BAB II

METODOLOGI PENELITIAN

2.1 Objek Penelitian

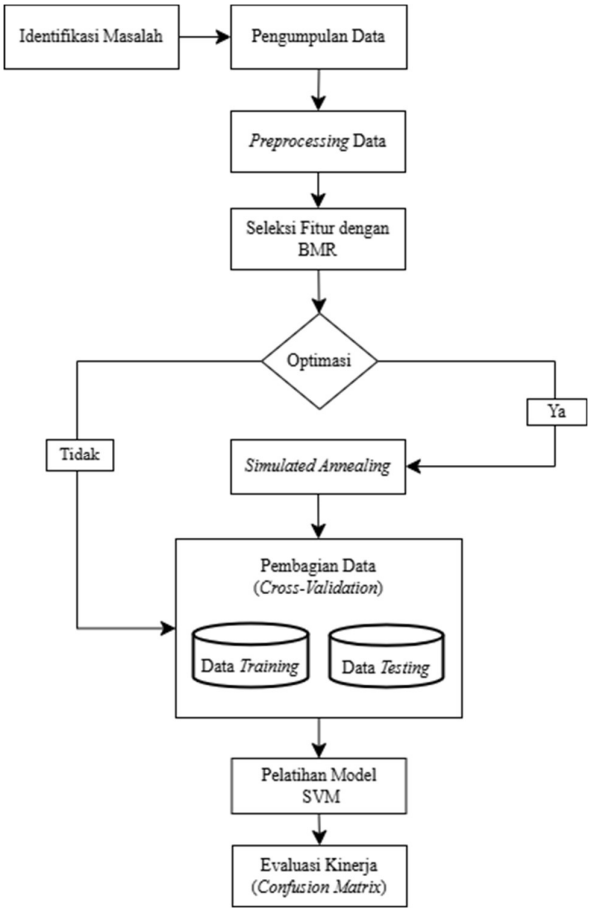
Data yang digunakan dalam penelitian ini adalah data stunting yang bersumber dari Dinas Kesehatan Jl. Milono No.1, Bugis, Kecamatan Samarinda Kota, Kota Samarinda, Kalimantan Timur 76112. Stunting adalah kondisi dimana pertumbuhan fisik seseorang, terutama anak-anak, terhambat sehingga tinggi badan tidak sesuai dengan usia mereka. Data diperoleh dari hasil pemeriksaan pasien setiap harinya baik yang mengidap penyakit stunting atau tidak. Dari 26 Puskesmas di Kota Samarinda dengan jumlah total ukur tinggi badan berdasarkan usia yaitu 15,285 jiwa seperti pada tabel berikut:

Tabel 2. 1 Sumber Data Dinas Kesehatan Kota Samarinda

No	Puskesmas	TB/U Jml Diukur
1	Palaran	351
2	Bantuas	254
3	Bukuan	560
4	Sidomulyo	442
5	Samarinda Kota	670
6	Sungai Kapih	31
7	Makroman	405
8	Sambutan	716
9	Kampung Baka	568
10	Mangkupalas	499
11	Harapan Baru	781
12	Trauma Center	977
13	Loa Bakung	1125
14	Karang Asam	630
15	Lok Bahu	456
16	Wonorejo	690
17	Pasundan	382
18	Air Putih	1628
19	Juanda	349
20	Segiri	778
21	Lempake	62
22	Sei Siring	382
23	Sempaja	379
24	Bengkuring	1080
25	Remaja	842
26	Temindung	248
	Total	15285

2.2 Prosedur Penelitian

Proses penelitian yang dilakukan secara terstruktur untuk merencanakan, melaksanakan, dan mengevaluasi suatu penelitian berdasarkan model yang digunakan, untuk memastikan bahwa penelitian berjalan lancar dan menghasilkan hasil yang dapat dipercaya. Alur penelitian ini tertera pada gamber diagram berikut:



Gambar 2. 1 Diagram Alur Penelitian

2.2.1 Identifikasi Masalah

Penanganan masalah klasifikasi data stunting dengan dimensi tinggi dan ketidakseimbangan kelas memerlukan pendekatan yang cermat. Metode SVM telah terbukti efektif dalam menangani masalah ini (J. R. Khan, p. 2021), tetapi untuk meningkatkan kinerjanya, penggunaan *Simulated Annealing* dan *Boundary Margin Relief* dalam seleksi fitur. *Simulated Annealing* digunakan untuk mencari konfigurasi parameter SVM yang optimal, sementara *Boundary Margin Relief* membantu mengidentifikasi fitur-fitur yang paling penting. Kombinasi ketiga metode ini diharapkan dapat meningkatkan akurasi klasifikasi data stunting dan memberikan dasar yang kuat untuk merancang metodologi penelitian yang efektif.

2.2.2 Pengumpulan Data

Teknik pengumpulan data berupa data sekunder, yaitu penggunaan data yang sudah ada dan dikumpulkan oleh Dinas Kesehatan Kota Samarinda dari lembaga lain untuk tujuan tertentu. Data sekunder dapat berupa data dari *survey*, laporan, basis data, rekaman administratif, atau sumber data lainnya yang tersedia untuk umum. Data yang diperoleh terdiri dari 20 kolom dengan total 150,465 *record*. Rincian data yang diperoleh adalah sebagai berikut:

Tabel 2. 2 Informasi Atribut

No	Atribut	Tipe Data	Keterangan
1	Nik	*String*	Nomor Induk Kependudukan
2	Nama	*String*	Nama
3	JK	*String*	Jenis Kelamin
4	Tel Lahir	*Date*	Tanggal Lahir
5	Nama Ortu	*String*	Nama Orang Tua
6	Provinsi	*String*	Provinsi
7	Kab/Kota	*String*	Kabupaten atau Kota
8	Kec	*String*	Kecamatan
9	Puskesmas	*String*	Lokasi Puskesmas
10	Posyandu	*String*	Lokasi Posyandu
11	Total Pengukuran	*Integer*	Total Pengukuran
12	Tanggal Pengukuran	*Integer*	Tanggal Pengukuran
13	Berat	*Integer*	Berat Badan
14	Tinggi	*Integer*	Tinggi Badan
15	BB/U	*Numeric*	Berat Badan Menurut Umur
16	ZS BB/U	*Numeric*	Z Score Berat Badan menurut Umur
17	TB/U	*Numeric*	Tinggi Badan menurut Umur
18	ZS TB/U	*Numeric*	Z Score Tinggi Badan menurut Umur
19	BB/TB	*Numeric*	Berat Badan menurut Tinggi Badan
20	ZS BB/TB	*Numeric*	Z Score Berat Badan menurut Tinggi Badan

Pada Tabel 2.2 fitur akan di seleksi lagi dan menyisakan 14 fitur yaitu Nama, JK, Berat, Tinggi, LiLA, BB/U, ZS BB/U, TB/U, ZS TB/U, BB/TB, ZS BB/TB, Naik Berat Badan, Jml Vit A, dan Tanggal Pengukuran.

Proses pengolahan data menggunakan *Python*, dengan memanfaatkan pustaka *Pandas*, yang merupakan salah satu pustaka paling populer untuk manipulasi dan analisis data yang di mulai dengan mengimpor data dari file CSV yang berjudul *'stunting.csv'*.

```
import pandas as pd
data = pd.read_csv('stunting.csv')
data.head()
```

Gambar 2. 2 Membaca Informasi Data di *Google Colab Python*

Pada Gambar 2.2, perintah tersebut adalah perintah untuk membaca file *stunting.csv* dan menyimpannya ke dalam sebuah *Dataframe* yang dinamakan data. Fungsi *head()* digunakan untuk menampilkan lima baris pertama dari *Dataframe*, memberikan gambaran awal tentang struktur dan isi data yang telah dibaca. Berikut tabel parameter dan keterangannya

Tabel 2. 3 Parameter Informasi Data Srunting

Parameter	Keterangan
pd	Alias untuk modul *Pandas* yang menyediakan fungsi untuk membaca, menulis, dan memanipulasi data.
stunting.csv	Nama file CSV yang akan dibaca
data	Nama variabel yang digunakan untuk menyimpan *Dataframe* yang berisi data dari file CSV.

head()	Fungsi yang digunakan untuk menampilkan lima baris pertama dataset dari *Dataframe*.

2.2.3 Data Preprocessing

Pada tahapan ini adalah serangkaian langkah yang dilakukan untuk membersihkan, menyiapkan, dan mengatur data mentah menjadi format yang sesuai untuk analisis lebih lanjut. Tahapan ini penting untuk memastikan data yang digunakan dalam analisis memiliki kualitas yang baik dan siap digunakan. Berikut adalah beberapa tahapannya:

A. Pembersihan Data

Pada dataset stunting di Kota Samarinda, dilakukan pembersihan data yang mencakup penanganan data yang tidak wajar, misalnya tinggi badan atau berat badan yang *ekstrem*. Penghapusan data yang memiliki nilai #N/A (nilai tidak tersedia) atau tidak memiliki nilai sama sekali, serta penghapusan data yang terduplikasi. Proses ini penting untuk memastikan bahwa data yang digunakan dalam analisis adalah *valid* dan akurat.

```
import pandas as pd

# Membaca data dari file CSV (ganti dengan lokasi file yang sesuai)
data = pd.read_csv('stunting.csv')

# Menghitung jumlah data sebelum penghapusan duplikat
total_data_sebelum = len(data)

# Mengurutkan data berdasarkan kolom "Nama" dan "Tanggal Pengukuran" secara menurun
data = data.sort_values(by=['Nama', 'Tanggal Pengukuran'], ascending=[True, False])

# Menghapus duplikat berdasarkan kolom "Nama" dan mempertahankan yang pertama (yang memiliki tanggal pengukuran terbaru)
data = data.drop_duplicates(subset='Nama', keep='first')

# Menghitung jumlah data setelah penghapusan duplikat
total_data_sesudah = len(data)

# Menyimpan data yang telah dihapus duplikatnya ke file CSV baru
data.to_csv('dataset_remove_duplikat.csv', index=False)

print(f"Total data sebelum penghapusan duplikat: {total_data_sebelum}")
print(f"Total data setelah penghapusan duplikat: {total_data_sesudah}")
```

Gambar 2. 3 Proses Penghapusan Data yang Terduplikat

Sebanyak 150465 *record* ditemukan pada Gambar 2.3 sebagai data terduplikasi dan dihapus dari dataset. Data yang terduplikasi dapat menyebabkan kesalahan dalam analisis karena informasi yang sama muncul lebih dari sekali.

```
dataset = pd.read_csv('dataset_remove_duplikat.csv')
dataset.info()
dataset = data.drop('Jml Vit A', axis=1)

# Menyimpan hasilnya ke file CSV baru
dataset.to_csv('dataset_without_vitA.csv', index=False)
```

Gambar 2. 4 Proses Penghapusan Atribut 'Jml Vit A'

Pada Gambar 2.4 proses penghapusan atribut *'*Jml Vit A*'* karena terdapat banyak data yang kosong atau *error 'NaN'*. Selanjutnya penghapusan data *missing* pada file *'dataset_without_vitA.csv'*

```
# Membaca data dari file CSV atau sumber data lainnya
data = pd.read_csv('dataset_without_vitA.csv')

# Menghitung jumlah data sebelum penghapusan
jumlah_data_sebelum = len(data)

# Menghapus baris yang memiliki setidaknya satu nilai yang hilang
data_tanpa_missing = data.dropna()

# Menghitung jumlah data setelah penghapusan
jumlah_data_sesudah = len(data_tanpa_missing)

# Menyimpan hasilnya ke file CSV baru
data_tanpa_missing.to_csv('nama_file_tanpa_missing.csv', index=False)

# Mencetak jumlah data sebelum dan setelah penghapusan
print(f"Jumlah data sebelum penghapusan: {jumlah_data_sebelum}")
print(f"Jumlah data setelah penghapusan: {jumlah_data_sesudah}")
```

Gambar 2. 5 Proses Penghapusan Data *Missing*

Terdapat 34,199 *record* yang memiliki nilai #N/A, yang juga dihapus karena tidak memberikan informasi yang berguna untuk analisis terlihat pada Gambar 2.5. Data yang memiliki nilai #N/A berarti informasi tersebut tidak tersedia atau hilang, sehingga tidak dapat digunakan untuk mendapatkan hasil analisis yang tepat. Setelah melalui proses pembersihan yang teliti ini, jumlah total data yang tersisa dan siap digunakan dalam analisis stunting di Kota Samarinda adalah 18,396 *record*. Dengan dataset yang sudah dibersihkan, analisis yang dilakukan akan lebih dapat efisien dan memberikan wawasan yang lebih akurat mengenai kondisi stunting di Kota Samarinda.

B. Transformasi Data

Langkah pertama adalah mengubah nilai-nilai atribut kategorikal menjadi bentuk numerik karena *library sklearn*, yang akan digunakan dalam analisis, hanya menerima atribut dengan nilai *numerik.* Misalnya, atribut seperti jenis kelamin, kenaikan berat badan, berat badan menurut umur, dan berat badan menurut tinggi badan harus diubah dari kategori teks menjadi angka. Selain itu, proses standarisasi atau normalisasi dilakukan pada data *numerik* seperti usia anak-anak, tinggi badan, dan berat badan. Standarisasi ini penting untuk memastikan bahwa semua data berada pada skala yang konsisten, sehingga memudahkan pembacaan dan analisis yang akurat. Transformasi variabel kategorikal menjadi numerik dilakukan dengan menggunakan fungsi *LabelEncoder* dari *library scikit-learn. LabelEncoder* adalah fungsi yang mengubah variabel kategorikal, seperti jenis kelamin dan berat badan berdasarkan umur, menjadi angka yang dapat dimengerti oleh algoritma *machine learning.*

```
# Proses Transformasi Data
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
ordinal = OrdinalEncoder()
labelencoder = LabelEncoder()

df_transform = stunting[['JK','Berat','Tinggi','LiLA','BB/U','ZS BB/U',
                         'ZS TB/U','Naik Berat Badan','BB/TB', 'ZS BB/TB','Kelas']]

stunting['JK'] = labelencoder.fit_transform(stunting['JK'])
stunting['BB/U'] = labelencoder.fit_transform(stunting['BB/U'])
stunting['BB/TB'] = labelencoder.fit_transform(stunting['BB/TB'])
stunting['Naik Berat Badan'] = labelencoder.fit_transform(stunting['Naik Berat Badan'])

df = stunting[['JK','Berat','Tinggi','LiLA','BB/U','ZS BB/U','ZS TB/U','Naik Berat Badan','BB/TB','ZS BB/TB','Kelas']]
```

Gambar 2. 6 Proses Transformasi Data di *Python*

Pada Gambar 2.6, perintah untuk melakukan transformasi data dengan mengubah kolom kategori dalam *Dataframe* *'*stunting*'* menjadi format numerik menggunakan *'LabelEncoder'*. Kolom yang diproses adalah *'*JK*'*, *'*BB/U*'*, *'*BB/TB*'*, dan *'*Naik Berat Badan*'*. Data yang telah diubah disimpan kembali dalam *Dataframe* *'*stunting*'*, dengan kolom lainnya tetap ada. Hasilnya adalah *Dataframe* *'*df*'* yang siap digunakan untuk analisis lebih lanjut. Berikut tabel parameter dan keterangannya.

Tabel 2. 4 Parameter Proses Transformasi

Parameter	Keterangan
from sklearn.preprocessing import OrdinalEncoder	Mengimpor modul *OrdinalEncoder* dari pustaka *sklearn.preprocessing*, yang digunakan untuk mengkodekan fitur kategori menjadi bilangan bulat.
ordinal OrdinalEncoder()	Membuat objek *OrdinalEncoder* yang akan digunakan untuk mentransformasikan fitur-fitur ordinal dalam *Dataframe*.
LabelEncoder LabelEncoder()	Membuat objek *LabelEncoder* yang akan digunakan untuk mentransformasikan fitur-fitur kategori dalam *Dataframe*.
df_transform	*Dataframe* yang berisi subset kolom-kolom yang akan digunakan untuk proses transformasi.
stunting['JK'] LabelEncoder.fit_transform(stunting['JK'])	Menggunakan *LabelEncoder* untuk mengubah nilai kategori dalam kolom *'*JK*'* menjadi nilai numerik.
stunting['BB/U'] LabelEncoder.fit_transform(stunting['BB/U'])	Menggunakan *LabelEncoder* untuk mengubah nilai kategori dalam kolom *'*BB/U*'* menjadi nilai numerik.
stunting['BB/TB'] LabelEncoder.fit_transform(stunting['BB/TB'])	Menggunakan *LabelEncoder* untuk mengubah nilai kategori dalam kolom *'*BB/TB*'* menjadi nilai numerik.
stunting['Naik Berat Badan'] LabelEncoder.fit_transform(stunting['Naik Berat Badan'])	Menggunakan *LabelEncoder* untuk mengubah nilai kategori dalam kolom *'*Naik Berat Badan*'* menjadi nilai numerik.
df	*Dataframe* yang berisi kolom-kolom hasil transformasi yang telah dilakukan.

C. Data *Balanced*

Teknik-teknik untuk mengatasi masalah saat satu atau beberapa kelas dalam dataset memiliki jumlah sampel yang jauh lebih sedikit dibandingkan kelas lainnya sangat penting dalam pembelajaran mesin. Ketidakseimbangan kelas ini dapat menyebabkan model menjadi cenderung tidak adil, di mana model lebih cenderung memprediksi kelas dengan lebih banyak sampel dan mengabaikan kelas dengan lebih sedikit sampel (*Anggrawan et al*., 2023). Akibatnya, performa model secara keseluruhan menurun, terutama dalam hal prediksi kelas yang kurang terwakili. Salah satu teknik yang efektif untuk menangani masalah ini adalah SMOTE (*Synthetic Minority Over-sampling Technique*), yang bekerja dengan menghasilkan sampel sintetis untuk kelas dengan lebih sedikit sampel, sehingga meningkatkan jumlah dan keberagaman kelas tersebut dalam dataset stunting.

```
X1 = df.drop(['Kelas'],axis=1)
Y1 = df['Kelas']
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, sampling_strategy=1)
X_imb, y_imb = sm.fit_resample(X1, Y1)
```

Gambar 2. 7 Proses Penanganan Ketidakseimbanagan Kelas

Implementasi pada Gambar 2.7 bertujuan untuk mengatasi masalah ketidakseimbangan kelas dengan menambahkan sampel sintetis pada kelas minoritas, sehingga meningkatkan performa model dalam memprediksi kelas yang kurang terwakili. Berikut adalah tabel parameter dan penjelasan dari setiap langkah yang dilakukan dalam proses tersebut:

Tabel 2. 5 Parameter Proses Ketidakseimbanagan Kelas

Parameter	Penjelasan
X1 = df.drop(['Kelas'], axis=1)	Membuat *Dataframe* X1 yang berisi semua kolom dari df kecuali kolom *'*Kelas*'*.
Y1 = df['Kelas']	Membuat seri Y1 yang berisi kolom *'*Kelas*'* dari *Dataframe* df.
from imblearn.over_sampling import SMOTE	Mengimpor fungsi SMOTE dari *library imblearn* untuk *oversampling* kelas minoritas.
sm = SMOTE(random_state=42, sampling_strategy=1)	Membuat objek SMOTE dengan seed random 42 dan strategi *sampling* 1 (kelas seimbang 1:1).
X_imb, y_imb = sm.fit_resample(X1, Y1)	Menerapkan SMOTE pada data X1 dan Y1, menghasilkan data baru *X_imb dan y_imb* yang seimbang.

Pada Tabel 2.5 kolom *'Kelas'* dihapus dari *Dataframe* untuk membuat *Dataframe* fitur *'X1'*, sementara kolom 'Kelas' tersebut disimpan sebagai label *'*Y1*'*. Selanjutnya, modul SMOTE diimpor dari *library 'imblearn.over_sampling',* dan objek SMOTE dibuat dengan menetapkan seed acak *'random_state=42'* dan strategi sampling *'sampling_strategy=1',* yang berarti kelas dengan sampel lebih sedikit dan kelas dengan sampel lebih banyak akan seimbang setelah *resampling*. Kemudian, metode *'fit_resample'* dari objek SMOTE diterapkan pada data fitur dan label, menghasilkan dataset

yang baru *('X_imb' dan 'y_imb')* yang telah *diresample* sehingga kelas dengan sampel lebih sedikit dan kelas dengan sampel lebih banyak menjadi seimbang.

2.2.4 Seleksi Fitur Menggunakan BMR

Seleksi fitur adalah langkah penting dalam pemrosesan data yang bertujuan untuk meningkatkan performa model prediktif dengan mengurangi jumlah fitur yang tidak relevan atau kurang signifikan. Salah satu metode yang dapat digunakan untuk seleksi fitur adalah *Boundary Margin Relief*. *Boundary Margin Relief* adalah teknik yang mengukur kepentingan setiap fitur berdasarkan margin batas antara kelas-kelas yang berbeda dalam dataset. Teknik ini membantu dalam mengidentifikasi fitur-fitur yang paling berpengaruh dalam klasifikasi, dengan mempertimbangkan jarak antara sampel dari kelas yang sama dan kelas yang berbeda. Berikut ini adalah implementasi dari metode *Boundary Margin Relief* yang digunakan untuk melakukan seleksi fitur pada dataset:

```
def boundary_margin_relief(X, y, feature_names):
    weights = np.zeros(X.shape[1])
    for i in range(len(X)):
        # Menghitung jarak ke semua sampel lain
        distances = np.linalg.norm(X - X[i], axis=1)
        # Mengabaikan jarak ke dirinya sendiri dengan mengatur ke nilai besar
        distances[i] = np.inf
        # Mencari tetangga terdekat dalam kelas yang sama (hit) dan kelas berbeda (miss)
        same_class_indices = np.where(y == y[i])[0]
        diff_class_indices = np.where(y != y[i])[0]
        nearest_hit = same_class_indices[np.argmin(distances[same_class_indices])]
        nearest_miss = diff_class_indices[np.argmin(distances[diff_class_indices])]
        weights += np.abs(X[i] - X[nearest_miss]) - np.abs(X[i] - X[nearest_hit])
    return weights

# Menggunakan fungsi yang telah dimodifikasi
feature_names = X.columns.tolist()
weights = boundary_margin_relief(X_scaled, y, feature_names)

# Mengurutkan fitur berdasarkan bobot
sorted_indices = np.argsort(weights)[::-1]
sorted_weights = weights[sorted_indices]
sorted_feature_names = [feature_names[i] for i in sorted_indices]
```

```
# Menampilkan fitur terbaik beserta bobotnya
for feature, weight in zip(sorted_feature_names[:n_features_to_display], sorted_weights[:n_features_to_display]):
    print(f"Feature: {feature}, Bobot: {weight}")
```

Gambar 2. 8 Penerapan *Boundary Margin Relief* (BMR) Untuk Bobot Fitur

Pada Gambar 2.8 menunjukkan penggunaan *Boundary Margin Relief* untuk menyeleksi fitur penting dalam dataset. Prosesnya dimulai dengan mengimpor *library* seperti *NumPy, Pandas, Matplotlib,* dan *StandardScaler* dari *scikit-learn* untuk *preprocessing* data. Dataset dibaca, dan kolom target *('Kelas')* dipisahkan. Nilai yang hilang diisi dengan rata-rata kolom, dan data distandarisasi menggunakan *StandardScaler.* Fungsi *boundary_margin_relief* menghitung bobot fitur berdasarkan jarak *Euclidean* antara sampel dari kelas yang sama dan berbeda. Jarak ke sampel sendiri diatur ke nilai tak hingga untuk mengabaikannya. Indeks tetangga terdekat dalam kelas yang sama *(nearest_hit)* dan kelas yang berbeda *(nearest_miss)* ditentukan, dan bobot fitur diperbarui dengan selisih jarak tersebut. Setelah bobot dihitung, fitur diurutkan berdasarkan bobot, dan visualisasi dibuat untuk menunjukkan fitur terpenting beserta bobotnya. Informasi ini membantu memahami fitur yang paling berpengaruh dalam klasifikasi.

Tabel 2. 6 Parameter BMR Untuk Menentukan Bobot Fitur

Parameter	Keterangan
X.fillna(X.mean(), inplace=True)	Mengisi nilai yang hilang dalam X dengan rata-rata dari masing-masing kolom.
scaler = StandardScaler()	Membuat *instance StandardScaler* untuk standarisasi data.
X_scaled = scaler.fit_transform(X)	Menstandarisasi fitur data sehingga memiliki skala yang sama.
boundary_margin_relief(X, y, feature_names)	Fungsi untuk menghitung bobot fitur menggunakan algoritma *Boundary Margin Relief.*
np.linalg.norm(X - X[i], axis=1)	Menghitung jarak *Euclidean* antara sampel i dengan semua sampel lainnya.
distances[i] = np.inf	Mengatur jarak ke sampel sendiri menjadi nilai tak terhingga untuk mengabaikannya.
same_class_indices = np.where(y == y[i])[0]	Mendapatkan indeks sampel dengan kelas yang sama.
diff_class_indices = np.where(y != y[i])[0]	Mendapatkan indeks sampel dengan kelas yang berbeda.
nearest_hit	Tetangga terdekat dalam kelas yang sama.
nearest_miss	Tetangga terdekat dalam kelas yang berbeda.
weights += np.abs(X[i] - X[nearest_miss]) - np.abs(X[i] - X[nearest_hit])	Memperbarui bobot fitur berdasarkan selisih jarak ke *nearest_miss* dan *nearest_hit.*
np.argsort(weights)[::-1]	Mengurutkan indeks fitur berdasarkan bobot dalam urutan menurun.

Selanjutnya, proses mengurangi kompleksitas model dan mempercepat proses pelatihan tanpa kehilangan informasi penting yang dapat mempengaruhi performa model. Fungsi ini dapat digunakan untuk fokus pada fitur-fitur yang paling relevan. Berikut implementasi dari fungsi tersebut:

```
# Fungsi boundary margin relief
def boundary_margin_relief(X_train, y_train):
    n_features = X_train.shape[1]
    features_to_remove = np.random.choice(n_features, n_features // 2, replace=False)
    return features_to_remove
```

Gambar 2. 9 Penerapan *Boundary Margin Relief* (BMR) Untuk Menghapus Setengah Fitur

Fungsi *boundary_margin_relief* bertujuan untuk menghapus setengah dari fitur yang ada dalam data *X_train* secara acak. Fungsi ini menghitung jumlah fitur dalam *X_train (n_features*), kemudian secara acak memilih indeks dari setengah fitur tersebut untuk dihapus dan mengembalikan *array* dari indeks fitur yang dipilih *(features_to_remove).*

2.2.5 Optimasi dengan Simulated Annealing

Simulated Annealing akan digunakan untuk mengoptimasi pemilihan fitur dalam proses pembelajaran mesin. Tujuan utamanya adalah untuk menemukan kumpulan fitur yang memberikan kinerja terbaik pada model SVM. *Simulated Annealing* adalah sebuah metode yang terinspirasi dari proses annealing dalam fisika, untuk mencari solusi optimal dalam ruang fitur dengan mengizinkan beberapa solusi yang lebih buruk diterima pada awalnya untuk menghindari terjebak di dalam daerah yang tidak optimal. Fungsi ini mengoptimalkan pemilihan fitur dengan mengevaluasi performa model pada data uji, mengubah suhu (T) dan faktor penurunan suhu (*alpha*) secara *iteratif*, dan memperbarui solusi terbaik berdasarkan skor akurasi.

```
# Simulated Annealing untuk optimasi
def simulated_annealing(X_train, y_train, X_test, y_test, svm, T=1.0, alpha=0.95, max_iter=100):
    best_score = 0.0
    best_features = None
    features_to_remove = []

    for _ in range(max_iter):
        # Pilih fitur yang akan dihapus
        new_features_to_remove = boundary_margin_relief(X_train, y_train)

        # Evaluasi model dengan fitur yang dipilih
        svm_copy = copy.deepcopy(svm)
        svm_copy.fit(X_train[:, [i for i in range(X_train.shape[1]) if i not in new_features_to_remove]], y_train)
        y_pred = svm_copy.predict(X_test[:, [i for i in range(X_test.shape[1]) if i not in new_features_to_remove]])
        score = accuracy_score(y_test, y_pred)

        # Simulated Annealing
        if score > best_score or np.exp((score - best_score) / T) > random.random():
            best_score = score
            best_features = new_features_to_remove
            features_to_remove = new_features_to_remove

        T *= alpha

    return best_score, features_to_remove

# Panggil fungsi simulated annealing untuk optimasi
best_score, selected_features = simulated_annealing(X_train_scaled, y_train, X_test_scaled, y_test, svm)
```

Gambar 2. 10 Penerapan *Simulated Annealing* (SA)

Pada Gambar 2.10 menunjukkan fungsi *simulated_annealing* yang mengoptimalkan pemilihan fitur untuk model SVM menggunakan algoritma *simulated annealing*. Fungsi ini menerima data pelatihan dan pengujian, model SVM, serta parameter temperatur awal, faktor pendinginan, dan jumlah iterasi maksimum. Dalam setiap *iterasi*, fitur yang akan dihapus dipilih dengan *boundary_margin_relief*, lalu model SVM dilatih dan diuji untuk menghitung akurasi. Jika akurasi lebih baik atau diterima berdasarkan probabilitas tertentu, fitur tersebut diterima dan skor diperbarui. Temperatur dikurangi setiap *iterasi* dengan mengalikan T dengan *alpha*. Fungsi ini mengembalikan skor terbaik dan fitur yang dipilih.

Tabel 2. 7 Parameter *Simulated Annealing*

Parameter	Keterangan
X_test , y_test	*X_test Array* fitur dari data pengujian yang digunakan untuk menguji model klasifikasi. *y_test Array* label atau kelas target dari data pengujian yang digunakan untuk mengevaluasi performa model klasifikasi

T	Temperatur awal dalam algoritma *Simulated Annealing*. Nilai *defaultnya* adalah 1.0.
alpha	Faktor penurunan temperatur dalam setiap *iterasi* algoritma *Simulated Annealing*. Nilai *defaultnya* adalah 0.95
max_iter	Jumlah iterasi maksimum yang akan dilakukan dalam algoritma *Simulated Annealing*. Nilai *defaultnya* adalah 100.
best_Score	Variabel yang menyimpan nilai akurasi terbaik yang dicapai setelah optimasi.
best_features	Variabel yang menyimpan fitur-fitur terbaik yang dipilih setelah optimasi.
features_to_remove	Variabel yang menyimpan *Array* indeks-indeks fitur yang akan dihapus dari data pelatihan setelah proses optimasi.

2.2.6 Pembagian Data dengan *Cross Validation*

Data stunting akan dibagi menjadi data pelatihan (*training*) dan data uji (*testing*) menggunakan metode *cross-validation*. Pembagian data dengan *cross-validation* yaitu pembagian dataset menjadi subset yang disebut (*folds*), di mana model dilatih pada beberapa *fold* dan diuji pada *fold* lainnya secara bergantian (*R. R. R. Arisandi*, 2022.).

```
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification

# Generate data
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=42)

# Preprocessing data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Inisialisasi Stratified K-Fold Cross-Validation dengan 10 lipatan
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Loop melalui setiap lipatan cross-validation
for fold, (train_index, test_index) in enumerate(kfold.split(X_scaled, y), 1):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Gambar 2. 11 Penerapan *Cross Validation* di *Python*

Proses pembagian data pada Gambar 2.11 dimulai dengan mengimpor pustaka yang diperlukan, yaitu *numpy* dan komponen dari *sklearn* untuk penanganan data dan model, termasuk *StratifiedKFold* untuk validasi silang, serta *StandardScaler* untuk normalisasi data. Data sintetik dikembangkan menggunakan *make_classification* dengan 1000 sampel dan 10 fitur, kemudian dilakukan *preprocessing* dengan *StandardScaler* untuk menyamakan skala fitur-fitur tersebut. Setelah itu, *Stratified K-Fold Cross-Validation* dengan 10 *fold* diinisialisasi, di mana data akan diacak dan dibagi menjadi 10 bagian, memastikan distribusi kelas yang seimbang pada setiap *fold*. Proses ini dilakukan dalam *loop,* di mana setiap *iterasi* mengambil kumpulan data pelatihan dan pengujian berdasarkan indeks yang dihasilkan oleh *StratifiedKFold,* memisahkan data untuk pelatihan dan pengujian dengan cara yang menjaga proporsi kelas di setiap *fold*.

Tabel 2. 8 Parameter *Cross Validation*

Parameter	Keterangan
n_samples=1000	Menentukan jumlah sampel dalam dataset yang dihasilkan.
n_features=10	Menentukan jumlah fitur (atribut) dalam dataset yang dihasilkan.
n_classes=2	Menentukan jumlah kelas dalam dataset, yaitu biner (2 kelas).
random_state=42	Menentukan *seed* untuk pengacakan, yang memastikan hasil yang konsisten dan dapat direproduksi.
n_splits=10	Menentukan jumlah *fold* dalam Stratified *K-Fold cross-validation.*
shuffle=True	Menentukan apakah data harus diacak sebelum dibagi menjadi *fold*.
scaler = StandardScaler()	Inisialisasi objek untuk menstandarisasi fitur data.
X_scaled = scaler.fit_transform(X)	Pengaplikasian standar skala pada data X, mengubah fitur menjadi skala standar (*mean 0, variance 1*).
train_index	Indeks dari sampel yang digunakan untuk pelatihan pada *fold* tertentu.
test_index	Indeks dari sampel yang digunakan untuk pengujian pada *fold* tertentu.
fold	Nomor *fold* yang sedang diproses dalam *loop*, dimulai dari 1.

2.2.7 Pelatihan Model SVM

Support Vector Machine (SVM) adalah algoritma pembelajaran yang digunakan untuk tugas-tugas klasifikasi dan regresi. Tujuan utama dari SVM adalah untuk menemukan *hyperplane* terbaik yang memisahkan dua kelas dalam ruang fitur sedemikian rupa sehingga jarak antara *hyperplane* dan *instance* terdekat dari masing-masing kelas (yang disebut sebagai *support vectors*) sebesar mungkin. Berikut persamaan yang dapat digunakan untuk mendapatkan hyperplane pada SVM.

$$(w \,.\, x_i) + b = 0 \tag{2.1}$$

Data *xi* yang termasuk dalam kelas -1 dapat dirumuskan sebagai berikut:

$$(w \,.\, x_i) + b < 1,\ y_i = -1 \tag{2.2}$$

Data *xi* yang termasuk dalam kelas +1 dapat dirumuskan sebagai berikut:

$$(w \,.\, x_i) + b > 1,\ y_i = 1 \tag{2.3}$$

Keterangan:

x_i = Data ke-i

w = Nilai bobot support vector yang tegak lurus dengan *hyperplane* (*weight vector*)

b = Nilai bias (merujuk pada parameter yang menentukan posisi hyperplane terhadap titik asal (origin) dalam ruang fitur)

y_i = Kelas/label data ke-i

Dalam proses klasifikasi dengan SVM, seringkali terjadi kondisi di mana kernel linear tidak bekerja dengan baik, yang menyebabkan hasil klasifikasi data yang buruk. Kondisi ini dapat diatasi dengan menggunakan kernel *Radial Basis Function* (RBF) (*Al-Mejibli et al*., 2020), yang dirumuskan sebagai berikut:

$$K(x_1, x_2) = \exp(-\gamma \parallel x_1 - x_2 \parallel^2) \quad (2.4)$$

Keterangan:

$K(x_1, x_2)$ = fungsi kernel RBF yang mengukur kesamaan antara dua sampel x_1 dan x_2

$\parallel x_1 - x_2 \parallel$ = Jarak Euclidean kuadrat antara dua vektor fitur x_1 dan x_2

γ (gamma) = Parameter yang menentukan "lebar" fungsi RBF

exp = Fungsi eksponensial

Pelatihan model SVM akan menggunakan karnel RBF, cost = 10, gamma = 5 untuk membuktikan teori dari (Rahmi et al., 2022). Model SVM akan di latih menggunakan data *training* setelah penanganan ketidakseimbangan kelas, *high dimension* dengan BMR dan optimasi dengan *Simulated Annealing.*

```
from sklearn.svm import SVC

# Parameter SVM
C = 10
gamma = 5

# Model SVM
svm = SVC(kernel='rbf', C=C, gamma=gamma)  # kernel 'rbf'
svm.fit(X_train, y_train)

y_pred_svm = svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm) * 100  # Menjadi persen
accuracies_svm.append(accuracy_svm)


# Model SVM dengan SA
accuracy_optimized, best_features = simulated_annealing(X_train, y_train, X_test, y_test)
accuracy_optimized *= 100  # Menjadi persen
accuracies_optimized.append(accuracy_optimized)

svm_optimized = SVC(kernel='rbf', C=C, gamma=gamma)  # kernel 'rbf'
svm_optimized.fit(X_train[:, [i for i in range(X_train.shape[1]) if i not in best_features]], y_train)
y_pred_optimized = svm_optimized.predict(X_test[:, [i for i in range(X_test.shape[1]) if i not in best_features]])
```

Gambar 2. 12 Penerapan Model SVM

Pada Gambar 2.12, terdapat dua proses pemodelan, pertama menggunakan *Support Vector Machine* (SVM) dengan *kernel* RBF untuk membuat, melatih, dan mengevaluasi model. Dengan menggunakan *library Scikit-Learn,* model SVM (SVC) didefinisikan dengan *kernel 'rbf'*. Setelah dilatih dengan data pelatihan (*X_train dan y_train),* model digunakan untuk memprediksi label dari data uji *(X_test)* dan kemudian menghitung akurasinya menggunakan *accuracy_score().* Akurasi hasilnya disimpan dalam list *accuracies_svm.*

Kedua, proses optimasi dilakukan dengan menggunakan *Simulated Annealing* (SA) untuk menentukan fitur terbaik yang akan digunakan dalam model SVM kedua *(svm_optimized).* Setelah dilatih dengan fitur terpilih, model tersebut digunakan untuk membuat prediksi terhadap *X_test,* dan hasilnya dievaluasi untuk mendapatkan akurasi yang dioptimalkan *(accuracy_optimized),* yang juga dimasukkan ke dalam list *accuracies_optimized.*

Tabel 2. 9 Parameter Model SVM

Parameter	Keterangan
SVC	Kelas dari *Scikit-Learn* yang digunakan untuk membuat model *Support Vector Machine*.
kernel='rbf'	Parameter yang menentukan jenis *kernel* yang digunakan oleh model SVM. *'rbf'* adalah *Radial Basis Function*, yang umum digunakan untuk masalah *non-linear.*
X_train, y_train	*X_train* adalah data latih yang digunakan untuk melatih model SVM, sedangkan *y_train* adalah label dari data latih tersebut.
X_test	*X_test* adalah data uji yang digunakan untuk melakukan evaluasi atau prediksi menggunakan model SVM yang sudah dilatih.
accuracy_score(y_test, y_pred_svm)	Fungsi yang digunakan untuk menghitung akurasi prediksi dengan membandingkan label sebenarnya *(y_test)* dengan label yang diprediksi *(y_pred_svm).*
simulated_annealing(X_train, y_train, X_test, y_test)	*simulated_annealing* adalah fungsi yang dioptimalkan untuk memperbaiki model SVM dengan cara *Simulated Annealing.* Mengembalikan akurasi terbaik dan fitur terbaik yang dipilih.

2.2.8 Evaluasi Kinerja Algoritma

Kinerja model SVM yang dioptimalkan akan dievaluasi menggunakan *Confusion Matrix. Confusion Matrix* adalah sebuah tabel yang digunakan dalam evaluasi klasifikasi untuk menampilkan performa model dengan membandingkan hasil prediksi dengan nilai sebenarnya dari data yang diuji.

```
# Confusion matrix
from sklearn.metrics import confusion_matrix

# Menghitung confusion matrix SVM
confusion_matrices_svm.append(confusion_matrix(y_test, y_pred_svm))

# Menghitung confusion matrix SVM dengan SA
confusion_matrices_optimized.append(confusion_matrix(y_test, y_pred_optimized))
```

Gambar 2. 13 Evaluasi Kinerja Model dengan *Confusion Matrix*

Pada Gambar 2.13 proses *confusion matrix* untuk prediksi model SVM standar dihitung dengan membandingkan label sebenarnya *(y_test)* dengan label yang diprediksi oleh model *(y_pred_svm),* dan hasilnya ditambahkan ke daftar confusion_matrices_svm. Selanjutnya, *confusion matrix* untuk model SVM yang dioptimalkan dihitung dengan membandingkan *y_test* dengan prediksi dari model yang dioptimalkan *(y_pred_optimized),* dan hasilnya disimpan dalam daftar *confusion_matrices_optimized*. Confusion Matrix ini memberikan gambaran tentang performa model dalam hal prediksi benar, salah positif, salah negatif, dan salah klasifikasi.

Tabel 2. 10 Parameter *Confusion Matrix*

Parameter	Keterangan
confusion_matrix	Fungsi dari *sklearn.metrics* yang digunakan untuk menghitung confusion matrix dari hasil prediksi model.
y_test	Label sebenarnya dari data uji yang digunakan sebagai referensi untuk menghitung akurasi prediksi model.
y_pred_svm	Label yang diprediksi oleh model SVM standar untuk data uji.
y_pred_optimized	Label yang diprediksi oleh model SVM yang dioptimalkan menggunakan Simulated Annealing (SA) untuk data uji.
confusion_matrices_svm	Daftar (list) yang menyimpan confusion matrix untuk setiap *fold* dari model SVM standar.
confusion_matrices_optimized	Daftar (list) yang menyimpan confusion matrix untuk setiap *fold* dari model SVM yang dioptimalkan dengan SA.

Confusion Matrix adalah alat penting untuk mengukur kinerja model. *Confusion matrix* menunjukkan jumlah prediksi yang benar dan salah dengan membandingkan hasil prediksi model dengan nilai sebenarnya. Berikut adalah tabel *confusion matrix*:

Tabel 2. 11 *Confusion Matrix*

Prediction	***True values***	
	True	***False***
True	TP *Correct result*	FP *Unexpected result*
False	FN *Missing result*	TN *Correct absence of result*

Dalam *confusion matrix*, terdapat beberapa istilah yang digunakan dalam kasus klasifikasi seperti yang dijelaskan oleh (*Anggrawan et al*., 2023).

Keterangan :

True Positive (TP) = Data positif pada confusion matrix yang terdeteksi dengan benar.

False Positive (FP) = Data negatif pada confusion matrix yang salah terdeteksi sebagai data positif.

False Negative (FN) = Data positif pada confusion matrix yang salah terdeteksi sebagai data negatif.

True Negative (TN) = Data negatif pada confusion matrix yang terdeteksi dengan benar

Evaluasi kinerja model yang akan dilakukan dengan mengukur nilai akurasi, berikut rumusnya

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.5)$$

Keterangan :

Accuracy = Mengukur sejauh mana model klasifikasi memberikan prediksi yang benar secara keseluruhan.

Performa model akan dibandingkan dengan metode SVM dengan metode SVM + BMR + *Simulated Annealing* untuk menilai efektivitas kombinasi model.

Berikut adalah kategori akurasi berdasarkan persentase:

Sangat Baik : 90% - 100%
Baik : 80% - 89%
Cukup : 70% - 79%
Kurang : 60% - 69%
Sangat Kurang : < 60%