BAB 5 IMPLEMENTASI

Pada bab ini akan dijleskan mengenai implementasi sistem yang terdiri dari spesifikasi sistem dan implementasi algoritma. Pembahasan lebih lanjut akan dijelaskan proses implementasi algoritma untuk menyelesaikan permasalahan klasifikasi jenis penyakit Skizofrenia dengan menggunakan algoritma *Support Vector Machine* dan konsep *One Against All*.

5.1 Spesifikasi Sistem

Spesifikasi sistem yang dibutuhkan dalam proses pembuatan program ini dibagi menjadi dua. Pertama adalah perangkat keras yang merupakan spesifikasi dari laptop atau komputer yang digunakan. Kedua adalah perangkat lunak yang digunakan dalam proses pembuatan program ini.

5.1.1 Spesifikasi Perangkat Keras

Perangkat keras yang digunakan dalam mendukung pembuatan program klasifikasi ini terdiri dari satu buah laptop dengan spesifikasi yang tertera pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras

Komponen	Spesifikasi
Prosesor	AMD APU FX-7600P 2.7 – 3.4 GHz
Memory	4 GB
Hardisk	500 GB

5.1.2 Spesifikasi Perangkat Lunak

Perangkat lunak yang digunakan dalam mendukung pembuatan program klasifikasi ini terdiri dari Sistem Operasi, IDE (*Integrated Development Environment*) dan *Java Toolkit*. Tabel 5.2 menjelaskan perangkat lunak apa saja yang digunakan.

Tabel 5.2 Spesifikasi Perangkat Lunak

Komponen	Spesifikasi
Sistem Operasi	Windows 10
IDE	Java
Java Toolkit	Netbeans

5.2 Implementasi Algoritma

Proses implementasi algoritma pada sistem klasifikasi ini terbagi menjadi beberapa bagian. Bagian-bagian ini menjelaskan setiap proses perhitungan pada algoritma *Support Vector Machine*.

5.2.1 Implementasi *Sequential Training* SVM

Proses ini merupakan proses utama dalam perhitungan data latih. Dalam proses ini terdapat perhitungan mulai dari perhitungan *kernel* hingga mendapat nilai *alpha* dan mendapat nilai bias pada setiap level. Proses implementasi dari *Sequential Training* SVM ditampilkan pada Gambar 5.1.

```
1   public void main_svm(double[] parameter_svm, double[][]
2           data_tranning, double[] aktual) {
3           this.aktual_semt = new double[aktual.length];
4           this.data_semt = new
5           double[data_tranning.length][data_tranning[0].length];
6           for (int i = 0; i < data_semt.length; i++) {
7               aktual_semt[i] = aktual[i];
8               for (int j = 0; j < data_semt[i].length; j++) {
9                   data_semt[i][j] = data_tranning[i][j];
10              }
11          }
12          saveKernel = new
13          double[4][data_semt.length][data_semt.length];
14          saveError = new double[4][data_semt.length];
15          saveBetaAlpa = new double[4][data_semt.length];
16          saveAlpa = new double[4][data_semt.length];
17          saveHasil = new int[4][aktual.length];
18          int[][] hasil = new int[4][aktual.length];
19          lambda = parameter_svm[0];
20          gamma = parameter_svm[1];
21          c = parameter_svm[2];
22          epsilon = parameter_svm[3];
23          b = new double[4];
24          // Proses level 1, 2, 3, & 4.
25          inisialisasi();
26          for (int r = 0; r < 4; r++) {
27              this.hasil = new int[data_tranning.length];
28              if (r == 0) {
29                  // level 1
30                  this.aktual = aktual;
31                  this.data = data_tranning;
32                  for (int i = 0; i < this.aktual.length; i++) {
33                      if (this.aktual[i] == 1) {
34                          this.aktual[i] = 1;
35                      } else if (this.aktual[i] == 2) {
36                          this.aktual[i] = -1;
37                      } else if (this.aktual[i] == 3) {
38                          this.aktual[i] = -1;
39                      } else if (this.aktual[i] == 4) {
40                          this.aktual[i] = -1;
41                      } else if (this.aktual[i] == 5) {
42                          this.aktual[i] = -1;
43                      }
44                  }
45              } else if (r == 1) {
46                  // level 2
47                  int kk = 0;
48                  for (int i = 0; i < aktual_semt.length; i++) {
49                      if (aktual_semt[i] != 1) {
```

```
50                          kk++;
51                      }
52                  }
53                  this.data = new double[kk][data[0].length];
54                  this.aktual = new double[kk];
55                  int jj = 0;
56                  for (int i = 0; i < aktual_semt.length; i++) {
57                      if (aktual_semt[i] != 1) {
58                          this.data[jj] = data_semt[i];
59                          this.aktual[jj++] = aktual_semt[i];
60                      }
61                  }
62                  for (int i = 0; i < this.aktual.length; i++) {
63                      if (this.aktual[i] == 2) {
64                          this.aktual[i] = 1;
65                      } else if (this.aktual[i] == 3) {
66                          this.aktual[i] = -1;
67                      } else if (this.aktual[i] == 4) {
68                          this.aktual[i] = -1;
69                      } else if (this.aktual[i] == 5) {
70                          this.aktual[i] = -1;
71                      }
72                  }
73              } else if (r == 2) {
74                  // level 3
75                  int kk = 0;
76                  for (int i = 0; i < aktual_semt.length; i++) {
77                   if (aktual_semt[i] != 1 && aktual_semt[i] != 2) {
78                          kk++;
79                      }
80                  }
81                  this.data = new double[kk][data[0].length];
82                  this.aktual = new double[kk];
83                  int jj = 0;
84                  for (int i = 0; i < aktual_semt.length; i++) {
85                   if (aktual_semt[i] != 1 && aktual_semt[i] != 2) {
86                          this.data[jj] = data_semt[i];
87                          this.aktual[jj++] = aktual_semt[i];
88                      }
89                  }
90                  for (int i = 0; i < this.aktual.length; i++) {
91                      if (this.aktual[i] == 3) {
92                          this.aktual[i] = 1;
93                      } else if (this.aktual[i] == 4) {
94                          this.aktual[i] = -1;
95                      } else if (this.aktual[i] == 5) {
96                          this.aktual[i] = -1;
97                      }
98                  }
99              } else if (r == 3) {
100                 // level 4
101                 int kk = 0;
102                 for (int i = 0; i < aktual_semt.length; i++) {
103                     if (aktual_semt[i] != 1 && aktual_semt[i] != 2
104                 && aktual_semt[i] != 3) {
105                         kk++;
106                     }
107                 }
108                 this.data = new double[kk][data[0].length];
109                 this.aktual = new double[kk];
110                 int jj = 0;
111                 for (int i = 0; i < aktual_semt.length; i++) {
112                     if (aktual_semt[i] != 1 && aktual_semt[i] != 2
113                  && aktual_semt[i] != 3) {
114                         this.data[jj] = data_semt[i];
```

```
                            this.aktual[jj++] = aktual_semt[i];
                        }
                    }
                    for (int i = 0; i < this.aktual.length; i++) {
                        if (this.aktual[i] == 4) {
                            this.aktual[i] = 1;
                        } else if (this.aktual[i] == 5) {
                            this.aktual[i] = -1;
                        }
                    }
                }
                panjang_alpa[r] = data.length;
                double[][] D;
                double[] error = null;
                double[] beta_alpa = null;
                kernel = hitung_kernel(kondisi);
                saveKernel[r] = kernel;
                D = hitung_d(kernel);
                for (int i = 0; i < learning_rate; i++) {
                    error = hitung_error(D, r);
                    beta_alpa = hitung_beta_alpa(error, r);
                    update_alpa(beta_alpa, r);
                    boolean kondisi = true;
                    for (int j = 0; j < beta_alpa.length; j++) {
                        if (beta_alpa[j] > 0) {
                            beta_alpa[j] = beta_alpa[j] * 1;
                            if (beta_alpa[j] > epsilon) {
                            kondisi = false;
                            break;
                            }
                        else {
                            beta_alpa[j] = beta_alpa[j] * -1;
                            if (beta_alpa[j] > epsilon) {
                            kondisi = false;
                            break;
                            }
                        }
                    }
                }
                if (kondisi == false) {
                        break;
                    }
                }
                saveError[r] = error;
                saveBetaAlpa[r] = beta_alpa;
                saveAlpa[r] = alpa[r];
                getError(kernel, r);
                System.out.println("Level " + (r + 1));
                System.out.println("Kernel");
                for (int i = 0; i < kernel.length; i++) {
                    for (int j = 0; j < kernel[0].length; j++) {
                        System.out.print(" " + kernel[i][j]);
                    }
                    System.out.println();
                }
                System.out.println();
                System.out.println("Matrik Hessian");
                for (int i = 0; i < D.length; i++) {
                    for (int j = 0; j < D[0].length; j++) {
                        System.out.print(" " + D[i][j]);
                    }
                    System.out.println();
                }
                System.out.println();
                System.out.println("Error");
```

```
180             for (int i = 0; i < error.length; i++) {
181                 System.out.println(error[i]);
182             }
183             System.out.println();
184             System.out.println("Delta Alpa");
185             for (int i = 0; i < beta_alpa.length; i++) {
186                 System.out.println(beta_alpa[i]);
187             }
188             System.out.println();
189             System.out.println("Alpa");
190             for (int i = 0; i < alpa.length; i++) {
191                 for (int j = 0; j < alpa[0].length; j++) {
192                     System.out.print(" " + alpa[i][j]);
193                 }
194                 System.out.println();
195             }
196             System.out.println("B = " + b[r]);
197         }
198     }
```

Kode Program 5.1 *Sequential Training* SVM

Penjelasan Gambar 5.1:

1. Pada baris 28 hingga 122 merupakan algoritma untuk memberikan nilai variabel y untuk setiap kelas pada setiap level.
2. Pada baris 126 hingga 143 merupakan algoritma perhitungan *Sequential Training* SVM.
3. Pada baris 149 hingga 184 menampilkan hasil perhitungan dari setiap fungsi yang digunakan pada *Sequential Training* SVM.

5.2.2 Implementasi Perhitungan *Kernel*

Proses ini merupakan proses perhitungan *kernel* yang digunakan pada metode SVM bersifat nonlinier. Pada algoritma ini jenis *kernel* terbagi menjadi 3 jenis yaitu *kernel Polynomial, kernel RBF, kernel linier*. Proses perhitungan *kernel* ditunjukan pada Gambar 5.2.

```
1   public double[][] hitung_kernel(int kondisi) {
2           double[][] kernel = new double[data.length][data.length];
3           double hasil = 0;
4           if (kondisi == 2) { //polynomial
5               int jj = 0;
6               while (jj < data.length) {
7                   for (int k = 0; k < data.length; k++) {
8                       hasil = 0;
9                       for (int j = 0; j < data[0].length; j++) {
10                          hasil += (data[jj][j] * data[k][j]);
11                      }
12                      kernel[k][jj] += Math.pow(hasil, 2);
13                  }
14                  jj++;
15              }
16          }
17          else if (kondisi == 1) { // Linier
18              int jj = 0;
19              while (jj < data.length) {
20                  for (int k = 0; k < data.length; k++) {
21                      for (int j = 0; j < data[0].length; j++) {
22                          kernel[k][jj] += (data[jj][j] *
```

```
23          data[k][j]);
24                      }
25                  }
26                  jj++;
27              }
28           }
29          else if (kondisi == 3) { // rbf
30              int jj = 0;
31              double sigma = 0.1;
32              while (jj < data.length) {
33                  for (int k = 0; k < data.length; k++) {
34                      for (int j = 0; j < data[0].length; j++) {
35                          kernel[k][jj] +=
36          (Math.pow(Math.abs(Math.abs(data[jj][j] - data[k][j])), 2)
37           / (2 * Math.pow(sigma, 2)));
38                      }
39                      kernel[k][jj] = Math.exp(-1 * kernel[k][jj]);
40                  }
41                  jj++;
42              }
43          }
44          return kernel;
45      }
46
```

Kode Program 5.2 Perhitungan *Kernel*

Penjelasan Gambar 5.2:

1. Pada baris 6 hingga 16 merupakan perhitungan *kernel Polynomial.*
2. Pada baris 18 hingga 27 merupakan perhitungan *kernel Linier*.
3. Pada baris 29 hingga 40 merupakan perhitungan *kernel* RBF.

5.2.3 Implementasi Perhitungan Matriks Hessian

Perhitungan Matriks Hessian ini didapatkan setelah melakukan perhitungan *Kernel*. Proses perhitungan Matriks Hessian ditunjukan pada Gambar 5.3.

```
1       public double[][] hitung_d(double[][] kernel) {
2           double[][] D = new double[data.length][data.length];
3           for (int i = 0; i < D.length; i++) {
4               for (int j = 0; j < D.length; j++) {
5                   D[i][j] = (aktual[i] * aktual[j]) *
6                  ((kernel[i][j]) + Math.pow(lambda, 2));
7               }
8           }
9           double maxD = 0;
10          for (int i = 0; i < D.length; i++) {
11              for (int j = 0; j < D[0].length; j++) {
12                  if (D[i][j] > maxD) {
13                      maxD = D[i][j];
14                  }
15              }
16          }
17          l_rate = gamma / maxD;
18          return D;
19      }
```

Kode Program 5.3 Perhitungan Matriks Hessian

Penjelasan Gambar 5.3:

1. Pada baris 4 hingga 7 merupakan perulangan untuk perhitungan matriks hessian sebanyak panjang data.
2. Pada baris 10 hingga 14 merupakan perhitungan untuk mencari nilai matriks hessian terbesar untuk digunakan pada perhitungan variabel γ.
3. Pada baris 18 merupakan proses perhitungan nilai variabel γ.

5.2.4 Implementasi Perhitungan E_i

Proses ini merupakan proses perhitungan nilai E_i. Dalam proses ini akan dilakukan iterasi hingga mencapai kondisi yang telah ditentukan. Proses perhitungan nilai E_i ditunjukan pada Gambar 5.4.

```
1       public double[] hitung_error(double[][] D, int r) {
2           double[] error = new double[data.length];
3           for (int i = 0; i < D.length; i++) {
4               for (int j = 0; j < D[0].length; j++) {
5                   error[i] += D[i][j] * alpa[r][j];
6               }
7           }
8           return error;
9       }
```

Kode Program 5.4 Perhitungan E_i

Penjelasan Gambar 5.4:

1. Pada baris 3 hingga 5 merupakan proses perulangan pada perhitungan nilai E_i sebanyak panjang data.

5.2.5 Implementasi Perhitungan $\delta\alpha_i$

Proses ini merupakan proses perhitungan nilai $\delta\alpha_i$. Dalam proses ini akan dilakukan iterasi hingga mencapai kondisi yang telah ditentukan. Proses perhitungan nilai $\delta\alpha_i$ ditunjukan pada Gambar 5.5.

```
1       public double[] hitung_beta_alpa(double[] error, int r) {
2           double[] beta_alpa = new double[data.length];
3           for (int i = 0; i < error.length; i++) {
4               beta_alpa[i] = Math.min(Math.max(gamma * (1 –
5               error[i]), -1 * alpa[r][i]), c - alpa[r][i]);
6           }
7           return beta_alpa;
8       }
```

Kode Program 5.5 Perhitungan $\delta\alpha_i$

Penjelasan Gambar 5.5:

1. Pada baris 3 hingga 5 merupakan proses perulangan pada perhitungan nilai $\delta\alpha_i$ sebanyak panjang data.

5.2.6 Implementasi Perhitungan α_i

Proses ini merupakan proses perhitungan nilai α_i yang baru, dimana nilai ini akan disimpan dan digunakan pada proses pengujian nanti. Dalam

proses ini juga akan dilakukan iterasi hingga mencapai kondisi yang telah ditentukan. Proses perhitungan nilai α_i ditunjukan pada Gambar 5.6.

```
1       public void update_alpa(double[] beta_alpa, int r) {
2           for (int i = 0; i < panjang_alpa[r]; i++) {
3               alpa[r][i] += beta_alpa[i];
4           }
5       }
```

Kode Program 5.6 Perhitungan α_i

Penjelasan Gambar 5.6:

1. Pada baris 2 hingga 3 merupakan proses perulangan pada perhitungan untuk *update* nilai $\boldsymbol{\alpha_i}$ sebanyak panjang data.

5.2.7 Implementasi Perhitungan $\boldsymbol{K(x_i.x^+)}$

Proses ini merupakan proses untuk mencari nilai *kernel* yang akan digunakan untuk masuk dalam proses perhitungan nilai bobot pada kelas positif. Pencarian nilai *kernel* ini berdasarkan nilai *alpha* terbesar pada kelas positif atau yang bernilai 1. Proses perhitungan ini ditunjukan pada Gambar 5.7.

```
1       public double[] hitung_w_plus(double[][] kernel, double[]
2           alpa) {
3           double[] w_plus = new double[data.length];
4           double max = 0;
5           int indek = 0;
6           for (int j = 0; j < aktual.length; j++) {
7               if (aktual[j] == 1) {
8                   if (alpa[j] > max) {
9                       max = alpa[j];
10                      indek = j;
11                  }
12              }
13          }
14          for (int i = 0; i < kernel[indek].length; i++) {
15              w_plus[i] = kernel[indek][i];
16          }
17          return w_plus;
18      }
```

Kode Program 5.7 Perhitungan $\boldsymbol{K(x_i.x^+)}$

Penjelasan Gambar 5.7:

1. Pada baris 6 hingga 10 merupakan perhitungan untuk mencari nilai alpha terbesar dari kelas positif.
2. Pada baris 14 hingga 15 merupakan kode untuk mengambil nilai *kernel* sesuai dengan indek pada alpha terbesar.

5.2.8 Implementasi Perhitungan $\boldsymbol{K(x_i.x^-)}$

Proses ini merupakan proses untuk mencari nilai *kernel* yang akan digunakan untuk masuk dalam proses perhitungan nilai bobot pada kelas negatif. Pencarian nilai *kernel* ini berdasarkan nilai *alpha* terbesar pada kelas

negatif atau yang bernilai -1. Proses perhitungan ini ditunjukan pada Gambar 5.8.

```
1           public double[] hitung_w_negatif(double[][] kernel,
2           double[] alpa) {
3           double[] w_negatif = new double[data.length];
4           double max = 0;
5           int indek = 0;
6           for (int j = 0; j < aktual.length; j++) {
7               if (aktual[j] == -1) {
8                   if (alpa[j] > max) {
9                       max = alpa[j];
10                      indek = j;
11                  }
12              }
13          }
14          for (int i = 0; i < kernel[indek].length; i++) {
15              w_negatif[i] = kernel[indek][i];
16          }
17          return w_negatif;
18      }
```

Kode Program 5.8 Perhitungan $K(x_i.x^-)$

Penjelasan Gambar 5.8:

1. Pada baris 6 hingga 10 merupakan perhitungan untuk mencari nilai alpha terbesar dari kelas negatif.
2. Pada baris 14 hingga 15 merupakan kode untuk mengambil nilai *kernel* sesuai dengan indek pada alpha terbesar.

5.2.9 Implementasi Perhitungan b (Bias)

Proses ini merupakan proses perhitungan nilai bobot pada kelas positif dan kelas negatif, dimana perhitungan tersebut akan menghasilkan nilai bias pada setiap level. Nilai bias ini akan disimpan dan digunakan pada proses pengujian. Proses perhitungan bias ditunjukan pada Gambar 5.9.

```
1           public double hitung_b(double[] k_plus, double[] k_negatif,
2            int r) {
3            double b = 0;
4            double plus = 0;
5            double neg = 0;
6            for (int i = 0; i < panjang_alpa[r]; i++) {
7                plus += (alpa[r][i] * aktual[i] * k_plus[i]);
8                neg += (alpa[r][i] * aktual[i] * k_negatif[i]);
9            }
10           b = -0.5 * (plus + neg);
11           return b;
12      }
```

Kode Program 5.9 Perhitungan Nilai Bias

Penjelasan Gambar 5.9:

1. Pada baris 6 hingga 8 merupakan perhitungan untuk mendapatkan nilai bobot pada kelas positif dan nilai bobot kelas negatif .
2. Pada baris 10 merupakan perhitungan untuk mencari nilai bias.

5.2.10 Implementasi Perhitungan *Kernel* Uji

Proses ini merupakan proses perhitungan *kernel* yang digunakan pada tahap pengujian. Pada dasarnya cara perhitungan *kernel* ini sama dengan perhitungan *kernel* pada proses *training*, perbedaannya terdapat pada proses perkalian data uji harus dikalikan dengan data training sebelumnya. Proses perhitungan *kernel* ini ditunjukan pada Gambar 5.10.

```
1     public double[] hitung_kernel_testing(int kondisi, double[][]
2     data_testing, int ii) {
3          double[] kernel = new double[data.length];
4          double hasil;
5          if (kondisi == 2) { //polynomial
6              for (int k = 0; k < data.length; k++) {
7                  hasil = 0;
8                  for (int j = 0; j < data[0].length; j++) {
9                      hasil += (data[k][j] * data_testing[ii][j]);
10                 }
11                 kernel[k] += Math.pow(hasil, 2);
12             }
13         } else if (kondisi == 1) { // Linier
14             for (int k = 0; k < data.length; k++) {
15                 for (int j = 0; j < data[0].length; j++) {
16                     kernel[k] += (data[k][j] *
17                     data_testing[ii][j]);
18                 }
19             }
20         } else if (kondisi == 3) { // rbf
21             double sigma = 0.1;
22             for (int k = 0; k < data.length; k++) {
23                 for (int j = 0; j < data[0].length; j++) {
24                     kernel[k] +=
25                 (Math.pow(Math.abs(Math.abs(data[k][j] -
26                 data_testing[ii][j])), 2) / (2 * Math.pow(sigma,
27                 2)));
28                 }
29                 kernel[k] = Math.exp(-1 * kernel[k]);
30             }
31         }
32         return kernel;
33     }
```

Kode Program 5.10 Perhitungan *Kernel* Uji

Penjelasan Gambar 5.10:

1. Pada baris 5 hingga 11 merupakan perhitungan *kernel polynomial*.
2. Pada baris 13 hingga 17 merupakan perhitungan *kernel linier*.
3. Pada baris 20 hingga 27 merupakan perhitungan *kernel* RBF.

5.2.11 Implementasi Perhitungan $f(x)$

Proses ini merupakan tahap pengujian dimana jika hasil akhir $f(x)$ bernilai positif maka data uji yang sedang diproses merupakan kelas 1, sebaliknya jika hasil akhir bernilai -1 maka data merupakan kelas -1. Proses perhitungan $f(x)$ ini ditunjukan pada Gambar 5.11.

```
1   public double[] getResult(double[][] data_testing, int r, double[]
2   results) {
3           double[] hasil = new double[data_testing.length];
4           for (int i = 0; i < hasil.length; i++) {
5               double result = 0;
6               if (r == 0) {
7                   double[] kernel = hitung_kernel_testing(kondisi,
8                   data_testing, i);
9                   for (int j = 0; j < data.length ; j++) {
10                      result += (alpa[r][j] * aktual[j] *
11                      kernel[j]);
12                  }
13                  result += b[r];
14              } else {
15                  if (results[i] == 1 || results[i] == 0) {
16                      result = 0;
17                  } else {
18                      double[] kernel =
19                  hitung_kernel_testing(kondisi, data_testing, i);
20                      for (int j = 0; j < data.length; j++) {
21                          result += (alpa[r][j] * aktual[j] *
22                         kernel[j]);
23                      }
24                      result += b[r];
25                  }
26              }
27              if (result < 0) {
28                  hasil[i] = -1;
29              } else if (result > 0) {
30                  hasil[i] = 1;
31              } else {
32                  hasil[i] = 0;
33              }
34          }
35          return hasil;
36      }
```

Kode Program 5.11 Perhitungan Nilai $f(x)$

Penjelasan Gambar 5.11:

1. Pada baris 6 hingga 13 merupakan perhitungan $f(x)$ pada level 1.
2. Pada baris 14 hingga 24 merupakan perhitungan $f(x)$ pada level selain level 1, dimana jika nilai $f(x)$ pada level sebelumya sudah bernilai 1 atau 0 maka akan melakukan perhitungan $f(x)$ ulang.
3. Pada baris 27 hingga 32 merupakan fungsi klasifikasi dimana jika nilai lebih dari 0 maka akan bernilai 1, sebaliknya jika kurang dari 0 maka akan bernilai -1.

5.2.12 Implementasi Perhitungan *One Against All*

Proses ini merupakan implementasi dari konsep *One Against All*. Konsep ini berfungsi pada SVM yang bersifat multikelas. Pada permasalahan ini digunakan perhitungan sebanyak 4 level. Proses perhitungan ini dilihat dari hasil akhir pada perhitungan $f(x)$ yang jika bernilai 1 maka data itu masuk pada kelas pada level tersebut, jika bernilai -1 maka data akan diteruskan pada proses perhitungan

pada level selanjutnya. Proses perhitungan konsep *One Against All* ini ditunjukan pada Gambar 5.12.

```
1     public double[] main_testing(double[][] data_tranning, double[]
2     aktual_tranning, double[][] data_testing) {
3          this.data_semt = data_tranning;
4          this.aktual_semt = aktual_tranning;
5          double[][] results = new double[4][data_testing.length];
6          double[] result = new double[data_testing.length];
7          for (int r = 0; r < 4; r++) {
8              if (r == 0) {
9                  this.aktual = new double[aktual_semt.length];
10                 // level 1
11                 for (int i = 0; i < this.aktual.length; i++) {
12                     if (this.aktual_semt[i] == 1) {
13                         this.aktual[i] = 1;
14                     } else if (this.aktual_semt[i] == 2) {
15                         this.aktual[i] = -1;
16                     } else if (this.aktual_semt[i] == 3) {
17                         this.aktual[i] = -1;
18                     } else if (this.aktual_semt[i] == 4) {
19                         this.aktual[i] = -1;
20                     } else if (this.aktual_semt[i] == 5) {
21                         this.aktual[i] = -1;
22                     }
23                 }
24                 data = new
25                 double[data_semt.length][data_semt[0].length];
26                 for (int i = 0; i < data_semt.length; i++) {
27                     for (int j = 0; j < data_semt[i].length; j++)
28                 {
29                         data[i][j] = data_semt[i][j];
30                     }
31                 }
32             } else if (r == 1) {
33                 // level 2
34                 int kk = 0;
35                 for (int i = 0; i < aktual_semt.length; i++) {
36                     if (aktual_semt[i] != 1) {
37                         kk++;
38                     }
39                 }
40                 this.data = new double[kk][data[0].length];
41                 this.aktual = new double[kk];
42                 int jj = 0;
43                 for (int i = 0; i < aktual_semt.length; i++) {
44                     if (aktual_semt[i] != 1) {
45                         this.data[jj] = data_semt[i];
46                         this.aktual[jj++] = aktual_semt[i];
47                     }
48                 }
49                 for (int i = 0; i < this.aktual.length; i++) {
50                     if (this.aktual[i] == 2) {
51                         this.aktual[i] = 1;
52                     } else if (this.aktual[i] == 3) {
53                         this.aktual[i] = -1;
54                     } else if (this.aktual[i] == 4) {
55                         this.aktual[i] = -1;
56                     } else if (this.aktual[i] == 5) {
57                         this.aktual[i] = -1;
58                     }
59                 }
60             } else if (r == 2) {
61                 // level 3
62                 int kk = 0;
```

```
63                  for (int i = 0; i < aktual_semt.length; i++) {
64                      if (aktual_semt[i] != 1 && aktual_semt[i] !=
65                      2) {
66                          kk++;
67                      }
68                  }
69                  this.data = new double[kk][data[0].length];
70                  this.aktual = new double[kk];
71                  int jj = 0;
72                  for (int i = 0; i < aktual_semt.length; i++) {
73                      if (aktual_semt[i] != 1 && aktual_semt[i] !=
74                  2) {
75                          this.data[jj] = data_semt[i];
76                          this.aktual[jj++] = aktual_semt[i];
77                      }
78                  }
79                  for (int i = 0; i < this.aktual.length; i++) {
80                      if (this.aktual[i] == 3) {
81                          this.aktual[i] = 1;
82                      } else if (this.aktual[i] == 4) {
83                          this.aktual[i] = -1;
84                      } else if (this.aktual[i] == 5) {
85                          this.aktual[i] = -1;
86                      }
87                  }
88              } else if (r == 3) {
89                  // level 4
90                  int kk = 0;
91                  for (int i = 0; i < aktual_semt.length; i++) {
92                      if (aktual_semt[i] != 1 && aktual_semt[i] != 2
93                  && aktual_semt[i] != 3) {
94                          kk++;
95                      }
96                  }
97                  this.data = new double[kk][data[0].length];
98                  this.aktual = new double[kk];
99                  int jj = 0;
100                 for (int i = 0; i < aktual_semt.length; i++) {
101                     if (aktual_semt[i] != 1 && aktual_semt[i] != 2
102                 && aktual_semt[i] != 3) {
103                         this.data[jj] = data_semt[i];
104                         this.aktual[jj++] = aktual_semt[i];
105                     }
106                 }
107                 for (int i = 0; i < this.aktual.length; i++) {
108                     if (this.aktual[i] == 4) {
109                         this.aktual[i] = 1;
110                     } else if (this.aktual[i] == 5) {
111                         this.aktual[i] = -1;
112                     }
113                 }
114             }
115             if (r == 0) {
116                 results[r] = getResult(data_testing, r,
117                 results[r]);
118             } else {
119                 results[r] = getResult(data_testing, r, results[r
120                 - 1]);
121             }
122         }
123         System.out.println("Hasil Testing");
124         for (int j = 0; j < results[0].length; j++) {
125             for (int i = 0; i < results.length; i++) {
126                 System.out.print("  " + results[i][j]);
127             }
```

```
128             System.out.println();
129         }
130         System.out.println();
131         for (int i = 0; i < results.length; i++) {
132             for (int j = 0; j < results[i].length; j++) {
133                 if (results[i][j] < 0) {
134                     results[i][j] = -1;
135                 } else {
136                     results[i][j] = 1;
137                 }
138             }
139         }
140         for (int i = 0; i < results[0].length; i++) {
141             // lihat f(1)
142             if (results[0][i] == -1) {
143                 // lihat f(2)
144                 if (results[1][i] == -1) {
145                     // lihat f(3)
146                     if (results[2][i] == -1) {
147                         // lihat f(4)
148                         if (results[3][i] == -1) {
149                             result[i] = 5;
150                         } else {
151                             result[i] = 4;
152                         }
153                     } else {
154                         result[i] = 3;
155                     }
156                 } else {
157                     result[i] = 2;
158                 }
159             } else {
160                 result[i] = 1;
161             }
162         }
163         return result;
164     }
```

Kode Program 5.12 Perhitungan *One Against All*

Penjelasan Gambar 5.12:

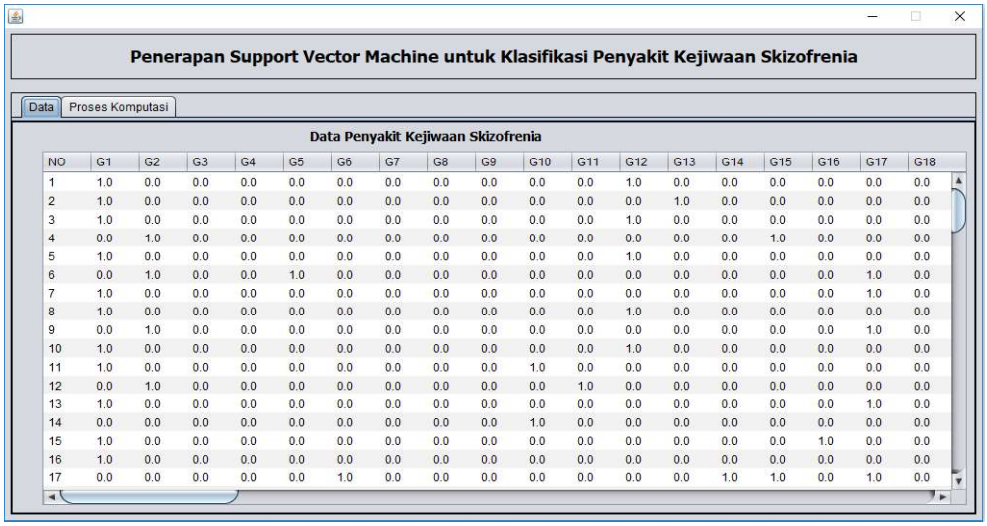
1. Pada baris 8 hingga 110 merupakan penentuan nilai y untuk setiap data pada setiap level.
2. Pada baris 114 hingga 119 memanggil fungsi perhitungan $f(x)$.
3. Pada baris 139 hingga 159 merupakan algoritma konsep *One Against All*.

5.3 Implementasi Antarmuka (*Interface)*

Antarmuka (*Interface*) adalah suatu fungsi yang digunakan untuk menghubungkan antara sistem dengan *user* dalam berinteraksi. Pada program klasifikasi jenis penyakit Skizofrenia dengan menggunakan metode *Support Vector Machine* ini dibagi menjadi 6 halaman. Halaman pertama menampilkan kesulurahan data. Pada halaman kedua dibagi menjadi 5 halaman yaitu halaman perhitungan level1, level 2, level 3, level 4, dan hasil pengujian.

5.3.1 Implementasi Antarmuka Halaman Data

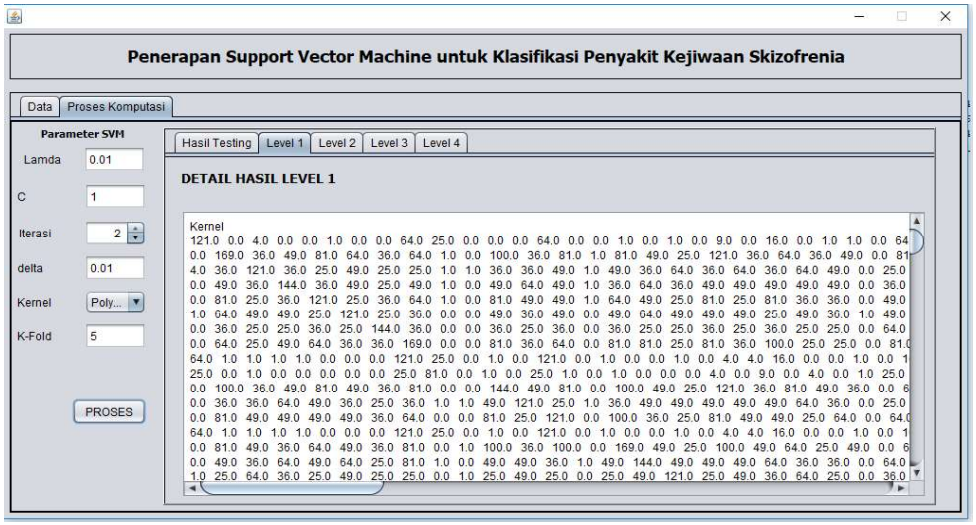
Pada halaman ini akan ditampikan semua data yang akan digunakan untuk perhitungan *Sequential Training SVM* dan proses *One Against All*. Data ini terdiri dari 111 data yang memiliki format .xls. Implementasi untuk antarmuka pada halaman yang pertama ini ditampilkan pada Gambar 5.13.



Gambar 5.1 Antarmuka Halaman Data

5.3.2 Implementasi Antarmuka Halaman Level 1

Pada halaman ini akan ditampikan semua hasil dari perhitungan *training* pada level 1. Perhitungan ini terdiri dari hasil *kernel* hingga nilai bias pada level 1. Implementasi untuk antarmuka pada halaman level 1 ini ditampilkan pada Gambar 5.14.

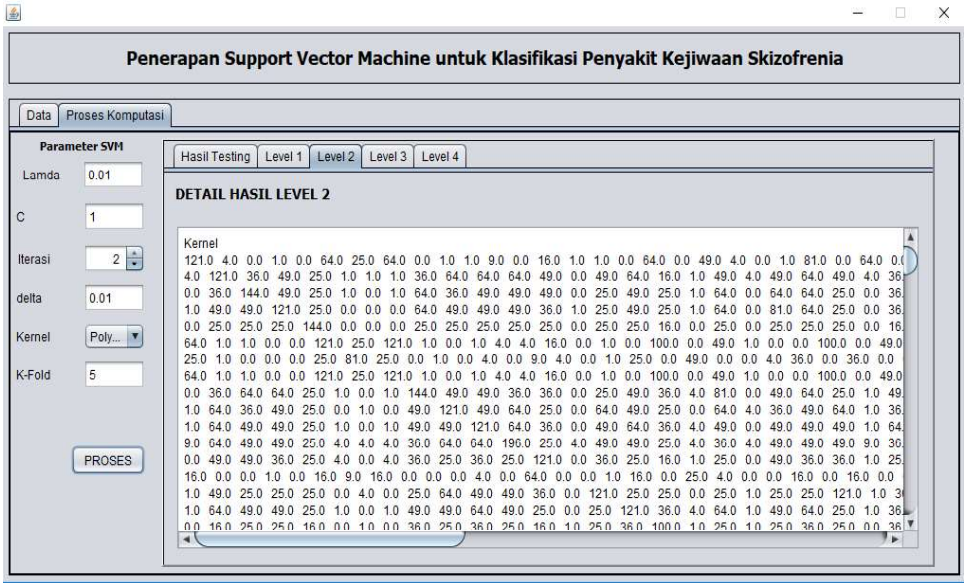


Gambar 5.2 Antarmuka Halaman Level 1

5.3.3 Implementasi Antarmuka Halaman Level 2

Pada halaman ini akan ditampikan semua hasil dari perhitungan *training* pada level 2. Perhitungan ini terdiri dari hasil *kernel* hingga nilai bias pada level

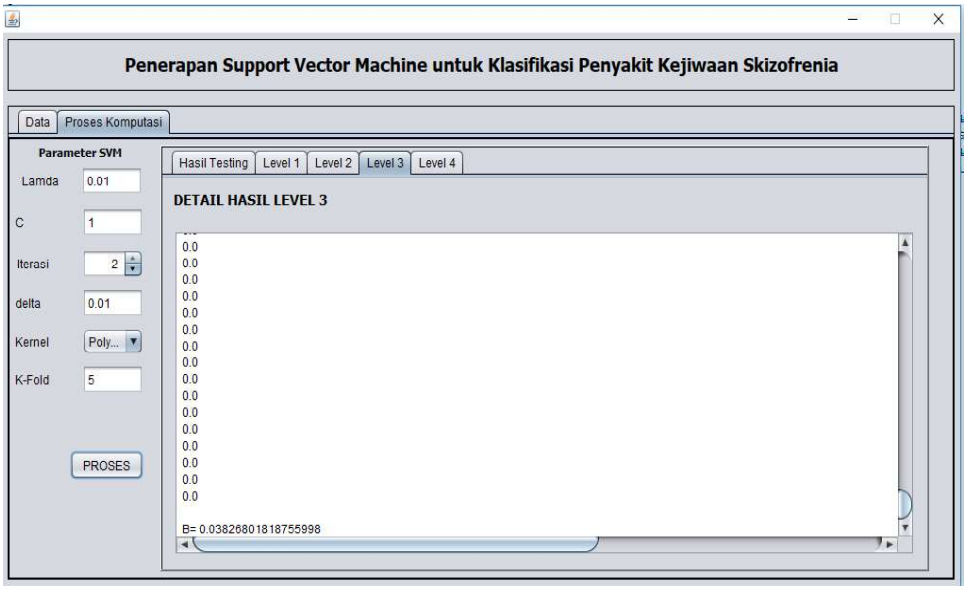
2. Implementasi untuk antarmuka pada halaman level 2 ini ditampilkan pada Gambar 5.15.



Gambar 5.3 Antarmuka Halaman Level 2

5.3.4 Implementasi Antarmuka Halaman Level 3

Pada halaman ini akan ditampikan semua hasil dari perhitungan *training* pada level 3. Perhitungan ini terdiri dari hasil *kernel* hingga nilai bias pada level 3. Implementasi untuk antarmuka pada halaman level 3 ini ditampilkan pada Gambar 5.16.

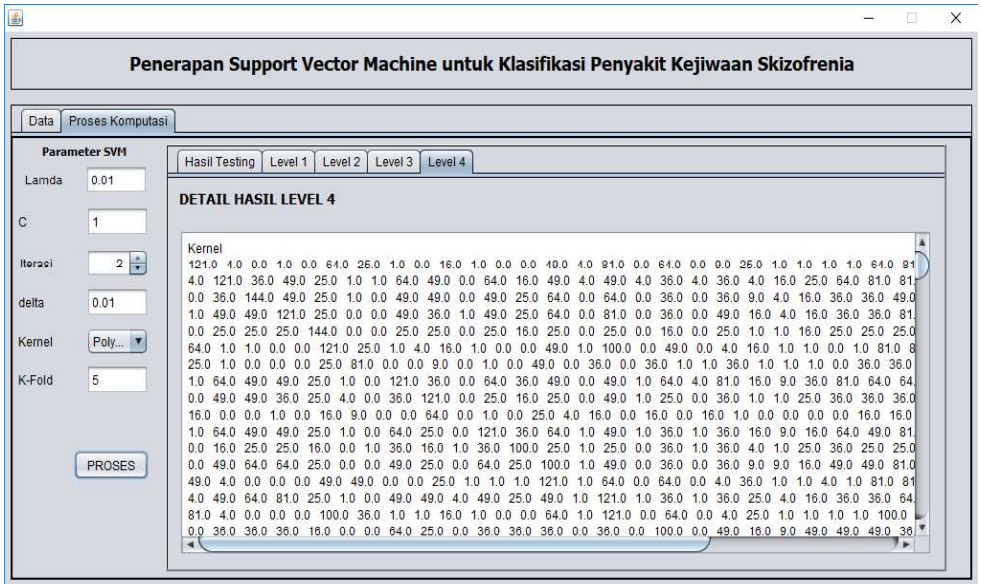


Gambar 5.4 Antarmuka Halaman Level 3

5.3.5 Implementasi Antarmuka Halaman Level 4

Pada halaman ini akan ditampikan semua hasil dari perhitungan *training* pada level 4. Perhitungan ini terdiri dari hasil *kernel* hingga nilai bias pada level

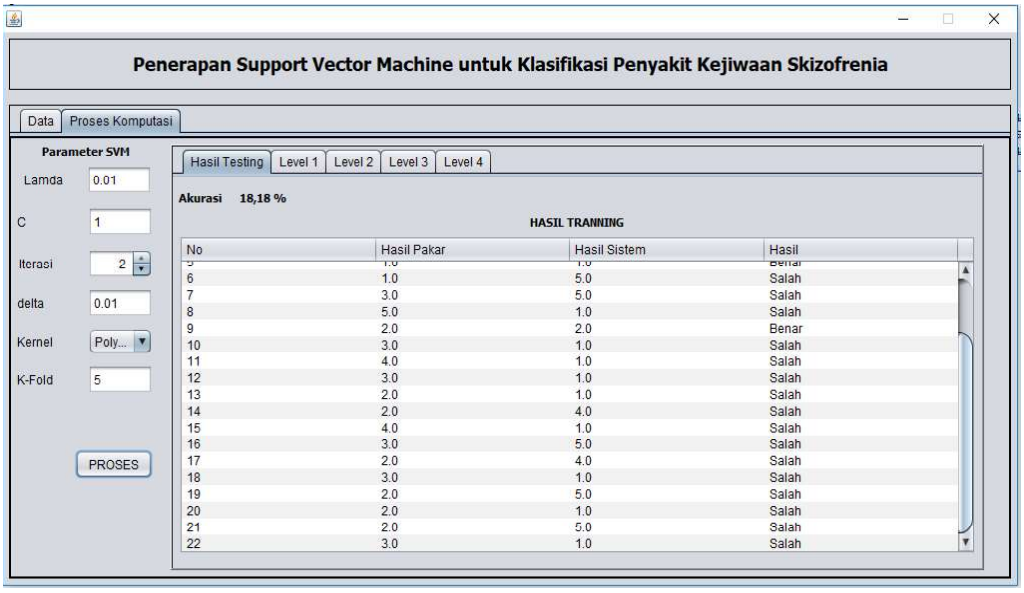
4. Implementasi untuk antarmuka pada halaman level 4 ini ditampilkan pada Gambar 5.17.



Gambar 5.5 Antarmuka Halaman Level 4

5.3.6 Implementasi Antarmuka Halaman Hasil Pengujian

Pada halaman ini akan ditampikan semua hasil yang dihasilkan sistem dan selanjutnya akan dibandingkan sesuai hasil dari pakar. Proses ini berfungsi mendapatkan hasil akurasi terakhir. Implementasi untuk antarmuka pada halaman hasil testing ditampilkan pada Gambar 5.18.



Gambar 5.6 Antarmuka Halaman Hasil Pengujian