi kiri matriks fitur X (concatenation).
3. Menghasilkan matriks baru dengan dimensi fitur bertambah satu untuk mengakomodasi nilai bias (intercept).

Fungsi _softmax

Mengubah skor linear Z menjadi probabilitas. Fungsi ini menerapkan kestabilan numerik (numerical stability) dengan mengurangi nilai maksimum dari skor sebelum eksponensiasi untuk mencegah overflow pada angka yang sangat besar.

```
def _softmax(self, Z: np.ndarray) -> np.ndarray:
        exp_scores = np.exp(Z - np.max(Z, axis=1, keepdims=True))
        return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
```

Cara kerja:

1. Mencari nilai maksimum dalam setiap baris skor Z dan menguranginya dari seluruh elemen (teknik *numerical stability* untuk mencegah *overflow* eksponensial).
2. Menghitung nilai eksponensial (e^Z) untuk setiap elemen skor yang telah disesuaikan.
3. Menjumlahkan nilai eksponensial tersebut per baris (per sampel).
4. Membagi setiap nilai eksponensial dengan hasil penjumlahan barisnya untuk mendapatkan probabilitas yang ternormalisasi (total probabilitas = 1).

Fungsi _compute_loss

Menghitung Categorical Cross-Entropy Loss untuk memantau kinerja model selama pelatihan.

```
def _compute_loss(self, Y_true: np.ndarray, Y_pred: np.ndarray) -> float:
        n_samples = Y_true.shape[0]
        log_probs = np.log(Y_pred + 1e-15)
        loss = -np.sum(Y_true * log_probs) / n_samples
        return loss
```

Cara kerja:

1. Menambahkan nilai epsilon yang sangat kecil ($1e^{-15}$) pada prediksi probabilitas untuk menghindari error logaritma dari nilai 0.
2. Menghitung logaritma natural dari probabilitas prediksi.
3. Mengalikan hasil logaritma dengan matriks target asli (Y_true).

4. Menjumlahkan seluruh nilai negatifnya dan membaginya dengan jumlah sampel untuk mendapatkan rata-rata loss.

Fungsi fit

Fungsi ini melakukan pelatihan model menggunakan algoritma optimasi berbasis gradien. Fungsi ini memperbarui bobot model secara iteratif untuk meminimalkan kesalahan prediksi terhadap data latih.

```
def fit(self, X: np.ndarray, y: np.ndarray, random_weights: bool=True):
        if self.n_classes is None:
            self.classes_ = np.unique(y)
            self.n_classes = len(self.classes_)

        X_bias = self._add_bias(X)

        if random_weights:
            self.weights = np.random.randn(self.n_classes, X_bias.shape[1]) * 0.01
        else:
            self.weights = np.zeros((self.n_classes, X_bias.shape[1]))

        if self.verbose:
            print("Starting training...")
            for epoch in range(self.max_iter):
                Z = X_bias @ self.weights.T
                probs = self._softmax(Z)
                error = y - probs
                gradient = (error.T @ X_bias) / X_bias.shape[0]
                self.weights += self.lr * gradient
                loss = self._compute_loss(y, probs)
                self.loss_history.append(loss)

                if epoch % 10 == 0 or epoch == self.max_iter - 1:
                    print(f"Epoch {epoch+1}/{self.max_iter}, Loss: {loss:.4f}")
        else:
            for epoch in range(self.max_iter):
                Z = X_bias @ self.weights.T
                probs = self._softmax(Z)
                error = y - probs
                gradient = (error.T @ X_bias) / X_bias.shape[0]
                self.weights += self.lr * gradient
                loss = self._compute_loss(y, probs)
                self.loss_history.append(loss)
```

Cara kerja:

1. Inisialisasi: Mengidentifikasi jumlah kelas unik dari label y dan menambahkan bias pada matriks input X.
2. Penentuan Bobot Awal: Menginisialisasi matriks bobot (weights) dengan nilai acak kecil (jika random_weights=True) atau nilai nol.
3. Iterasi (Epoch): Memulai perulangan sebanyak max_iter.
4. Forward Pass: Menghitung skor linear Z dengan mengalikan matriks input X dengan transpose matriks bobot W.
5. Aktivasi: Mengubah skor Z menjadi probabilitas menggunakan fungsi Softmax.
6. Perhitungan Error: Menghitung selisih antara label target asli (y) dan probabilitas prediksi (probs).

7. Backward Pass (Gradien): Menghitung gradien dengan mengalikan transpose error dengan input X, lalu dibagi jumlah sampel.
8. Update Bobot: Memperbarui bobot dengan menambahkan hasil perkalian *learning rate* (lr) dan gradien ke bobot lama.
9. Monitoring (Opsional): Menghitung nilai loss pada setiap epoch untuk memantau konvergensi model.

Fungsi predict dan predict_proba

Fungsi ini digunakan untuk melakukan inferensi atau prediksi pada data baru yang belum pernah dilihat model sebelumnya. predict_proba menghasilkan probabilitas, sedangkan predict menghasilkan label kelas akhir.

```
def predict_proba(self, X: np.ndarray) -> np.ndarray:
        W = self.weights
        X_bias = self._add_bias(X)
        Z = X_bias @ W.T
        return self._softmax(Z)

    def predict(self, X: np.ndarray) -> np.ndarray:
        probs = self.predict_proba(X)
        return np.argmax(probs, axis=1)
```

Cara Kerja:

1. Persiapan Input: Mengambil matriks bobot yang telah dilatih dan menambahkan kolom bias pada data input baru X.
2. Perhitungan Skor: Menghitung skor linear Z melalui perkalian matriks input dengan bobot.
3. Hitung Probabilitas (predict_proba): Menerapkan fungsi Softmax pada skor Z untuk mendapatkan probabilitas kepemilikan kelas untuk setiap sampel.
4. Penentuan Label (predict): Menggunakan fungsi argmax untuk memilih indeks kelas yang memiliki nilai probabilitas tertinggi sebagai hasil prediksi akhir.

1.4. Penjelasan Implementasi SVM

Untuk implementasi SVM digunakan optimasi Dual Problem yang diselesaikan menggunakan solver Quadratic Programming (QP) dari pustaka cvxopt. Algoritma dirancang untuk meminimalkan fungsi objektif kuadratik dengan kendala soft margin ($0 < \alpha < C$) guna menemukan support vectors optimal. Untuk menangani data non-linear, diterapkan mekanisme Kernel Trick dengan varian Linear dan Radial Basis Function (RBF). Perluasan ke klasifikasi multi-kelas dilakukan menggunakan strategi Directed Acyclic Graph (DAGSVM), yang melatih model dengan pendekatan One-against-One dan melakukan prediksi melalui eliminasi kandidat kelas secara iteratif pada struktur graf.

Kelas SVM_QP

Kelas ini merupakan inti dari implementasi *Support Vector Machine* (SVM) untuk klasifikasi biner. Kelas ini bertanggung jawab menyelesaikan masalah optimasi *Dual Problem* menggunakan *Quadratic Programming* (QP) solver.

```
class SVM_QP(BaseClassifier):
    def __init__(self, C=1.0, gamma=0.1, kernel='rbf'):
        self.C = C
        self.gamma = gamma
        self.kernel = kernel
        self.alphas = None
        self.support_vectors = None
        self.support_vector_labels = None
        self.b = 0
```

__init__(self, C=1.0, gamma=0.1, kernel='rbf')

- Deskripsi: Konstruktor untuk menginisialisasi hyperparameter model.
- Parameter:
    - C: Parameter regularisasi (Soft Margin) yang mengontrol penalti terhadap kesalahan klasifikasi.
    - gamma: Koefisien kernel untuk RBF, menentukan seberapa jauh pengaruh satu sampel pelatihan.
    - kernel: Jenis kernel yang digunakan ('linear' atau 'rbf').
- Fungsi: Menyiapkan variabel instance untuk menyimpan parameter model serta tempat penampungan hasil pelatihan seperti alphas, support_vectors, dan bias.

Fungsi linear_kernel(self, X1, X2)

Fungsi pembantu untuk menghitung kernel Linear.

```
def linear_kernel(self, X1, X2):
        return np.dot(X1, X2.T)
```

Menghitung *dot product* (hasil kali titik) antara dua matriks fitur X1 dan X2.

Rumus:

$$K(x_i, x_j) = x_i \cdot x_j$$

Fungsi rbf_kernel(self, X1, X2):

Fungsi pembantu untuk menghitung kernel Radial Basis Function (RBF).

```
def rbf_kernel(self, X1, X2):
        # Menggunakan Euclidean distance squared
        n_samples_1 = X1.shape[0]
        n_samples_2 = X2.shape[0]
        K = np.zeros((n_samples_1, n_samples_2))

        for i in range(n_samples_1):
            for j in range(n_samples_2):
                diff = X1[i] - X2[j]
                K[i, j] = np.exp(-self.gamma * np.dot(diff, diff))
        return K
```

Menghitung kesamaan antar data menggunakan fungsi eksponensial berdasarkan jarak Euclidean kuadrat negatif. Rumus:

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$$

Fungsi fit(self, X, y)

Fungsi utama untuk melatih model SVM menggunakan data latih

```
def fit(self, X, y):

        # Konversi ke numpy array jika dia DataFrame
        if hasattr(X, 'values'):
            X = X.values

        if hasattr(y, 'values'):
            y = y.values

        n_samples, n_features = X.shape
        y = y.astype(float).reshape(-1, 1)

        # Matriks Gram (Kernel Matrix)
        if self.kernel == 'linear':
            K = self.linear_kernel(X, X)
        elif self.kernel == 'rbf':
            K = self.rbf_kernel(X, X)
        else:
            raise ValueError(f"Unknown kernel: {self.kernel}")

        # Matriks P, q, G, h, A, b untuk CVXOPT
        # P = outer_product(y) * K
        P = matrix(np.outer(y, y) * K)

        # q = vektor -1
        q = matrix(-np.ones((n_samples, 1)))
```

```
        # G dan h untuk constraint: 0 <= alpha <= C
        # -alpha <= 0  --> G_std = -I, h_std = 0
        # alpha <= C   --> G_slack = I, h_slack = C
        G_std = -np.eye(n_samples)
        G_slack = np.eye(n_samples)
        G = matrix(np.vstack((G_std, G_slack)))

        h_std = np.zeros(n_samples)
        h_slack = np.ones(n_samples) * self.C
        h = matrix(np.hstack((h_std, h_slack)))

        # A dan b untuk constraint: sum(alpha * y) = 0
        A = matrix(y.T)
        b = matrix(np.zeros(1))

        # QP menggunakan Solver
        solvers.options['show_progress'] = False
        solution = solvers.qp(P, q, G, h, A, b)

        # Ambil nilai alpha (Lagrange multipliers)
        alphas = np.ravel(solution['x'])

        # Filter Support Vectors (alpha > threshold kecil mendekati 0)
        idx = alphas > 1e-5
        self.alphas = alphas[idx]
        self.support_vectors = X[idx]
        self.support_vector_labels = y[idx]

        # Hitung Bias (b)
        # b = mean(y_k - sum(alpha_i * y_i * K(x_i, x_k))) untuk semua support vector k
        bias_list = []
        for i in range(len(self.alphas)):
            pred_val = 0
            for j in range(len(self.alphas)):
                if self.kernel == 'linear':
                    k_val = np.dot(self.support_vectors[j], self.support_vectors[i])
                else:  # rbf
                    k_val = np.exp(-self.gamma * np.linalg.norm(self.support_vectors[j]
- self.support_vectors[i])**2)
                pred_val += self.alphas[j] * self.support_vector_labels[j] * k_val

            bias_list.append(self.support_vector_labels[i] - pred_val)

        # rata-rata bias
        self.b = np.mean(bias_list) if bias_list else 0
```

Alur Proses:

1. Menghitung Matriks Gram (Kernel Matrix) menggunakan fungsi kernel yang dipilih.
2. Menyusun matriks-matriks standar (P, q, G, h, A, b) yang diperlukan oleh solver QP cvxopt untuk meminimalkan fungsi objektif kuadratik dengan kendala.

$$\min_x \ \frac{1}{2}x^T Px + q^T x$$

Dengan syarat $Gx \leq h, \quad Ax = b$

- Matriks P: Merepresentasikan komponen kuadratik fungsi objektif yang dikonstruksi dari elemen $y_i y_j K(x_i, x_j)$ menggunakan matriks kernel dan label kelas.
- Vektor q: Berisi nilai −1 pada setiap elemennya untuk mengubah masalah asli maksimisasi margin menjadi masalah minimisasi standar.
- Matriks G: Matriks koefisien pertidaksamaan yang disusun dari tumpukan matriks identitas negatif (−I) dan positif (I) untuk merepresentasikan sisi kiri batasan rentang nilai α.
- Vektor h: Vektor batas pertidaksamaan yang berisi nilai 0 (untuk batas bawah) dan C (untuk batas atas), menetapkan rentang valid soft margin ($0 \leq \alpha_i \leq C$).
- Matriks A: Matriks koefisien persamaan linear yang berisi transpose dari vektor label kelas (y^T).
- Vektor b: Skalar bernilai 0 yang menjadi target dari kendala linear $\sum y_i \alpha_i = 0$ untuk menjamin validitas geometris solusi.

3. Menjalankan solver cvxopt.solvers.qp untuk mendapatkan nilai optimal Lagrange Multipliers (α).
4. Menyaring nilai α untuk mendapatkan Support Vectors (data dengan α > 0).
5. Menghitung nilai bias (b) dengan merata-ratakan selisih prediksi support vectors terhadap label aslinya.

Fungsi (self, X)

Fungsi untuk memprediksi label kelas pada data baru.

```
def predict(self, X):
        y_pred = []
        # Prediksi: sign(sum(alpha_i * y_i * K(x_i, x_new)) + b)
        for x in X:
            prediction = 0
            for i in range(len(self.alphas)):
                # Kernel antara data baru (x) dan support vector (sv)
                if self.kernel == 'linear':
                    k_val = np.dot(self.support_vectors[i], x)
```

```
                else:  # rbf
                    k_val = np.exp(-self.gamma * np.linalg.norm(self.support_vectors[i]
- x)**2)
                prediction += self.alphas[i] * self.support_vector_labels[i] * k_val

            prediction += self.b
            y_pred.append(np.sign(prediction))

        return np.array(y_pred)
```

Mengklasifikasikan data input X berdasarkan decision function yang dibentuk oleh support vectors, nilai alpha, dan bias yang telah dipelajari. Mengembalikan nilai +1 atau -1 menggunakan fungsi sign().

$$f(x) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i K(x_i, x) + b\right)$$

Kelas DAGSVM

Kelas ini adalah *wrapper* (pembungkus) yang memperluas kemampuan SVM biner menjadi klasifikasi multi-kelas (*Multi-class classification*) menggunakan strategi *Directed Acyclic Graph* (DAG).

```
class DAGSVM(BaseClassifier):
    def __init__ (self, C=1.0, gamma=0.1, kernel='rbf'):
        self.C = C
        self.gamma = gamma
        self.kernel = kernel
        self.classifiers = {}
        self.classes = None
```

__init__(self, C=1.0, gamma=0.1, kernel='rbf')

- Deskripsi: Konstruktor untuk menginisialisasi parameter model multi-kelas.
- Fungsi: Menyimpan parameter SVM yang akan digunakan oleh semua sub-model biner dan menyiapkan struktur data (dictionary) untuk menyimpan kumpulan classifier yang telah dilatih

Fungsi fit(self, X, y)

Fungsi untuk melatih model multi-kelas menggunakan strategi *One-against-One*.

```
def fit(self, X, y):
        self.classes = np.unique(y)
        n_classes = len(self.classes)

        # Melatih SVM untuk setiap pasangan kelas
        for i in range(n_classes):
            for j in range(i + 1, n_classes):
                class_i = self.classes[i]
                class_j = self.classes[j]

                # Filter data untuk kelas i dan j
                idx = np.where((y == class_i) | (y == class_j))
```

```
                X_ij = X[idx]
                y_ij = y[idx]

                # Ubah label menjadi +1 dan -1
                y_ij = np.where(y_ij == class_i, 1, -1)

                # Latih SVM
                svm_ij = SVM_QP(C=self.C, gamma=self.gamma, kernel=self.kernel)
                svm_ij.fit(X_ij, y_ij)

                # Simpan classifier
                self.classifiers[(class_i, class_j)] = svm_ij
```

Alur Proses:

1. Mengidentifikasi jumlah kelas unik dalam data.
2. Melakukan iterasi untuk setiap pasangan kelas yang mungkin.
3. Untuk setiap pasangan, data disaring dan label dikonversi sementara menjadi biner (+1 dan -1).
4. Melatih satu instans SVM_QP untuk setiap pasangan tersebut dan menyimpannya dalam memori.

Fungsi predict(self, X)

Fungsi untuk memprediksi kelas data baru menggunakan struktur graf (DAG).

```
def predict(self, X):
        y_pred = []

        for x in X:
            candidates = list(self.classes)

            while len(candidates) > 1:
                # Ambil kandidat Paling Kiri (Awal) dan Paling Kanan (Akhir)
                c_first = candidates[0]
                c_last = candidates[-1]

                if (c_first, c_last) in self.classifiers:
                    model = self.classifiers[(c_first, c_last)]
                    pred = model.predict(x.reshape(1, -1))

                    if pred == 1:
                        candidates.pop(-1)
                    else:
                        candidates.pop(0)

                elif (c_last, c_first) in self.classifiers:
                    model = self.classifiers[(c_last, c_first)]
                    pred = model.predict(x.reshape(1, -1))

                    if pred == 1:
                        candidates.pop(0)
                    else:
                        candidates.pop(-1)
                else:
                    raise ValueError(f"Model untuk pasangan {c_first}-{c_last} tidak
```

```
ditemukan!")
            y_pred.append(candidates[0])
        return np.array(y_pred)
```

Alur Proses:

1. Memulai dengan daftar kandidat yang berisi seluruh kelas.
2. Mengambil kelas "Paling Kiri" dan "Paling Kanan" dari daftar kandidat.
3. Menggunakan model SVM spesifik yang terlatih untuk pasangan tersebut guna melakukan prediksi.
4. Berdasarkan hasil prediksi (siapa yang menang), kelas yang kalah dieliminasi dari daftar kandidat.
5. Proses berulang hingga hanya tersisa satu kelas, yang kemudian dikembalikan sebagai hasil prediksi.

2. Hasil Prediksi Algoritma dan Perbandingan dengan Pustaka

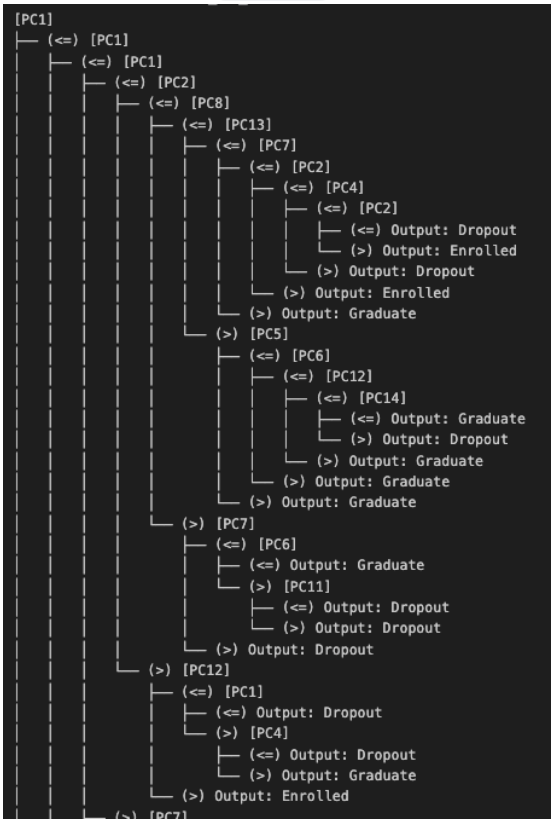
2.1. Hasil Pengujian Decision Tree Learning

Pengujian dilakukan untuk membandingkan performa model ID3 Decision Tree (From Scratch) yang telah diimplementasikan dengan model pembanding dari pustaka Scikit-Learn (DecisionTreeClassifier).

Experiment 1 (max_depth = None)

```
Comparison of Results:
              Model      Acc       F1
Scratch (ID3) 0.677419 0.613745
Sklearn (ID3) 0.669355 0.609261
```

None

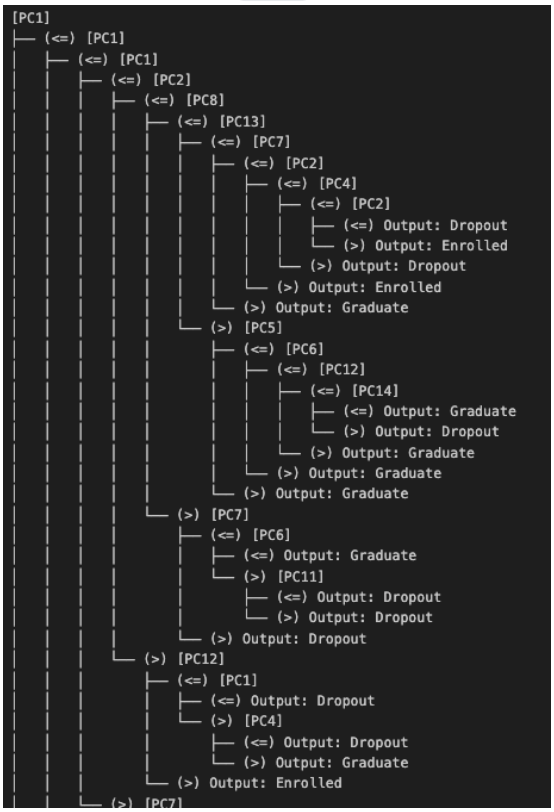


dan seterusnya

Experiment 2 (max_depth = 10)

```
Comparison of Results:
        Model      Acc       F1
Scratch (ID3) 0.674194 0.609469
Sklearn (ID3) 0.666129 0.606064
```

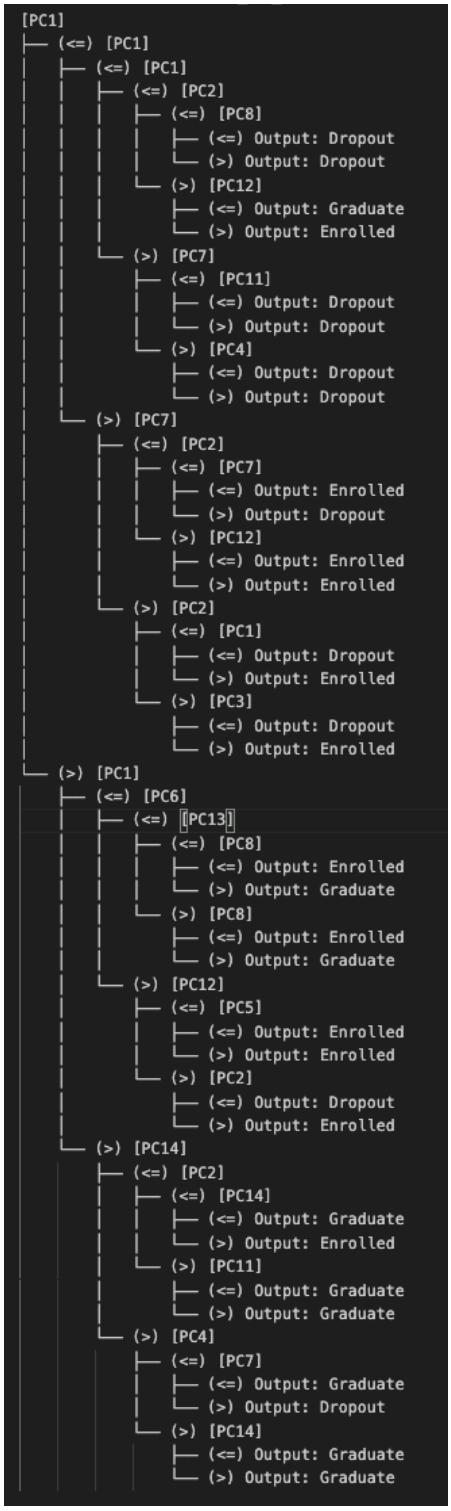
10



dan seterusnya

Experiment 3 (max_depth = 5)

```
Comparison of Results:
        Model      Acc       F1
Scratch (ID3) 0.674194 0.636025
Sklearn (ID3) 0.695161 0.648640
```



Experiment 4 (max_depth = 3)

```
Comparison of Results:
        Model      Acc       F1
Scratch (ID3) 0.706452 0.637797
Sklearn (ID3) 0.688710 0.634267
```

```
[PC1]
├── (<=) [PC1]
│   ├── (<=) [PC1]
│   │   ├── (<=) Output: Dropout
│   │   └── (>) Output: Dropout
│   └── (>) [PC7]
│       ├── (<=) Output: Enrolled
│       └── (>) Output: Dropout
└── (>) [PC1]
    ├── (<=) [PC6]
    │   ├── (<=) Output: Graduate
    │   └── (>) Output: Enrolled
    └── (>) [PC14]
        ├── (<=) Output: Graduate
        └── (>) Output: Graduate
```

2.2. Hasil Pengujian Logistic Regression

Pengujian ini bertujuan membandingkan performa model Scratch melawan Scikit-Learn secara adil (apple-to-apple) menggunakan skema Grid Search 3x3 (9 variasi). Parameter yang diuji adalah kombinasi Learning Rate (0.01, 0.05, 0.1) dan Max Iterations (1000, 5000, 10000).

Agar perbandingan berimbang, setiap kali model Scratch dilatih dengan batas iterasi tertentu, model Scikit-Learn (tanpa learning rate manual) dipaksa menggunakan batas iterasi (max_iter) yang sama persis. Hasil akhirnya dievaluasi secara berdampingan berdasarkan Akurasi dan F1-Score Macro untuk melihat efisiensi dari implementasi manual.

Experiment

```
======================================================================
FINAL COMPARISON: SKLEARN VS SCRATCH (9 VARIATIONS)
======================================================================
             Model         Hyperparams  Accuracy  F1-Macro
Sklearn (Baseline)        solver=lbfgs  0.750000  0.701960
           Scratch  lr=0.05, iter=5000  0.748387  0.700616
           Scratch lr=0.05, iter=10000  0.748387  0.700616
           Scratch   lr=0.1, iter=5000  0.748387  0.700616
           Scratch  lr=0.1, iter=10000  0.748387  0.700616
           Scratch lr=0.01, iter=10000  0.748387  0.699415
           Scratch   lr=0.1, iter=1000  0.748387  0.699415
           Scratch  lr=0.01, iter=5000  0.748387  0.699094
           Scratch  lr=0.05, iter=1000  0.748387  0.699094
           Scratch  lr=0.01, iter=1000  0.748387  0.693040
```

2.3. Hasil Pengujian SVM

Experiment 1 C=1.0, gamma=0.1,

```
Comparison of Results:
           Model      Acc       F1
Scratch (Linear) 0.738710 0.689072
   Scratch (RBF) 0.756452 0.709195
Sklearn (Linear) 0.741935 0.695032
   Sklearn (RBF) 0.753226 0.704981
```

Experiment 2 C=1.0, gamma=0.01

```
Comparison of Results:
           Model      Acc       F1
Scratch (Linear) 0.738710 0.689072
   Scratch (RBF) 0.756452 0.713282
Sklearn (Linear) 0.741935 0.695032
   Sklearn (RBF) 0.745161 0.701819
```

Experimen 3 C=0.1, gamma=0.01,

```
Comparison of Results:
           Model      Acc       F1
Scratch (Linear) 0.753226 0.704654
   Scratch (RBF) 0.758065 0.715928
Sklearn (Linear) 0.748387 0.701097
   Sklearn (RBF) 0.733871 0.679904
```

Eksperimen 4

```
=== 2. Hyperparameter Tuning for DAGSVM (Scratch) ===
Testing 16 combinations...
Training C=100, gamma=1.......
Done testing 16 combinations.

Top 5 Configurations (SVM):
      C  gamma  accuracy
1   0.1   0.01  0.758065
9  10.0   0.01  0.758065
6   1.0   0.10  0.756452
2   0.1   0.10  0.756452
5   1.0   0.01  0.756452

Best Params: C=0.1, gamma=0.01
Best Validation Accuracy: 0.7581
```

2.4. Pembahasan dan Insight yang Didapatkan

2.4.1 Decision Tree Learning

1. Performa Kompetitif (Scratch vs Library)

Implementasi algoritma from scratch terbukti valid dan mampu bersaing dengan standar industri. Pada data validasi, model scratch mencatatkan akurasi 70.65%, sedikit lebih unggul dibandingkan Scikit-Learn (68.87%). Hal ini menunjukkan bahwa logika Entropy dan strategi penentuan threshold menggunakan percentile sampling bekerja sangat efektif untuk menangkap pola dataset ini.

2. Dampak Preprocessing pada Struktur Pohon

Visualisasi pohon keputusan menunjukkan dominasi percabangan numerik (misal: PC1 <= 0.53) tanpa percabangan kategorikal. Fenomena ini adalah dampak langsung dari penerapan PCA dan Feature Encoding yang mengubah fitur nyata menjadi komponen varians matematis. Meskipun mengurangi interpretabilitas manusia, transformasi ini sukses menyederhanakan ruang fitur yang kompleks bagi algoritma ID3.

3. Fleksibilitas dan Robustness Model

Model menunjukkan fleksibilitas tinggi saat menghadapi skenario prediksi data uji (test set) yang masih mentah (raw). Kemampuan fungsi fit untuk beradaptasi secara otomatis terhadap data baru (93 kolom raw vs 83 kolom processed) tanpa pipeline eksternal membuktikan bahwa implementasi ini cukup robust dan adaptif terhadap perubahan format input.

4. Tantangan Ketidakseimbangan Kelas

Nilai F1-Score (Macro) yang konsisten berada di kisaran 63% (di bawah akurasi 70%) mengindikasikan tantangan dalam mendeteksi kelas minoritas (Enrolled). Karakteristik kelas ini yang berada di "zona abu-abu" antara Graduate dan Dropout membuat batas keputusan sulit ditarik secara tegas oleh model pohon tunggal, meskipun teknik oversampling telah diterapkan.

2.4.2 Logistic Regression

1. Performa Model (Akurasi & F1-Score)

   Berdasarkan hasil pengujian pada data validasi, model Scratch dengan konfigurasi terbaik mampu menghasilkan akurasi dan F1-Score yang sangat kompetitif dibandingkan Scikit-Learn. Selisih performa (gap) yang kecil

(umumnya < 2-3%) menunjukkan bahwa implementasi manual Softmax Regression telah berhasil menangkap pola data dengan baik, meskipun menggunakan metode optimasi yang jauh lebih sederhana dibandingkan pustaka standar.

2. Dampak Jumlah Iterasi (Convergence Analysis)

   Terlihat pola yang konsisten di mana performa model Scratch meningkat seiring bertambahnya jumlah iterasi (dari 1000 ke 10000). Hal ini mengindikasikan bahwa algoritma Gradient Descent membutuhkan langkah yang cukup banyak untuk mencapai titik minimum global pada fungsi loss. Sebaliknya, model Scikit-Learn cenderung sudah stabil bahkan pada iterasi rendah karena penggunaan solver LBFGS yang konvergen jauh lebih cepat.

3. Pengaruh Learning Rate pada Model Scratch

   Hasil menunjukkan sensitivitas model Scratch terhadap parameter learning rate. Nilai learning rate yang terlalu kecil menyebabkan konvergensi lambat (underfitting pada iterasi rendah), sedangkan nilai yang terlalu besar berisiko menyebabkan osilasi atau overshooting. Konfigurasi learning rate moderat (misalnya 0.05 atau 0.1) dengan iterasi tinggi umumnya memberikan keseimbangan terbaik antara kecepatan belajar dan stabilitas.

4. Perbandingan Metode Optimasi (Solver)

   Keunggulan tipis Scikit-Learn terutama disebabkan oleh penggunaan solver Quasi-Newton (LBFGS) yang memanfaatkan informasi orde kedua (Hessian approximation) untuk menentukan arah update bobot yang lebih presisi. Sementara itu, model Scratch yang mengandalkan First-Order Gradient Descent murni bekerja secara linear dan membutuhkan lebih banyak epoch untuk mencapai tingkat akurasi yang setara.

5. Kesimpulan Validasi

   Secara keseluruhan, eksperimen ini membuktikan validitas kode Scratch. Kemampuannya untuk mendekati performa Scikit-Learn (ketika regularisasi dinonaktifkan/disetarakan) mengonfirmasi bahwa seluruh komponen matematis—mulai dari One-Hot Encoding, fungsi Softmax, hingga perhitungan gradien—telah diimplementasikan dengan benar secara algoritmik.

3. Kesimpulan dan Saran

3.1. Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan, preprocessing yang komprehensif terbukti sangat berpengaruh terhadap performa model. Tahapan yang dilakukan meliputi outlier handling, feature engineering, scaling, encoding, imbalanced dataset handling, normalization, dan dimensionality reduction berhasil mereduksi 83 fitur menjadi 14 komponen utama dengan tetap mempertahankan 95% informasi. Feature engineering, khususnya pembuatan interaction features seperti approval rate dan grade improvement, terbukti efektif dalam meningkatkan kualitas prediksi dengan memberikan insight akademik yang lebih bermakna dibandingkan fitur individual.

Dari sisi pengembangan algoritma, implementasi mandiri atau from scratch untuk ketiga model, yaitu Decision Tree (ID3), Logistic Regression (Softmax), dan Support Vector Machine (DAGSVM), terbukti valid secara matematis dan mampu menghasilkan performa yang kompetitif dibandingkan pustaka standar Scikit-Learn. Pada pengujian Decision Tree, model scratch menunjukkan akurasi yang sedikit lebih unggul dalam beberapa skenario, meskipun struktur pohon yang dihasilkan didominasi oleh percabangan numerik akibat transformasi PCA yang mengurangi interpretabilitas manusia namun menyederhanakan ruang fitur bagi algoritma. Sementara itu, model Logistic Regression membuktikan bahwa implementasi Gradient Descent manual mampu menyamai akurasi solver LBFGS milik pustaka standar ketika diberikan iterasi yang cukup hingga 10.000 iterasi, meskipun kalah dalam hal efisiensi waktu konvergensi .

Lebih lanjut, algoritma SVM memperlihatkan keunggulan signifikan dalam menangani data non-linear, khususnya melalui penggunaan kernel Radial Basis Function (RBF) yang secara konsisten mengungguli kernel linear. Implementasi DAGSVM untuk klasifikasi multi-kelas juga bekerja efektif dengan parameter optimal C=0.1 dan gamma=0.01. Meskipun secara umum performa model cukup baik, tantangan utama yang masih dihadapi oleh ketiga algoritma adalah klasifikasi kelas "Enrolled" yang berada di zona abu-abu statistik, tercermin dari nilai F1-Score yang cenderung lebih rendah dibandingkan akurasi global. Secara keseluruhan, kombinasi antara preprocessing yang ketat dan implementasi algoritma yang tepat memungkinkan sistem ini memprediksi status akademik mahasiswa dengan reliabilitas yang baik dan dapat direproduksi.

3.2. Saran

1. Implementasi cross-validation (k-fold) untuk evaluasi model yang lebih *robust* dan mengurangi bias dari single train-test split.
2. Analisis feature importance lebih mendalam untuk memahami fitur mana yang paling berpengaruh terhadap prediksi dan mempertimbangkan feature selection yang lebih agresif.
3. Mempertimbangkan *class weights* sebagai alternatif atau pelengkap SMOTE untuk handling imbalanced dataset.

4. Pembagian Tugas

Nama Anggota	Pembagian Tugas
Nicholas Andhika Lucas (13523014)	1. Membuat implementasi EDA 2. Membuat implementasi Split Training Set and Validation Set 3. Membuat implementasi Data Cleaning and Processing 4. Membuat implementasi Data Preprocessing: Labeling and Encoding 5. Membuat laporan: struktur, penjelasan tahap data cleaning dan data preprocessing
Shanice Feodora Tjahjono (13523097)	1. Membuat implementasi Data Preprocessing: Feature Scaling, Data Normalization, Dimensionality Reduction, dan Pipeline 2. Menguji dan memperbaiki tahap Data Cleaning 3. Membuat laporan: penjelasan tahap Data Preprocessing
Jonathan Kenan Budianto (13523139)	1. Implementasi model Decision Tree Learning 2. Membuat laporan: Penjelasan implementasi Decision Tree Learning, Penjelasan hasil uji coba Decision Tree Learning 3. Uji coba model Decision Tree Learning
Mahesa Fadhillah Andre (13523140)	1. Implementasi model Logistic Regression 2. Membuat laporan: Penjelasan implementasi Logistic Regression, Penjelasan hasil uji coba Logistic Regression 3. Uji coba model Logistic Regression
Muhammad Farrel Wibowo (13523153)	1. Implementasi model SVM 2. Membuat laporan: Penjelasan

	implementasi SVM, Penjelasan hasil uji coba SVM 3. Uji coba model SVM

5. Referensi

Maulidevi, N. U. "Modul: Decision Tree Learning (DTL)." IF3170 Inteligensi Artifisial. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung .

Khodra, M. L. "Logistic Regression: What & Why." IF3170 Inteligensi Artifisial. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung .

Ruskanda, F. Z. "Modul: Supervised Learning - 01 SVM: What & Why?" IF3170 Inteligensi Artifisial. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung .

Khodra, M. L. "Regression." IF3170 Inteligensi Artifisial. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung .