BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi perangkat lunak berdasarkan perancangan yang telah dijelaskan pada bab sebelumnya.

4.1. Lingkungan Implementasi

Lingkungan yang dijelaskan pada sub bab ini adalah lingkungan implementasi dari perangkat keras dan perangkat lunak yang digunakan pada implementasi algoritma SVM untuk penentuan potensi bencana tsunami.

4.1.1. Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan implementasi algoritma SVM untuk penentuan potensi bencana tsunami ini, antara lain:

1. Processor Intel Core i3-2310M CPU 2.10GHz
2. Memory 2048 MB
3. Harddisk 500 GB
4. Graphics Intel HD 3000

4.1.2. Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan implementasi algoritma SVM untuk penentuan potensi bencana tsunami ini, antara lain:

1. Sistem Operasi Windows 7 Home Premium
2. Notepad++
3. XAMPP
4. Web Browser Google Chrome
5. Microsoft Office Excel 2007

4.2. Implementasi Aplikasi

Pada sub bab ini menjelaskan tentang implementasi program dalam pembuatan aplikasi untuk implementasi algoritma SVM untuk penentuan potensi bencana tsunami dengan menggunakan bahasa PHP. Pada aplikasi yang dibuat

memiliki tiga proses utama, yaitu proses normalisasi data, proses *training*, dan proses *testing*.

4.3. Proses Normalisasi Data

Pada proses ini dilakukan normalisasi pada setiap fitur. Metode yang digunakan untuk normalisasi yaitu metode *Min-Max Normalization*. Pada program yang dibuat terdapat fungsi yang digunakan untuk melakukan normalisasi data. Fungsi yang digunakan adalah fungsi "normalisasi". Pada fungsi tersebut diinisialisasi batas bawah dan batas atas, di mana nilai batas bawahnya adalah 0 dan batas atasnya adalah 1. Implementasi normalisasi data ditunjukkan pada *source code* 4.1.

```
1  function normalisasi($isi_data){
2        $bb = 0;    $ba = 1;
3        for($i=0; $i<count($isi_data); $i++){
4              $max[$i] = max($isi_data[$i]);
5              $min[$i] = min($isi_data[$i]);
6        }
7        for($j=0; $j<count($isi_data); $j++){
8              for($k=0; $k<count($isi_data[$j]); $k++){
9                    $normalisasi[$j][$k] = ($isi_data[$j][$k] -
10 $min[$j]) / ($max[$j] - $min[$j]) * ($ba-$bb) - $bb;
11             }
12       }
13       return $normalisasi;
14 }
```

Source Code 4.1 Implementasi Perhitungan Normalisasi Data

4.4. Proses *Training*

Pada proses *training* ini metode yang digunakan adalah metode *Sequential Training* SVM. Nilai-nilai yang harus didapatkan dalam proses training ini adalah nilai kernel, nilai matriks hessian, nilai α, dan nilai b. *Source code* 4.2 menunjukkan implementasi dari perhitungan *kernel Polynomial Degree*. Fungsi yang digunakan dalam perhitungan *kernel* ini bernama "kernelPoly".

```
1  function kernelPoly($x, $depth, $mw, $ms, $mb, $mmi, $d,
2  $jmlh_baris_dLatih){
```

```
3       for($i=0; $i<$jmlh_baris_dLatih; $i++){
4             $krnl[$i]                                    =
5 pow(($depth[$x]*$depth[$i])+($mw[$x]*$mw[$i])+($ms[$x]*$ms[$
6 i])+($mb[$x]*$mb[$i])+($mmi[$x]*$mmi[$i]),$d);
7       }
8       return $krnl;
9 }
10
11 function kernelRbf($x, $depth, $mw, $ms, $mb, $mmi, $sigma,
12 $jmlh_baris_dLatih){
13       for($i=0; $i<$jmlh_baris_dLatih; $i++){
14             $krnl[$i]      =      exp((-1)*((pow(($depth[$x]-
15 $depth[$i]),2)+pow(($mw[$x]-$mw[$i]),2)+pow(($ms[$x]-
16 $ms[$i]),2)+pow(($mb[$x]-$mb[$i]),2)+pow(($mmi[$x]-
17 $mmi[$i]),2))/(2*pow($sigma,2))));
18       }
19       return $krnl;
20 }
21
22 for($i=0; $i<$jmlh_baris_dLatih; $i++){
23       if($model_kernel == 'poly'){
24             $kernel[$i] = kernelPoly($i, $depth, $mw, $ms,
25 $mb, $mmi, $d, $jmlh_baris_dLatih);
26       }
27       else{
28             $kernel[$i] = kernelRbf($i, $depth, $mw, $ms,
29 $mb, $mmi, $sigma, $jmlh_baris_dLatih);
30       }
31 }
```

Source Code 4.2 Implementasi Perhitungan *Kernel Polynomial Degree*

Source code 4.3 menunjukkan implementasi dari perhitungan matriks hessian. Fungsi yang digunakan dalam perhitungan matriks hessian ini bernama “hessian”.

```
function hessian($kelas, $kernel, $lamda){
      for($i=0; $i<count($kelas); $i++){
            for($j=0; $j<count($kelas); $j++){
                  $hessian[$i][$j] =
$kelas[$i]*$kelas[$j]*($kernel[$i][$j]+(pow($lamda,2)));
            }
      }
      return $hessian;
}

$hessian = hessian($kelas, $kernel, $lamda);
```

Source Code 4.3 Implementasi Perhitungan Matriks Hessian

Source code 4.4 menunjukkan implementasi dari perhitungan nilai b.

```
      // mencari letak array kelas positif pertama pada alfa
terkhir
      for($i=0; $i<$jmlh_baris_dLatih; $i++){
      if($kelas[$i] > 0){
            $positif_awal = $i;
            break;
      }
      }

      for($i=0; $i<$jmlh_baris_dLatih; $i++){
      if($kelas[$i] < 0){
            $negatif_awal = $i;
            break;
      }
      }

      // mencari letak array nilai terbesar pada pada alfa
terkhir
      $max_kelas_positif = $alfa[$positif_awal];
      for($i=0; $i<$jmlh_baris_dLatih; $i++){
      if($kelas[$i] > 0 && $alfa[$i] > $max_kelas_positif){
            $max_kelas_positif = $alfa[$i];
            $positif = $i;
      }
```

```
25        }
26
27        $max_kelas_negatif = $alfa[$negatif_awal];
28        for($i=0; $i<$jmlh_baris_dLatih; $i++){
29        if($kelas[$i] < 0 && $alfa[$i] > $max_kelas_negatif){
30              $max_kelas_negatif = $alfa[$i];
31              $negatif = $i;
32        }
33        }
34
35  if($model_kernel == 'poly'){
36        $k_positif =
37  kernelPoly($positif,$depth,$mw,$ms,$mb,$mmi,$d,$jmlh_baris_d
38  Latih);
39        $k_negatif =
40  kernelPoly($negatif,$depth,$mw,$ms,$mb,$mmi,$d,$jmlh_baris_d
41  Latih);
42  }
43  else{
44        $k_positif =
45  kernelRbf($positif,$depth,$mw,$ms,$mb,$mmi,$sigma,$jmlh_bari
46  s_dLatih);
47        $k_negatif =
48  kernelRbf($negatif,$depth,$mw,$ms,$mb,$mmi,$sigma,$jmlh_bari
49  s_dLatih);
50  }
51
52        function sigmaayk($alfa, $kelas, $jenis_kelas,
53  $jmlh_baris_dLatih){
54        $ayk = 0;
55        for($i=0; $i<$jmlh_baris_dLatih; $i++){
56              $ayk = $ayk +
57  $alfa[$i]*$kelas[$i]*$jenis_kelas[$i];
58        }
          return $ayk;
          }

          $w_positif = sigmaayk($alfa, $kelas, $k_positif,
    $jmlh_baris_dLatih);
```

```
      $w_negatif = sigmaayk($alfa, $kelas, $k_negatif,
$jmlh_baris_dLatih);

      $b = -(0.5)*($w_positif+$w_negatif);
```

Source Code 4.4 Implementasi Perhitungan Nilai b

4.5. Proses *Testing*

Source code 4.5 menunjukkan implementasi dari perhitungan proses klasifikasi sesuai dengan nilai *f*(*x*).

```
for($i=$jmlh_baris_dLatih; $i<$banyak_data; $i++){
      if($model_kernel == 'poly'){
            $kernel_test[$i-$jmlh_baris_dLatih] =
kernelPoly($i, $depth, $mw, $ms, $mb, $mmi, $d,
$jmlh_baris_dLatih);
      }
      else{
            $kernel_test[$i-$jmlh_baris_dLatih] =
kernelRbf($i, $depth, $mw, $ms, $mb, $mmi, $sigma,
$jmlh_baris_dLatih);
      }
}
for($i=0; $i<count($kernel_test);$i++){
      $sigmaayk[$i] = sigmaayk($alfa, $kelas,
      $kernel_test[$i], $jmlh_baris_dLatih);
}

for($i=0; $i<count($sigmaayk); $i++){
      $fx[$i] = $sigmaayk[$i]+$b;

      if($fx[$i] >= 0){
            $kelas_baru[$i] = 1;
      }
      else{
            $kelas_baru[$i] = -1;
      }
}
```

Source Code 4.5 Implementasi Perhitungan Nilai *f(x)*

Source code 4.6 menunjukkan implementasi dari perhitungan akurasi.

```
1  $sama = 0; $beda = 0;
2  for($i=$jmlh_baris_dLatih; $i<$banyak_data; $i++){
3        if($kelas[$i]==$kelas_baru[$i-$jmlh_baris_dLatih]){
4            $sama++;
5        }
6        else {
7            $beda++;
8        }
9  }
10 $akurasi = ($sama/$jml_baris_dUji)*100;
```

Source Code 4.6 Implementasi Perhitungan Akurasi

4.6. Implementasi Antarmuka

Implementasi antarmuka program deteksi tsunami memiliki beberapa bagian utama, yaitu:

1. Antarmuka input data

   Antarmuka ini digunakan untuk memasukkan data. *Input* data bisa berupa data tunggal dan data banyak. *Input* data secara banyak dibagi menjadi 2 yaitu data secara keseluruhan di mana belum diketahui termasuk dalam data latih atau data uji dan data terpisah di mana data latih dan data uji dimasukkan secara terpisah.

2. Antarmuka proses *training* dan *testing* data dengan metode *Support Vector Machine*

   Antarmuka ini digunakan untuk memproses data dengan menggunakan metode *Support Vector Machine.*

3. Antarmuka daftar uji akurasi

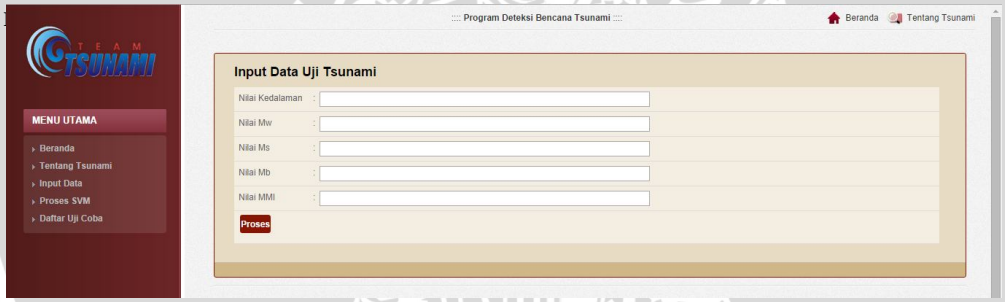
   Antarmuka ini bertujuan untuk menampilkan daftar akurasi dari uji coba yang pernah diuji sebelumnya. Dari daftar tabel tersebut dapat diketahui nilai yang digunakan pada tiap-tiap variabel pada algoritma *Support Vector Machine*, jumlah data latih dan jumlah data uji.

4.6.1. Antarmuka Input Data

Pada interface ini terdapat dua model data yang ingin diinputkan. Data yang inputkan bisa berupa data tunggal dan data banyak. Data tunggal digunakan untuk mengetahui kelas hasil klasifikasi dari nilai dari tiap-tiap parameter. *Interface* pemilihan data input tunggal ditunjukkan pada Gambar 4.1

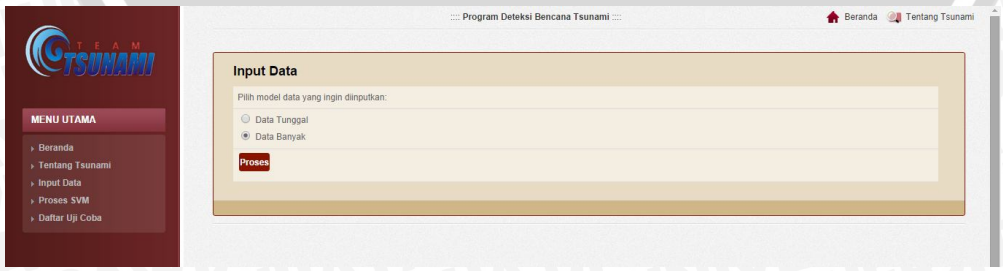
Gambar 4.1 *Interface* pemilihan data input tunggal

Setelah memilih model data tunggal, maka akan menuju pada interface input data tunggal yang ditunjukkan pada Gambar 4.2. Pada *interface* ini akan



Gambar 4.2 *Interface* input data tunggal

Pada input data terdapat pemilihan model data yang diinputkan dengan banyak data. Sehingga pengujian yang dilakukan dengan menggunakan banyak data. *Interface* pemilihan data input banyak ditunjukkan pada Gambar 4.3.

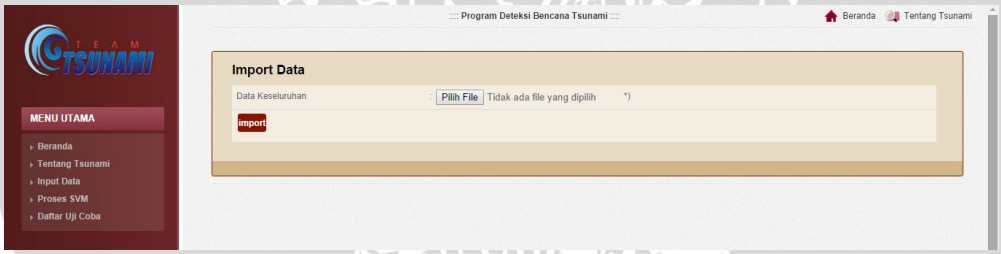


Gambar 4.3 *Interface* pemilihan data input banyak

Setelah memilih data input banyak, maka menuju interface *import* data. *Import* data digunakan untuk memasukkan data dengan jumlah banyak. Pada *import* data terdapat dua model. Data tersebut dimasukkan secara keseluruhan atau secara terpisah. *Interface* pemilihan *import* data secara keseluruhan ditunjukkan pada Gambar 4.4.

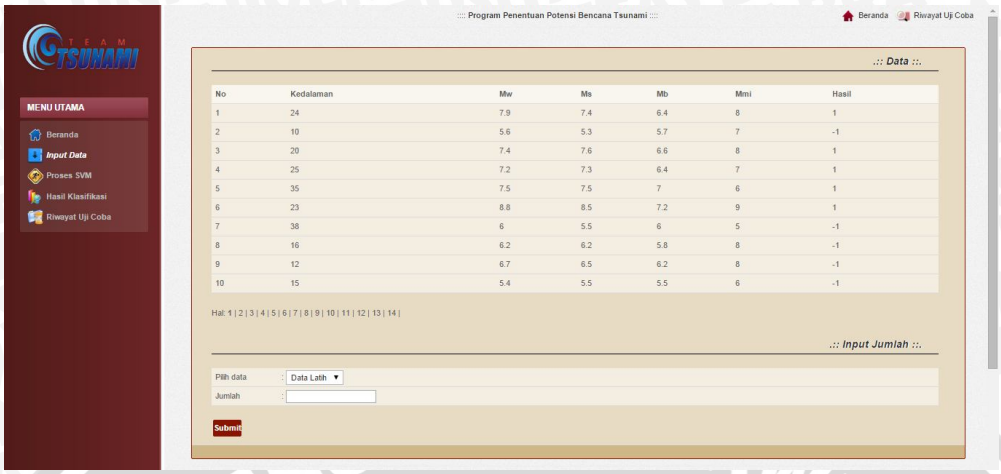
Gambar 4.4 *Interface* pemilihan *import* data secara keseluruhan

Setelah memilih model import data secara keseluruhan, maka munuju *Interface import* data secara keseluruhan. *Interface import* data secara keseluruhan ditunjukkan pada Gambar 4.5.



Gambar 4.5 *Interface import* data secara keseluruhan

Data keseluruhan tersebut akan dibagi ke dalam data latih dan data uji oleh *user* sesuai kebutuhan. *User* memilih salah satu opsi yaitu data latih dan data uji. Kemudian *user* memasukkan jumlah data yang digunakan. Selisih dari jumlah data secara keseluruhan dengan jumlah data opsi pilihan *user* akan dimasukkan ke dalam opsi selain pilihan *user*. Misalkan data secara keseluruhan berjumlah 132 dan *user* memilih data latih kemudian jumlah data yang dimasukkan 66, maka 66 data yang lainnya akan masuk ke dalam data uji. *Interface* pembagian keseluruhan data ke dalam data latih dan data uji ditunjukkan pada Gambar 4.6.



Gambar 4.6 *Interface* pembagian keseluruhan data ke dalam data latih dan data uji

Model *import* data yang lainnya yaitu *import* data secara terpisah. *Interface* pemilihan *import* data secara terpisah ditunjukkan pada Gambar 4.7.

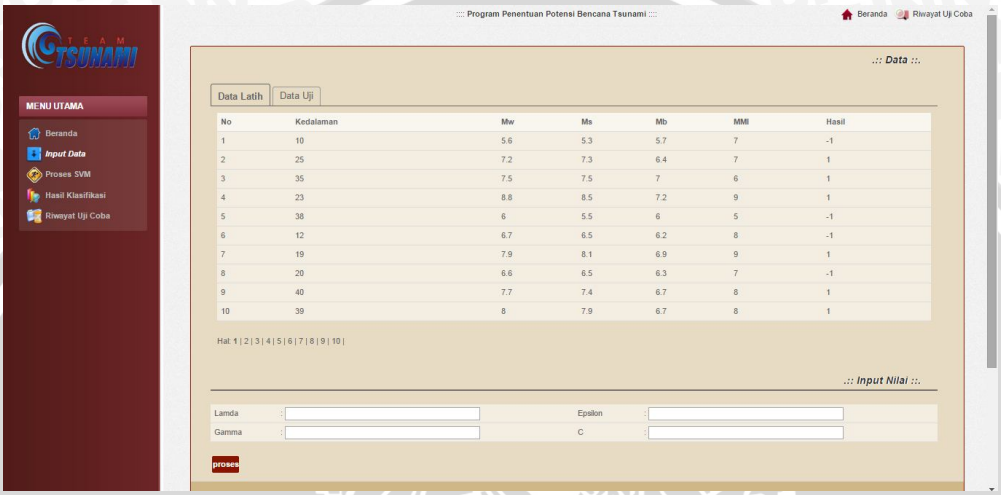
Gambar 4.7 *Interface* pemilihan *import* data secara terpisah

Kemudian menuju *Interface import* data secara terpisah. File data yang dimasukkan dapat dimasukkan secara terpisah. *Interface import* data secara terpisah ditunjukkan pada Gambar 4.8.

Gambar 4.8 *Interface import* data secara terpisah

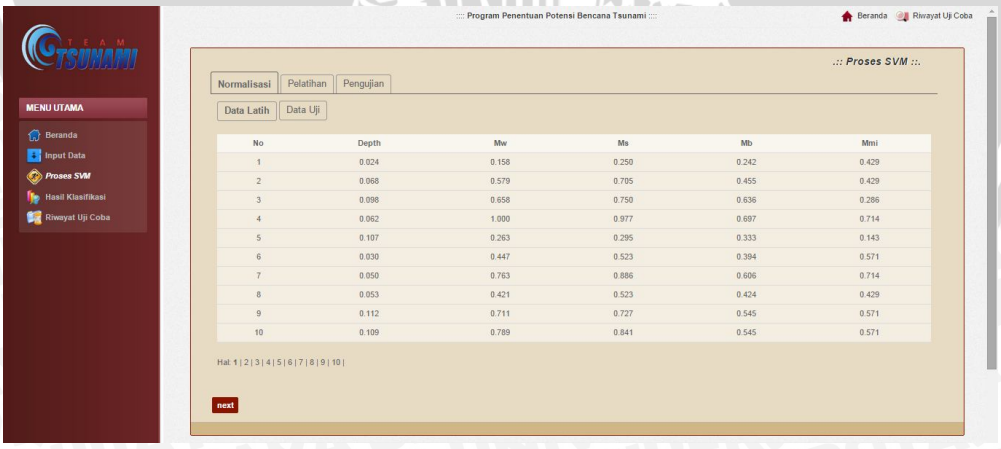
4.6.2. Antarmuka Proses SVM

Pada antarmuka ini digunakan untuk melakukan uji coba dengan memproses menggunakan metode SVM. Sebelum dilakukan proses SVM, *user* harus memasukkan nilai dari tiap variabel SVM. *Interface input* nilai variabel metode SVM ditunjukkan pada Gambar 4.9.



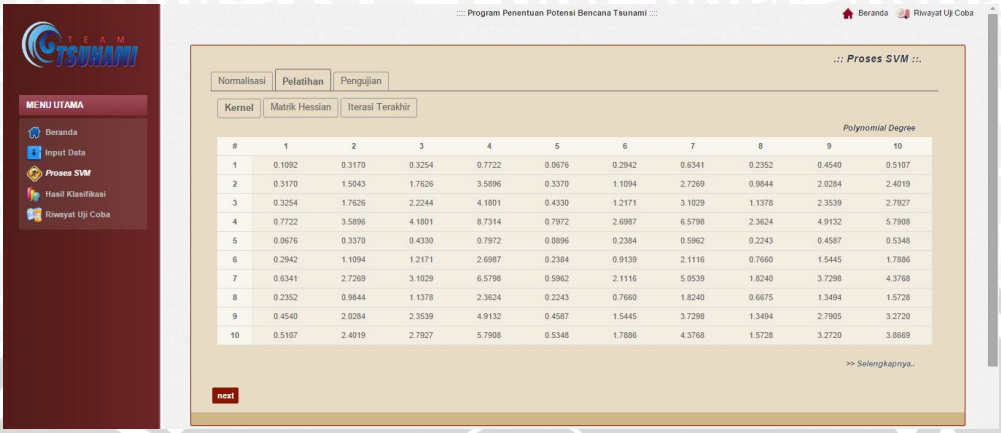
Gambar 4.9 *Interface input* nilai variabel metode SVM

Setelah dilakukan klik pada tombol proses, maka dilakukan proses SVM. Sebelum menuju proses *training* dan *testing*, dilakukan proses normalisasi. Hasil dari proses normalisasi ditunjukkan pada Gambar 4.10.



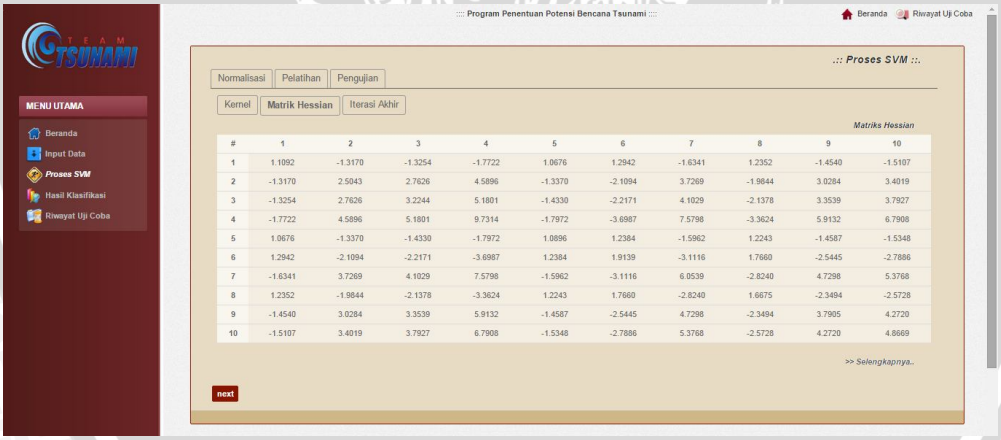
Gambar 4.10 *Interface* normalisasi data latih dan data uji

Pada *training* SVM ditunjukkan hasil perhitungan kernel *polynomial degree*, matrik hessian, dan nilai Ei, δαi, αi pada iterasi terakhir. Gambar 4.11 *Interface* menunjukkan nilai kernel *polynomial degree.*



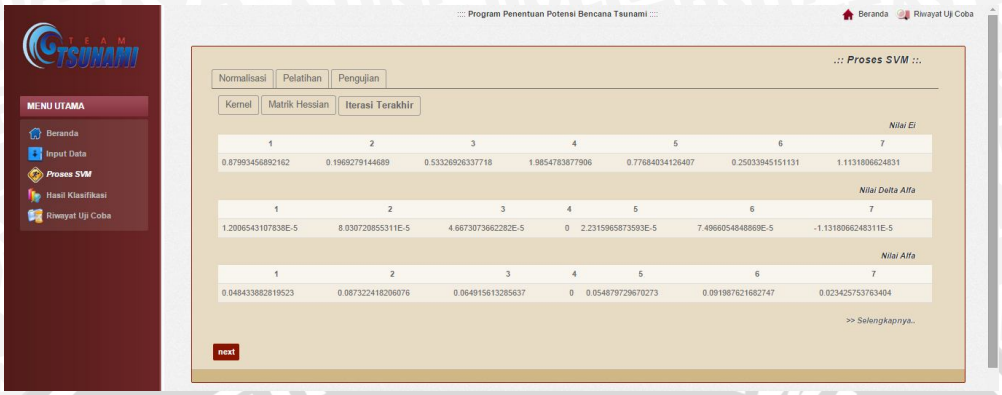
Gambar 4.11 *Interface* nilai kernel *polynomial degree*

Pada Gambar 4.12 menunjukkan *Interface* hasil perhitungan nilai matriks hessian.



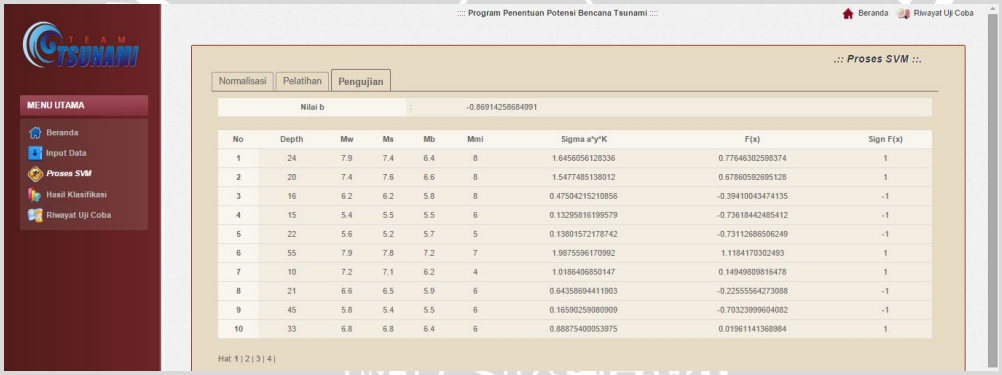
Gambar 4.12 *Interface* nilai matriks hessian

Pada Gambar 4.13 menunjukkan *Interface* hasil perhitungan nilai Ei, δαi, αi pada iterasi terakhir.



Gambar 4.13 *Interface* nilai Ei, δi, αi

Pada proses *testing* dapat ditunjukkan pada Gambar 4.14.



Gambar 4.14 *Interface* hasil pengujian

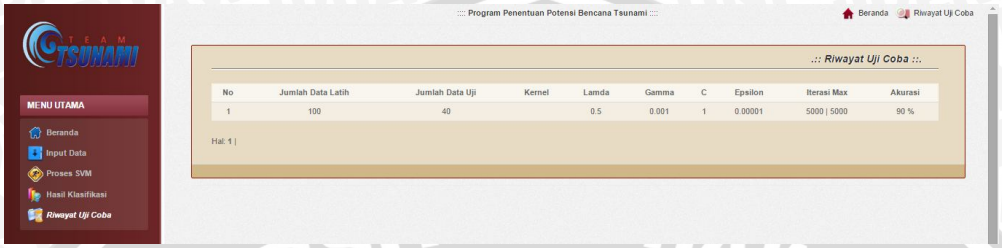
4.6.3. Antarmuka Riwayat Hasil Klasifikasi

Antarmuka ini menunjukkan hasil dari proses klasifikasi yang membandingkan antar kelas hasil klasifikasi dan kelas *actual*-nya.

Gambar 4.15 *Interface* nilai hasil klasifikasi

4.6.4. Antarmuka Riwayat Uji Coba

Antarmuka ini digunakan untuk menyimpan hasil uji coba. Sehingga uji coba yang pernah dilakukan *user* dapat dilihat pada antarmuka ini. Antarmuka daftar uji coba ditunjukkan pada Gambar 4.16.



Gambar 4.16 *Interface* riwayat uji coba