BAB 5 IMPLEMENTASI

Pada bagian ini menjelaskan implementasi sistem yang akan dibuat berdasarkan analisis kebutuhan dan proses perancangan sistem. Implementasi yang dibahas dalam bab ini meliputi implementasi algoritme *Support Vector Machine* untuk klasifikasi penyakit gagal ginjal dan implementasi dari antarmuka sistem.

5.1 Spesifikasi Sistem

Spesifikasi sistem dibutuhkan untuk acuan dalam pembuatan sistem cerdas agar sistem dapat berjalan dengan baik serta sesuai dengan kebutuhan sistem.

5.1.1 Spesifikasi Perangkat Keras

Perangkat keras yang digunakan dalam implementasi algoritme SVM untuk klasifikasi penyakit gagal ginjal ditunjukkan pada Tabel 0.1.

Tabel 0.1 Spesifikasi perangkat keras

Komponen	Spesifikasi
Prosesor	Intel(R) Celeron(R) CPU 1007U @ 1.50GHz 1.50GHz
RAM	2 GB
Monitor	12 Inch

5.1.2 Spesifikasi Perangkat Lunak

Perangkat lunak yang digunakan dalam implementasi algoritme SVM untuk klasifikasi penyakit gagal ginjal ditunjukkan pada Tabel 0.2.

Tabel 0.2 Spesifikasi perangkat lunak

Komponen	Spesifikasi
Sistem Operasi	Windows 7
Bahasa Pemograman	Java
Tool Pemograman	Netbeans IDE 8.0.2

5.2 Batasan Imlementasi

Batasan implementasi untuk membangun sistem klasifikasi tingkat risiko pasien gagal ginjal, yaitu:

1. Sistem dibangun dengan menggunakan bahasa pemograman java dan GUI untuk tampilan *interface* nya.
2. Metode klasifikasi yang digunakan adalah metode *Support Vector Machine* dengan strategi *one-againts-all*.
3. Input dalam sistem berupa parameter SVM yang digunakan untuk proses *sequential training* dan proses *testing*.

4. Output berupa hasil klasifikasi pasien gagal ginjal yang mana berasal dari hasil perhitungan *training* dan *testing* SVM. Klasifikasi pasien gagal ginjal terdiri dari 3 tingkat risiko dan 5 parameter.

5.3 Implementasi Algoritme

Implementasi algoritme mengacu pada perancangan proses yang telah dijelaskan yaitu proses normalisasi data, perhitungan *sequential training* dan proses perhitungan *testing*.

5.3.1 Implementasi Algoritme Normalisasi

Proses pertama adalah perhitungan normalisasi data. Normalisasi data ini digunakan untuk menyetarakan sebaran data sehingga jarak dari data seimbang. Proses ini dilakukan pada perhitungan data latih dan data uji. Implementasi dari algoritme perhitungan normalisasi dapat dilihat pada Gambar 0.1.

```
1   public void MinMax() {
2   System.out.println("NORMALISASI LEVEL 1");
3           double nilai_max[] = new double[6];
4           double nilai_min[] = new double[6];
5           for (int i = 0; i < dtLevel1[0].length-1 ; i++) {
6               double max = dtLevel1[0][i];
7               double min = dtLevel1[0][i];
8               for (int j = 0; j < dtLevel1.length; j++) {
9                   if (max < dtLevel1[j][i]) {
10                      max = dtLevel1[j][i]; }
11                  if (min > dtLevel1[j][i]) {
12                      min = dtLevel1[j][i]; }
13              }
14              nilai_max[i  ] = max;
15              nilai_min[i  ] = min;
16          }
17          for (int i = 0; i < nilai_max.length; i++) {
18              System.out.println("max " + nilai_max[i]);
19              System.out.println("min " + nilai_min[i]);
20          }
21          for (int i = 0; i < dtLevel1[0].length -1 ; i++) {
22              for (int j = 0; j < dtLevel1.length; j++) {
23        normalisasi1[j][i] = (dtLevel1[j][i ] - nilai_min[i   ]) /
24  (nilai_max[i  ] - nilai_min[i ]);
25              }
26          }
27          for (int i = 0; i < normalisasi1.length; i++) {
28              for (int j = 0; j < normalisasi1[0].length; j++) {
29                  System.out.print(normalisasi1[i][j] + "|");
30              }
31              System.out.println("\t"); }}
```

Gambar 0.1 Implementasi algoritme perhitungan normalisasi

Berikut penjabaran dari source code pada Gambar 0.1.

1. Baris 1-4 inisialisasi variabel yang digunakan dalam perhitungan normalisasi.
2. Baris 5-20 proses pencarian nilai minimal dan maksimal pada setiap fitur data.
3. Baris 21-31 proses normalisasi dari masing-masing data.

5.3.2 Implementasi Algoritme Perhitungan *Kernel*

Pada implementasi algoritme perhitungan *kernel* melibatkan perhitungan *dot product* antara 2 data latih pada ruang vector yang berdimensi. *Kernel* yang digunakan adalah *kernel gausian* RBF. Source code algoritme *kernel* RBF dapat dilihat pada Gambar 0.2.

```
1  public void KernelRBF() {
2          System.out.println("KernelRBF LEVEL 1");
3          for (int i = 0; i < dtLevel1.length; i++) {
4              for (int j = 0; j < dtLevel1.length; j++) {
5                  kernelL1[i][j]        =        Math.exp(-1         *
6  ((Math.pow(dtLevel1[i][0]      -      dtLevel1[j][0],      2)      +
7  Math.pow(dtLevel1[i][1] - dtLevel1[j][1], 2)
8                          + Math.pow(dtLevel1[i][2] - dtLevel1[j][2],
9  2) + Math.pow(dtLevel1[i][3] - dtLevel1[j][3], 2)
10                         + Math.pow(dtLevel1[i][4] - dtLevel1[j][4],
11 2)) / (2 * Math.pow(sigma, 2))));
12             }
13         }
14         for (int i = 0; i < kernelL1.length; i++) {//
15             for (int j = 0; j < kernelL1[0].length; j++) {
16                 System.out.print(kernelL1[i][j] + "|");
17             }
18             System.out.println("");
19         }
20     }
```

Gambar 0.2 Implementasi algoritme perhitungan *kernel* RBF

Berikut penjabaran dari source code pada Gambar 0.2.

1. Baris 3-4 proses perulangan untuk data ke-*i* dan ke-*j* sampai dengan panjang data.
2. Baris 5-20 proses perhitungan *kernel* RBF.

5.3.3 Implementasi Algoritme Perhitungan *Sequential Training SVM*

Sequential Training SVM digunakan untuk proses pencarian nilai *hyperplane* yang optimal. Pada tahap ini akan dilakukannya iterasi dengan melakukan proses perhitungan dalam mencari nilai matriks *hessian*, *Ei*, nilai *delta alpha* dan mencari nilai *alpha* yang terbaru.

.3.3.1 Implementasi Algoritme Perhitungan Matrik *Hessian*

Implementasi dari algoritme perhitungan matrik *hessian* menggunakan variabel yang didapat dari hasil perhitungan fungsi *kernel* dan nilai *lambda*. Source code algoritme *kernel* RBF dapat dilihat pada Gambar 0.3.

```
1  public void MatrixHessian() {
2          System.out.println("");
3          for (int i = 0; i < dtLevel1.length; i++) {
4              for (int j = 0; j < dtLevel1.length; j++) {
5                  mhessianL1[i][j] = dtLevel1[i][5] * dtLevel1[j][5] *
6  (kernelL1[i][j] + Math.pow(lambda, 2));
7              }
8          }
9          System.out.println("Matrix Hessian LEVEL 1");
10         for (int i = 0; i < mhessianL1.length; i++) {
11             for (int j = 0; j < mhessianL1.length; j++) {
```

```
12                  System.out.print(mhessianL1[i][j] + "|");
13              }
14              System.out.println("");
15          }
16      }
```

Gambar 0.3 Implementasi algoritme perhitungan matrik *hessian*

Berikut penjabaran dari source code pada Gambar 0.3.

1. Baris 3-4 proses perulangan untuk data latih ke-*i* dan ke-*j* sesuai banyaknya data.
2. Baris ke 5-8 proses perhitungan matrik *hessian.*
3. Baris ke 9-16 proses pencetakan dari perhitungan matrik *hessian.*

.3.3.2 Implementasi Algoritme Perhitungan Nilai Ei

Implementasi dari algoritme perhitungan nilai Ei dilakukan dengan menjumlahkan hasil perkalian dari matrik *hessian* dengan *alpha* ke-*i.* Awal iterasi dari *sequential training* SVM dilakukan pada proses Ei. Source code perhitungan nilai Ei ditunjukkan pada Gambar 0.4.

```
1   public void Ei() {
2           System.out.println("");
3           double hasil_ei = 0;//untuk menyimpan hasil penjumlahan
4   ai*matrixhessian
5           for (int i = 0; i < dtLevel1.length; i++) {
6               for (int j = 0; j < dtLevel1.length; j++) {
7                   hasil_ei += ai1[j] * mhessianL1[i][j];
8               }
9               Ei1[i] = hasil_ei;
10              hasil_ei = 0;
11          }
12          System.out.println("nilai EI LEVEL 1");
13          for (int i = 0; i < Ei1.length; i++) {
14              System.out.println(Ei1[i]);
15          }
16      }
```

Gambar 0.4 Implementasi algoritme perhitungan nilai Ei

Berikut penjabaran dari source code pada Gambar 0.4.

1. Baris 5-10 proses perulangan untuk data latih ke-*i* dank e-*j* sampai dengan banyaknya data.
2. Baris 7-11 proses perhitungan nilai Ei.
3. Baris ke 12-16 proses pencetakan dari perhitungan nilai Ei.

.3.3.3 Implementasi Algoritme Perhitungan Nilai $\delta\alpha_i$

Implementasi algoritme perhitungan nilai $\delta\alpha_i$ dilakukan dengan terlebih dulu mencari nilai maksimum dari nilai *gamma,* jumlah nilai Ei, dan nilai *alpha*. Kemudian jika nilai maksimum sudah ditemukan, maka selanjutnya mencari nilai minimum dari nilai maksimu yang didapat, nilai c, dan nilai *alpha.* Source code perhitungan nilai $\delta\alpha_i$ ditunjukkan pada Gambar 0.5.

```
1   public void delta_i() {
2           System.out.println("");
3           for (int i = 0; i < dtLevel1.length; i++) {
4               delta_i1[i] = Math.min(Math.max(gamma * (1 - Ei1[i]), -
5   ai1[i]), C - ai1[i]);
6           }
7           System.out.println("Nilai Delta LEVEL 1");
8           for (int i = 0; i < delta_i1.length; i++) {
9               System.out.println(delta_i1[i]);
10          }
11      }
```

Gambar 0.5 Implementasi Algoritme Perhitungan Nilai $\delta\alpha_i$

Berikut penjabaran dari source code pada Gambar 0.5.

1. Baris 3 proses perulangan untuk data latih ke-*i* sesuai dengan banyaknya data.
2. Baris 4-6 proses perhitungan nilai $\delta\alpha_i$.
3. Baris ke 7-11 proses pencetakan dari perhitungan nilai $\delta\alpha_i$.

.3.3.4 Implementasi Algoritme Perhitungan Pembaharuan Nilai α_i

Implementasi algoritme α_i digunakan untuk menentukan nilai α_i yang baru. Dimana dengan menjumlahkan nilai $\delta\alpha_i$ dengan nilai α_i sebelumnya. Source code perhitungan nilai α_i dapat dilihat pada Gambar 0.6.

```
1   System.out.println("");
2           for (int i = 0; i < dtLevel1.length; i++) {
3               alfa_i1[i] = ai1[i] + delta_i1[i];
4           }
5           System.out.println("");
6           System.out.println("Nilai Alpha LEVEL 1");
7           for (int i = 0; i < alfa_i1.length; i++) {
8               System.out.println(alfa_i1[i]);
9           }
10          System.out.println("");
11          ai1 = alfa_i1;
12          return ai1;
13      }
```

Gambar 0.6 Implementasi algoritme perhitungan α_i

Berikut penjabaran dari source code pada Gambar 0.6.

1. Baris ke-2 proses perulangan data latih ke-*i* sesuai dengan panjang data.
2. Baris 3-4 proses perhitungan nilai α_i.
3. Baris ke 5-13 proses pencetakan dari perhitungan nilai α_i.

5.3.4 Impementasi Algoritme Perhitungan *Testing* SVM

.3.4.1 Implementasi Algoritme Perhitungan Nilai Kernel+, kernel- dan Bias

Data latih mempunyai bobot sejumlah 2, yaitu bobot *training* di *dot product* dengan nilai data latih yang mempunyai alpha teringgi di kelas positif wx^+ dan alpha tertinggi di kelas negatif wx^-. Dalam proses pencarian nilai wx^+ dan nilai wx^- menggunakan perhitungan kernel masing-masing dari kelas positif dan negatif yang memiliki alpha tertinggi dari masing-masing kelas tersebut. Source code perhitungan nilai α_i dapat dilihat pada Gambar 0.7.

```
1   public void Kernelnegatifpositif() {
2           double kelas_positif[][] = new double[80][2];
3           double kelas_negatif[][] = new double[160][2];
4           int positif = 0; //plus
5           int negatif = 0; //minus
6           for (int i = 1; i < dtLevel1.length; i++) {
7               if (dtLevel1[i][5] == 1.0) {
8                   kelas_positif[positif][0] = i;
9                   kelas_positif[positif][1] = alfa_i1[i];
10                  positif++;
11              } else if (dtLevel1[i][5] == -1) {
12                  kelas_negatif[negatif][0] = i;
13                  kelas_negatif[negatif][1] = alfa_i1[i];
14                  negatif++;
15              }
16          }
17          //mencari nilai maksimal
18          double maxPositif = kelas_positif[0][1];
19          double urutanPositif = kelas_positif[0][0];
20          for (int i = 0; i < kelas_positif.length; i++) {
21              if (maxPositif < kelas_positif[i][1]) {
22                  maxPositif = kelas_positif[i][1];
23                  urutanPositif = kelas_positif[i][0];
24              }
25          }
26          double maxNegatif = kelas_negatif[0][1];
27          double urutanNegatif = kelas_negatif[0][0];
28          for (int i = 0; i < kelas_negatif.length; i++) {
29              if (maxNegatif < kelas_negatif[i][1]) {
30                  maxNegatif = kelas_negatif[i][1];
31                  urutanNegatif = kelas_negatif[i][0];
32              }
33          }
34          //kernel X+
35          double x_positif = 0.0;
36          for (int i = 0; i < dtLevel1.length; i++) {
37              double hasil_positif = alfa_i1[i] * dtLevel1[i][5] *
38  kernelL1[i][(int) urutanPositif];
39              x_positif += hasil_positif;
40          }
41
42          //kernel X-
43          double x_negatif = 0.0;
44          for (int i = 0; i < dtLevel1.length; i++) {
45              double hasil_negatif = alfa_i1[i] * dtLevel1[i][5] *
46  kernelL1[i][(int) urutanNegatif];
47              x_negatif += hasil_negatif;
48          }
49          System.out.println("Nilai Kx+ = " + urutanPositif);
50          System.out.println("Nilai Kx- = " + urutanNegatif);
51          System.out.println("Kernel(xi,x+)   = " + x_positif);
52          System.out.println("Kernel(xi,x-)  = " + x_negatif);
53          //menghitung b
54          bias = -0.5 * (x_positif + x_negatif);
55          System.out.println("bias = " + bias);
56      }
```

Gambar 0.7 Implementasi algoritme perhitungan nilai *kernel+, kernel-* dan *bias*

Berikut penjabaran dari source code pada Gambar 0.7.

1. Baris 2-5 inisialisasi variabel untuk nilai kelas positif dan kelas negatif pada data.

2. Baris 6-33 proses pencarian nilai positif dan negatif.
3. Baris 34-48 proses perhitungan nilai kernel+ dan kernel -.
4. Baris 49-52 proses mencetak nilai hasil dari perhitungan kernel+ dan kernel -.
5. Baris 54 proses perhitungan nilai bias.

.3.4.2 Implementasi Algoritme Perhitungan Nilai *Testing* Kernel

Implementasi algoritme untuk perhitungan nilai *testing kernel* ini menggunakan rumus perhitungan *kernel* RBF melainkan data yang dihitung adalah data latih dan data uji. Source code perhitungan *testing kernel* dapat dilihat pada Gambar 0.8.

```
1  public void testingkernelRBF1() {
2          System.out.println("");
3          for (int i = 0; i < dt_testing.length; i++) {
4              for (int j = 0; j < dtLevel1.length;) {
5                  Hasil1 = Math.exp(-1 * ((Math.pow(dt_testing[i][0] -
6  dtLevel1[j][0], 2)
7                          +        Math.pow(dt_testing[i][1]        -
8  dtLevel1[j][1], 2)
9                          +        Math.pow(dt_testing[i][2]        -
10 dtLevel1[j][2], 2)
11                         +        Math.pow(dt_testing[i][3]        -
12 dtLevel1[j][3], 2)
13                         +        Math.pow(dt_testing[i][4]        -
14 dtLevel1[j][4], 2)
15                         ) / (2 * Math.pow(sigma, 2))));
16                 kernelRBFtesting1[i][j] = Hasil1;
17                 j++;
18             }
19         }
20
21         System.out.println("testing kernel level 1");
22         for (int j = 0; j < dt_testing.length; j++) {
23             for (int i = 0; i < dtLevel1.length; i++) {//
24                 System.out.println(kernelRBFtesting1[j][i] + "\t");
25             }
26             System.out.println("");
27         }
28         System.out.println("");
29     }
```

Gambar 0.8 Implementasi algoritme perhitungan nilai *testing kernel*

Berikut penjabaran dari source code pada Gambar 0.8.

1. Baris 3-4 proses perulangan data latih ke-*i* dan ke-*j* dengan data uji ke-*i* dan ke-*j* sesuai banyaknya data.
2. Baris 5-19 proses perhitungan *testing kernel.*
3. Baris 21-29 proses mencetak nilai perhitungan testing kernel

.3.4.3 Implementasi Algoritme Perhitungan Nilai aiyk

Dalam implementasi algoritme perhitunganjinilai aiyk dilakukan dengan menjumlahkan nilai *alpha,* kelas pada data latih, serta nilai *kernel* RBF dalam proses training. Kemudian menghitung nilai f(x) dengan menjumlahkan nilai aiyk dan bias. Hasil dari nilai f(x) dibulatkan menjadi nilai positif atau negatif. Dimana

jika hasilnya positif maka data uji masuk kelas 1 dan apabila hasilnya negatif maka data uji tersebut masuk kelas -1. Pada source program aiyk ini terdapat proses *one-againts-all* dimana data *training* dari kelas ke-*i* di *training* dengan diberi tanda positif dan data *training* yang bukan dari kelas *i* diberi tanda negatif. Source code perhitungan nilai aiyk dapat dilihat pada Gambar 0.9.

```
1  public void a_iyk() {
2          System.out.println("");
3          double jumlah = 0, jumlahAiyk = 0;
4          double fx1 = 0, fx2 = 0;
5          double jumlahaiyk3 = 0, jumlahaiyk4 = 0;
6
7          for (int i = 0; i < dt_testing.length; i++) {
8              System.out.println("Nilai aiyk");
9              for (int j = 0; j < dtLevel1.length; j++) {
10                 for (int k = 0; k < ai1.length; k++) {
11                     jumlah    =    ai1[k]    *    dtLevel1[j][5]    *
12 kernelRBFtesting1[i][j];
13                 }
14                 System.out.println(jumlah);
15                 jumlahaiyk3 += jumlah;
16             }
17             System.out.println("Jumlah");
18             System.out.println(jumlahaiyk3);
19             System.out.println("");
20             fx1 = (jumlahaiyk3 + bias);
21             System.out.println("Nilai fx");
22             System.out.println(fx1);
23             jumlah = 0;
24             jumlahaiyk3 = 0;
25
26             if (fx1 >= 0) {
27                 hUji[i][1] = fx1;
28                 hUji[i][0] = 1;
29             } else {
30                 System.out.println("level 2");
31                 System.out.println("nilai aiyk");
32                 for (int j = 0; j < dtLevel2.length; j++) {
33                     for (int k = 0; k < ai2.length; k++) {
34                         jumlahAiyk  =  ai2[k]  *  dtLevel2[j][5]  *
35 kernelRBFtesting2[i][j];
36                     }
37                     System.out.println(jumlahAiyk);
38                     jumlahaiyk4 += jumlahAiyk;
39                 }
40                 System.out.println("Jumlah");
41                 System.out.println(jumlahaiyk4);
42                 System.out.println("");
43                 fx2 = (jumlahaiyk4 + b);
44                 System.out.println("Nilai fx");
45                 System.out.println(fx2);
46                 jumlahAiyk = 0;
47                 jumlahaiyk4 = 0;
48                 if (fx2 >= 0) {
49                     hUji[i][1] = fx2;
50                     hUji[i][0] = 2;
51                 } else {
52                     hUji[i][1] = fx2;
53                     hUji[i][0] = 3;
54                 }
55             }
56
57         }
58
```

```
59          for (int i = 0; i < dt_testing.length; i++) {
60              System.out.println("Hasil Uji = " + hUji[i][0]);
61              if (hUji[i][0] == dt_testing[i][5]) {
62                  akurasi++;
63
64              }
65
66          }
67          akurasi = (akurasi * 100) / dt_testing.length;
68          System.out.println("Akurasi=" + akurasi);
69      }
```

Gambar 0.9 Implementasi algoritme perhitungan nilai aiyk

Penjelasan dari source code pada Gambar 0.9 adalah sebagai berikut:

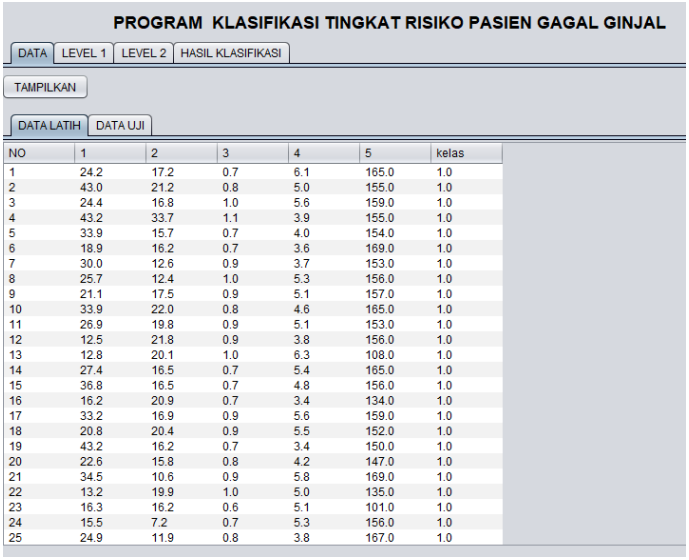
1. Baris 7-10 proses perulangan data uji ke-*i*, data latih level 1 ke-*j,* dan nilai alpha sesuai dengan panjang data.
2. Baris 11-19 proses perhitungan aiyk level 1.
3. Baris 20-24 proses perhitungan f(x) level 1 serta mencetak dan menyimpan nilai fx level 1.
4. Baris 26-42 suatu kondisi jika f(x) tidak bernilai 1 maka masuk ke proses perhitungan aiyk level 2.
5. Baris 43-47 proses perhitungan f(x) level 2 serta mencetak dan menyimpan nilai fx level 2.
6. Baris 48-55 suatu kondisi jika f(x) tidak bernilai 2 maka masuk ke hasil uji yang ke 3.
7. Baris 59-69 proses perulangan untuk pencarian hasil uji yang sama dengan *actual classnya* dan dilakukan proses perhitungan akurasi.

5.4 Implementasi Antar Muka

Terdapat 4 tampilan yang dirancang. Antarmuka pertama merupakan halaman utama sistem untuk menampilkan halaman data. Antarmuka kedua adalah halaman untuk melakukan inisialisasi dari setiap parameter yang dibutuhkan serta hasil perhitungan metode SVM pada level 1. Antarmuka ketiga merupakan halaman untuk menampilkan hasil perhitungan metode SVM level 2 dan halaman keempat merupakan hasil klasifikasi.

5.4.1 Implementasi Perancangan Halaman Data

Pada halaman data terdapat button yang berfungsi untuk menampilkan data uji dan data latih yang digunakan. Pada Gambar 0.10 dapat dilihat tampilan halaman data.

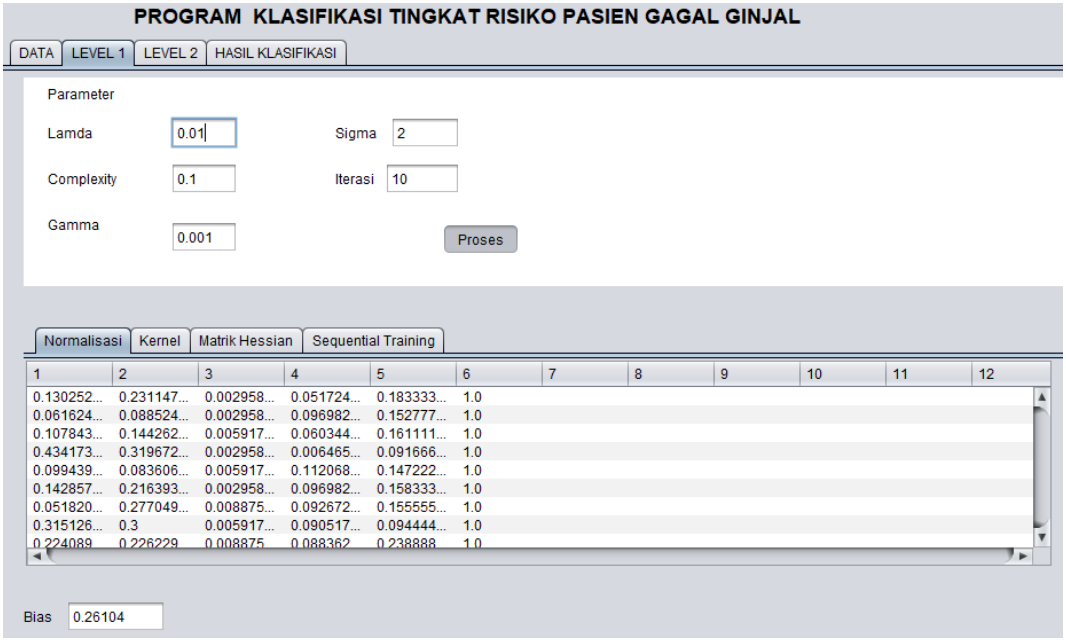


NO	1	2	3	4	5	kelas
1	24.2	17.2	0.7	6.1	165.0	1.0
2	43.0	21.2	0.8	5.0	155.0	1.0
3	24.4	16.8	1.0	5.6	159.0	1.0
4	43.2	33.7	1.1	3.9	155.0	1.0
5	33.9	15.7	0.7	4.0	154.0	1.0
6	18.9	16.2	0.7	3.6	169.0	1.0
7	30.0	12.6	0.9	3.7	153.0	1.0
8	25.7	12.4	1.0	5.3	156.0	1.0
9	21.1	17.5	0.9	5.1	157.0	1.0
10	33.9	22.0	0.8	4.6	165.0	1.0
11	26.9	19.8	0.9	5.1	153.0	1.0
12	12.5	21.8	0.9	3.8	156.0	1.0
13	12.8	20.1	1.0	6.3	108.0	1.0
14	27.4	16.5	0.7	5.4	165.0	1.0
15	36.8	16.5	0.7	4.8	156.0	1.0
16	16.2	20.9	0.7	3.4	134.0	1.0
17	33.2	16.9	0.9	5.6	159.0	1.0
18	20.8	20.4	0.9	5.5	152.0	1.0
19	43.2	16.2	0.7	3.4	150.0	1.0
20	22.6	15.8	0.8	4.2	147.0	1.0
21	34.5	10.6	0.9	5.8	169.0	1.0
22	13.2	19.9	1.0	5.0	135.0	1.0
23	16.3	16.2	0.6	5.1	101.0	1.0
24	15.5	7.2	0.7	5.3	156.0	1.0
25	24.9	11.9	0.8	3.8	167.0	1.0

Gambar 0.10 Implementasi perancangan halaman data

5.4.2 Implementasi Halaman SVM Level 1

Halaman SVM level 1 antarmuka sistem yang digunakan *user* untuk memasukkan nilai parameter pada proses perhitungan SVM. Serta pada halaman ini menampilkan hasil perhitungan *kernel*, matriks *Hessian*, nilai *ei*, *delta alpha*, nilai *alpha* dan *bias.* Pada Gambar 0.11 dapat dilihat tampilan halaman SVM level 1.

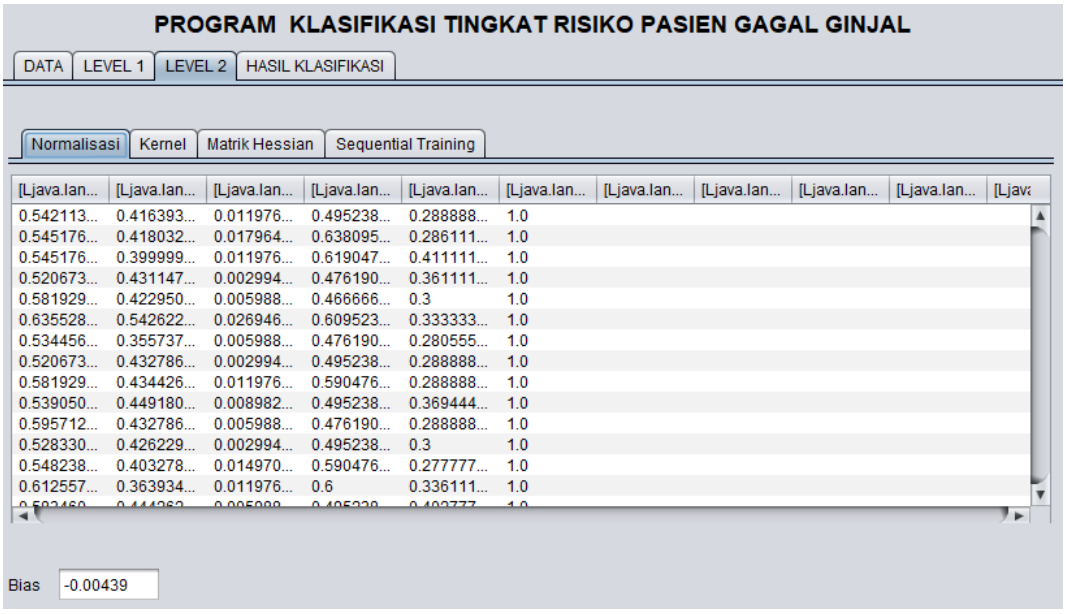


1	2	3	4	5	6	7	8	9	10	11	12
0.130252...	0.231147...	0.002958...	0.051724...	0.183333...	1.0						
0.061624...	0.088524...	0.002958...	0.096982...	0.152777...	1.0						
0.107843...	0.144262...	0.005917...	0.060344...	0.161111...	1.0						
0.434173...	0.319672...	0.002958...	0.006465...	0.091666...	1.0						
0.099439...	0.083606...	0.005917...	0.112068...	0.147222...	1.0						
0.142857...	0.216393...	0.002958...	0.096982...	0.158333...	1.0						
0.051820...	0.277049...	0.008875...	0.092672...	0.155555...	1.0						
0.315126...	0.3	0.005917...	0.090517...	0.094444...	1.0						
0.224089...	0.226229...	0.008875...	0.088362...	0.238888...	1.0						

Gambar 0.11 Implementasi perancangan halaman SVM level 1

5.4.3 Implementasi Halaman SVM Level 2

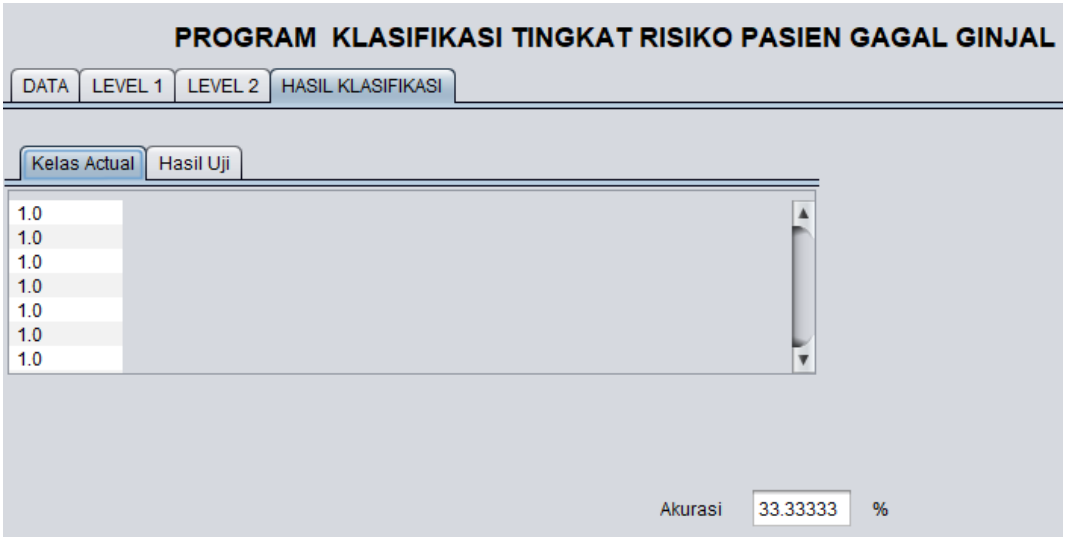
Halaman SVM level 2 ini memiliki tampilan yang sama dengan halaman SVM level 1. Tetapi pada halaman ini user tidak melakukan *input* parameter SVM. Pada Gambar 0.12 merupakan tampilan halaman SVM level 2.



Gambar 0.12 Implementasi perancangan halaman SVM level 2

5.4.4 Implementasi Halaman Hasil Klasifikasi

Pada halaman ini menampilkan hasil akurasi dari perhitungan metode SVM. User dapat melihat perbandingan dari kelas sebenarnya dan kelas hasil klasifikasi sistem. Pada Gambar 0.13 adalah tampilan halaman hasil klasifikasi.



Gambar 0.13 Implementasi halaman hasil klasifikasi