# BAB II

# METODE PENELITIAN

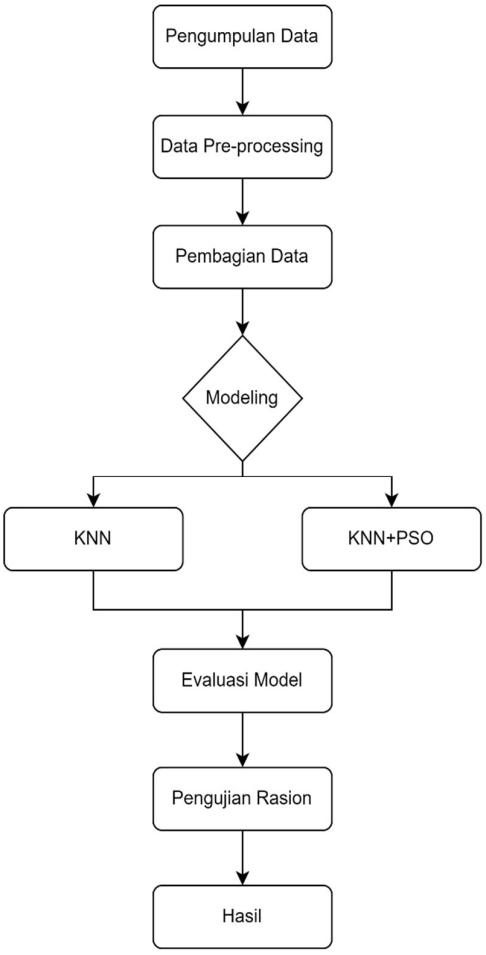
## 2.1 Objek Penelitian

Objek penelitian ini adalah balita di Kota Samarinda yang memiliki beragam status gizi, yang terdiri dari berbagai parameter seperti tinggi, berat, lingkar kepala dan usia. Penelitian ini bertujuan untuk mengklasifikasikan status gizi balita menjadi kategori gizi buruk, gizi kurang, gizi baik atau gizi lebih berdasarkan data yang dikumpulkan.

Data yang digunakan dalam penelitian ini adalah rekam medis hasil pemeriksaan status gizi dan stunting pada balita di Puskesmas Kota Samarinda. Kota Samarinda terdiri dari 10 kecamatan, dengan total 26 puskesmas yang menyelenggarakan pemeriksaan kesehatan balita. Data yang diperoleh mencapai 15.593 data pada periode 3 Agustus 2022 hingga 31 Juli 2023. Data dikategorikan dalam beberapa klasifikasi yang mencakup enam kelas: gizi baik, gizi buruk, gizi kurang, gizi lebih, obesitas, dan makan berlebihan.

## 2.2 Prosedur Penelitian

Penelitian ini melibatkan beberapa langkah untuk mencapai tujuan penelitian. Tahapan penelitian ini dimulai dari pengumpulan dan analisis data hingga tahap akhir. Berikut langkah prosedur penelitian:




**Gambar 2.1** Desain Penelitian

### 2.2.1 Metode Pengumpulan Data

Data yang digunakan dalam penelitian ini adalah rekam medis hasil pemeriksaan status gizi dan *stunting* pada balita di Puskesmas Kota Samarinda. Kota Samarinda terdiri dari 10 kecamatan, dengan total 26 puskesmas yang menyelenggarakan pemeriksaan kesehatan balita. Data yang diperoleh mencapai 15.593 data pada periode 3 Agustus 2022 hingga 31 Juli 2023 dan terdapat sebanyak 17 atribut dan 1 target yaitu "BB/TB". Data target dikategorikan dalam beberapa klasifikasi yang mencakup enam kelas: gizi baik, gizi buruk, gizi kurang, gizi lebih, obesitas, dan resiko gizi lebih. Informasi atribut ditampilkan pada Tabel 2.1

**Tabel 2.1** Data status gizi balita

| No | Atribut | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | JK | *String* | Jenis Kelamin |
| 2 | Tgl Lahir | *String* | Tanggal Lahir |
| 3 | Provinsi | *String* | Provinsi |
| 4 | Kab/Kota | *String* | Kabupaten / Kota |
| 5 | Kec | *String* | Kecamatan |
| 6 | Puskesmas | *String* | Lokasi Puskesmas |
| 7 | Posyandu | *String* | Lokasi Posyandu |
| 8 | Usia saat diukur | *Integer* | Usia balita saat dilakukan pemeriksaan |
| 9 | Berat | *Integer* | Berat Badan |
| 10 | Tinggi | *Integer* | Tinggi Badan |
| 11 | BB/U | *Integer* | Berat Badan menurut Umur |
| 12 | ZS BB/U | *Integer* | Z Score Berat Badan menurut Umur |
| 13 | TB/U | *Integer* | Tinggi Badan menurut Umur (Stunting) |
| 14 | ZS TB/U | *Integer* | Z Score Tinggi Badan menurut Umur |
| 15 | BB/TB | *Integer* | Berat Badan menurut Tinggi Badan (Status Gizi) |
| 16 | ZS BB/TB | *Integer* | Z Score Berat Badan menurut Tinggi Badan |
| 17 | Naik Berat Badan | *Kategorikal* | Tinggi Badan Kenaikan berat badan dibandingkan pemeriksaan sebelumnya |

### 2.2.2 Data *Pre-Processing*

Data dari Dinas Kesehatan Kota Samarinda harus diolah lebih dulu sebelum memasuki tahap pemodelan, yaitu dengan cara data *pre-processing*, tahap *pre-processing* mengacu pada langkah-langkah yang digunakan untuk membersihkan, mengubah dan menyiapkan data mentah sebelum digunakan untuk pemodelan (Mishra et al., 2020), sehingga menghasilkan sebuah data yang lebih relevan (A'yuniyah & Reza, 2023). Langkah-langkah pemrosesan data tersebut meliputi data *Cleaning,* tugas nya adalah menghapus data yang salah, rusak dan tidak lengkap. Jadi data *cleaning* digunakan untuk mengatasi nilai yang hilang. Pada proses ini bisa dilihat pada Gambar 2.2

```
missing_values = data.isna().sum()

# Menampilkan jumlah missing values untuk setiap kolom
print("Jumlah missing values untuk setiap kolom:")
print(missing_values)
data.dropna(inplace=True)
data.drop_duplicates(inplace=True)
```

**Gambar 2.2** Data *cleaning*

Pada Gambar 2.2 menampilkan proses data *cleaning* menggunakan metode '.isna)' untuk mengidentifikasi *missing values* dalam dataset. Setiap elemen dalam dataset yang bernilai 'NaN' atau '*none'* akan dianggap sebagai *missing values*. Kemudian metode '.sum()' digunakan untuk menghitung jumlah *missing* dalam setiap kolom. Selanjutnya pada metode penghapusan menggunakan '.dropna()' dengan parameter 'inplace=True', yang menghapus semua baris yang mengandung *missing values* dan pada metode '.datadrop_duplicates()' menghapus semua baris duplikat pada dataframe

Langkah selanjutnya yaitu data *transformation*, adalah proses mengubah nilai pada data dari *string* menjadi *numerik*. Tujuannya untuk mempermudah proses pemodelan, pada tahap ini data yang akan ditransformasi adalah 'Jenis Kelamin', 'BB/U', 'TB/U', 'BB/TB', 'Naik Berat Badan'. Seperti pada Gambar 2.3

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

# Ubah Jenis Kelamin
data['JK'] = data['JK'].str.strip()
data['JK'] = le.fit_transform(data['JK'])

# Ubah BB/U
data['BB/U'] = le.fit_transform(data['BB/U'])

# Ubah TB/U
data['TB/U'] = le.fit_transform(data['TB/U'])

# Ubah BB/TB
data['BB/TB'] = le.fit_transform(data['BB/TB'])

# Ubah Naik Berat Badan
data['Naik Berat Badan'] = le.fit_transform(data['Naik Berat Badan'])
```

**Gambar 2.3** Data *transformation*

Kode pada Gambar 2.3 merupakan serangkaian langkah untuk mengubah jenis data dalam beberapa kolom khusus dari dataset yang disimpan dalam variabel data. Pertama-tama, dilakukan pengolahan pada kolom 'JK' yang merupakan singkatan dari jenis kelamin. Langkah pertama dalam proses ini adalah menghilangkan spasi yang mungkin terdapat di sekitar nilai-nilai dalam kolom menggunakan metode .str.strip(). Kemudian, nilai-nilai dalam kolom tersebut diubah dari teks menjadi nilai numerik, di mana 'L' yang mewakili laki-laki diganti dengan angka 1, sedangkan 'P' yang mewakili perempuan diganti dengan angka 2. Selanjutnya, terdapat perubahan dalam kolom 'BB/U' yang merupakan kategori berat badan untuk umur. Nilai-nilai kategori teks seperti 'Berat Badan Normal', 'Kurang', 'Sangat Kurang', dan 'Risiko Lebih' diubah menjadi nilai numerik yang lebih mudah diinterpretasikan, yaitu 1, 2, 3, dan 4 secara berurutan. Langkah yang serupa dilakukan pada kolom 'TB/U', yang merupakan kategori tinggi badan untuk umur. Kategori teks seperti 'Pendek' dan 'Sangat Pendek' diubah menjadi nilai numerik 1 dan 2. Terakhir, dalam kolom 'BB/TB' yang mewakili indeks gizi berdasarkan berat badan dan tinggi badan, nilai-nilai teks seperti 'Gizi Baik', 'Gizi Buruk', 'Gizi

Kurang', 'Gizi Lebih', 'Obesitas', dan 'Risiko Gizi Lebih' diubah menjadi nilai *numerik* yang relevan, yaitu 1, 2, 3, 4, 5, dan 6 secara berurutan.

Semua perubahan ini bertujuan untuk menyederhanakan representasi data dan memungkinkan analisis lebih lanjut dengan menggunakan nilai numerik yang terstruktur.

### 2.2.3 Pembagian Data

```
1 from sklearn.model_selection import train_test_split
2
3 x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state = 42)
```

**Gambar 2.4** Pembagian data

Pada Gambar 2.4 adalah proses pembagian data dimana dataset dibagi menjadi dua subset eksklusif: data latih dan data uji. Data latih digunakan untuk melatih model, sedangkan data uji digunakan untuk menguji kinerja model yang sudah dilatih. Pembagian jumlah data menjadi salah satu faktor yang menentukan hasil akurasi yang akan diperolah (Musu et al., 2021). Proporsi pembagian adalah 80% untuk data latih dan 20% untuk data uji. Ini berarti 80% dari dataset akan digunakan untuk melatih model, sementara 20% sisanya akan digunakan untuk menguji model. Penggunaan nilai random_state memastikan konsistensi dalam pembagian data setiap kali kode dijalankan, sehingga hasilnya dapat direproduksi dengan tepat.

### 2.2.4 Modeling

Pada penelitian ini, digunakan sebuah model klasifikasi yang menghubungkan *K-Nearest Neighbors* dengan *Particle Swarm Optimization* sebagai metode optimasi, sebelum proses pemrosesan dataset, data sebelumnya sudah di *pro-processing* terlebih dahulu agar tidak terjadi kesalahan dalam proses pemodelan. Berikut adalah rincian mengenai penggunaan model ini dalam penelitian, tahapan pertama. adalah mempersiapkan dataset status gizi balita.

Pada tahap kedua. melakukan pemodelan menggunakan KNN, dalam implementasi KNN menggunakan *scikit-learn*, langkah-langkah utama mencakup inisialisasi dan evaluasi model. KNN adalah algoritma pembelajaran mesin yang digunakan untuk klasifikasi dan regresi, yang bekerja dengan mengidentifikasi sejumlah k tetangga terdekat dari titik data yang ingin diklasifikasikan. Untuk proses implementasi bisa dilihat pada Gambar 2.5

```
for k in k_values:
    for weight in weight_options:
        for p in p_values:
            knn = KNeighborsClassifier(n_neighbors=k, weights=weight, p=p)
            scores = cross_val_score(knn, x_train, y_train, cv=cv_folds, scoring='accuracy')
            mean_score = scores.mean()

            knn.fit(x_train, y_train)
            test_accuracy = knn.score(x_test, y_test)

            results.append((k, weight, p, mean_score, test_accuracy))

            # Print hasil untuk setiap kombinasi parameter
            print(f'k: {k}, weight: {weight}, p: {p}, Cross-validated accuracy: {mean_score}, Test accuracy: {test_accuracy}')

            if mean_score > best_score:
                best_score = mean_score
                best_params = (k, weight, p)
```

**Gambar 2.5** Implementasi *K-Nearest Neighbors*

Pada Gambar 2.5, dilakukan iterasi untuk mencoba setiap kombinasi parameter yang mungkin untuk model KNN. Pada setiap iterasi, sebuah model KNN baru dibangun dengan parameter-parameter yang sedang diuji. Kemudian, model tersebut dievaluasi menggunakan metode *cross-validation* dengan

menggunakan cross_val_score. Hasil evaluasi berupa akurasi dari setiap lipatan *cross-validation* dihitung, dan nilai rata-rata akurasi dari seluruh lipatan diambil.

Selama iterasi, nilai akurasi validasi silang yang paling baik (tertinggi) beserta parameter-parameter yang sesuai disimpan menggunakan variabel best_score dan best_params. Setelah iterasi selesai, parameter-parameter yang memberikan akurasi validasi silang tertinggi dipilih sebagai parameter optimal untuk model KNN.

Selanjutnya Tahapan ketiga. melakukan tahapan optimasi menggunakan PSO untuk mengoptimalkan parameter model KNN. PSO adalah metode optimasi berbasis populasi yang terinspirasi oleh perilaku sosial hewan, seperti kawanan burung dan digunakan untuk menemukan parameter optimal yang meminimalkan atau memaksimalkan suatu fungsi objektif. Proses implementasi bisa di lihat pada Gambar 2.6

```
def optimize_knn(params):
    k = int(params[0])
    weight = 'uniform' if params[1] < 0.5 else 'distance'
    algorithm = 'auto'
    p = int(params[2])

    knn = KNeighborsClassifier(n_neighbors=k, weights=weight, algorithm=algorithm, p=p)

    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('knn', knn)
    ])

    # Menggunakan cross-validation untuk mendapatkan mean accuracy
    scores = cross_val_score(pipeline, x_train, y_train, cv=5, scoring='accuracy')
    penalty = 0 if k > 4 else (4 - k) * 0.01

    # Evaluasi model pada data uji
    pipeline.fit(x_train, y_train)
    test_accuracy = pipeline.score(x_test, y_test)

    # Simpan semua hasil yang dihitung
    all_results.append((k, weight, p, scores.mean(), penalty, test_accuracy))

    return -scores.mean() + penalty
```

**Gambar 2.6** Fungsi *Optimize_Knn*

pada Gambar 2.6 adalah proses implementasi optimasi parameter untuk model *K-Nearest Neighbors* (KNN) menggunakan *Particle Swarm Optimization* (PSO). Fungsi 'optimize_knn' adalah fungsi tujuan yang akan dioptimalkan algoritma PSO.

Pertama, fungsi ini mengonversi nilai parameter yang diberikan oleh algoritma PSO menjadi nilai-nilai yang dapat digunakan dalam pembangunan model KNN. Parameter yang dioptimalkan meliputi jumlah tetangga (k), jenis bobot (uniform atau distance), dan parameter p untuk metrik jarak. Selanjutnya, model KNN dimasukan dalam pipeline dengan 'StandardScaler' dan dievaluasi menggunakan cross-validation untuk mendapatkan *mean accuracy*. Penalti diberikan jika nilai 'K' kurang dari 4 untuk menghindari *underfitting*. Hasil evaluasi disimpan dalam list 'all_results'.

```
# Batasan untuk parameter
lb = [1, 0, 1]
ub = [50, 1, 2]
```

**Gambar 2.7** Batasan Parameter

Pada Gambar 2.7, batasan nilai parameter didefinisikan sebagai 'lb' dan 'ub' yaitu jumlah tetangga terdekat 'k' dengan rentang 1 hingga 50, bobot (*weight*) yang berupa '*uniform* dan *distance*' serta metrik jarak (*distance metric*) dengan rentang 1 hingga 2.

```
optimal_params, optimal_score = pso(optimize_knn, lb, ub, swarmsize=20, maxiter=50)
optimal_k = int(optimal_params[0])
optimal_weight = 'uniform' if optimal_params[1] < 0.5 else 'distance'
optimal_p = int(optimal_params[2])
```

**Gambar 2.8** Implementasi *Particle Swarm Optimization*

Pada Gambar 2.8 menjelaskan implementasi dilakukan dengan menetapkan ukuran populasi partikel (*swarmsize*=20) dan jumlah iterasi maksimum (*maxiter*=50). Setiap partikel dalam populasi ini memiliki posisi dan kecepatan awal yang dipilih secara acak dalam ruang parameter yang telah ditentukan. Setelah proses optimasi selesai, parameter optimal yang ditemukan disimpan dalam variabel.

### 2.2.5 Evaluasi Model

Evaluasi model merupakan langkah krusial setelah pembentukan model, di tahap ini. Performa model akan diukur untuk mengevaluasi akurasi dan kualitas data pelatihan yang digunakan. Pengujian akan dilakukan dengan *Confusion Matrix* yang memberikan pemahaman kesalahan klasifikasi secara mendalam melalui kalkulasi nilai positif benar (TP), positif salah (FP), negatif benar (TN), negatif salah (FN).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \quad (1)$$

Keterangan:

TP : *True Positive*

TN : *True Negative*

FP : *False Positive*

FN : *False Negative*

Mengingat penelitian ini menggunakan berbagai metode, mulai dari algoritma klasifikasi *K-Nearest Neighbors* hingga metode optimasi *Particle Swarm Optimization.* Tahap evaluasi ini bertujuan untuk membandingkan keefektifan setiap algoritma yang digunakan.