BAB 5 IMPLEMENTASI

Pada bab ini akan dibahas bagaimana implementasi dan antarmuka dari penelitian yang telah dirancang pada bab sebelumnya yaitu bab perancangan.

5.1 Implementasi Sistem

Dalam sub-bab implementasi sistem ini terdapat pembahasan mengenai 2 metode, yaitu Algoritme Genetika dan *Extreme Learning Machine*. Pada implementasi Algoritme Genetika terdapat 4 proses perhitungan yang akan dibahas, diantaranya adalah inisialisasi populasi awal, proses reproduksi, proses evaluasi serta proses seleksi. Sedangkan pada *Extreme Learning Machine* terdapat 7 proses perhitungan yang meliputi normalisasi data, pembangkitan bobot awal/*input weight*, pembangkitan bias, proses perhitungan keluaran *hidden layer*, proses perhitungan fungsi aktivasi, proses perhitungan *output weight* serta yang terakhir terdapat proses perhitungan menghitung *output layer*.

Implementasi sistem ini menggunakan bahasa pemrograman Java dan editor Netbeans IDE 8.0,2. Pada implementasi sistem ini juga menggunakan beberapa *library* diantaranya yaitu JAMA dan JXL. Dimana *library* JAMA akan digunakan dalam perhitungan yang melibatkan matriks, sedangkan *library* JXL digunakan untuk mengakses *file Excel*.

5.1.1 Implementasi Inisialisasi Populasi Awal

Populasi awal diinisialisasi secara *random* antara -1 s.d. 1. *Parent* pertama dibangkitkan dengan memanggil *file Excel* yang berisi bobot awal/*input weight* pada penelitian sebelumnya, sedangkan *parent-parent* selanjutnya dibangkitkan secara *random*. Operasi pengaksesan *file Excel* pada implementasi inisialisasi awal ini menggunakan *library* JXL. Implementasi inisialisasi populasi awal dapat dilihat pada Kode Program 5.1.

Kode Program 5.1 Implementasi Inisialisasi Populasi Awal

```
1   Random populasiAwal = new Random();
2   double max = 1, min = -1;
3   double range = max - min;
4
5   try {
6     File foldChoosen = new File
7                        ("C:\\Users\\home\\Desktop\\Rere\\File
8                        Input\\inputWeight.xls");
9     Workbook foldExcel = Workbook.getWorkbook(foldChoosen);
10    Sheet sheet = foldExcel.getSheet(0);
11
12    int columns = sheet.getColumns();
13    int rows = sheet.getRows();
14    double[][] tempKromosomExc = new double[rows][columns];
15
16    for (int i = 0; i < rows; i++) {
17      for (int j = 0; j < columns; j++) {
18        Cell cell = sheet.getCell(j, i);
19        tempKromosomExc[i][j]=
20                          Double.valueOf(cell.getContents());
```

```
21      }
22    }
23
24    int countArray = 0;
25    for (int i = 0; i < rows; i++) {
26      for (int j = 0; j < columns; j++) {
27        kromosomExcel[0][countArray] = tempKromosomExc[i][j];
28        countArray++;
29      }
30    }
31
32    for (int i = 0; i < populasi; i++) {
33      for (int j = 0; j < panjangKromosom; j++) {
34        if (i == 0) {
35          kromosomVar[0][j] = kromosomExcel[0][j];
36        } else {
37          double scaled = populasiAwal.nextDouble() * range;
38          double shifted = scaled + min;
39          kromosomVar[i][j] = shifted;
40        }
41      }
42    }
43  } catch (IOException | BiffException | NumberFormatException
44  e) {
45  System.out.println(e);
46  }
```

Penjelasan Kode Program 5.1 tentang implementasi inisialisasi populasi awal adalah sebagai berikut:

Baris 1 s.d 3 : Inisialisasi variabel yang digunakan untuk membangkitkan populasi awal secara *random* dengan *range* -1 s.d. 1.

Baris 6 s.d 10 : Inisialisasi variabel yang digunakan untuk memilih dan membaca *file Excel*.

Baris 12 s.d 14 : Inisialisasi variabel untuk menyimpan kromosom *parent* sementara.

Baris 16 s.d 22 : Proses menyimpan data yang dibaca dari *Excel* ke dalam variabel tempKromosomExc (variabel yang digunakan untuk menyimpan nilai kromosom *parent* secara sementara).

Baris 24 s.d 30 : Proses mengubah *array* tempKromosomExc dengan ukuran i x j menjadi 1 x (ixj), agar sesuai dengan kromosom *parent* yang lain.

Baris 32 s.d 42 : Proses membangkitkan populasi pertama dengan bobot awal yang didapat dengan membaca *file Excel*, dan populasi selanjutnya dibangkitkan secara *random*.

Baris 43 s.d 45 : Menampilkan proses *error* pada proses *try* yang telah ditangani oleh *catch*.

5.1.2 Implementasi Reproduksi

Implementasi Reproduksi pada penelitian ini terdiri dari 2 proses, yaitu implementasi proses *crossover* dan implementasi proses mutasi.

5.1.2.1 Implementasi Proses *Crossover*

Implementasi proses *crossover* pada penelitian ini menggunakan *Extended Intermediate Crossover*, dimana dalam prosesnya dibutuhkan 2 *parent* untuk menghasilkan 2 *child/offspring*. Pada *Extended Intermediate Crossover* dibutuhkan variabel tambahan yaitu *alpha* untuk melakukan proses perhitungan. Nilai *alpha* dibangkitkan secara *random* dengan *range* -0,25 hingga 1,25. Implementasi proses *crossover* dapat dilihat pada Kode Program 5.2.

Kode Program 5.2 Implementasi Proses *Crossover*

```
1   public double[][] ExtendedIntermediateCrossover(double[][]
2   kromosom) {
3     Random crossover = new Random();
4     int panjangKromosom = kromosom[0].length;
5     int popsize = kromosom.length;
6     double[] newAlpha = new double[panjangKromosom];
7     double[][] child = new double[2][panjangKromosom];
8     double[][] p1 = new double[1][panjangKromosom];
9     double[][] p2 = new double[1][panjangKromosom];
10    int minPopsize = 0;
11    int maxPopsize = popsize - 1;
12    double maxAlpha = 1.25, minAlpha = -0,25;
13    double range = maxAlpha - minAlpha;
14    int firstParentIndex = 0, secondParentIndex = 0;
15    boolean run = true;
16
17    while (run) {
18      firstParentIndex   =   crossover.nextInt(maxPopsize   -
19                           minPopsize + 1) + minPopsize;
20      secondParentIndex   =   crossover.nextInt(maxPopsize   -
21                           minPopsize + 1) + minPopsize;
22      if (firstParentIndex != secondParentIndex) {
23        run = false;
24      }
25    }
26
27    for (int k = 0; k < panjangKromosom; k++) {
28      double scaledC = crossover.nextDouble() * range;
29      double shiftedC = scaledC + minAlpha;
30      double alpha = shiftedC;
31      newAlpha[k] = alpha;
32
33      for (int i = 0; i < panjangKromosom; i++) {
34        p1[0][i] = kromosom[firstParentIndex][i];
35      }
36      for (int i = 0; i < panjangKromosom; i++) {
37        p2[0][i] = kromosom[secondParentIndex][i];
38      }
39
40      /*   Perhitungan   child/offspring   dengan   Extended
41           Intermediate Crossover
42           C1 = P1 + a (P2 – P1)
```

```
43            C2 = P2 + a (P1 - P2)
44        */
45        for (int i = 0; i < panjangKromosom; i++) {
46          child[0][i] = p1[0][i] + (newAlpha[k] * (p2[0][i] -
47                        p1[0][i]));
48          child[1][i] = p2[0][i] + (newAlpha[k] * (p1[0][i] -
49                        p2[0][i]));
50        }
51      }
52      return child;
53    }
```

Penjelasan Kode Program 5.2 tentang implementasi proses *crossover* adalah sebagai berikut:

Baris 3 s.d 15 : Inisialisasi variabel yang digunakan untuk perhitungan proses *Extended Intermediate Crossover*.

Baris 17 s.d 25 : Proses memilih 2 indeks *parent* secara *random* diantara populasi yang telah dibangkitkan.

Baris 28 s.d 38 : Proses membangkitkan nilai *alpha* secara *random* dengan *range* antara -0,25 s.d 1,25.

Baris 40 s.d 51 : Proses Perhitungan *child/offspring crossover* dengan *Extended Intermediate Crossover* menggunakan persamaan C1 = P1 + a (P2 – P1) dan C2 = P2 + a (P1 – P2).

Baris 52 : Mengembalikan nilai dari variabel *child*.

5.1.2.2 Implementasi Proses Mutasi

Implementasi proses mutasi pada penelitian ini menggunakan *Random Mutation*, dimana dalam prosesnya akan dipilih 1 *parent* secara *random*. Pada *parent* yang terpilih, ada 1 gen yang akan dimutasi, untuk kemudian dijadikan sebagai *child/offspring*. Implementasi proses mutasi dapat dilihat pada Kode Program 5.3.

Kode Program 5.3 Implementasi Proses Mutasi

```
1   public double[][] RandomMutation(double[][] kromosom) {
2     Random mutasi = new Random();
3     int panjangKromosom = kromosom[0].length;
4     int popsize = kromosom.length;
5     double[][] child = new double[1][panjangKromosom];
6
7     int minPopsize = 0;
8     int maxPopsize = popsize - 1;
9
10    int p = mutasi.nextInt(maxPopsize - minPopsize + 1) +
11              minPopsize;
12
13    int xMin = 0, xMax = panjangKromosom - 1;
14    int x = 0;
15
16    x = mutasi.nextInt(xMax - xMin + 1) + xMin;
17    int xIndex = x + 1;
18
```

```
19    double maxR = 1, minR = -1;
20    double rangeR = maxR - minR;
21    double scaledR = mutasi.nextDouble() * rangeR;
22    double shiftedR = scaledR + minR;
23    double r = shiftedR;
24
25    for (int i = 0; i < panjangKromosom; i++) {
26      child[0][i] = kromosom[p][i];
27    }
28
29    double max = kromosom[0][0], min = kromosom[0][0];
30
31    for (int i = 0; i < kromosom.length; i++) {
32      if (kromosom[i][x] > max) {
33        max = kromosom[i][x];
34      }
35      if (kromosom[i][x] < min) {
36        min = kromosom[i][x];
37      }
38    }
39    child[0][x] = kromosom[p][x] + (r * (max - min));
40
41    return child;
42  }
```

Penjelasan Kode Program 5.3 tentang implementasi proses mutasi adalah sebagai berikut:

Baris 2 s.d 8 : Inisialisasi variabel yang digunakan untuk perhitungan proses *Random Mutation* .

Baris 10 s.d 11 : Proses memilih indeks *parent* secara *random* diantara populasi yang telah dibangkitkan.

Baris 13 s.d 17 : Proses memilih posisi gen yang akan dimutasi, proses pemilihan dilakukan secara *random* pada *parent* yang terpilih sebelumnya.

Baris 19 s.d 23 : Proses membangkitkan nilai r secara *random* dengan *range* antara -1 s.d 1.

Baris 25 s.d 27 : Proses penyimpanan nilai *parent* yang terpilih sebagai *child* secara sementara.

Baris 29 s.d 38 : Proses mencari nilai minimal dan maksimal pada kolom yang sama dengan indeks terpilih pada seluruh *parent*.

Baris 39 : Proses Perhitungan *child/offspring* mutasi dengan *Random Mutation* menggunakan persamaan $x'_i = x'_i + r (max_i – min_j)$.

Baris 41 : Mengembalikan nilai dari variabel *child*.

5.1.3 Implementasi Proses Evaluasi

Pada implementasi proses evaluasi akan dihitung nilai *fitness* dari setiap individu yang ada. Implementasi proses evaluasi dapat dilihat pada Kode Program 5.4.

Kode Program 5.4 Implementasi Proses Evaluasi

```
1  double[][] hitungFitness(double[][] inpWeight) throws
2     IOException, BiffException {
3    int fold = 5;
4    int panjangKromosom = inpWeight.length *
5     inpWeight[0].length;
6    double[] tmp = new double[panjangKromosom + 1];
7    double[][] temp = new double[fold][panjangKromosom + 1];
8
9    // Fold 1-5
10   for (int i = 0; i < fold; i++) {
11     tmp = proses(inpWeight, (i + 1));
12     for (int j = 0; j < panjangKromosom + 1; j++) {
13       temp[i][j] = tmp[j];
14     }
15   }
16   return temp;
17 }
```

Penjelasan Kode Program 5.4 tentang implementasi proses evaluasi adalah sebagai berikut:

Baris 3 s.d 7 : Inisialisasi variabel yang digunakan untuk perhitungan nilai *fitness*.

Baris 10 s.d 11 : Memanggil method proses untuk menghitung nilai *fitness* dengan persamaan ELM, kemudian nilai *fitness* disimpan ke variabel tmp.

Baris 12 s.d 14 : Karena nilai *fitness* hanya ada di kolom *index* ke 0 pada variabel tmp, maka dilakukan perulangan untuk menyimpan hanya nilai *fitness*-nya saja.

Baris 16 : Mengembalikan nilai dari variabel tempFitness.

5.1.4 Implementasi Proses Seleksi

Pada penelitian ini, implementasi proses seleksi yang digunakan adalah jenis seleksi *elitism*/*Elitisim Selection*. Dimana seluruh individu (*parent* dan *child/offspring*) diurutkan berdasarkan nilai *fitness*-nya. Kemudian akan diambil individu terbaik sejumlah *popsize* untuk dijadikan populasi baru di generasi berikutnya. Implementasi proses seleksi dapat dilihat pada Kode Program 5.5.

Kode Program 5.5 Implementasi Proses Seleksi

```
1  public double[][] ElitismSelection(double[][]
2  individuGabungan, int popsize, double[] fitness) {
3    int panjangKromosom = individuGabungan[0].length;
4    int jmlIndividu = individuGabungan.length;
5    double[][]result = new
6  double[popsize][panjangKromosom+1];
7    double[] hasilFitness = new double[popsize];
8    int[] firstIndex = new int[jmlIndividu];
9
10   for (int i = 0; i < jmlIndividu; i++) {
11     firstIndex[i] = i;
12   }
13
```

```
14    for (int i = 0; i < jmlIndividu; i++) {
15      double tempFitness = fitness[i];
16      int tempFirstIndex = firstIndex[i];
17      double[][] tempResult = new double[1][panjangKromosom];
18
19      for (int j = (i + 1); j < jmlIndividu; j++) {
20        if (fitness[j] > tempFitness) {
21          tempFitness = fitness[j];
22          fitness[j] = fitness[i];
23          fitness[i] = tempFitness;
24          tempFirstIndex = firstIndex[j];
25          firstIndex[j] = firstIndex[i];
26          firstIndex[i] = tempFirstIndex;
27          for (int k = 0; k < panjangKromosom; k++) {
28            tempResult[0][k] = individuGabungan[j][k];
29          }
30          for (int k = 0; k < panjangKromosom; k++) {
31            individuGabungan[j][k] = individuGabungan[i][k];
32          }
33          for (int k = 0; k < panjangKromosom; k++) {
34            individuGabungan[i][k] = tempResult[0][k];
35          }
36        }
37      }
38    }
39
40    for (int i = 0; i < popsize; i++) {
41      for (int j = 0; j < panjangKromosom; j++) {
42        result[i][j] = individuGabungan[i][j];
43      }
44    }
45    for (int i = 0; i < popsize; i++) {
46      result[i][panjangKromosom] = fitness[i];
47    }
48    return result;
  }
```

Penjelasan Kode Program 5.5 tentang implementasi proses seleksi adalah sebagai berikut:

Baris 3 s.d 7 : Inisialisasi variabel yang digunakan untuk perhitungan *Elitism Selection*.

Baris 9 s.d 11 : Proses memberi indeks pada seluruh individu.

Baris 13 s.d 37 : Proses *sorting* atau mengurutkan individu berdasarkan nilai *fitness* tertinggi.

Baris 39 s.d 46 : Proses mengambil individu dengan nilai *fitness* terbaik sejumlah *popsize*.

Baris 47 : Mengembalikan nilai dari variabel *result*.

5.1.5 Implementasi Normalisasi Data

Implementasi normalisasi data diawali dengan mencari nilai minimal dan maksimal pada kolom yang sama kemudian dilakukan perhitungan normalisasi

agar *range* antar data tidak terlalu jauh. Implementasi proses seleksi dapat dilihat pada Kode Program 5.6.

Kode Program 5.6 Implementasi Normalisasi Data

```
1   void normalisasiData(int x) {
2     for (int i = 0; i < jumlahData; i++) {
3       if (isiData[i][x] > max) {
4         max = isiData[i][x];
5       }
6       if (isiData[i][x] < min) {
7         min = isiData[i][x];
8       }
9     }
10
11    for (int i = 0; i < jumlahData; i++) {
12      isiData[i][x] = (isiData[i][x]-min)/(max - min);
13    }
14
15    max = 0;
16    min = 10000;
17  }
```

Penjelasan Kode Program 5.6 tentang implementasi normalisasi data adalah sebagai berikut:

Baris 2 s.d 9 : Proses mencari nilai minimal dan maksimal dengan parameter x sebagai representasi dari tiap kolom atau tiap kriteria.

Baris 11 s.d 13 : Proses perhitungan normalisasi *min-max*.

Baris 15 s.d 16 : Menentukan kembali nilai dari variabel *max* dan *min*.

5.1.6 Implementasi Pembangkitan *Input Weight*/Bobot Awal

Proses pada implementasi pembangkitan *input weight*/bobot awal adalah proses utama yang akan digunakan untuk proses-proses perhitungan selanjutnya. Implementasi pembangkitan *input weight* dapat dilihat pada Kode Program 5.7.

Kode Program 5.7 Implementasi Pembangkitan *Input Weight*

```
1   void pembangkitanInputWeight(double[][] inputW) {
2     this.inputWeight = inputW;
3   }
```

Penjelasan Kode Program 5.7 tentang implementasi pembangkitan *input weight* adalah sebagai berikut:

Baris 1 s.d 3 : Proses me-*refer* variabel dalam parameter ke variabel global.

5.1.7 Implementasi Pembangkitan Bias

Pada implementasi pembangkitan bias, bias dibangkitkan dengan cara menginputkan data dari *file Excel*. Pengaksesan *file Excel* pada implementasi

pembangkitan bias ini menggunakan *library* JXL. Implementasi pembangkitan bias dapat dilihat pada Kode Program 5.8.

Kode Program 5.8 Implementasi Pembangkitan Bias

```
1   void pembangkitanBias() throws IOException, BiffException {
2      File  biasInit  =  new  File  ("C:\\Users\\home\\Desktop
3                           \\Rere\\File Input\\bias.xls");
4      Workbook biasExcel = Workbook.getWorkbook(biasInit);
5      Sheet sheet = biasExcel.getSheet(0);
6
7      int k = 0;
8      for (int j = 0; j < bias.length; j++) {
9        Cell cell = sheet.getCell(0, k++);
10       bias[j] = Double.valueOf(cell.getContents());
11     }
12  }
```

Penjelasan Kode Program 5.8 tentang implementasi pembangkitan bias adalah sebagai berikut:

Baris 2 s.d 5 : Proses memilih *file Excel*.

Baris 7 s.d 11 : Mendapatkan nilai dari data *file* bias.

5.1.8 Implementasi Proses Perhitungan Keluaran *Hidden Layer*

Implementasi proses perhitungan keluaran *hidden layer* merupakan proses pertama pada Extreme Learning Machine. Proses perhitungan ini akan digunakan pada proses *training* dan *testing*. Di dalam proses ini terdapat proses perkalian antara data *training/testing* dengan *input weight*/bobot awal, kemudian akan ditambahkan dengan biasnya. Operasi perkalian pada proses perhitungan keluaran *hidden layer* ini menggunakan *library* JAMA. Implementasi proses perhitungan keluaran *hidden layer* dapat dilihat pada Kode Program 5.9.

Kode Program 5.9 Implementasi Proses Perhitungan Keluaran *Hidden Layer*

```
1   void keluaranHiddenLayer(int jmlData, double[][] data) {
2     outputHidden = new double[jmlData][jmlHiddenLayer];
3     Matrix dataA = new Matrix(data);
4     Matrix inpWei = new Matrix(inputWeight);
5     Matrix dataKaliInput = dataA.times(inpWei);
6
7     hiddenOutput = dataKaliInput.getArray();
8     for (double[] hiddenOutput 1 : hiddenOutput) {
9       for (int j = 0; j < hiddenOutput[0].length; j++) {
10        hiddenOutput[j] += bias[j];
11      }
12    }
13  }
```

Penjelasan Kode Program 5.9 tentang implementasi proses perhitungan keluaran *hidden layer* adalah sebagai berikut:

Baris 2 s.d 5 : Proses perkalian data (N) dengan *input weight*.

Baris 7 s.d 12 : Proses penambahan bias pada hasil perkalian data dengan *input weight*.

5.1.9 Implementasi Proses Perhitungan Fungsi Aktivasi

Implementasi proses perhitungan fungsi aktivasi yang digunakan adalah fungsi aktivasi sigmoid. Implementasi proses perhitungan fungsi aktivasi dapat dilihat pada Kode Program 5.10,

Kode Program 5.10 Implementasi Proses Perhitungan Fungsi Aktivasi

```
1  void aktivasiSigmoid() {
2    for (double[] outputHidden1 : hiddenOutput) {
3      for (int j = 0; j < hiddenOutput[0].length; j++) {
4        outputHidden1[j]    =    1    /    (1    +    Math.exp(-
5                                hiddenOutput1[j]));
6      }
7    }
8  }
```

Penjelasan Kode Program 5.10 tentang implementasi proses perhitungan fungsi aktivasi adalah sebagai berikut:

Baris 2 s.d 7 : Proses perhitungan aktivasi sigmoid menggunakan Persamaan 2.1.

5.1.10 Implementasi Proses Perhitungan *Output Weight*

Dalam implementasi proses perhitungan *output weight* terdapat proses perhitungan matriks *Moore-Penrose*. Kemudian mengalikan hasil dari perhitungan *Moore-Penrose* dengan target data *training*. Operasi perhitungan *array* (yang meliputi *transpose*, perkalian, dan *inverse*) pada proses perhitungan *output weight* ini menggunakan *library* JAMA. Implementasi proses perhitungan *output weight* dapat dilihat pada Kode Program 5.11.

Kode Program 5.11 Implementasi Proses Perhitungan *Output Weight*

```
1  void outputWeight() {
2    Matrix h = new Matrix(hiddenOutput);
3    Matrix trans = h.transpose();
4    Matrix hTh = trans.times(h); // hT x h
5
6    Matrix hInvers = hTh.inverse();
7
8    Matrix hCross = hInvers.times(trans); // hTh Inverse x hT
9
10   convertTarget();
11   Matrix target = new Matrix(convertedTarget);
12   Matrix beta = hCross.times(target);
13 }
```

Penjelasan Kode Program 5.11 tentang implementasi proses perhitungan *output weight* adalah sebagai berikut:

Baris 2 s.d 8 : Proses perhitungan *Moore-Penrose* dengan Persamaan 2.3.
Baris 10 : Memanggil fungsi untuk mengubah matriks target.
Baris 11 s.d 12 : Proses *perhitungan output weight* dengan mengalikan hasi perhitungan matriks Moore-Penrose (H^+) dengan target yang telah dikonversi.

5.1.11 Implementasi Proses Perhitungan *Output Layer*

Implementasi proses perhitungan *output layer* digunakan pada proses *testing*. Operasi perkalian pada perhitungan *output layer* ini menggunakan *library* JAMA. Implementasi proses perhitungan *output layer* dapat dilihat pada Kode Program 5.12.

Kode Program 5.12 Implementasi Proses Perhitungan *Output Layer*

```
1  void outputLayer() {
2    Matrix outputH = new Matrix(hiddenOutput);
3    Matrix outputW = new Matrix(outputWeight);
4    Matrix result = outputH.times(outputW);
5  }
```

Penjelasan Kode Program 5.12 tentang implementasi proses perhitungan *output layer* adalah sebagai berikut:

Baris 2 s.d 4 : Proses perkalian antara keluaran *hidden layer* dari proses *testing* dan *output weight* yang dihasilkan dari proses *training*.

5.1.12 Implementasi Hasil Klasifikasi

Implementasi hasil klasifikasi akan menyaring/mem-*filter* hasil dari *output layer* yang dilakukan saat proses *testing*. *Output* dari implementasi ini berupa kelas 1, 2 atau 3. Implementasi hasil klasifikasi dapat dilihat pada Kode Program 5.13.

Kode Program 5.13 Implementasi Hasil Klasifikasi

```
1  void classification() {
2    kelas = new int[output.length];
3    int index = 0;
4    max = output[0][0];
5
6    for (int i = 0; i < output.length; i++) {
7      for (int j = 0; j < output[0].length; j++) {
8        if (output[i][j] > max) {
9          max = output[i][j];
10         index = j;
11       }
12     }
13     kelas[i] = index;
14   }
15 }
```

Penjelasan Kode Program 5.13 tentang implementasi hasil klasifikasi adalah sebagai berikut:

Baris 2 s.d 4 : Inisialisasi variabel yang digunakan untuk perhitungan hasil klasifikasi.

Baris 6 s.d 14 : Proses klasifikasi kelas kualitas air.

5.1.13 Implementasi Proses Perhitungan Akurasi

Implementasi proses perhitungan akurasi digunakan untuk mengetahui seberapa akurat tingkat klasifikasi yang dihasilkan pada penelitian ini. Implementasi proses perhitungan akurasi dapat dilihat pada Kode Program 5.13.

Kode Program 5.13 Implementasi Proses Perhitungan Akurasi

```
1  void accuracy(int fold) {
2    int prediksiBenar = 0;
3    for (int i = 0; i < kelas.length; i++) {
4      if (targetTesting[i][0] == (kelas[i]+1)) {
5        prediksiBenar++;
6      }
7    }
8    double accuracy = ((double)prediksiBenar/kelas.length)*
9                       100;
1  }
0
```

Penjelasan Kode Program 5.13 tentang implementasi proses seleksi adalah sebagai berikut:

Baris 2 s.d 7 : Proses membandingkan hasil klasifikasi/prediksi baru dengan klasifikasi sebenarnya, kemudian menghitung jumlah data benarnya/data yang sama.

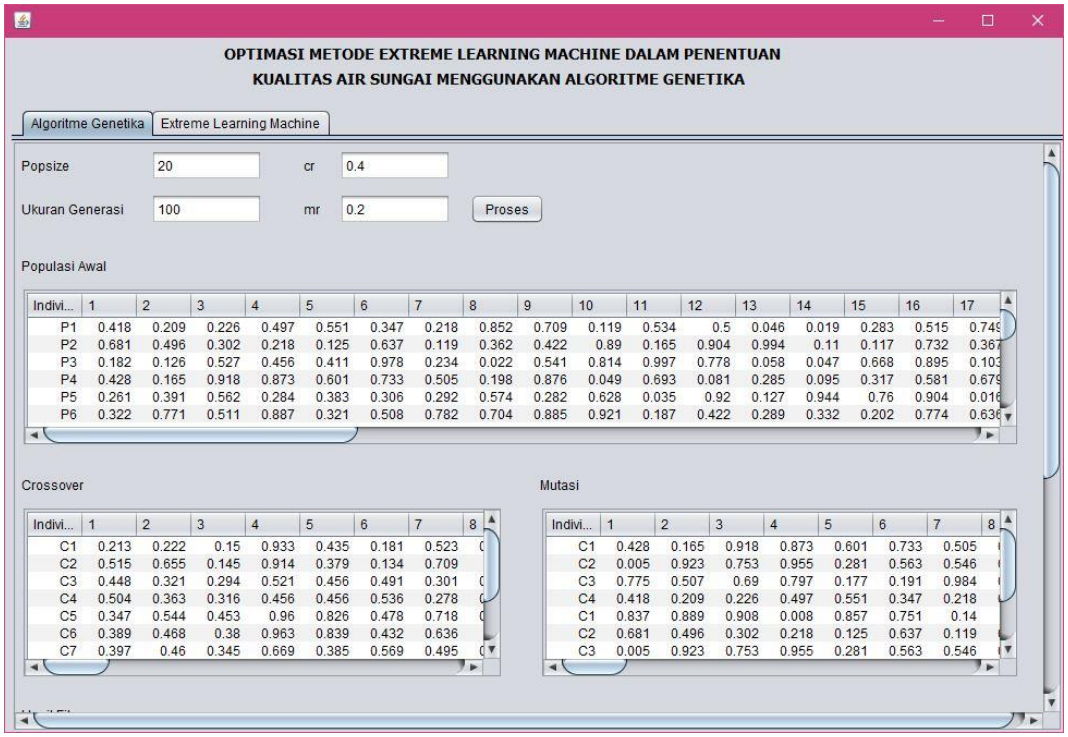
Baris 8 s.d 9 : Proses perhitungan akurasi menggunakan Persamaan 2.8.

5.2 Implementasi Antarmuka

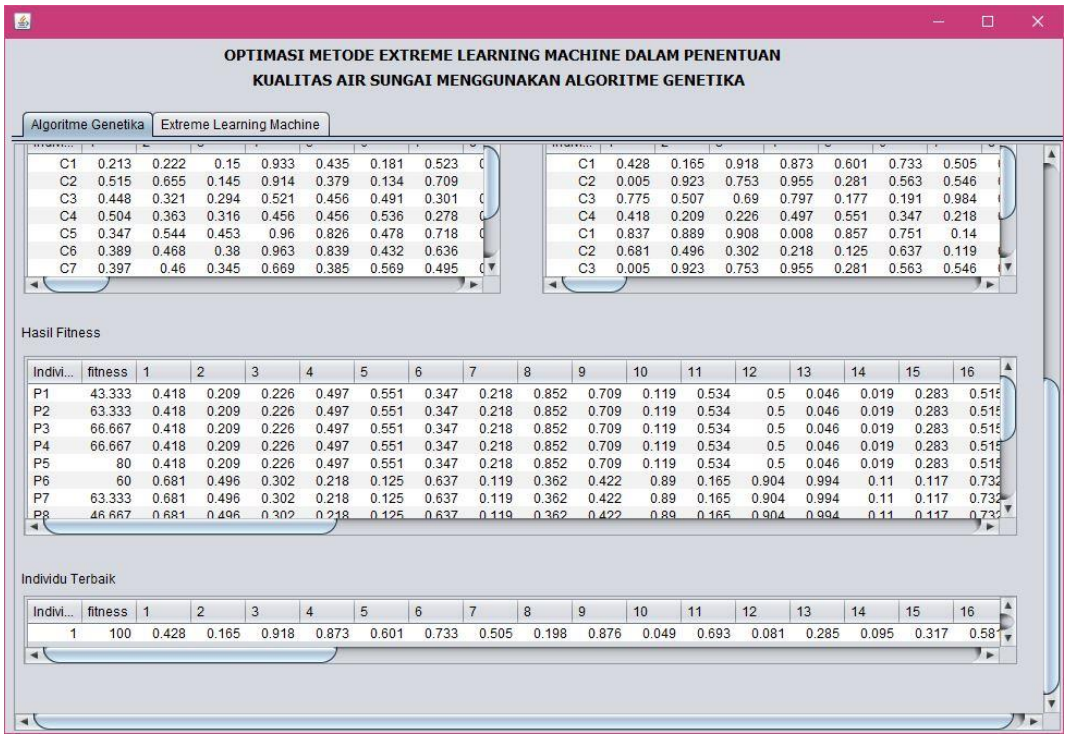
Pada sub-bab implementasi antarmuka akan membahas dan menampilkan hasil implementasi dari rancangan yang telah dibuat pada bab perancangan antarmuka sebelumnya. Implementasi antarmuka pada penelitian ini terdiri dari 2 bagian, yaitu Algoritme Genetika dan *Extreme Learning Machine* serta 4 sub bagian dari *Extreme Learning Machine* yang meliputi normalisasi data, data *fold*, *input weight* dan bias, evaluasi dan hasil prediksi.

5.2.1 Implementasi Antarmuka Algoritme Genetika

Antarmuka Algoritme Genetika merupakan halaman antarmuka pertama pada penelitian ini. Antarmuka ini berfungsi untuk menerima masukan parameter dari *user* serta menampilkan proses perhitungan yang terjadi dalam Algoritme Genetika. Implementasi antarmuka Algoritme Genetika dapat dilihat pada Gambar 5.1 dan 5.2.



Gambar 5.1 Implementasi antarmuka algoritme genetika (1)



Gambar 5.2 Implementasi antarmuka algoritme genetika (2)

Pada Gambar 5.1 ditunjukkan implementasi antarmuka Algoritme Genetika bagian pertama. Dalam tersebut terdapat 4 *textfield* yang digunakkan sebagai parameter masukan *user*, diantaranya *popsize* (ukuran populasi), ukuran generasi, cr (*crossover rate*) dan mr (*mutation rate*). Disamping *textfield*

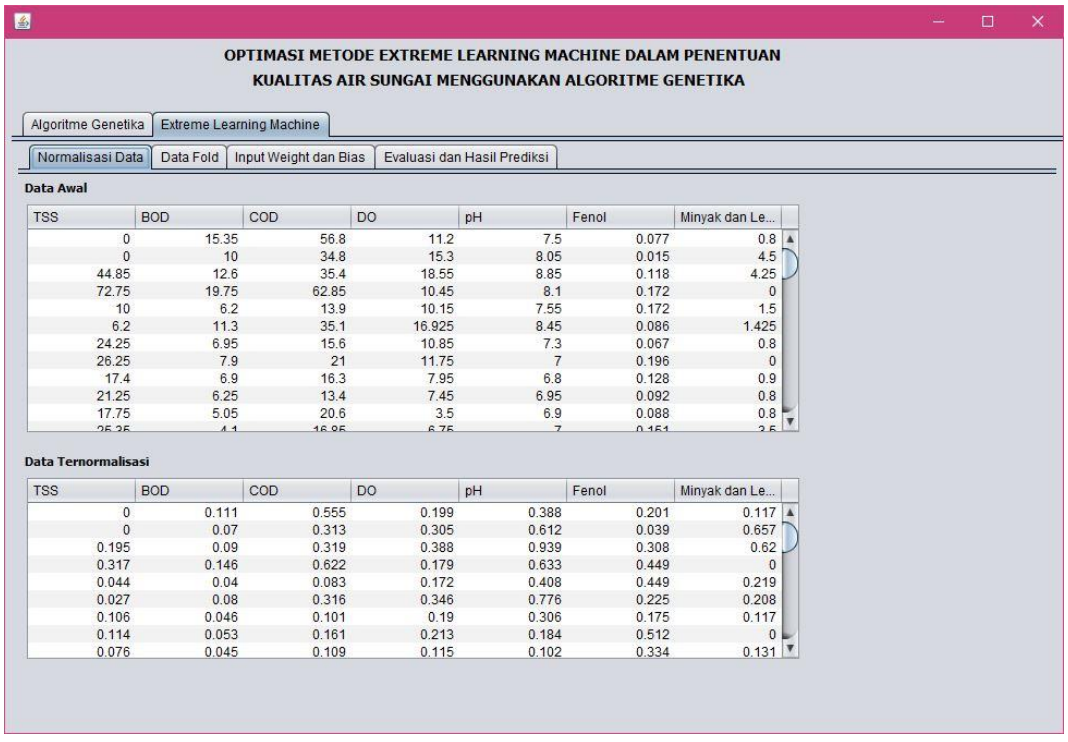
parameter, terdapat tombol proses yang berfungsi untuk melakukan perhitungan dalam sistem. Tahap awal akan dibangkitkan populasi awal, dan kemudian ditampilkan pada tabel populasi awal. Tahap selanjutnya yaitu reproduksi, yang meliputi *crossover* dan mutasi. Kedua proses perhitungan reproduksi ini juga akan ditampilkan pada tabel.

Sedangkan pada Gambar 5.2 ditunjukkan bagian lanjutan atau bagian 2 dari proses Algoritme Genetika yaitu proses evaluasi dan seleksi. Perhitungan proses evaluasi atau menghitung nilai fitness, ditampilkan pada tabel hasil *fitness*. Kemudian akan dilakukan proses seleksi individu untuk memilih individu terbaik. Individu terbaik ditampilkan pada tabel individu terbaik. Individu ini merupakan *input weight* yang menghasilkan *fitness* terbaik.

5.2.2 Implementasi Antarmuka *Extreme Learning Machine*

Antarmuka *Extreme Learning Machine* merupakan halaman antarmuka kedua pada penelitian ini. Antarmuka ini berfungsi untuk menampilkan proses perhitungan yang terjadi dalam *Extreme Learning Machine* (ELM). Pada halaman ini terdapat 4 sub menu, diantaranya normalisasi data, data *fold*, *input weight* dan bias, evaluasi dan hasil prediksi.

5.2.2.1 Implementasi Antarmuka Normalisasi Data



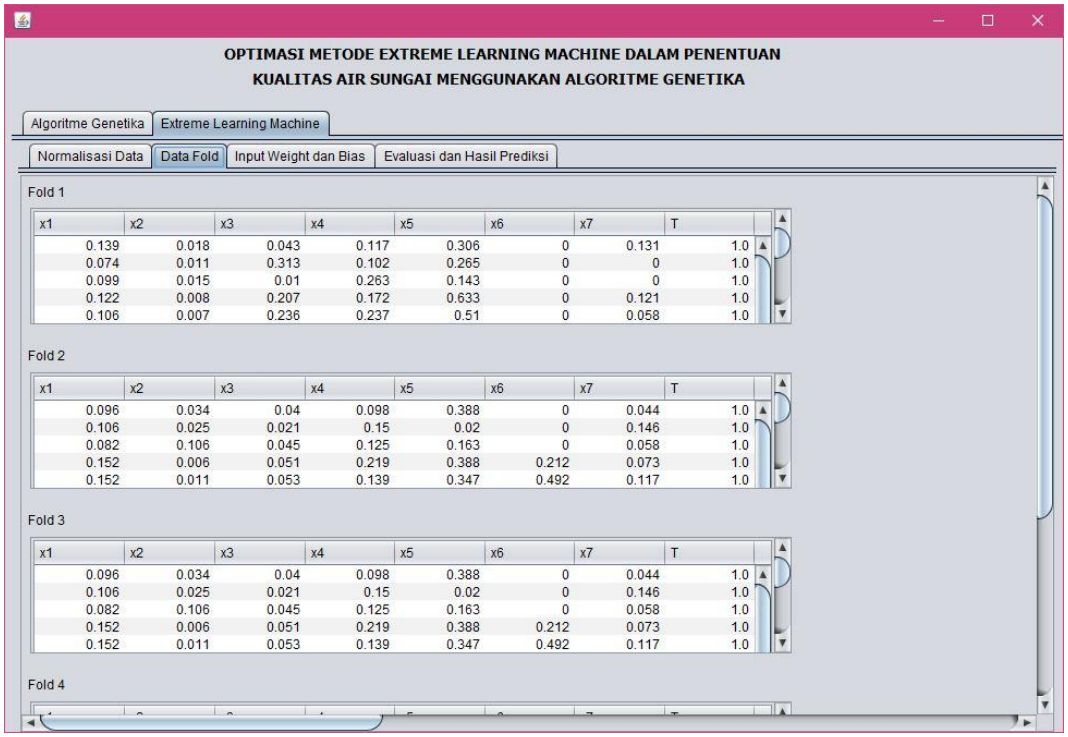
Gambar 5.3 Implementasi antarmuka normalisasi data

Antarmuka normalisasi data bertujuan untuk menampilkan data awal berbentuk *file* Excel dengan ekstensi xls yang dibaca oleh sistem dan menampilkan data ternormalisasi. Data awal adalah data sebelum dilakukan proses perhitungan.

Data awal akan ditampilkan pada tabel data awal. Sedangkan data ternormalisasi merupakan hasil dari proses perhitungan normalisasi yang kemudian akan ditampilkan pada tabel data ternomalisasi. Implementasi antarmuka normalisasi data dapat dilihat pada Gambar 5.3.

5.2.2.2 Implementasi Antarmuka Data *Fold*

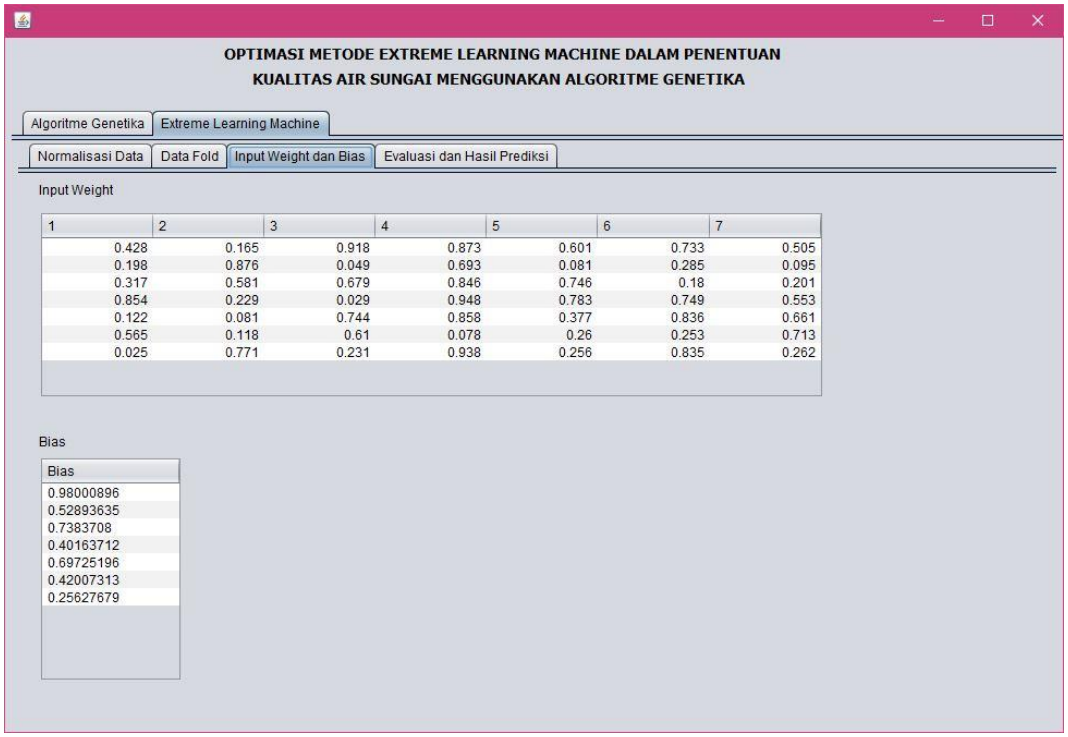
Pada penelitian ini data yang digunakan dibagi menjadi 5 *fold*. Halaman antarmuka data *fold* ini berfungsi untuk menampilkan ke 5 *fold* (*fold* 1-5) tersebut. Data *fold* ini ditampilkan dengan membaca *file* Excel. Masing-masing *fold* akan menempati 1 tabel. Implementasi antarmuka data *fold* dapat dilihat pada Gambar 5.4.



Gambar 5.4 Implementasi antarmuka data *fold*

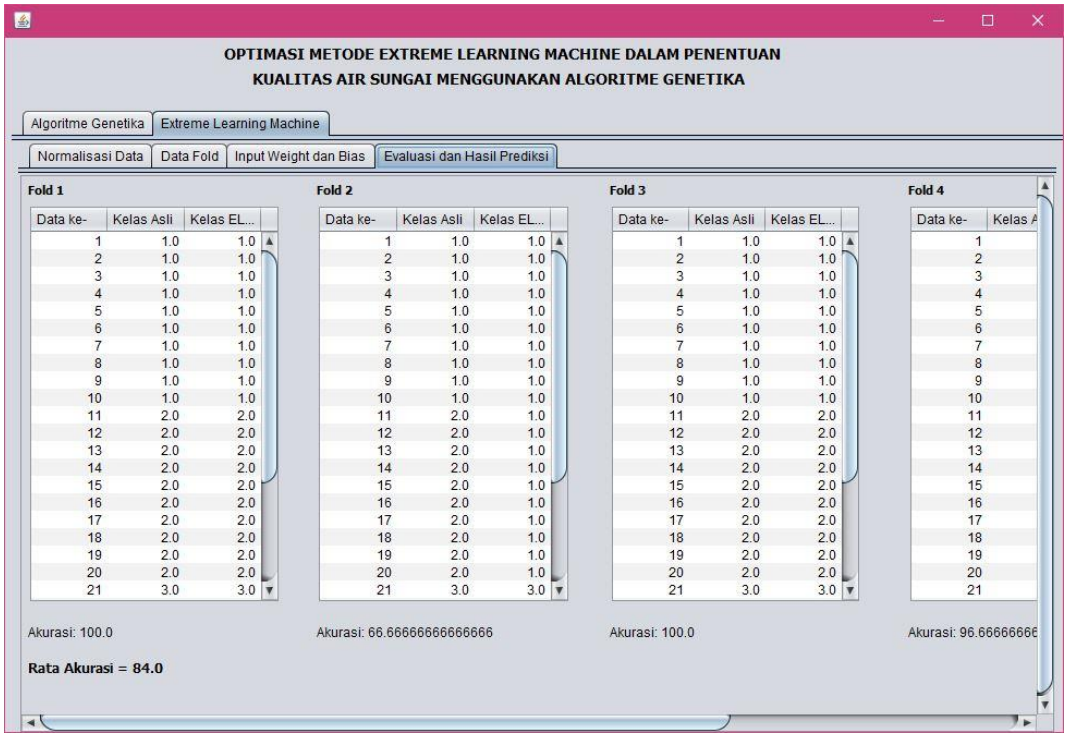
5.2.2.3 Implementasi Antarmuka *Input Weight* dan Bias

Antarmuka *input weight* dan bias bertujuan untuk menampilkan *input weight* dan bias optimal yang digunakan dalam proses perhitungan sistem. Bias didapat dengan membaca *file* Excel, sedangkan *input weight* merupakan nilai dari individu terbaik yang didapatkan dari perhitungan Algoritme Genetika. Hasil pembacaan nilai bias kemudian akan dimasukkan ke dalam tabel bias, dan nilai individu terbaik akan dimasukkan ke dalam tabel *input weight*. Karena ukuran matriks individu terbaik dan *input weight* berbeda, maka sistem akan melakukan proses perubahan ukuran matriks terlebih dahulu. Implementasi antarmuka *input weight* dan bias dapat dilihat pada Gambar 5.5.



Gambar 5.5 Implementasi antarmuka input weight dan bias

5.2.2.4 Implementasi Antarmuka Evaluasi dan Hasil Prediksi



Gambar 5.6 Implementasi antarmuka evaluasi dan hasil prediksi

Pada antarmuka evaluasi dan hasil prediksi yang ditunjukkan pada Gambar 5.6, akan ditampilkan perbandingan kelas asli dengan kelas hasil prediksi tiap *fold*, akurasi pada tiap *fold* dan rata-rata akurasinya. Data kelas asli didapat dengan membaca *file* data *testing* yang berbentuk *file* Excel, sedangkan data hasil prediksi didapatkan dari perhitungan ELM menggunakan *input weight* terbaik. Data kelas asli dimasukan dalam kolom kelas asli pada tabel tiap *fold*, dan data hasil prediksi dimasukkan ke kolom kelas ELM pada tiap *fold*.