# BAB 5 IMPLEMENTASI

Bab ini menjelaskan implementasi sistem sesuai dengan perancangan. Pembahasan pada bab ini meliputi implementasi *Extreme Learning Machine* (ELM) dan implementasi *Improved-Particle Swarm Optimization* (IPSO).

## 5.1 Implementasi *Extreme Learning Machine* (ELM)

Implementasi *Extreme Learning Machine* (ELM) dibagi menjadi dua bagian utama, yaitu proses pelatihan lalu testing ELM dan prediksi.

### 5.1.1 Proses Normalisasi data

Sebelum digunakan *dataset* dinormalisasi terlebih dahulu menggunakan *Min-Max Normalization*.

```
1  double[][] normalisasiMinMax(double[][] matriks) {
2     double[][] X = new
      double[matriks.length][matriks[0].length];
3     double xmin = 0, xmax;
4     for (int j = 0; j < matriks[0].length; j++) {
5        xmin = matriks[0][j];
6        xmax = matriks[0][j];
7        for (int i = 0; i < matriks.length; i++) {
8        if (matriks[i][j] < xmin) {
9           xmin = matriks[i][j];
10       }
11       if (matriks[i][j] > xmax) {
12          xmax = matriks[i][j];
13       }
14       }
15       for (int i = 0; i < matriks.length; i++) {
16       x[i][j] = (matriks[i][j] - xmin)/(xmax - xmin);
17       }
18    }
19    return x;
20 }
```

**Kode Program 5.1 Normalisasi *Dataset***

Penjelasan Kode Program 5.1 sebagai berikut:

1. Baris 1 membuat *method* dengan parameter *array* dua dimensi dengan tipe data *double*.
2. Baris 2 membuat *array* dengan ukuran dan tipe data yang sama dengan parameter untuk menyimpan hasil normalisasi data.
3. Baris 3-14 mencari nilai minimum dan maksimum.
4. Baris 15-17 melakukan normalisasi terhadap *dataset* menggunakan *Min-Max Normalization*.
5. Baris 19 mengembalikan array hasil normalisasi.

### 5.1.2 Proses Pelatihan *Extreme Learning Machine* (ELM)

```
1  private double[][] hitungH(double[][] X) {
2    double[][] hInit = Helper.perkalianMatriks(X,
     wTransposed);
3    double[][] H = new double[X.length][hiddenNode];
4    for (int i = 0; i < hInit.length; i++) {
5       for (int j = 0; j < hInit[0].length; j++) {
6           hInit[i][j] = hInit[i][j] + bias[j];
7           H[i][j] = 1 / (1 + Math.exp(-hInit[i][j]));
8       }
9    }
10   return H;
11 }
12
13 private double[][] MoorePenrose(double[][] H) {
14     double[][] Htranpose = Helper.tranpose(H);
15     double[][] HTranposeKaliH =
       Helper.perkalianMatriks(Htranpose, H);
16     double[][] HTranposeKaliHInverse =
       Helper.inverse(HTranposeKaliH);
17     double[][] HPlus =
       Helper.perkalianMatriks(HTranposeKaliHInverse,
       Htranpose);
18     return HPlus;
19 }
20
21 private double[][] hitungBeta(double[][] HPlus) {
22     double[][] beta = Helper.perkalianMatriks(HPlus,
       data.YDataLatih);
23     return beta;
24 }
```

**Kode Program 5.2 Pelatihan *Extreme Learning Machine* (ELM)**

Penjelasan Kode Program 5.2 sebagai berikut:

1. Baris 1 membuat *method* `hitungH` dengan parameter *array* dua dimensi dengan tipe data double `X`.
2. Baris 2 mendeklarasikan *array* dua dimensi `hInit` bertipe data *double* dan dihitung dengan cara mengalikan *array* `X` dengan *array* `wTransposed`. `wTransposed` merupakan hasil *transpose* dari matriks bobot.
3. Baris 3 mendeklarasikan *array* dua dimensi `H` bertipe data *double* dengan dimensi panjang *array* `X` dikali `hiddenNode`.
4. Baris 4-9 menghitung matriks `H`. Baris 6 menjumlahkan `hInit` dengan bias. Matriks `H` dihitung menggunakan Persamaan 2.1.
5. Baris 10 mengembalikan matriks `H`.
6. Baris 13-19 menghitung `HPlus`. Baris 13 membuat *method MoorePenrose* dengan parameter matriks `H`. Baris 14 mendeklarasikan matriks `Htranpose`. Matriks `Htranpose` merupakan hasil *transpose* dari matriks `H`.

7. Baris 15 mengalikan matriks *Htranpose* dengan matriks *H* lalu disimpan dalam matriks *HTranposeKaliH*.
8. Baris 16 hasil *inverse* matriks *HTranposeKaliH* disimpan dalam matriks *HTranposeKaliHInverse*.
9. Baris 17 menghitung matriks HPlus dengan cara mengalikan matriks *HTranposeKaliHInverse* dengan matriks *Htranpose*.
10. Baris 18 mengembalikan matriks HPlus.
11. Baris 21 membuat *method* *hitungBeta* dengan parameter *array* dua dimensi dengan tipe data double *HPlus*.
12. Baris 22 mendeklarasikan matriks *beta* dan dihitung dengan cara mengalikan matriks *HPlus* dengan target data latih.
13. Baris 23 mengembalikan matriks *beta*.

### 5.1.3 Proses Pengujian *Extreme Learning Machine* (ELM)

```
1  private double evaluasi(double[][] H, double[][] beta)
   {
2    double[][] YPrediksi = Helper.perkalianMatriks(H,
     beta);
3    double MSEProtein;
4    double MSELemak;
5    double MSELaktosa
6    double MSEDensity;
7    double totalSelisihProtein = 0;
8    double totalSelisihLemak = 0;
9    double totalSelisihLaktosa = 0;
10   double totalSelisihDensity = 0;
11   for (int i = 0; i < data.banyakDataLatih; i++) {
12     for (int j = 0; j < data.banyakfitur; j++) {
13       totalSelisihProtein = totalSelisihProtein +
         Math.abs(YPrediksi[i][0] -
         data.YDataLatih[i][0]);
14       totalSelisihLemak = totalSelisihLemak +
         Math.abs(YPrediksi[i][1] -
         data.YDataLatih[i][1]);
15       totalSelisihLaktosa = totalSelisihLaktosa +
         Math.abs(YPrediksi[i][2] -
         data.YDataLatih[i][2]);
16       totalSelisihDensity = totalSelisihDensity +
         Math.abs(YPrediksi[i][3] -
         data.YDataLatih[i][3]);
17     }
18   }
19   MSEProtein = totalSelisihProtein/data.banyakDataUji;
20   MSELemak = totalSelisihLemak/data.banyakDataUji;
21   MSELaktosa = totalSelisihLaktosa/data.banyakDataUji;
22   MSEDensity = totalSelisihDensity/data.banyakDataUji;
23   meanMSE = (MSEProtein + MSELemak + MSELaktosa +
     MSEDensity) / 4;
24   return meanMSE;}
```

**Kode Program 5.3 Pengujian *Extreme Learning Machine* (ELM)**

Penjelasan **Error! Reference source not found.** sebagai berikut:

1. Baris 1 Membuat *method* `evaluasi` dengan parameter matriks `H` dan matriks `beta`.
2. Baris 2 mendeklarasikan matriks `YPrediksi` dan dihitung dengan cara mengalikan matriks `H` dengan matriks `beta`.
3. Baris 3-6 mendeklarasikan variabel-variabel untuk menyimpan nilai MSE.
4. Baris 11-18 menghitung semua total selisih kandungan gizi.
5. Baris 19-22 menghitung semua nilai MSE.
6. Baris 23 menghitung rata-rata nilai MSE.
7. Baris 24 mengembalikan nilai MSE.

### 5.1.4 Proses Prediksi Kandungan Susu menggunakan *Extreme Learning Machine* (ELM)

```
1  void prediksi(double[] giziPakan){
2     double[][] untukH = new double[1][giziPakan.length];
3     System.arraycopy(giziPakan, 0, untukH[0], 0,
      giziPakan.length);
4     double[][] H = hitungH(untukH);
5     double[][] hasilPrediksi = Helper.perkalianMatriks(H,
      beta);
6     proteinPrediksi = hasilPrediksi[0][0];
7     lemakPrediksi = hasilPrediksi[0][1];
8     LaktosaPrediksi = hasilPrediksi[0][2];
9     densityPrediksi = hasilPrediksi[0][3];
10 }
```

**Kode Program 5.4 Prediksi Kandungan Susu menggunakan *Extreme Learning Machine* (ELM)**

Penjelasan Kode Program 5.4 sebagai berikut:

1. Baris 1 membuat *method* `prediksi` dengan parameter matriks bertipe data double `giziPakan`.
2. Baris 2-3 menyalin isi matriks `giziPakan` ke matriks `untukH`.
3. Baris 4 mendeklarasikan matriks dua dimensi bertipe data double `H` dan dihitung dengan cara memasukkan matriks `untukH` ke dalam *method* `hitungH`.
4. Baris 5 mendeklarasikan matriks dua dimensi bertipe data double `hasilPrediksi` dan dihitung dengan cara mengalikan matriks `H` dengan matriks `beta`.
5. Baris 6-9 menyalin nilai kandungan gizi hasil prediksi ke masing-masing variabel.

## 5.2 Implementasi *Improved-Particle Swarm Optimization* (IPSO)

### 5.2.1 Proses Mencari Bobot dan Bias Terbaik Menggunakan *Improved-Particle Swarm Optimization* (IPSO)

```
1  IPSOBobotDanBias(int IterasiMaksimum, int ukuran
   populasi) {
2    this.IterasiMaksimum = IterasiMaksimum;
3    this.ukuran populasi = ukuran populasi;
4    inisialisasi();
5    for (int Iterasi = 0; Iterasi <= this.IterasiMaksimum;
     Iterasi++) {
6      updateKecepatan(Iterasi);
7      updatePosisi();
8      hitungFitness();
9      updatePBestdanGbest();
10   }
11 }
12 void inisialisasi() {
13   posisi = new double[ukuran populasi][panjangPartikel];
14   kecepatan = new double[ukuran
     populasi][panjangPartikel];
15   fitness = new double[ukuran populasi];
16   pBest = new double[ukuran populasi][panjangPartikel];
17   fitnessPBest = new double[ukuran populasi];
18   Gbest = new double[panjangPartikel];
19   for (int i = 0; i < ukuran populasi; i++) {
20     for (int j = 0; j < panjangPartikel; j++) {
21     posisi[i][j] = Math.random();
22     }
23   }
24   System.arraycopy(posisi, 0, pBest, 0, ukuran
     populasi);
25 }
26 void updateKecepatan(int Iterasi) {
27   double w, K, r1 = 0.550462113, r2 = 0.507555189;
28   for (int i = 0; i < ukuran populasi; i++) {
29     for (int j = 0; j < panjangPartikel; j++) {
30       if (Iterasi < IterasiMaksimum / 2) {
31         w = (0.857143 + ((1 - 0.857143) * (1 - Iterasi /
           IterasiMaksimum)));
32       kecepatan[i][j] = w * kecepatan[i][j] + 2 * r1 *
         (pBest[i][j] - posisi[i][j]) - 2 * r2 * (Gbest[j]
         - posisi[i][j]);
33       } else {
34         K = (Math.cos(2 * 3.14 / IterasiMaksimum *
           (Iterasi - IterasiMaksimum / 2)) + 2.428571) /
           4;
35         kecepatan[i][j] = K * (0.7 * kecepatan[i][j] + 2
           * r1 * (pBest[i][j] - posisi[i][j]) + 2 * r2 *
           (Gbest[j] - posisi[i][j]));
```

```
36        }
37      }
38    }
39  }
40    void updatePosisi() {
41    for (int i = 0; i < ukuran populasi; i++) {
42      for (int j = 0; j < panjangPartikel; j++) {
43      posisi[i][j] = posisi[i][j] + kecepatan[i][j];
44      }
45    }
46  void hitungFitness() {
47    ELM elmpemodelan = new ELM();
48    for (int i = 0; i < ukuran populasi; i++) {
49      elmpemodelan.pelatihan(posisi[i]);
50      fitness[i] = 1 / (elmpemodelan.meanMSE + 1);
51    }
52  }
53  void updatePBestdanGbest() {
54    for (int i = 0; i < ukuran populasi; i++) {
55      if (fitness[i] > fitnessPBest[i]) {
56      System.arraycopy(posisi[i], 0, pBest[i], 0,
        panjangPartikel);
57      fitnessPBest[i] = fitness[i];
58      }
59    }
60    int indeksTerbesar = 0;
61    for (int i = 0; i < ukuran populasi; i++) {
62      if (fitnessPBest[i] > fitnessPBest[indeksTerbesar])
        {
63      indeksTerbesar = i;
64      }
65    }
66    if (fitnessPBest[indeksTerbesar] > fitnessGbest) {
67      System.arraycopy(pBest[indeksTerbesar], 0, Gbest, 0,
        panjangPartikel);
68      fitnessGbest = fitnessPBest[indeksTerbesar];
69    }
70  }
```

**Kode Program 5.5 Mencari Bobot dan Bias Terbaik Menggunakan *Improved-Particle Swarm Optimization* (IPSO)**

Penjelasan Kode Program 5.5 sebagai berikut:

1. Baris 1 membuat *method* `IPSOBobotDanBias` dengan parameter `IterasiMaksimum` dan `ukuran populasi`.
2. Baris 2-3 menyalin parameter ke variabel dalam *class*.
3. Baris 4 memanggil *method* `inisialisasi`.
4. Baris 5 mengulang *syntax* pada baris 6-9 selama Iterasi lebih kecil dari `IterasiMaksimum`.

5. Baris 6 memanggil *method* `PerbaruiKecepatan` dengan argumen `Iterasi` sekarang.
6. Baris 7 memanggil *method* `PerbaruiPosisi`.
7. Baris 8 memanggil *method* `hitungFitness`.
8. Baris 9 memanggil *method* `PerbaruiPBestdanGbest`.
9. Baris 12 membuat *method* tanpa parameter dan pengembalian `inisialisasi`.
10. Baris 13-18 menentukan ukuran matriks `Posisi`, `kecepatan`, `fitness`, `pBest`, `fitnessPBest`, `Gbest` berdasarkan `ukuran populasi` dan `panjangPartikel`.
11. Baris 19-23 mengisi matriks `Posisi` dengan nilai acak.
12. Baris 24 menyalin isi matriks `Posisi` ke matriks `pBest`.
13. Baris 26 membuat *method* `PerbaruiKecepatan` dengan parameter `Iterasi`.
14. Baris 27 mendeklarasikan variabel `w`, `K`, serta nilai acak `r1` dan `r2`.
15. Baris 30 memeriksa apakah `Iterasi` berada pada setengah Iterasi awal atau akhir. Baris 31-32jika iya, maka hitung variabel `w` menggunakan Persamaan 2.7 lalu hitung kecepatan menggunakan Persamaan 2.8. Baris 34-35 jika tidak, maka hitung variabel `K` menggunakan Persamaan 2.9 lalu hitung kecepatan menggunakan Persamaan 2.8.
16. Membuat *method* tanpa parameter dan pengembalian `PerbaruiPosisi`.
17. Baris 41-45 Perbarui posisi dengan cara menambahkan posisi lama dengan kecepatan.
18. Baris 46 membuat *method* `hitungFitness`.
19. Baris 47 membuat model `elmpemodelan` dari *class* `ELM` dengan parameter posisi ke-*i*
20. Memanggil *method* `pelatihan` yang dimiliki `elmpemodelan`.
21. Baris 50 menghitung nilai *fitness* dengan menggunakan Persamaan 2.13.
22. Baris 53 membuat *method* `updatePBestdanGbest`.
23. Baris 55 memeriksa apakah *fitness* partikel Iterasi sekarang lebih baik daripada *fitness pBest* Iterasi sebelumnya.
24. Baris 56-57 jika iya, salin posisi dan *fitness* partikel Iterasi sekarang ke *pBest*.
25. Baris 60-65 mencari indeks *pBest* yang mempunyai nilai *fitness* paling besar.
26. Baris 66 membandingkan apakah nilai *fitness pBest* ke-`indeksTerbesar` lebih besar dari nilai *fitness Gbest* pada Iterasi sebelumnya. Baris 67-68 jika iya, salin posisi dan nilai *fitness pBest* ke-`indeksTerbesar` ke *Gbest*.

### 5.2.2 Proses Optimasi Komposisi Kandungan Gizi Pakan Menggunakan *Improved-Particle Swarm Optimization* (IPSO)

```
1  void inisialisasi() {
2    posisi = new double[ukuran populasi][panjangPartikel];
3    kecepatan = new double[ukuran
     populasi][panjangPartikel];
4    fitness = new double[ukuran populasi];
5    pBest = new double[ukuran populasi][panjangPartikel];
6    fitnessPBest = new double[ukuran populasi];
```

```
  Gbest = new double[panjangPartikel];
  for (int i = 0; i < ukuran populasi; i++) {
    for (int j = 0; j < panjangPartikel; j++) {
      posisi[i][j] = Math.random();
    }
  }
  System.arraycopy(posisi, 0, pBest, 0, ukuran
  populasi);
}
void updateKecepatan(int Iterasi) {
  double w, K, r1 = 0.918182794, r2 = 0.299676349;
  for (int i = 0; i < ukuran populasi; i++) {
    for (int j = 0; j < panjangPartikel; j++) {
      if (Iterasi < IterasiMaksimum / 2) {
        w = (0.857143 + ((1 - 0.857143) * (1 - Iterasi /
        IterasiMaksimum)));
        kecepatan[i][j] = w * kecepatan[i][j] + 2 * r1 *
        (pBest[i][j] - posisi[i][j]) - 2 * r2 *
        (Gbest[j] - posisi[i][j]);
      } else {
        K = (Math.cos(2 * 3.14 / IterasiMaksimum *
        (Iterasi - IterasiMaksimum / 2)) + 2.428571) /
        4;
        kecepatan[i][j] = K * (0.7 * kecepatan[i][j] + 2
        * r1 * (pBest[i][j] - posisi[i][j]) + 2 * r2 *
        (Gbest[j] - posisi[i][j]));
      }
    }
  }
}
void updatePosisi() {
  for (int i = 0; i < ukuran populasi; i++) {
    for (int j = 0; j < panjangPartikel; j++) {
      posisi[i][j] = posisi[i][j] + kecepatan[i][j];
    }
  }
}
void hitungFitness() {
  ELM = new ELM();
  elm.pelatihan(bobotdanbias);
  for (int i = 0; i < ukuran populasi; i++) {
    elm.prediksi(posisi[i]);
    fitness[i] = elm.proteinPrediksi +
    1/elm.lemakPrediksi + elm.LaktosaPrediksi +
    elm.densityPrediksi;
  }
}
void updatePBestdanGbest() {
  for (int i = 0; i < ukuran populasi; i++) {
    if (fitness[i] > fitnessPBest[i]) {
      System.arraycopy(posisi[i], 0, pBest[i], 0,
```

```
        panjangPartikel);
48      fitnessPBest[i] = fitness[i];
49    }
50  }
51  int indeksTerbesar = 0;
52  for (int i = 0; i < ukuran populasi; i++) {
53    if (fitnessPBest[i] > fitnessPBest[indeksTerbesar])
      {
54    indeksTerbesar = i;
55    }
56  }
57  if (fitnessPBest[indeksTerbesar] > fitnessGbest) {
58    System.arraycopy(pBest[indeksTerbesar], 0, Gbest,
      0, panjangPartikel);
59    fitnessGbest = fitnessPBest[indeksTerbesar];
60 }
```

**Kode Program 5.6 Optimasi Komposisi Kandungan Gizi Pakan Menggunakan *Improved-Particle Swarm Optimization* (IPSO)**

Penjelasan Kode Program 5.6 sebagai berikut:

1. Semua *syntax* sama seperti pada Proses Mencari Bobot dan Bias Terbaik Menggunakan *Improved-Particle Swarm Optimization* (IPSO) kecuali pada bagian hitung *fitness*.
2. Baris 36 membuat *method* `hitung`*`Fitness`*.
3. Baris 37-38 membuat model *`elm`* dari *class `ELM`*.
4. Baris 40 memanggil *method* *`prediksi`* yang dimiliki *`elm`* dengan argumen posisi ke-*i*.
5. Baris 41 menghitung *fitness* menggunakan Persamaan 2.14.
6. Baris 44 membuat *method* `update`*`PBest`*`dan`*`Gbest`*.
7. Baris 46 memeriksa apakah *fitness* partikel Iterasi sekarang lebih baik daripada *fitness pBest* Iterasi sebelumnya.
8. Baris 47-48 jika iya, salin posisi dan *fitness* partikel Iterasi sekarang ke *pBest*.
9. Baris 51-56 mencari indeks *pBest* yang mempunyai nilai *fitness* paling besar.
10. Baris 57 membandingkan apakah nilai *fitness pBest* ke-*`indeksTerbesar`* lebih besar dari nilai *fitness Gbest* pada Iterasi sebelumnya. Baris 58-59 jika iya, salin posisi dan nilai *fitness pBest* ke-*`indeksTerbesar`* ke *Gbest*.

### 5.2.3 Proses Pembagian Porsi

```
1  void konversiKeKomposisiPakan(IPSOKandunganSusu
   ipsokandungansusu) {
2    int indeksPakan1 = 1, indeksPakan2 = 3;
3    double persentasePakan1 = 0.7;
4    double persentasePakan2 = 0.3;
5    double totalNutrisiPakan = data.abuPakan[indeksPakan1]
     + data.abuPakan[indeksPakan2] +
     data.lkPakan[indeksPakan1] +
     data.lkPakan[indeksPakan2];
6    double totalNutrisiOptimasi = ELM.proteinPrediksi +
```

```
    ELM.lemakPrediksi + ELM.laktosaPrediksi +
    ELM.densityPrediksi;
7   double totalPemberianPakan = totalNutrisiOptimasi /
    totalNutrisiPakan;
8   double pemberianPakan1 = persentasePakan1 *
    totalPemberianPakan;
9   double pemberianPakan2 = persentasePakan2 *
    totalPemberianPakan;
1 }
```

**Kode Program 5.7 Konversi Ke Komposisi Pakan**

Penjelasan Kode Program 5.7 sebagai berikut:

1. Baris 2-4 menerima masukkan dari jenis dan persentase pakan dari pengguna.
2. Baris 5 menjumlahkan kandungan gizi pakan.
3. Baris 6 menjumlahkan kandungan gizi hasil optimasi.
4. Baris 7 hitung berat pemberian pakan.
5. Baris 8 bagi porsi pakan berdasarkan persentase yang telah dimasukkan.