BAB 5 IMPLEMENTASI

Bab ini akan menjelaskan tentang proses implementasi berdasarkan hasil perancangan yang telah dilakukan. Bab ini terdiri dari implementasi algritma *backpropagation* dan algoritma genetika serta implementasi antarmuka untuk proses peramalan jumlah kasus penyakit.

5.1 Implementasi Algoritma

5.1.1 Implementasi Algoritma Genetika

Implementasi algoritma genetika ini dijalankan melalui beberapa proses untuk mendapatkan individu terbaik dengan melakukan beberapa perulangan atau generasi keculai proses inisialisasi populasi awal. Proses ini terdiri dari inisialisasi populasi awal, *crossover*, mutasi, evaluasi, dan seleksi.

5.1.1.1 Implementasi Algoritma Inisialisasi Populasi Awal

Inisialisasi populasi awal hanya dilakukan satu kali pada awal proses optimasi menggunakan algoritma genetika. Untuk melakukan inisialisasi populasi awal dibutuhkan *popSize* untuk menentukan jumlah populasi awal yang diinginkan. Implementasi algoritma inisialisasi populasi awal ditunjukkan dengan Kode Program 5.1:

```
1     public Algen(int popSize, double cr, double mr, int
2  generasi, int pData) {
3          this.popSize = popSize;
4          this.cr = cr;
5          this.mr = mr;
6          this.generasi = generasi;
7          this.pData = pData;
8
9          Random r = new Random();
10         populasi                    =                    new
11 double[popSize][(int)Math.pow((pData+1),2)];
12         for (int i = 0; i < populasi.length; i++) {
13             for (int j = 0; j < populasi[i].length; j++) {
14                 populasi[i][j] = -1 + (double) Math.random()
15 * 1;
16             }
17         }
18     }
```

Kode Program 5.1 Implementasi Algoritma Inisialisasi Populasi Awal

Berikut penjelasan dari Kode Program 5.2:

1. Baris 9 sampai 17 digunakan untuk membangkitkan populasi dari range -1 sampai 1.

5.1.1.2 Implementasi Algoritma *Crossover*

Pada tahap *crossover* dilakukan reproduksi dengan memilih 2 induk atau *parent* secara acak dari populasi. Jumlah *child* yang dihasilkan dipengaruhi oleh

nilai *cr* atau *crossover rate* dengan jumlah populasi atau *popSize*. Dibutukan pula *cut point* untuk menentukan gen mana yang akan ditukarkan. Implementasi algritma *crossover* ditunjukkan dengan Kode Program 5.2:

```
1  public double[][] crossover(double[][] populasi, double cr) {
2          double[]           parent1           =           new
3  double[(int)Math.pow((pData+1),2)];//169
4          double[]           parent2           =           new
5  double[(int)Math.pow((pData+1),2)];//169
6
7          int cut1 = new Random().nextInt(pData) + 1;
8          int cut2 = new Random().nextInt((pData*(pData+1))-1)
9  + 1;//155
10
11         int jumlahChild = (int) Math.round(populasi.length *
12 cr);
13         if (jumlahChild <= 1 && jumlahChild >= 0) {
14             jumlahChild = 1;
15         }
16         double[][]           child           =           new
17 double[jumlahChild][(int)Math.pow((pData+1),2)];
18
19         for (int i = 0; i < child.length; i++) {
20             parent1              =              populasi[new
21 Random().nextInt(populasi.length)];
22             parent2              =              populasi[new
23 Random().nextInt(populasi.length)];
24             while (parent1 == parent2) {
25                 parent2            =            populasi[new
26 Random().nextInt(populasi.length)];
27             }
28             for (int j = 0; j < cut1; j++) {
29                 child[i][j] = parent1[j];
30             }
31             for (int j = cut1; j < (pData+1); j++) {
32                 child[i][j] = parent2[j];
33             }
34             for (int j = (pData+1); j < cut2 + (pData+1); j++)
35 {
36                 child[i][j] = parent1[j];
37             }
38             for   (int   j   =   cut2   +   (pData+1);   j   <
39 child[0].length; j++) {
40                 child[i][j] = parent2[j];
41             }
42         }
43         return child;
44     }
```

Kode Program 5.2 Implementasi Algoritma *Crossover*

Berikut penjelasan Kode Program 5.2:

1. Baris 7 dan 8 digunakan untuk menentukan *cut point* 1 dan *cut point* 2 secara acak.
2. Baris 11 sampai 17 digunakan untuk menghitung jumlah *child* yang akan dihasilkan.

3. Baris 19 sampai 27 digunakan untuk menentukan *parent* mana yang akan dijadikan induk untuk proses *crossover* secara acak.
4. Baris 28 sampai 42 digunakan untuk memasukkan gen dari induk 1 ke induk 2 ke *child* hasil *crossover* sesuai dengan *cut point* yang sudah ditentukan.

5.1.1.3 Implementasi Algoritma Mutasi

Pada tahap mutasi dilakukan reproduksi dengan memilih satu *parent* sebagai induk untuk proses mutasi serta gen yang akan dimutasi secara acak. Jumlah *child* yang dihasilkan dipengaruhi oleh nilai *mr* atau *mutation rate* dengan jumlah populasi atau *popSize.* Dibutuhkan juga nilai *r* yang dibangkitkan dengan nilai antara -0,1 sampai 0,1 serta nilai maksimum dan minimum dari induk yang terpilih. Impementasi algoritma mutasi ditunjukkan dengan Kode Program 5.3:

```
1  public double[][] mutasi(double[][] populasi, double mr) {
2
3          double[]            temp            =            new
4  double[(int)Math.pow((pData+1),2)];
5          int jumlahChild = (int) Math.round(populasi.length *
6  mr);
7          if (jumlahChild <= 1 && jumlahChild >= 0) {
8              jumlahChild = 1;
9          }
10         double[][]           child           =            new
11 double[jumlahChild][(int)Math.pow((pData+1),2)];
12         double[][]          parent           =            new
13 double[jumlahChild][(int)Math.pow((pData+1),2)];
14         int[] gen = new int[jumlahChild];
15         double[] max = new double[jumlahChild];
16         double[] min = new double[jumlahChild];
17         double[] r = new double[jumlahChild];
18         double[] mutasi = new double[jumlahChild];
19
20         for (int i = 0; i < parent.length; i++) {
21             parent[i]             =             populasi[new
22 Random().nextInt(populasi.length)];
23         }
24
25         for (int i = 0; i < gen.length; i++) {
26             gen[i]                    =                    new
27 Random().nextInt(populasi[0].length);
28         }
29
30         for (int i = 0; i < r.length; i++) {
31             r[i] = (-0.1) + ((double) Math.random() * 0.1);
32         }
33
34         for (int i = 0; i < max.length; i++) {
35             max[i] = parent[i][0];
36             for (int j = 0; j < parent[i].length; j++) {
37                 if (parent[i][j] > max[i]) {
38                     max[i] = parent[i][j];
39                 }
40             }
41         }
42
```

```
43          for (int i = 0; i < min.length; i++) {
44              min[i] = parent[i][0];
45              for (int j = 0; j < parent[i].length; j++) {
46                  if (parent[i][j] < min[i]) {
47                      min[i] = parent[i][j];
48                  }
49              }
50          }
51
52          for (int i = 0; i < mutasi.length; i++) {
53              mutasi[i] = parent[i][gen[i]] + (r[i] * (max[i] -
54  min[i]));
55          }
56
57          for (int i = 0; i < child.length; i++) {
58              for (int j = 0; j < child[i].length; j++) {
59                  if (j == gen[i]) {
60                      child[i][j] = mutasi[i];
61                  } else {
62                      child[i][j] = parent[i][j];
63                  }
64              }
65          }
66          return child;
67      }
```

Kode Program 5.3 Implementasi Algoritma Mutasi

Berikut penjelasan dari Kode Program 5.3:

1. Baris 5 sampai 9 digunakan untuk menghitung jumlah *child* yang akan dihasilkan.
2. Baris 20 sampai 23 digunakan untuk menentukan *parent* yang akan dijadikan induk secara acak.
3. Baris 25 sampai 28 digunakan untuk menentukan gen yang akan dimutasi secara acak.
4. Baris 30 sampai 32 digunakan untuk menentukan nilai *r* secara acak.
5. Baris 34 sampai 41 digunakan untuk mengitung nilai maksimum dari induk yang terpilih.
6. Baris 43 sampai 50 digunakan untuk menghitung nilai minimum dari induk yang terpilih.
7. Baris 52 sampai 55 digunakan untuk menghitung nilai mutasi dari gen yang terpilih.
8. Baris 57 sampai 65 digunakan untuk memasukkan nilai mutasi kedalam *child* yang dihasilkan.

5.1.1.4 Implementasi Algoritma Evaluasi

Proses evaluasi dibagi menjadi dua yaitu memperbarui populasi awal dengan *child* yang dihasilkan dari proses reproduksi kemudian menghitung nilai

*fitness*nya. Impelementasi algoritma evaluasi ditunjukkan dengan Kode Program 5.4:

```
1   |     public     double[][]     updatePop(double[][]     popLama,
2   | double[][] crossover, double[][] mutasi) {
3   |         double[][]  popBaru  =  new  double[popLama.length  +
4   | crossover.length                                              +
5   | mutasi.length][(int)Math.pow((pData+1),2)];
6   |
7   |         for (int i = 0; i < popBaru.length - (crossover.length
8   | + mutasi.length); i++) {
9   |             for (int j = 0; j < popBaru[i].length; j++) {
10  |                 popBaru[i][j] = popLama[i][j];
11  |             }
12  |         }
13  |
14  |         for  (int  i  =  popLama.length;  i  <  popBaru.length  -
15  | mutasi.length; i++) {
16  |             for (int j = 0; j < popBaru[i].length; j++) {
17  |                 popBaru[i][j]      =      crossover[i      -
18  | popLama.length][j];
19  |             }
20  |         }
21  |
22  |         for (int i = popLama.length + crossover.length; i <
23  | popBaru.length; i++) {
24  |             for (int j = 0; j < popBaru[i].length; j++) {
25  |                 popBaru[i][j] = mutasi[i - (popLama.length +
26  | crossover.length)][j];
27  |             }
28  |         }
29  |         return popBaru;}
30  |     public double fitness(double[] populasi, double[][] data,
31  | int pData) {
32  |         double[][] normal = normalisasi(data);
33  |         double[] bw = bobotw(populasi, pData);
34  |         double[][] bv = bobotv(populasi, pData);
35  |         double[][] z_in = new double[data.length][bv.length];
36  |         double[][]            z            =            new
37  | double[z_in.length][z_in[0].length];
38  |         double[] y_in = new double[z.length];
39  |         double[] y = new double[y_in.length];
40  |         double[] denorm = new double[y.length];
41  |         for (int i = 0; i < data.length; i++) {
42  |             z_in[i] = z_in(i, normal, bv);
43  |             z[i] = z(z_in[i]);
44  |             y_in[i] = y_in(z[i], bw);
45  |             y[i] = y(y_in[i]);
46  |             denorm[i] = denormalisasi(y[i], data);
47  |         }
48  |         double mse = 0;
49  |         mse = hitMSE(data, denorm);
50  |         double fitness = 1 / mse;
51  |         return fitness;
52  |     }
```

Kode Program 5.4 Implementasi Algoritma Evaluasi

Berikut penjelasan dari Kode Program 5.4:

1. Baris 1 sampai 29 digunakan untuk memasukkan nilai dari populasi lama, *child* hasil *crossover*, dan *child* hasil mutasi kedalam populasi baru.
2. Baris 30 sampai 52 digunakan untuk menghitung nilai *fitness* dengan menggunakan proses *feedforward*.

5.1.1.5 Implementasi Algoritma Seleksi

Pada tahap seleksi dilakukan pemilihan individu terbaik sebanyak *popSize*. Untuk memilih individu terbaik dibutuhkan nilai *fitness* yang akan diurutkan dari besar ke kecil. Implementasi algoritma seleksi ditunjukkan dengan Kode Program 5.5:

```
1     public double[][] seleksi(double[][] popBaru, double[]
2  fitness) {
3          double[][]         populasiBaru         =         new
4  double[popSize][(int)Math.pow((pData+1),2)];
5          double[][]         fitnessAkhir         =         new
6  double[fitness.length][2];
7
8          for (int i = 0; i < fitnessAkhir.length; i++) {
9              fitnessAkhir[i][0] = fitness[i];
10             fitnessAkhir[i][1] = i;
11         }
12
13         double[] temp = new double[2];
14         for (int i = 0; i < fitnessAkhir.length - 1; i++) {
15             for (int j = 1; j < fitnessAkhir.length - i; j++)
16 {
17                 if     (fitnessAkhir[j     -     1][0]     <
18 fitnessAkhir[j][0]) {
19                     temp[0] = fitnessAkhir[j - 1][0];
20                     temp[1] = fitnessAkhir[j - 1][1];
21                     fitnessAkhir[j      -      1][0]      =
22 fitnessAkhir[j][0];
23                     fitnessAkhir[j      -      1][1]      =
24 fitnessAkhir[j][1];
25                     fitnessAkhir[j][0] = temp[0];
26                     fitnessAkhir[j][1] = temp[1];
27                 }
28             }
29         }
30
31         for (int i = 0; i < populasiBaru.length; i++) {
32             populasiBaru[i]          =          popBaru[(int)
33 fitnessAkhir[i][1]];
34         }
35
36         return populasiBaru;
37     }
```

Kode Program 5.5 Implementasi Algoritma Seleksi

Berikut penjelasan dari Kode Program 5.5:

1. Baris 8 hingga 11 digunakan untuk memasukkan nilai *fitness* ke dalam *fitness*Akhir dan membuat nilai *index* yang akan digunakan untuk proses pengurutan.

2. Baris 13 sampai 29 digunakan untuk mengurutkan nilai *fitness*Akhir dari besar ke kecil.
3. Baris 31 sampai 34 digunakan untuk mengambil individu terbaik sesuai dengan *index* yang telah diurutkan.

5.1.2 Implementasi Algoritma *Backpropagation*

Pada implementasi algoritma *backpropagation* dilakukan beberapa proses perulangan atau iterasi untuk proses pelatihan. Proses ini terdiri dari *feedforward* dan *backproapgation error* dan pembaruan bobot. Selain itu terdapat proses normalisasi data, denormalisasi data, serta peritungan nilai MSE.

5.1.2.1 Implementasi Algoritma Normalisasi Data

Normalisasi data digunakan untuk merubah data kedalam suatu interval tertentu, dalam penelitian ini data dinormalisasi menjadi interval (0,1). Untuk melakukan proses normalisasi data dibutuhkan nilai maksimum dan minimum dari data yang ingin dinormalisasi. Setelah itu untuk menghitung normalisasi data dengan mengurangi data yang akan dinormalisasi dengan nilai minum kemudian membaginya dengan hasil pengurangan nilai maksimum dengan nilai minimum. Implementasi algoritma untuk normalisasi data ditunjukan dengan Kode Program 5.1.

```
1   private double[][] normalisasi(double data[][]) {
2           double max = data[0][0];
3           for (int i=0; i<data.length; i++){
4               for (int j=0; j<data[0].length; j++){
5                   if (data[i][j] > max){
6                       max = data[i][j];
7                   }
8               }
9           }
10          double min = data[0][0];
11          for (int i=0; i<data.length; i++){
12              for (int j=0; j<data[0].length; j++){
13                  if (data[i][j] < min){
14                      min = data[i][j];
15                  }
16              }
17          }
18          double[][]            norm            =            new
19  double[data.length][data[0].length];
20          for (int i = 0; i < norm.length; i++) {
21              for (int j = 0; j < norm[i].length; j++) {
22                  norm[i][j] = (data[i][j] - min) / (max - min);
23              }
24          }
25          return norm;
26      }
```

Kode Program 5.6 Implementasi Normalisasi Data

Berikut penjelasan dari Kode Program 5.6:

1. Baris 2 sampai 9 digunakan untuk mencari nilai maksimum.

2. Baris 10 sampai 17 digunakan untuk mencari nilai minimum.

3. Baris 18 sampai 25 digunakan untuk mengitung nilai normalisasi.

5.1.2.2 Implementasi Algoritma *Feedforward*

Pada tahap *feedforward* terdiri dari beberapa proses, yaitu menghitung nilai *z_in, z, y_in ,* dan *y*. Implementasi algoritma *feedforward* ditunjukkan dengan Kode Program 5.7:

```
1     private double[] z_in(int a, double[][] normal,
2   double[][] bobotv) {
3           double[] z_in = new double[bobotv.length];
4           for (int i = 0; i < z_in.length; i++) {
5               for (int j = 0; j < bobotv[0].length-1; j++) {
6                   z_in[i] = z_in[i] + (normal[a][j] *
7   bobotv[i][j]);
8               }
9               z_in[i] = z_in[i] + bobotv[i][6];
10          }
11          return z_in;
12      }
13      private double[] z(double[] z_in) {
14          double[] z = new double[z_in.length];
15          for (int i = 0; i < z.length; i++) {
16              z[i] = 1 / (1 + Math.exp(-(z_in[i])));
17          }
18          return z;
19      }
20      private double y_in(double[] z, double[] bobotw) {
21          double y_in = 0;
22          for (int i = 0; i < z.length; i++) {
23              y_in = y_in + (z[i] * bobotw[i]);
24          }
25          y_in = y_in + bobotw[6];
26          return y_in;
27      }
28      private double y(double y_in) {
29          double y = 1 / (1 + Math.exp(-(y_in)));
30          return y;
31      }
```

Kode Program 5.7 Implementasi Algoritma *Feedforward*

Berikut penjelasan Kode Program 5.7:

1. Baris 4 sampai 112 digunakan untuk menghitung nilai *z_in*.

2. Baris 15 sampai 17 digunakan untuk menghitung nilai *z*.

3. Baris 22 sampai 26 digunakan untuk menghitung nilai *y_in*.

4. Baris 29 digunakan untuk menghitung nilai *y*.

5.1.2.3 Implementasi Algoritma *Backpropagation Error*

Pada tahap *backpropagation error* terdiri dari beberapa proses, yaitu menghitung nilai error, faktor koreski output, delta bobot w, delta_in, delta, dan

delta bobot v. Implementasi algoritma *backpropagation error* ditunjukkan pada Kode Program 5.8:

```
1  private double nError(int a, double y, double[][] normal) {
2          double error = normal[a][normal[0].length-1] - y;
3          return error;
4  }
5  private double faktorKoreksiO(double error, double y) {
6          double koreksi = error * (y * (1 - y));
7          return koreksi;
8      }
9  private double[] deltaBobotW(double koreksi, double[] z) {
10         double[] deltaBobotw = new double[z.length + 1];
11 for (int i = 0; i < deltaBobotw.length - 1; i++) {
12 deltaBobotw[i] = alfa * koreksi * z[i];        }
13         deltaBobotw[deltaBobotw.length - 1] = alfa * koreksi;
14         return deltaBobotw;
15     }
16 private double[] deltaIn(double koreksi, double[] bobotw) {
17         double[] deltain = new double[bobotw.length - 1];
18         for (int i = 0; i < deltain.length; i++) {
19             deltain[i] = koreksi * bobotw[i];
20         }
21         return deltain;
22     }
23 private double[] delta(double[] deltain, double[] z) {
24         double[] delta = new double[deltain.length];
25         for (int i = 0; i < delta.length; i++) {
26             delta[i] = deltain[i] * z[i] * (1 - z[i]);
27         }
28         return delta;
29     }
30 private double[][] deltaBobotV(int a, double[][] data,
31 double[] delta, double[][] bobotv) {
32         double[][] deltaBobotv = new
33 double[bobotv.length][bobotv[0].length];
34         for (int i = 0; i < deltaBobotv.length; i++) {
35             for (int j = 0; j < deltaBobotv[0].length - 1;
36 j++) {
37                 deltaBobotv[i][j] = alfa * delta[i] *
38 data[a][j];
39             }
40             deltaBobotv[i][deltaBobotv.length] = alfa *
41 delta[i];
42         }
43         return deltaBobotv;
44     }
```

Kode Program 5.8 Implementasi Algoritma *Backpropagation Error*

Berikut penjelasan dari Kode Program 5.8:

1. Baris 2 digunakan untuk menghitung nilai error.
2. Baris 6 digunakan untuk menghitung nilai faktor koreksi *output*.
3. Baris 12 sampai 15 digunakan untuk menghitung nilai delta bobot w.
4. Baris 20 sampai 22 digunakan untuk menghitung nilai delta_in.

5. Baris 27 sampai 29 digunakan untuk menghitung nilai delta.
6. Baris 36 sampai 44 digunakan untuk menghitung nilai delta bobot v.

5.1.2.4 Implementasi Algoritma Pembaruan Bobot

Pada tahap pembaruan bobot dilakuan pembaruan bobot w dan bobot v dengan menjumlakan delta bobot v dan w dengan bobot v dan w. Implementasi algoritma pembaruan bobot ditunjukkan dengan Kode Program 5.9:

```
1  private double[][] bobotVBaru(int a, double[][] deltabobotv,
2  double[][] bobotv) {
3          double[][]          bobotVbaru          =          new
4  double[bobotv.length][bobotv[0].length];
5          for (int i = 0; i < bobotVbaru.length; i++) {
6              for (int j = 0; j < bobotVbaru[0].length; j++) {
7                  bobotVbaru[i][j]   =   deltabobotv[i][j]   +
8  bobotv[i][j];
9              }
10         }
11         return bobotVbaru;
12     }
13 private  double[]  bobotWBaru(double[]  deltabobotw,  double[]
14 bobotw) {
15         double[] bobotWbaru = new double[bobotw.length];
16         for (int i = 0; i < bobotWbaru.length; i++) {
17             bobotWbaru[i] = deltabobotw[i] + bobotw[i];
18         }
19         return bobotWbaru;
20     }
21
```

Kode Program 5.9 Implementasi Algoritma Pembaruan Bobot

Berikut penjelasan dari Kode Program 5.9:

1. Baris 6 sampai 11 digunakan untuk memperbarui bobot v.
2. Baris 17 sampai 19 digunakan untuk memperbarui bobot w.

5.1.2.5 Implementasi Algoritma Denormalisasi Data

Proses denormalisasi data digunakan untuk merubah data hasil normalisasi menjadi data aktual. Implementasi algoritma denormalisasi data ditunjukkan dengan Kode Program 5.10:

```
1  public double denormalisasi(double y, double[][] dataUji){
2          double max = dataUji[0][0];
3          for (int i=0; i<dataUji.length; i++){
4              for (int j=0; j<dataUji[0].length; j++){
5                  if (dataUji[i][j] > max){
6                      max = dataUji[i][j];
7                  }
8              }
9          }
10         double min = dataUji[0][0];
11         for (int i=0; i<dataUji.length; i++){
12             for (int j=0; j<dataUji[0].length; j++){
13                 if (dataUji[i][j] < min){
14                     min = dataUji[i][j];
```

```
15                  }
16              }
17          }
18          double denorm = (y * (max-min))+min;
19          return denorm;
20      }
```

Kode Program 5.10 Implementasi Algoritma Denormalisasi Data

Berikut penjelasan Kode Program 5.10:

1. Baris 3 sampai 9 digunakan untuk menghitung nilai maksimum dari data.
2. Baris 11 sampai 17 digunakan untuk menghitung nilai minimum dari data.
3. Baris 18 digunakan untuk menghitung nilai denormalisasi.

5.1.2.6 Implementasi Algoritma Perhitungan Nilai MSE

Peritungan nilai MSE dikerjakan dengan cara menghitung jumlah selisih kuadrat antara data target dan data hasil peramalan lalu dibagi oleh jumlah data. Implementasi algoritma perhitungan nilai MSE ditunjukkan dengan Kode Program 5.11:

```
1  public double hitMSE(double[][] dataUji, double[] denorm){
2          double MSE =0;
3          double error =0;
4          for (int i=0; i<dataUji.length; i++){
5              error            =            error            +
6  Math.pow((dataUji[i][dataUji[0].length-1]-denorm[i]),2);
7          }
8          return MSE=error/dataUji.length;
9      }
```

Kode Program 5.11 Implementasi Algoritma Perhitungan Nilai MSE

Berikut penjelasandari Kode Program 5.11:

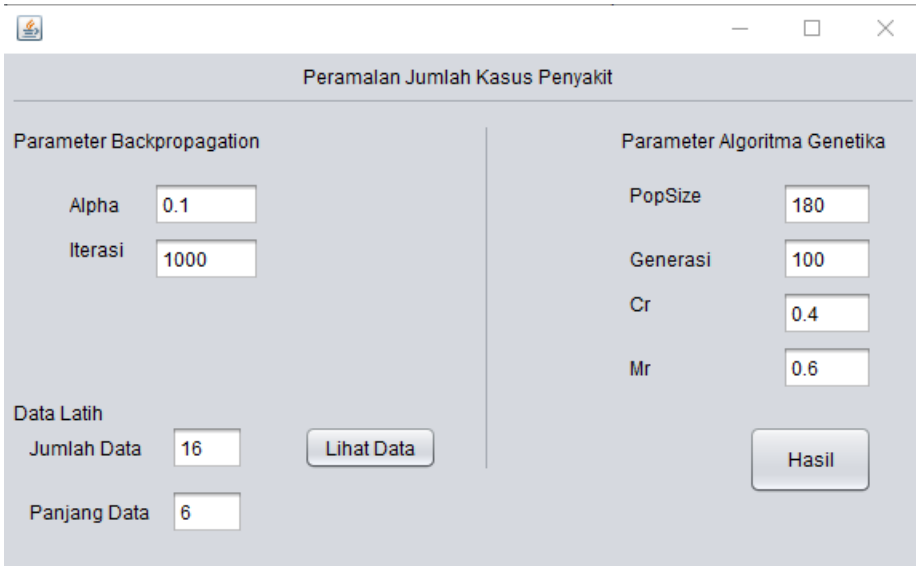
1. Baris 4 sampai 7 digunankan untuk mencari nilai error dari selisih kuadrat data target dikurangi data hasil peramalan.
2. Baris 8 digunakan untuk menghitung nilai MSE.

5.2 Implementasi Antarmuka

Antarmuka adalah bagian dari sistem yang digunakan sebagai perantara untuk berkomunikasi antara sistem dengan pengguna. Terdapat beberapa bagian pada antarmuka untuk sistem peramalan jumlah kasus penyakit ini yaitu antarmuka halaman utama, lihat data, dan hasil. Antarmuka diimplementasikan menggunakan JFrame pada Java.

5.2.1 Implementasi Antarmuka Halaman Utama

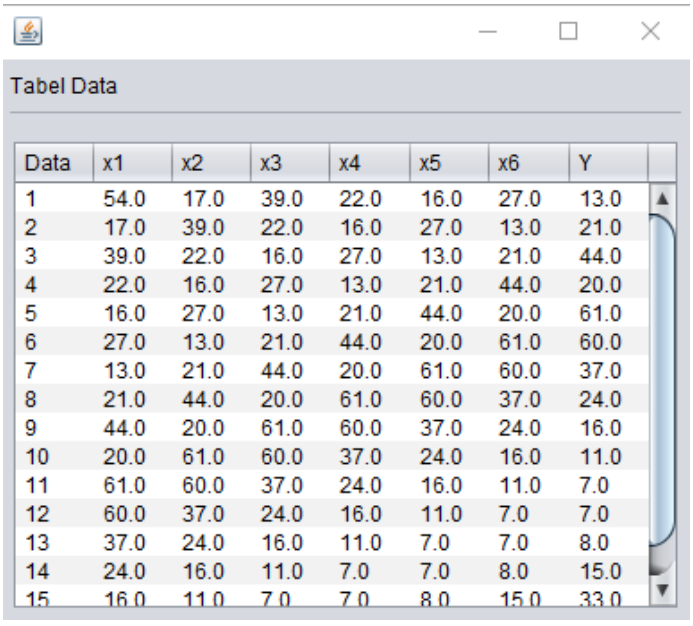
Pada halaman ini pengguna akan memasukkan parameter untuk proses algoritma *backpropagation* dan algoritma genetika serta data yang dibutuhkan untuk proses peramalan. Terdapat tombol untuk melihat data yang ingin dimasukkan serta tombol untuk menampilkan hasil peramalan. Implementasi antarmuka halaman utama ditunjukkan dengan Gambar 5.1.



Gambar 5.1 Antarmuka Halaman Utama

5.2.2 Implementasi Antarmuka Halaman Data

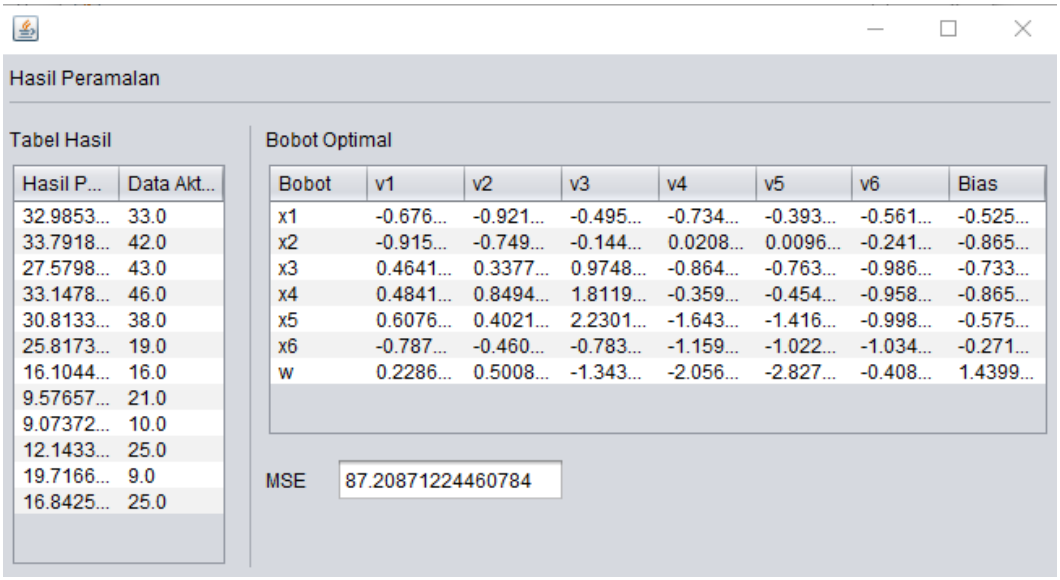
Pada halaman ini pengguna dapat melihat data yang akan digunakan untuk proses peramalan. Halaman ini berisi tabel data yang diambil dari *database* sesuai dengan masukkan yang diterima dari pengguna. Implementasi halaman data ditunjukkan dengan Gambar 5.2.



Gambar 5.2 Antarmuka Halaman Data

5.2.3 Implementasi Antarmuka Halaman Hasil

Pada halaman ini pengguna dapat melihat hasil dari proses peramalan yang terdiri dari tabel hasil peramalan, parameter optimal, dan nilai MSE. Implementasi halaman hasil ditunjukkan dengan Gambar 5.3.



Gambar 5.3 Antarmuka Halaman Hasil