BAB IV
IMPLEMENTASI

Bab ini berisi tentang lingkungan implentasi dan implementasi metode AHP-SVM untuk klasifikasi penerima RASKIN. Lingkungan implementasi menjelaskan tentang implementasi perangkat keras dan lunak. Implementasi aplikasi berisi algoritma yang digunakan dalam metode AHP-SVM.

4.1 Lingkungan Implementasi

Pada bagian sub bab ini membahas tentang lingkungan perangkat keras dan perangkat lunak. Lingkungan perangkat keras dan perangkat lunak digunakan untuk memenuhi kebutuhan sistem.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini adalah sebagai berikut.

1. *Processor* Inter Core Duo @1.73 Ghz
2. *Memory* RAM 3 GB
3. *Hardisk* 250 MB
4. Monitor Laptop 14 inch
5. *Keyboard*
6. *Mouse*

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan adalah sebagai berikut.

1. Microsoft Office Excel 2013 digunakan sebagai proses manualisasi perhitungan metode AHP-SVM.
2. Microsoft Office Word 2013 digunakan sebagai proses pembuatan laporan.
3. Microsoft Office Visio 2007 digunakan sebagai proses pembuatan diagram alir proses metode AHP-SVM.
4. Microsoft Visual Studio 2010 digunakan sebagai proses pembuatan aplikasi untuk klasifikasi penerima RASKIN dengan metode AHP-SVM.

4.2 Batasan Implementasi

Implementasi Aplikasi untuk klasifikasi penerima RASKIN di Kelurahan Ronggomulyo Kabupaten Tuban dengan menggunakan metode AHP-SVM memiliki batasan sebagai berikut.

1. Program yang dikembangkan menggunakan 100 data yang diperoleh dari hasil observasi warga dan interview dengan pihak Kelurahan Ronggomulyo Kabupaten Tuban.
2. Data yang digunakan memiliki format *file* .xls yang di *Load* ketika akan menjalankan. Data tersebut tidak dapat berubah karena bersifat statis dan sesuai dengan analisis kriteria yang ditetapkan oleh pemerinta pusat.
3. Terdapat 10 kriteria yang yaitu pekerjaan, ukuran rumah, status rumah, kondisi dinding, kondisi lantai, atap rumah, penghasilan, jumlah tanggungan, aset pribadi, dan aset elektronik. Dalam tiap kriteria terdapat subkriteria yang digunakan dalam perhitungan proses AHP.
4. Normalisasi *dataset* yang digunakan dengan skala [0.1 ,0.9].
5. Metode *Training* SVM menggunakan metode *Sequential Training* SVM.
6. Klasifikasi yang dilakukan untuk menentukan Layak atau Tidak Layak masyarakat tersebut menerima RASKIN.

4.3 Implementasi Aplikasi

Pada bagian sub bab implementasi aplikasi menjelaskan tentang aplikasi yang akan dibuat berdasarkan rancangan yang telah dipaparkan dalam BAB III yaitu metodologi dan perancangan. Dalam sub bab ini implementasi yang dilakukan adalah proses *load* data matriks perbandingan berpasangan, proses perhitungan bobot dengan metode AHP, proses normalisasi, proses *training* SVM, proses *testing* SVM, dan proses menghitung nilai akurasi sistem.

4.3.1 Implementasi Proses Load Matriks Perbandingan Kriteria

Proses awal dalam aplikasi ini adalah melakukan *load* matriks perbandingan untuk kriteria. Kemudian akan dihitung bobot kriteria dan diketahui nilai konsistensi dengan menggunakan metode AHP. Data matriks yang diinputkan berasal dari Microsoft Excel dipilih dengan format data berupa *file* load matriks

dapat dilihat pada *Sourcecode* 4.1. Fungsi yang akan dijalankan adalah fungsi "loadMatriks()".

```
1  private void loadMatriks()
2          {
3  string pathConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
4  txtPath.Text + ";Extended Properties=\"Excel 8.0;HDR=Yes;\";";
5              OleDbConnection conn = new OleDbConnection(pathConn);
6              OleDbDataAdapter myDataAdapter = new
7  OleDbDataAdapter("select * from [" + txtSheetMatrix.Text + "$]",
8  conn);
9                  DataTable dt = new DataTable();
10                 myDataAdapter.Fill(dt);
11                 dgvMatrix.DataSource = dt;
12 }
```

Source Code 4.1 *Listing Code* proses load matriks kriteria

Berikut adalah penjelasan *Source Code* 4.1.

1. Baris 3-4 merupakan inisialisasi variabel untuk mengambil data dari file *excel*.
2. Baris 5-6 adalah membuat variabel baru untuk mengambil data dari *file excel* berdasarkan nama *sheet* pada *texboxt*.
3. Baris 9-11 adalah menampilkan data dari file *excel* ke *datagridview*.

4.3.2 Implementasi Proses Perhitungan AHP

Proses perhitungan AHP diawalai dengan proses normalisasi matriks yang sebelumnya sudah di *load* dari *file Excel*. Setelah data matriks dinormalisasi dilakukan perhitungan bobot kriteria. Hasil perhitungan bobot kriteria digunakan untuk menentukan nilai konsistensi dari matriks tersebut. Proses perhitungan AHP untuk mendapatkan nilai bobot sub kriteria mempunyai langkah yang sama dengan proses perhitungan bobot kriteria. Pada *Source Code* 4.2 ditunjukkan implementasi proses perhitungan bobot menggunakan AHP. Fungsi yang akan dijalankan adalah fungsi "ahpbobot()" yang diproses pada *class* "p_ahp.cs".

```
1  public void ahpBobot()
2          {
3              //Menghitung Bobot Kriteria
4              bobotKriteria = new double[jmldata];
5              double tempTotal = 0;
6
7              for (int i = 0; i < jmldata; i++)
8              {
```

```
9                   tempTotal = 0;
10                  for (int j = 0; j < jmldata; j++)
11                  {
12                      tempTotal += matrikNormalisasi[j, i];
13                  }
14                  bobotKriteria[i] = tempTotal / jmldata;
15              }
16          }
17
18          public void aX()
19          {
20              //Menghitung Nilai AX
21              ax = new double[jmldata];
22              double tempTotal = 0;
23              for (int i = 0; i < jmldata; i++)
24              {
25                  tempTotal = 0;
26                  for (int j = 0; j < jmldata; j++)
27                  {
28                      tempTotal += matrikKriteria[j, i] *
29  bobotKriteria[j];
30                  }
31                  ax[i] = tempTotal;
32              }
33          }
34
35          public void Lamda()
36          {
37              //Hitung Nilai Lamda
38              double tempTotl = 0;
39              for (int i = 0; i < jmldata; i++)
40              {
41                  tempTotl += ax[i] / bobotKriteria[i];
42              }
43              lamda = tempTotl / jmldata;
44          }
45
46          public void Ci()
47          {
48              //Hitung CI
49              ci = (lamda - jmldata) / (jmldata - 1);
50          }
51
52          public void Cr()
53          {
54              //Hitung CR
55              if (jmldata == 1 || jmldata == 2)
56              {
57                  ir = 0;
58              }
59              else if (jmldata == 3)
60              {
61                  ir = 0.58;
62              }
63              else if (jmldata == 4)
64              {
65                  ir = 0.90;
66              }
67              else if (jmldata == 5)
```

```
68              {
69                  ir = 1.12;
70              }
71              else if (jmldata == 6)
72              {
73                  ir = 1.24;
74              }
75              else if (jmldata == 7)
76              {
77                  ir = 1.32;
78              }
79              else if (jmldata == 8)
80              {
81                  ir = 1.41;
82              }
83              else if (jmldata == 9)
84              {
85                  ir = 1.45;
86              }
87              else if (jmldata == 10)
88              {
89                  ir = 1.49;
90              }
91              else if (jmldata == 11)
92              {
93                  ir = 1.51;
94              }
95              else if (jmldata == 12)
96              {
97                  ir = 1.48;
98              }
99              else if (jmldata == 13)
100             {
101                 ir = 1.56;
102             }
103             else if (jmldata == 14)
104             {
105                 ir = 1.57;
106             }
107             else if (jmldata == 15)
108             {
109                 ir = 1.59;
110             }
111             cr = ci / ir;
112         }
```

Source Code 4.2 *Listing Code* Proses Perhitungan AHP

Berikut adalah penjelasan *Source Code* 4.2.

1. Baris 4-5 merupakan inisialisasi variabel bobot kriteria, dan baris 7-15 proses perhitungan bobot kriteria AHP.
2. Baris 22-24 adalah inisialisasi variabel Ax, dan baris 25-33 adalah proses perhitungan nilai Ax.
3. Baris 38-46 adalah proses perhitungan nilai *lamda*.

4. Baris 49-53 adalah proses perhitungan nilai *CI*.
5. Baris 56-117 adalah proses seleksi kondisi jumlah data dan perhitungan nilai *CR*.

Setelah diperoleh nilai bobot kriteria proses selanjutnya adalah *sorting* nilai bobot berdasarkan nilai *threshold* yang ditentukan. Pada *Source Code* 4.3 ditunjukkan implementasi proses *sorting* bobot kriteria terpilih. Fungsi yang akan dijalankan adalah fungsi "p_sort()"yang diproses pada *class* "p_sorting.cs".

```
1   public void p_sort()
2           {   //Proses Sorting
3               double tempMaks = 0;
4               string strTemp = "-1";
5               int indeksTerpilih = 0;
6
7               for (int i = 0; i < tempJum; i++)
8               {
9                   tempMaks = 0;
10                  for (int j = 0; j < tempJum; j++)
11                  {
12                      strTemp = tempSubBobotK[j].ToString();
13                      if (strTemp != "-1")
14                      {
15                          if (Convert.ToDouble(tempSubBobotK[j]) >
16  tempMaks)
18                          {
19                              tempMaks =
20  Convert.ToDouble(tempSubBobotK[j]);
21                              indeksTerpilih = j;
22                          }
23                      }
24                  }
25                  tempSubBobotK[indeksTerpilih] = "-1";
26                  sortingResult[i] = indeksTerpilih;
27              }
28          }
```

Source Code 4.3 *Listing Code* Proses *sorting* bobot kriteria

Berikut adalah penjelasan *Source Code* 4.3.

1. Baris 4-6 merupakan inisialisasi variabel.
2. Baris 8-29 adalah merupakan proses *sorting* data bobot terpilih.

4.3.3 Implementasi Proses Normalisasi

Proses Normalisasi dilakukan berdasarkan nilai interval tertentu. Proses normalisasi dilakukan setelah kriteria yang digunakan dipilih berdasarkan nilai

threshold yang telah ditentukan pada proses sebelumnya. Pada *Source Code* 4.4 ditunjukkan implementasi proses normalisasi. Fungsi yang akan dijalankan adalah fungsi "normalisasi()" yang diproses pada *class* "p_normalisasi.cs".

```
1   public void normalisasi()
2           {
3               int tempMax = 0;
4               int tempMin = 0;
5               tempMax = dataAll[0, 0];
6               tempMin = dataAll[0, 0];
7
8               for (int i = 0; i < totSubKTerpilih; i++)
9               {
10                  for (int j = 0; j < jumData; j++)
11                  {
12                      if (tempMax < dataAll[i, j])
13                      {
14                          tempMax = dataAll[i, j];
15                      }
16                      if (tempMin > dataAll[i, j])
18                      {
19                          tempMin = dataAll[i, j];
20                      }
21                  }
22              }
23              dataNormalisasi = new double[totSubKTerpilih, jumData];
24              double tempNormal = 0;
25              for (int i = 0; i < totSubKTerpilih; i++)
26              {
27                  for (int j = 0; j < jumData; j++)
28                  {
29                      tempNormal = ((0.8 * (dataAll[i, j] - tempMin)) /
30  (tempMax - tempMin)) + 0.1;
31                      dataNormalisasi[i, j] = tempNormal;
32                  }
33              }
34          }
```

Source Code 4.4 *Listing Code* proses normalisasi

Berikut adalah penjelasan *Source Code* 4.4.

1. Baris 3-6 merupakan inisialisasi variabel.
2. Baris 8-20 adalah proses mencari nilai *maximum* dan *minimum*.
3. Baris 24-34 adalah proses perhitungan normalisasi.

Setelah perhitungan normalisasi data, proses selanjutnya adalah perkalian data normalisasi dengan bobot kriteria terpilih. Pada *Source Code* 4.5 ditunjukkan implementasi proses perkalian antara data normalisasi dengan bobot kriteria

terpilih. Fungsi yang akan dijalankan adalah fungsi "bobotNorm()" yang diproses pada *class* p_bobotNormalisasi.cs.

```
1   public void bobotNorm()
2           {
3               dataBobot = new double[totSubKTerpilih, jumData];
4               double tempBobot = 0;
5
6               for (int i = 0; i < totSubKTerpilih; i++)
7               {
8                   for (int j = 0; j < jumData; j++)
9                   {
10                      tempBobot = dataNormalisasi[i, j] *
11  Convert.ToDouble(bobotSubK[Convert.ToInt32(subKTerpilih[i])]);
12                      dataBobot[i, j] = tempBobot;
13                  }
14              }
15          }
```

Source Code 4.5 *Listing Code* Proses data bobot ternormalisasi

Berikut adalah penjelasan *Source Code* 4.5.

1. Baris 3-4 merupakan inisialisasi variabel.
2. Baris 6-12 adalah proses perkalian data normalisasi dengan bobot kriteria yang terpilih.

4.3.4 Implementasi *Sequential Training* SVM

Pengujian aplikasi diawali dengan memilih perbandingan rasio data *training* dan data *testing* yang akan digunakan. Kemudian melakukan inisialisasi parameter proses *training* SVM. Proses *training* menggunakan metode *sequential training* SVM. Data yang digunakan untuk proses pengujian adalah data bobot yang merupakan hasil perkalian data ternormalisasi dengan data bobot kriteria.

Proses perhitungan menggunakan metode *sequential training* SVM, dimulai dengan inisialisasi variabel, perhitungan kernel yang digunakan, matriks hessian, nilai E_i, $\delta\alpha_i$, nilai α_i baru, iterasi serta perhitungan untuk mendapatkan nilai *b*.

Pada *Source Code* 4.6 ditunjukkan implementasi proses perhitungan *kernel polynomial*. Fungsi yang akan dijalankan adalah fungsi "K_Poli()" yang diproses pada *class* "p_kernel.cs".

```
1   public void K_Poli()
2           {
3               kernel = new double[jumDatTrain, jumDatTrain];
4               //Menghitung Kernel Polinomial
5               for (int i = 0; i < jumDatTrain; i++)
6               {
7                   indeksTerpilih =
8   Convert.ToInt32(datTrainTerpilih[i]);
9                   for (int j = 0; j < jumDatTrain; j++)
10                  {
11                      tempTotal = 0;
12                      tempIndeks =
13  Convert.ToInt32(datTrainTerpilih[j]);
14                      for (int l = 0; l < totSubKTerpilih; l++)
15                      {
16                          tempTotal += dataBobot[l, indeksTerpilih] *
18  dataBobot[l, tempIndeks];
19                      }
20                      //hasil = tempTotal * tempTotal;
21                      hasil = Math.Pow(tempTotal, degree);
22                      kernel[i, j] = hasil;
23                  }
24              }
25          }
```

Source Code 4.6 *Listing Code* Proses perhitungan *kernel polynomial*

Berikut adalah penjelasan *Source Code* 4.6.

1. Baris 3 merupakan inisialisasi variabel.
2. Baris 5-21 adalah proses perhitungan *kernel polynomial*.

Pada *Source Code* 4.7 ditunjukkan implementasi proses perhitungan *kernel Gaussian RBF*. Fungsi yang akan dijalankan adalah fungsi "K_RBF()" yang diproses pada *class* "p_kernel.cs".

```
1   public void K_RBF()
2               {
3                   kernel = new double[jumDatTrain, jumDatTrain];
4                   //Menghitung Kernel RBF
5                   for (int i = 0; i < jumDatTrain; i++)
6                   {
7                       indeksTerpilih =
8   Convert.ToInt32(datTrainTerpilih[i]);
9                       for (int j = 0; j < jumDatTrain; j++)
10                      {
11                          tempTotal = 0;
12                          tempIndeks =
13  Convert.ToInt32(datTrainTerpilih[j]);
14                          for (int l = 0; l < totSubKTerpilih; l++)
15                          {
16                              tempTotal += dataBobot[l, indeksTerpilih]
18  - dataBobot[l, tempIndeks];
```

```
19                          }
20                          hasil = (-1 / (2 * sigma * sigma)) *
21  (tempTotal * tempTotal);
22                          hasil = Math.Exp(hasil);
23
24                          kernel[i, j] = hasil;
25                      }
26                  }
27              }
```

Source Code 4.7 *Listing Code* Proses perhitungan *kernel Gaussian RBF*.

Berikut adalah penjelasan *Source Code* 4.7.

1. Baris 3 merupakan inisialisasi variabel.
2. Baris 5-21 adalah proses perhitungan *kernel polynomial*.

Pada *Source Code* 4.8 ditunjukkan implementasi proses perhitungan *Matriks Hessian*. Fungsi yang akan dijalankan adalah fungsi "matr_hes()"yang diproses pada *class* "p_matriksHes.cs".

```
1   public void matr_hes()
2           {
3               matrikHeisian = new double[jumDatTrain, jumDatTrain];
4
5               double tempTotal = 0;
6               int indeksTerpilih = 0, tempIndeks = 0;
7
8               //Menghitung Matrik Heissian
9               for (int i = 0; i < jumDatTrain; i++)
10              {
11                  indeksTerpilih =
12  Convert.ToInt32(datTrainTerpilih[i]);
13                  for (int j = 0; j < jumDatTrain; j++)
14                  {
15                      tempIndeks =
16  Convert.ToInt32(datTrainTerpilih[j]);
18                      tempTotal = (dataClass[indeksTerpilih] *
19  dataClass[tempIndeks]) * (kernel[i, j] + (valLamda * valLamda));
20                      matrikHeisian[i, j] = tempTotal;
21                  }
22              }
23          }
```

Source Code 4.8 *Listing Code* Proses perhitungan *Matriks Hessian.*

Berikut adalah penjelasan *Source Code* 4.8.

1. Baris 3-6 merupakan inisialisasi variabel.
2. Baris 8-19 merupakan proses perhitungan *matriks hessian*.

Pada *Source Code* 4.9 ditunjukkan implementasi proses perhitungan Maximal Diagonal Hessian. Fungsi ini digunakan untuk proses perhitungan nilai γ *(gamma)*, yang merupakan fungsi untuk mengontrol kecepatan proses *learning* Fungsi yang akan dijalankan adalah fungsi "MaxDiag_Hessian()".

```
1   public void MaxDiag_Hessian()
2           {
3               konstantaGamma = Convert.ToDouble(tbKonstantaGamma.Text);
4               maxHess = 0;
5               //Hitung MaxDiagonal Gamma
6               for (int i = 0; i < jumDatTrain; i++)
7               {
8                   maxHess = Math.Max(maxHess, matrikHeisian[i, i]);
9               }
10              valGamma = konstantaGamma / maxHess;
11          }
```

Source Code 4.9 *Listing Code* Proses perhitungan Maximal Diagonal Hessian.

Berikut adalah penjelasan *Source Code* 4.9.

1. Baris 3-4 merupakan inisialisasi variabel.
2. Baris 8-19 merupakan proses perhitungan *maximum diagonal hessian.*

Pada *Source Code* 4.10 ditunjukkan implementasi proses *sequential training* SVM pada tahap perhitungan nilai E_i. Fungsi yang akan dijalankan adalah fungsi "Val_Ei()".

```
1   public void Val_Ei()
2           {//Hitung Matrik Ei
3               double tempTotal = 0;
4               for (int i = 0; i < jumDatTrain; i++)
5               {
6                   for (int j = 0; j < jumDatTrain; j++)
7                   {
8                       matrikEi[i, j] = matrikHeisian[i, j] * valAi[i];
9                   }               }
10              //Hitung Nilai Ei
11              tempTotal = 0;
12              for (int i = 0; i < jumDatTrain; i++)
13              {
14                  tempTotal = 0;
15                  for (int j = 0; j < jumDatTrain; j++)
16                  {
18                      tempTotal += matrikEi[j, i];
19                  }
20                  valEi[i] = tempTotal;
21              }
                return;         }
```

Source Code 4.10 Implementasi proses perhitungan nilai E_i

Berikut adalah penjelasan *Source Code* 4.10.

1. Baris 3 merupakan inisialisasi variabel.
2. Baris 4-8 merupakan proses perhitungan matriks E_i.
3. Baris 10-19 merupakan proses perhitungan nilai E_i.

Pada *Source Code* 4.11 ditunjukkan implementasi proses *sequential training* SVM pada tahap perhitungan nilai $\delta\alpha_i$, nilai α_i baru dan Iterasi yang dilakukan pada data data *training* jika telah mencapai nilai konvergen ($|\delta\alpha_i| < \varepsilon$), atau mencapai nilai *itermax* maka iterasi akan dihentikan. Fungsi yang akan dijalankan adalah fungsi "sequen_train()".

```
1     private void sequen_train()
2         {
3             maxDeltaAi = 0;
4             valEpsilon = Convert.ToDouble(tbepsilon.Text);
5             iterMax = Convert.ToInt32(tbitermax.Text);
6 
7             MaxDiag_Hessian();
8             //Isi Matrix Ai
9             for (int i = 0; i < jumDatTrain; i++)
10            {
11                valAi[i] = 0;
12            }
13            //Inisialisasi Variabel Stop
14            int iterasiStop = 0;
15            x = 0;
16            double tempTotal = 0;
17
18            while (x <= iterMax)
19            {
20                Val_Ei();
21                //Hitung Delta ai
22                tempTotal = 0;
23                maxDeltaAi = 0;
24                for (int i = 0; i < jumDatTrain; i++)
25                {
26                    tempTotal = Math.Min((Math.Max(valGamma * (1 -
27 valEi[i]), -1 * valAi[i])), (valC - valAi[i]));
28                    valDeltaAi[i] = tempTotal;
29                    maxDeltaAi = Math.Max(maxDeltaAi,
30 Math.Abs(tempTotal));
31                }
32                //Hitung Nilai Ai yang baru
33                tempTotal = 0;
34                for (int i = 0; i < jumDatTrain; i++)
35                {
36                    tempTotal = valAi[i] + valDeltaAi[i];
37                    valAi[i] = tempTotal;
38                }
39                if (maxDeltaAi <= valEpsilon)
40
41
```

```
42                  {
43                      iterasiStop = x;
44                      x = iterMax;
45                  }
46                  else
47                  {
48                      iterasiStop = x;
49                      x++;
50                  }
51              }
52  laResult.Text = "Iterasi Berhenti pada Iterasi Ke - " +
53  iterasiStop.ToString();
54              return;
55          }
56
```

Source Code 4.11 *Listing Code* Proses perhitungan sequential training SVM.

Berikut adalah penjelasan Source Code 4.11.

1. Baris 3-5 merupakan inisialisasi variabel.
2. Baris 7 merupakan fungsi untuk memanggil nilai pada *method maximum diagonal hessian*.
3. Baris 8-12 merupakan proses untuk memberikan nilai pada matriks Ai.
4. Baris 14-17 merupakan inisialisasi untuk variabel stop pada iterasi.
5. Baris 19-50 merupakan proses perhitungan nilai $\delta\alpha_i$ dan nilai α_i baru yang dilakukan didalam proses iterasi.
6. Baris 52-54 menampilkan berhentinya proses iterasi pada *label*.

Pada *Source Code* 4.12 ditunjukkan implementasi proses *sequential training* SVM pada tahap ini dilakukan perhitungan untuk mendapatkan nilai *support vector positive* dan *negative* serta nilai *b* (bias). Fungsi yang akan dijalankan adalah fungsi "SV_Pos_Neg()".

```
1   public void SV_Pos_Neg()
2           {
3               //Inisialisasi Variabel Kernel dan Perhitungan SV
4               int tempClass = 0, tempXPost = 0, tempXNeg;
5               double tempPost = 0, tempNeg = 0, hasilPost = 0, hasilNeg
6   = 0, svmPos, svmNeg, sigmaPost = 0, sigmaNeg = 0;
7
8               tempXPost = Convert.ToInt32(dataClassPositif[0]);
9               tempXNeg = Convert.ToInt32(dataClassNegatif[0]);
10              for (int i = 0; i < jumDatTrain; i++)
11              {
12                  hasilPost = 0;
13                  hasilNeg = 0;
14                  tempPost = 0;
15                  tempNeg = 0;
```

```
16
17                  tempClass = Convert.ToInt32(datTrainTerpilih[i]);
18
19                  for (int j = 0; j < totSubKTerpilih; j++)
20                  {
21                      tempPost += dataBobot[j, tempClass] *
22  dataBobot[j, tempXPost];
23                      tempNeg += dataBobot[j, tempClass] * dataBobot[j,
24  tempXNeg];
25                  }
26                  hasilPost = tempPost * tempPost;
27                  hasilNeg = tempNeg * tempNeg;
28                  svmPos = hasilPost * dataClass[tempClass] * valAi[i];
29                  svmNeg = hasilNeg * dataClass[tempClass] * valAi[i];
30
31                  sigmaPost += svmPos;
32                  sigmaNeg += svmNeg;
33              }
34              //Menghitung Nilai b
35              bias = (-0.5) * (sigmaPost + sigmaNeg);
36              return;
37          }
38
```

Source Code 4.12 *Listing Code* proses perhitungan *support vector* dan nilai *b*.

Berikut adalah penjelasan Source Code 4.12.

1. Baris 3-10 merupakan inisialisasi variabel.
2. Baris 11-15 merupakan fungsi untuk memberikan niai awal pada variabel.
3. Baris 19-33 merupakan proses perhitungan *support vector positive* dan *negative*.
4. Baris 36 merupakan perhitungan nilai *b (bias)*.

4.3.5 Implementasi Proses *Testing* SVM

Proses *testing* menggunakan data yang terpilih berdasarkan rasio perbandingan. Data yang digunakan untuk proses pengujian adalah data bobot yang merupakan hasil perkalian data ternormalisasi dengan data bobot kriteria. Proses perhitungan *testing* selanjutnya adalah menghitung *kernel* data *testing*, nilai *f(x)*, dan *sign f(x)*. Hasil dari proses *testing* adalah keputusan berupa nilai klasifikasi atau *predicted class* yang menentukan kelayakan penerima RASKIN Kelurahan Ronggomulyo Kabupaten Tuban. Nilai -1 menunjukkan *class* layak menerima RASKIN, sedangkan 1 menunjukkan *class* tidak layak menerima RASKIN.

Pada *Source Code* 4.13 ditunjukkan implementasi proses *testing* SVM. Fungsi yang akan dijalankan adalah fungsi “testing()”.

```
1   private void testing()
2           {
3               //Inisialisasi Variabel
4               classNode = new int[jumDatTest];
5
6               x = 0;
7               int tempDatTest = 0, tempDatTrain = 0;
8               double tempKernelTest = 0, valAYK = 0, sigmaTest;
9               string strKeputusan = "";
10
11              rbprostesting.Text += "Hasil Data Testing\n\n";
12              while (x < jumDatTest)
13              {
14                  tempDatTest = Convert.ToInt32(datTestTerpilih[x]);
15                  sigmaTest = 0;
16                  for (int i = 0; i < jumDatTrain; i++)
17                  {
18                      tempKernelTest = 0;
19                      tempDatTrain =
20  Convert.ToInt32(datTrainTerpilih[i]);
21                      valAYK = 0;
22                      for (int j = 0; j < totSubKTerpilih; j++)
23                      {
24                          tempKernelTest += dataBobot[j, tempDatTest] *
25  dataBobot[j, tempDatTrain];
26                      }
27
28                      tempKernelTest = tempKernelTest * tempKernelTest;
29                      valAYK = valAi[i] * tempKernelTest *
30  dataClass[tempDatTrain];
31
32                      sigmaTest += valAYK;
33                  }
34
35                  valFx = bias + sigmaTest;
36                  classNode[x] = Math.Sign(valFx);
37
38                  rbprostesting.Text += "\nData Testing KL_" +
39  (tempDatTest + 1).ToString() + "\n\n";
40                  rbprostesting.Text += "Sigma = " +
41  Math.Round(sigmaTest, 7).ToString() + "\n";
42                  rbprostesting.Text += "f(x) = " + Math.Round(valFx,
43  7).ToString() + "\n";
44                  rbprostesting.Text += "Predicted Node = " +
45  classNode[x].ToString() + "\n";
46
47                  if (classNode[x] < 0)
48                  {
49                      strKeputusan = "Layak";
50                  }
51                  else
52                  {
53                      strKeputusan = "Tidak Layak";
54                  }
55                  rbprostesting.Text += "Keputusan = " +
56  strKeputusan.ToString() + "\n";
57                  rbprostesting.Text +=
58  "=============================================================
59  ======================\n";
60
61
62
```

```
63                  x++;
64              }
65              return;
66          }
```

Source Code 4.13 *Listing Code* Proses perhitungan *testing* SVM.

Berikut adalah penjelasan *Source Code* 4.13.

1. Baris 3-9 merupakan inisialisasi variabel.
2. Baris 13-34 merupakan fungsi untuk menjumlahkan nilai *kernel* data *testing*.
3. Baris 37 adalah perhitungan nilai *f(x)*.
4. Baris 38 merupakan perhitungan dalam menentukan *predicted class*.
5. Baris 40-48 adalah menampilkan data *testing*, *sigma*, nilai *f(x)*, dan *predicted class* pada data yang telah diuji.
6. Baris 50-57 merupakan proses seleksi kondisi antara hasil klasifikasi Layak atau Tidak Layak.
7. Baris 58-62 menampilkan keputusan hasil klasifikasi.

Pada *Source Code* 4.14 ditunjukkan implementasi proses akurasi sistem. Fungsi yang akan dijalankan adalah fungsi “cek_akurasi ()”.

```
1   private void cek_akurasi()
2           {
3               //Inisialisasi Variabel
4               double akurasi = 0;
5               dataBenar = 0;
6               int tempDatTest = 0;
7
8               //Proses Perhitungan Akurasi
9               for (int i = 0; i < jumDatTest; i++)
10              {
11                  tempDatTest = Convert.ToInt32(datTestTerpilih[i]);
12
13                  if (dataClass[tempDatTest] == classNode[i])
14                  {
15                      if (dataClass[tempDatTest] == 1)
16                      {
17                          tp += 1;
18                      }
19                      else
20                      {
21                          tn += 1;
22                      }
23                  }
24
25
26
```

```
27                  else
28                  {
29                      if (classNode[i] == 1)
30                      {
31                          fp += 1;
32                      }
33                      else
34                      {
35                          fn += 1;
36                      }
37                  }
38              }
39
40              akurasi = Math.Round ((Convert.ToDouble(tp + tn) /
41  Convert.ToDouble(tp + fp + tn + fn)) * 100,2);
42              lbAkurasi.Text = akurasi.ToString() + " %";
43
44              //Menampilkan data benar
45              for (int i = 0; i < jumDatTest; i++)
46              {
47                  tempDatTest = Convert.ToInt32(datTestTerpilih[i]);
48                  if (dataClass[tempDatTest] == classNode[i])
49                  {
50                      dataBenar += 1;
51                  }
52              }
53              lbdtbenar.Text = dataBenar.ToString();
54          }
55
```

Source Code 4.14 *Listing Code* proses perhitungan akurasi sistem.

Berikut adalah penjelasan *Source Code* 4.14.

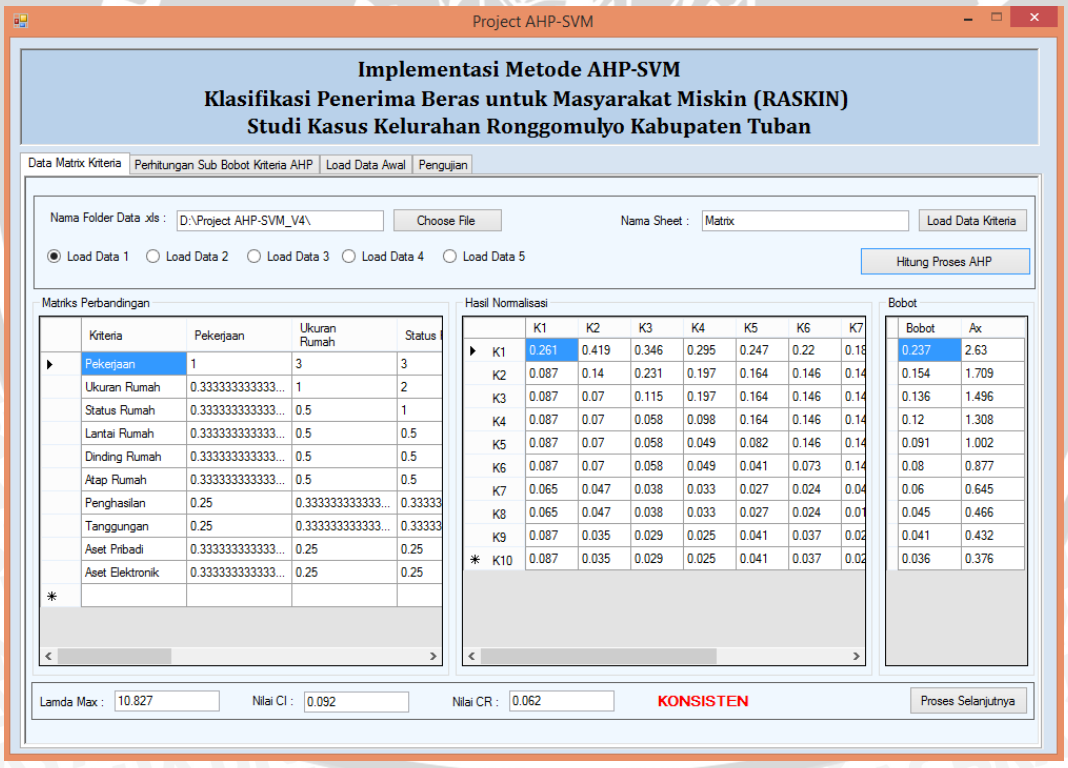
1. Baris 3-6 merupakan inisialisasi variabel.
2. Baris 9-37 merupakan fungsi untuk menentukan hasil klasifikasi *predicted class* data *testing*.
3. Baris 40-41 adalah perhitungan nilai akurasi.
4. Baris 33 menampilkan nilai akurasi pada *label*.
5. Baris 46-53 adalah proses perhitungan data benar antara hasil dari *class actial* dan hasil klasifikasi *predicted class*.
6. Baris 54 menampilkan jumlah data benar pada *label*.

4.4 Implementasi Antar Muka

Implementasi antar muka terdiri dari beberapa bagian diantaranya proses data Matriks, perhitungan data bobot kriteria AHP, *load* data awal, dan pengujian. Berikut penjelasan dari setiap implementasi antarmuka yang terdapat dalam sistem.

4.4.1 Antarmuka *Load* Matriks dan perhitungan AHP level 1

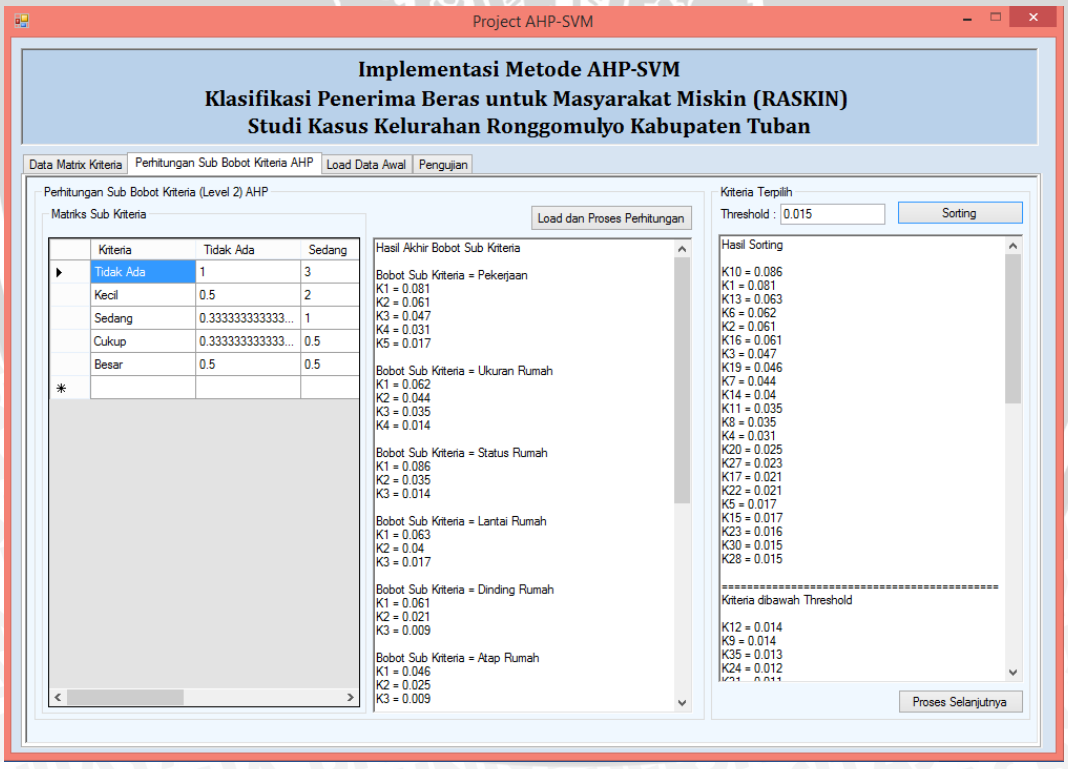
Antarmuka (*interface*) proses *load* matriks dan perhitungan AHP level 1 adalah antarmuka yang menjelaskan proses perhitungan AHP level 1. *Button Choose File* adalah memilih lokasi file matriks perbandingan berpasangan yang berasal dari Microsoft Excel dengan format data berupa *file* .xls. *Button Load* Data Kriteria adalah menampilkan data matriks yang telah dipilih sebelumnya untuk dilakukan proses selanjutnya berdasarkan nama sheet. Terdapat pilihan load data yang akan digunakan dalam pengujian. *Button* Hitung Proses AHP menjelaskan tahapan perhitungan AHP secara keseluruhan. Proses perhitungan diawali dengan normalisasi data matriks kemudian akan dihitung bobot kriteria dan proses terakhir mengukur nilai konsistensi matriks. *Button* proses selanjutnya menunjukkan proses selanjutnya dalam sistem. Gambar 4.1 menunjukkan antarmuka data matriks kriteria AHP.



Gambar 4.1 Antarmuka Data Matriks Kriteria AHP

4.4.2 Antarmuka Perhitungan Sub Bobot Kriteria AHP

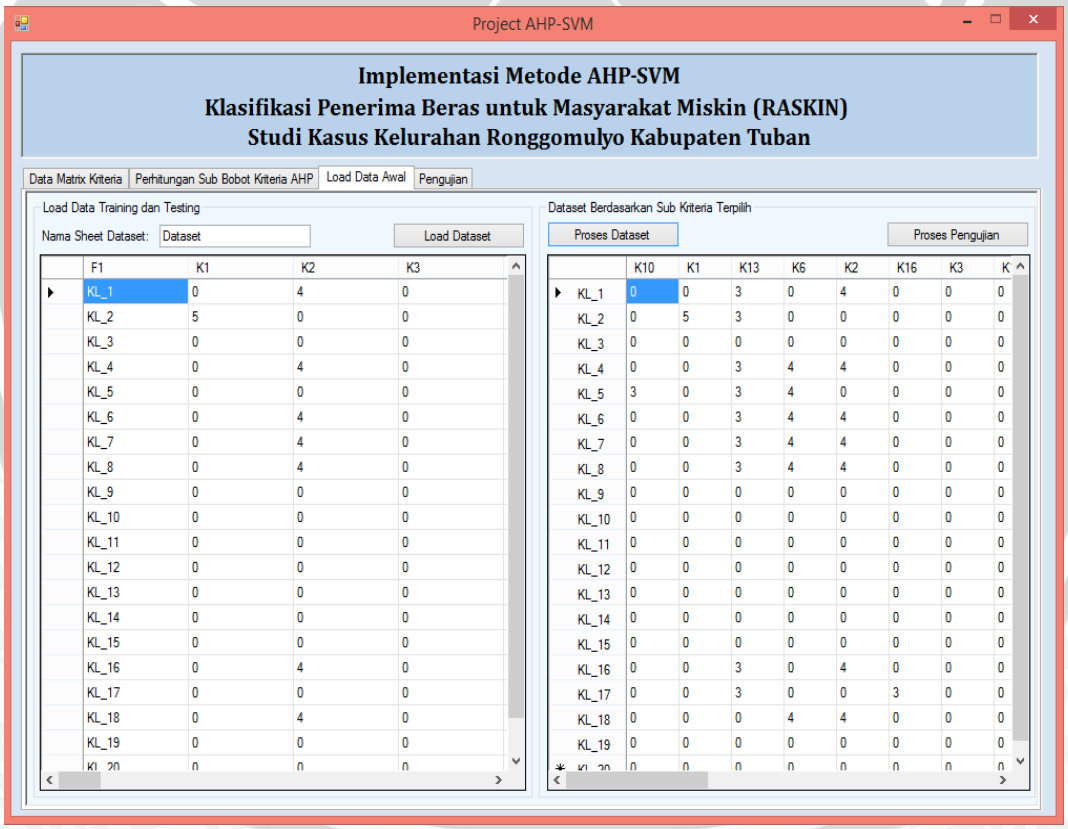
Antarmuka (*interface*) proses perhitungan bobot sub kriteria AHP adalah antarmuka yang menjelaskan proses perhitungan sub kriteria AHP. Pada proses ini Matriks Sub Kriteria akan ditampilkan beserta hasil perhitungan bobot sub kriteria. *Button Load* dan Proses Perhitungan sub kriteria menunjukkan proses *load* matriks perbandingan berpasangan. Tahapan proses perhitungan matriks sub kriteria sama dengan perhitungan sebelumnya, namun pada proses ini tidak sampai menghitung konsistensi matrik. Setelah diproses hasil yang diperoleh adalah nilai bobot sub kriteria. *Textboxt threshold* untuk menginputkan nilai *threshold* sebagai batasan kriteria yang akan digunakan. *Button sorting* akan melakukan proses *sorting* nilai bobot sub kriteria yang bernilai lebih besar sama dengan nilai *threshold* yang telah diinputkan sebelumnya. Gambar 4.2 menunjukkan antarmuka perhitungan sub bobot kriteria AHP.



Gambar 4.2 Antarmuka Perhitungan Sub Bobot Kriteria AHP

4.4.3 Antarmuka *Load* Data Awal

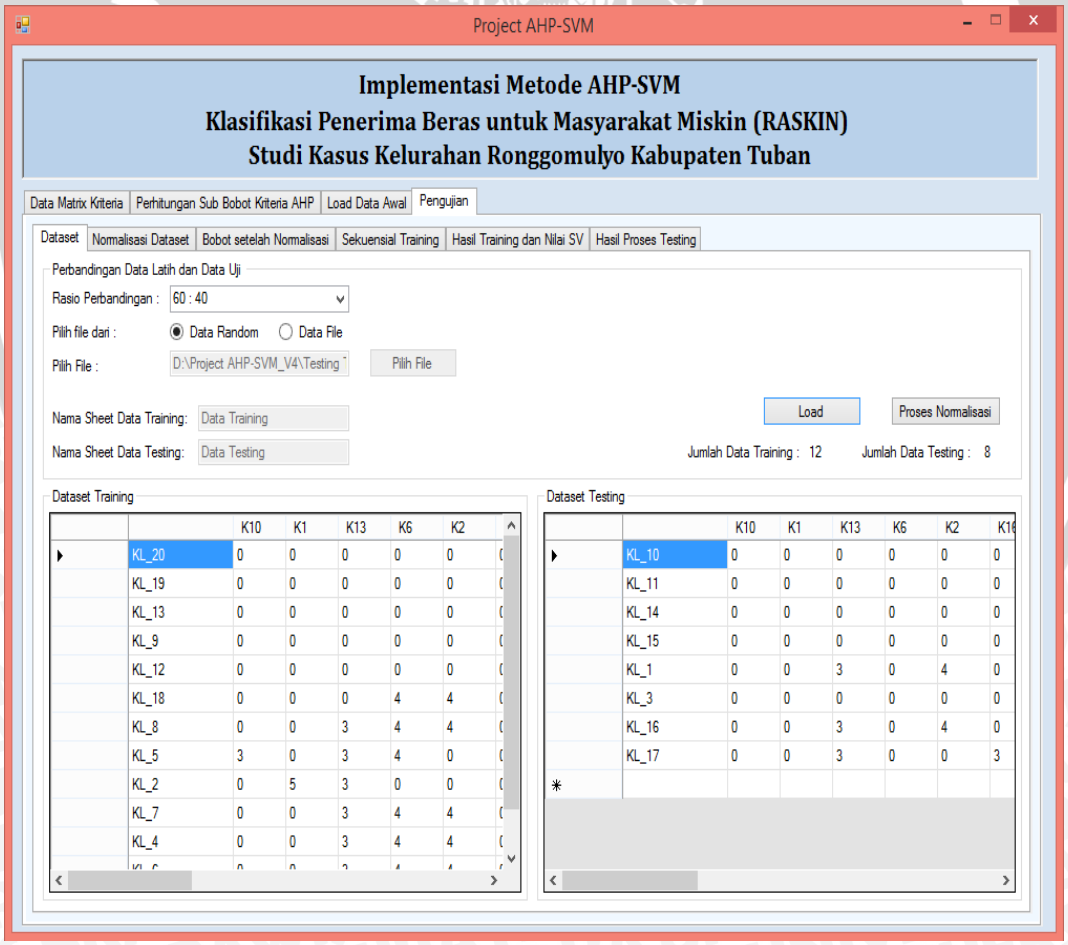
Antarmuka (*interface*) proses *load* data awal adalah antarmuka yang menampilkan data awal sebelum dilakukan proses perhitungan. Jumlah data yang akan digunakan adalah sebanyak 100 data. Data awal akan ditampilkan pada bagian *load* data *training* dan data *testing*. *Button* proses dataset menunjukkan proses pemilihan dataset sesuai dengan bobot nilai kriteria yang telah terpilih pada proses sebelumnya. Sehingga tidak semua kriteria digunakan untuk proses perhitungan selanjutnya, hanya kriteria terpilih yang akan digunakan. Semua data akan ditampilkan pada bagian yang telah disediakan berdasarkan kriteria terpilih. *Button* pengujian menunjukkan proses selanjutnya untuk pengujian data dengan menggunakan metode SVM. Gambar 4.3 menunjukkan antarmuka *load* data awal.



Gambar 4.3 Antarmuka *Load* Data Awal

4.4.4 Antar Muka Normalisasi *Dataset*

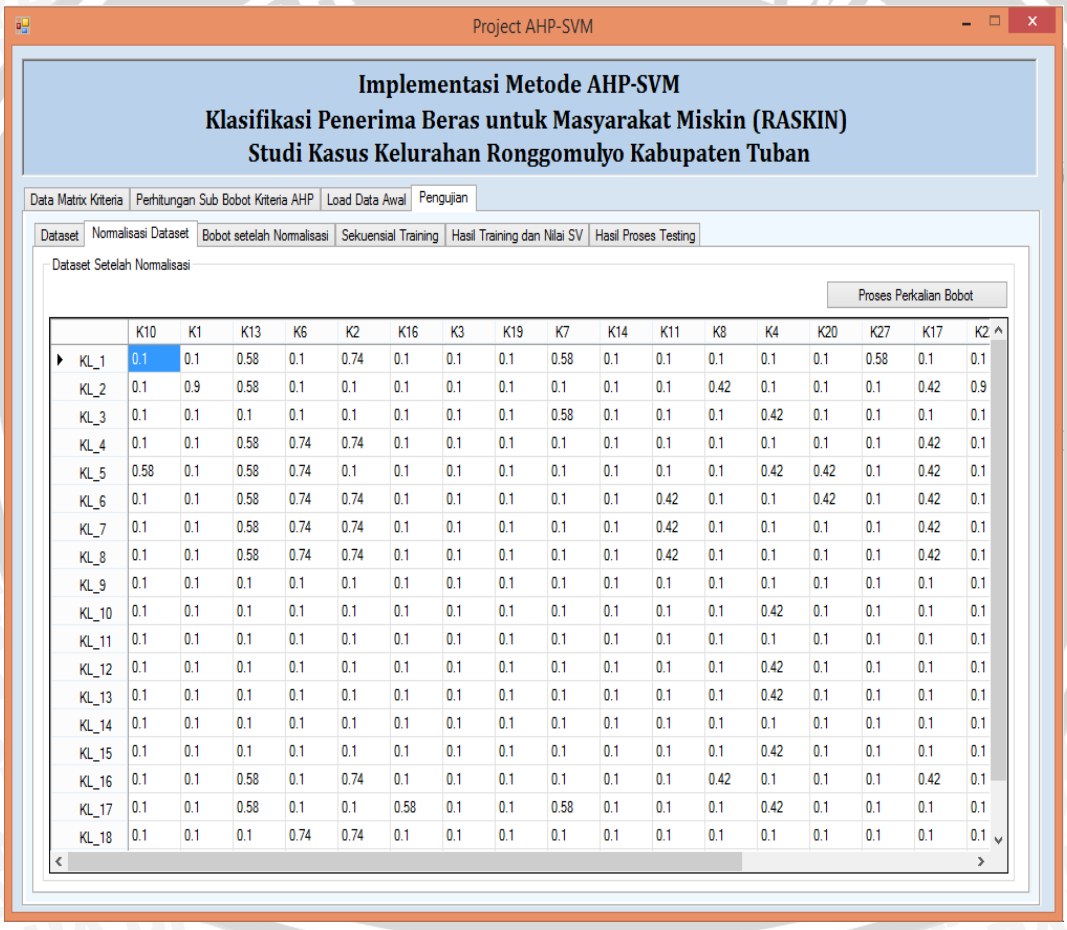
Antarmuka (*interface*) proses *load* dataset adalah antarmuka yang menampilkan dataset yang akan dipilih berdasarkan rasio perbandingan. Jumlah data yang digunakan disesuaikan berdasarkan rasio perbandingan antara data *training* dan data *testing*. Rasio perbandingan yang dapat digunakan adalah rasio perbandingan mulai dari perbandingan 90%:10%, 80%:20%, 70%:30%, 60%:40%, dan 50%:50%. *Button load* akan menampilkan data *training* dan data *testing* berdasarkan rasio perbandingan yang telah dipilih sebelumnya. Jumlah data *training* dan data *testing* akan diketahui dari label yang telah tersedia. data *training* dan data *testing*. *Button* proses Normalisasi akan menunjukkan proses normalisasi yang akan ditunjukkan pada *tab* selanjutnya. Gambar 4.4 menunjukkan antarmuka data awal berdasarkan rasio perbandingan yang dipilih.



Gambar 4.4 Antarmuka *Load* Data berdasarkan rasio perbandingan

4.4.5 Antarmuka Normalisasi *Dataset*

Antarmuka (*interface*) proses normalisasi *dataset* adalah antarmuka yang menampilkan dataset yang telah terpilih dan dinormalisasi berdasarkan rasio perbandingan. Data yang dinormalisasi adalah data data *training* dan data *testing*. Normalisasi dilakukan pada nilai setiap atribut data dengan skala data input sesuai dengan *range* fungsi penilaian yang digunakan. *Button* proses perkalian bobot akan menunjukkan hasil data ternormalisasi yang dikalikan dengan data bobot kriteria terpilih dan hasilnya ditunjukkan pada *tab* selanjutnya. Gambar 4.5 menunjukkan antarmuka normalisasi *dataset*.

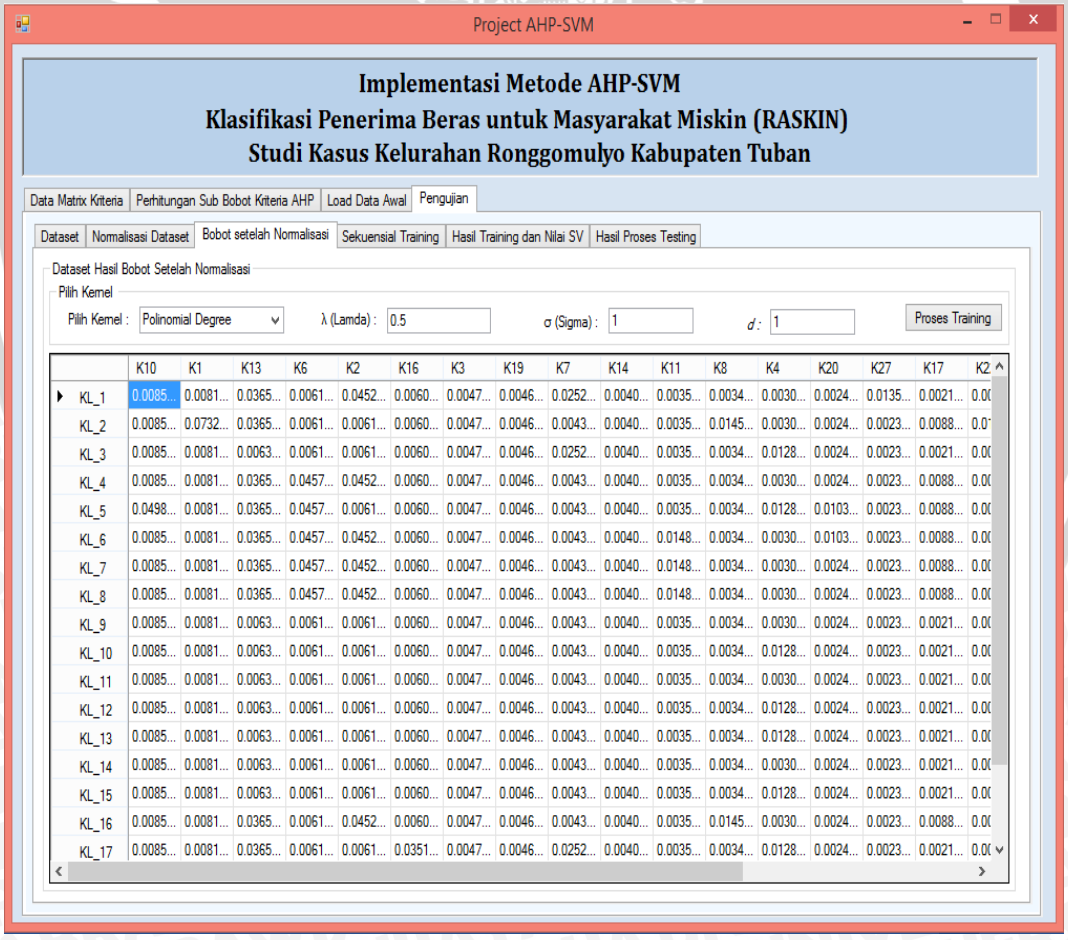


	K10	K1	K13	K6	K2	K16	K3	K19	K7	K14	K11	K8	K4	K20	K27	K17	K2
KL_1	0.1	0.1	0.58	0.1	0.74	0.1	0.1	0.1	0.58	0.1	0.1	0.1	0.1	0.1	0.58	0.1	0.1
KL_2	0.1	0.9	0.58	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.42	0.9
KL_3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.58	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_4	0.1	0.1	0.58	0.74	0.74	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1
KL_5	0.58	0.1	0.58	0.74	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.42	0.1	0.42	0.1
KL_6	0.1	0.1	0.58	0.74	0.74	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.42	0.1	0.42	0.1
KL_7	0.1	0.1	0.58	0.74	0.74	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1	0.42	0.1
KL_8	0.1	0.1	0.58	0.74	0.74	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1	0.42	0.1
KL_9	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
KL_10	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_11	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
KL_12	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_13	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_14	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
KL_15	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_16	0.1	0.1	0.58	0.1	0.74	0.1	0.1	0.1	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.42	0.1
KL_17	0.1	0.1	0.58	0.1	0.1	0.58	0.1	0.1	0.58	0.1	0.1	0.1	0.42	0.1	0.1	0.1	0.1
KL_18	0.1	0.1	0.1	0.74	0.74	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Gambar 4.5 Antarmuka Normalisasi *Dataset*

4.4.6 Antarmuka Perkalian Bobot setelah Normalisasi

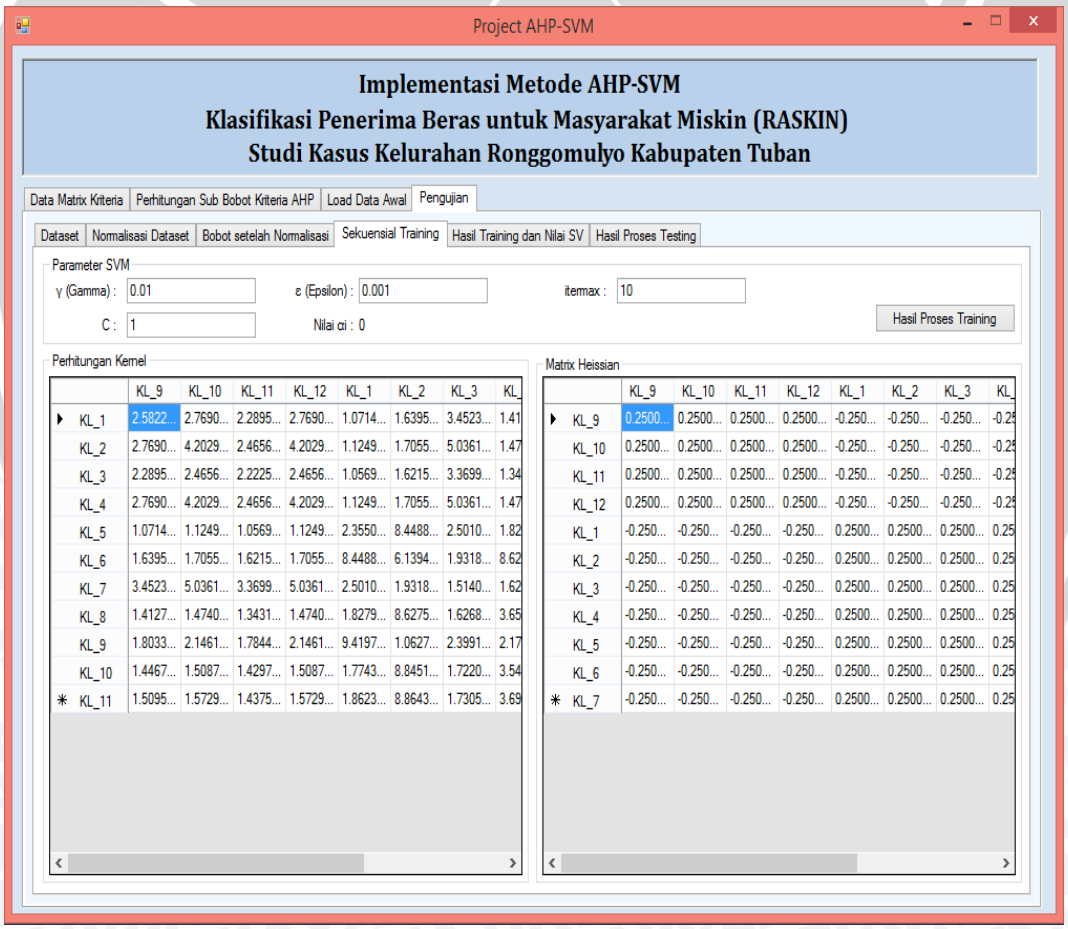
Antarmuka (*interface*) merupakan data hasil perkalian bobot hasil perkalian data ternormalisasi dengan nilai bobot kriteria. Pada bagian ini terdapat beberapa *textboxt* yang digunakan untuk menginputkan nilai yang akan digunakan sebagai parameter dalam proses perhitungan *kernel* SVM. *Kernel* yang akan digunakan dalam proses perngujian adalah *kernel polynomial* dan *kernel Gaussian RBF*. Nilai yang dapat diinputkan antara lain adalah nilai λ (*lamda*), σ (*sigma*), dan *d* (pangkat) adalah pangkat yang digunakan pada *kernel polynomial*. *Button* proses *training* menunjukkan proses perhitungan metode *sequential training* setelah memilih *kernel* yang akan digunakan dan semua parameter *kernel* telah diinput nilai. Proses *sequential training* akan ditunjukkan pada *tab* selanjutnya. Gambar 4.6 menunjukkan antarmuka bobot setelah normalisasi.



Gambar 4.6 Antarmuka Bobot Setelah Normalisasi

4.4.7 Antarmuka Proses *Sequential Training*

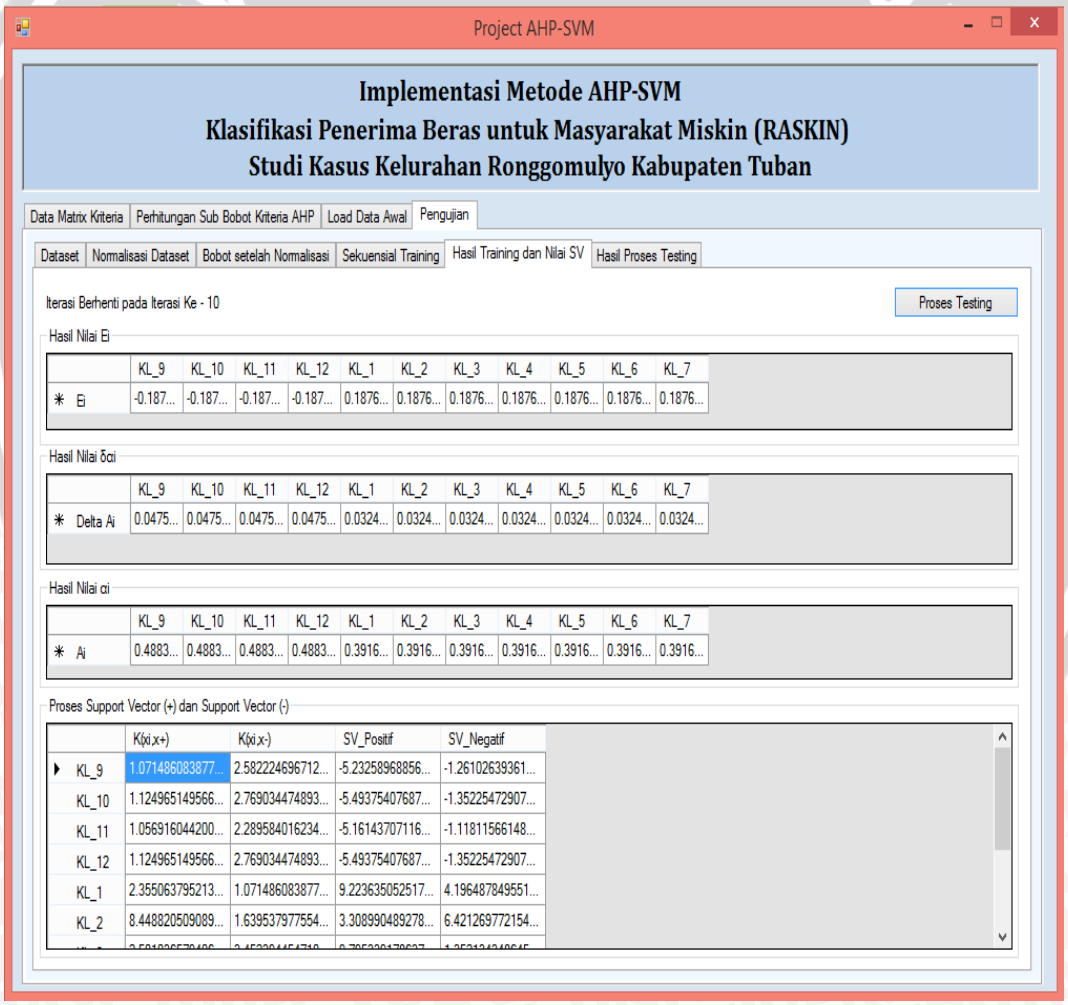
Antarmuka (*interface*) merupakan data hasil perhitungan *kernel* dan matriks hessian. Pada bagian ini terdapat beberapa *textboxt* yang digunakan untuk menginputkan nilai yang akan digunakan sebagai parameter dalam proses perhitungan *sequential training* SVM. Nilai parameter yang dapat diinputkan antara lain adalah nilai γ (*gamma*), C (*Complexity*), ε (*Epsilon*), dan *IterMax* (Banyaknya Iterasi Maximum). *Button* hasil *training* menunjukkan hasil dari proses perhitungan metode *sequential training* setelah tahapan pemilihan *kernel*, perhitungan matriks hessian dan input nilai parameter *sequential*. Proses *sequential training* akan ditunjukkan pada *tab* selanjutnya. Gambar 4.7 menunjukkan antarmuka proses *sequential training*.



Gambar 4.7 Antarmuka Proses *Sequential Training*

4.4.8 Antarmuka Hasil *Training* dan Nilai *Support Vector*

Antarmuka (*interface*) merupakan data hasil perhitungan *sequential training*. Pada bagian ini merupakan proses iterasi pada metode *sequential training*. Terdapat label yang menunjukkan proses iterasi akan berhenti pada iterasi ke berapa dan ditunjukkan dengan hasil nilai E_i, $\delta\alpha_i$, dan nilai α_i baru dari iterasi tersebut. Setelah diketahui nilai α_i baru, proses selanjutnya adalah proses *testing* untuk menghitung nilai *support verctor positive* dan *support verctor negative.* Proses ini dilakukan untuk mendapatkan nilai *b* (bias) yang akan digunakan pada proses *testing*. *Button* proses *testing* menunjukkan proses dan hasil klasifikasi data *testing.* Gambar 4.8 menunjukkan antarmuka proses *sequential training*.



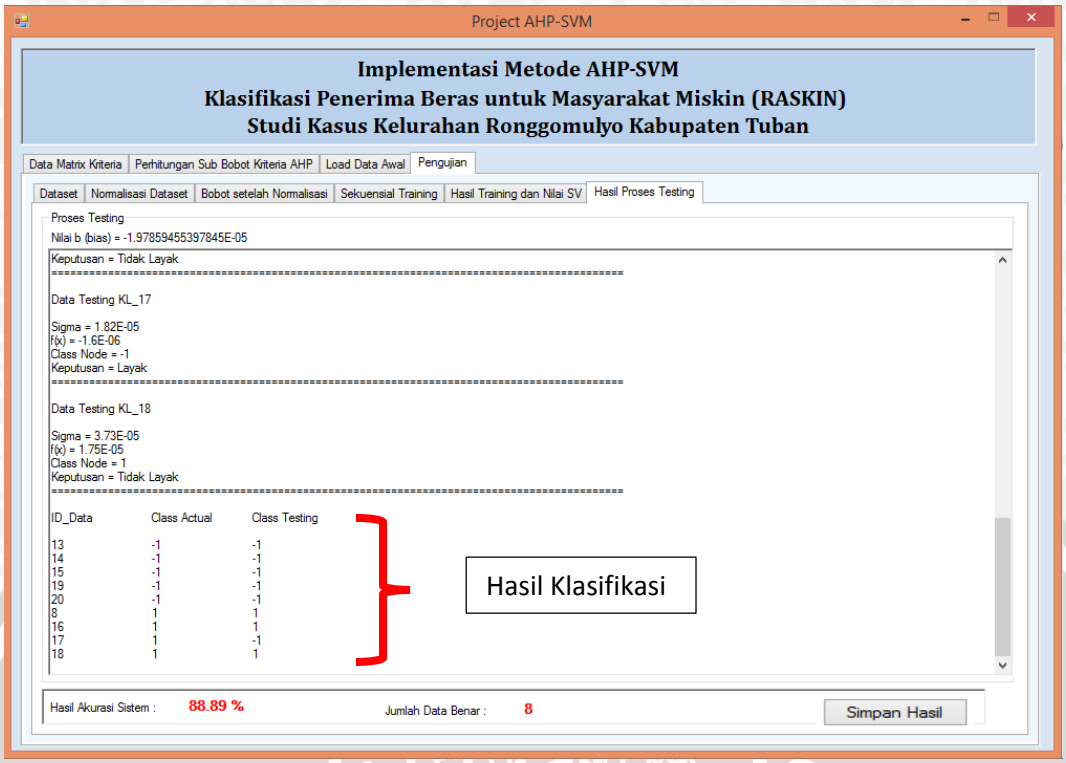
Gambar 4.8 Antarmuka Proses *Sequential Training*

4.4.9 Antarmuka Hasil Proses *Testing*

Antarmuka (*interface*) merupakan data hasil proses *testing* dan akurasi sistem. Pada bagian ini menunjukkan hasil perhitungan data *testing* untuk mendapatkan nilai *f(x)* dan hasil *predicted class* atau klasifikasi *class* dari perhitungan sign *f(x)*. *Class* -1 diklasifikasikan sebagai masyarakat yang layak menerima RASKIN. Sedangkan *class* 1 diklasifikasikan sebagai masyarakat yang tidak layak menerima RASKIN.

Pada bagian ini terdapat hasil akurasi sistem yang menunjukkan ketepatan hasil antara data *class* asli dan data *class* hasil klasifikasi sistem. Selain itu terdapat label yang menunjukkan jumlah data benar antara *class* asli dan hasil *class* klasifikasi dari keseluruhan data *testing* yang digunakan. Hasil proses *testing* dan hasil *class* klasifikasi data dari proses *testing* ditunjukkan pada Gambar 4.9 dan Gambar 4.10.

Gambar 4.9 Antarmuka Hasil Proses *Testing*



Gambar 4.10 Antarmuka Hasil Klasifikasi Proses *Testing*