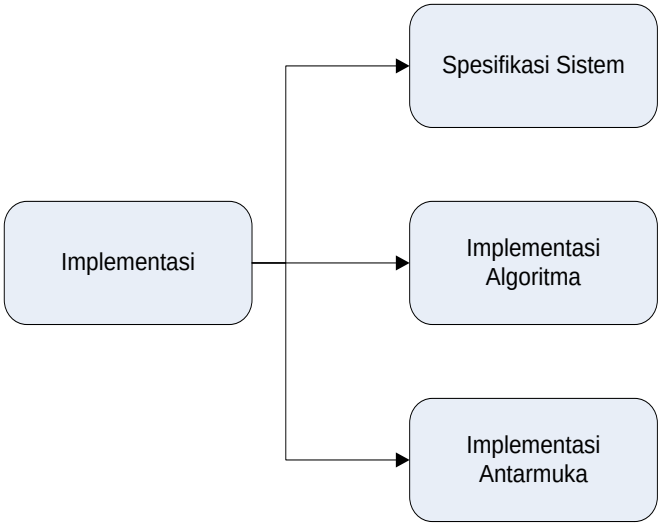
BAB V
IMPLEMENTASI

Bab ini menjelaskan tentang implementasi sistem berdasarkan proses perancangan. Diagram implementasi sebagai gambaran umum pokok bahasan pada bab 5 ditunjukkan pada Gambar (5.1).



Gambar 5.1 Diagram Implementasi
Sumber : Implementasi

Implementasi terdiri dari spesifikasi sistem, implementasi algoritma, dan implementasi antarmuka. Implementasi sistem pengelompokan penyakit kanker payudara ini dibangun mengacu pada pemodelan menggunakan metode *fuzzy c-means clustering* dan metode *fuzzy inference system* sugeno orde-satu dan sesuai dengan model perancangan.

5.1 Spesifikasi Sistem

Spesifikasi sistem yang dibahas meliputi spesifikasi perangkat keras dan perangkat lunak yang digunakan untuk implementasi. Spesifikasi dibahas secara detail agar implementasi berjalan sesuai dengan perancangan.

5.1.1 Spesifikasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebuah *notebook* dengan spesifikasi sebagai berikut:

1. *Prosesor* Intel(R) Core(TM)2 Duo CPU T8300 @2.40GHz

2. *Memori* 3.00 Gb RAM
3. *Harddisk* 250 GB
4. *Monitor* 14”
5. *Mouse*

5.1.2 Spesifikasi Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem diantaranya adalah sebagai berikut:

1. Sistem operasi Windows XP professional Version 2002
2. Java(TM) SE Development Kit 6 Update 16
3. NetBeans IDE 6.9.1
4. XAMPP 1.7.1

5.2 Implementasi Algoritma

Berdasarkan analisa dan perancangan sistem yang telah dibangun pada bab 4, maka pada bab implementasi ini akan dijelaskan bagaimana menerapkan hasil perancangan ke dalam sebuah program yang di dalamnya berisikan *source code* yang membentuk sistem. Algoritma diimplementasikan ke dalam sebuah *package* yang didalamnya berisi kelas. Secara umum kelas yang dibentuk merupakan kelas untuk mengimplementasikan tampilan antarmuka sistem dan kelas untuk mengoperasikan proses jalannya sistem.

5.2.1 Implementasi kelas FCM

Kelas FCM merupakan kelas yang berisi proses pembelajaran data latih menggunakan *fuzzy c-means clustering.* Proses FCM *clustering* diterapkan pada kelas `FCM.java` berdasarkan diagram alir pada Gambar (4.3), dimana pada kelas ini terdapat 7 proses utama yang diterapkan pada beberapa *method*. Penjelasan *method* pembentuk kelas `FCM.java` dijelaskan pada Tabel (5.1).

Tabel 5.1 *Method – method* pembentuk kelas `FCM.java`

Nama *method*	Deskripsi
`BacaDataLatih()`	*Method* yang digunakan untuk membaca dan menampung data

	latih dari *database*.
`matriksPartisiAwal`	*Method* yang digunakan untuk membentuk matriks partisi awal
`cariPusatCluster (double[][] matriksU, double[][] matriksData)`	*Method* yang digunakan untuk menghitung nilai pusat cluster
`hitungFungsiObyektif(double[][] matriksAwal, double[][] matriksV, double[][] matriksX)`	*Method* yang digunakan untuk menghitung nilai fungsi objektif
`hitungPerubahanMatriksPartisi(double[][] matriksAwal, double[][] matriksData, double[][] matriksV)`	*Method* yang digunakan untuk menghitung perubahan matriks partisi. Matriks partisi selalu diperbaiki sampai syarat kondisi berhenti terpenuhi
`isBerhenti(double Pt, double PtMin1, int iterasiKe)`	*Method* yang digunakan untuk mengecek syarat kondisi berhenti
`tentukanIdxCluster(double[][] matriksU)`	*Method* yang digunakan untuk menentukan id cluster dari data mamografi yang digunakan untuk pelatihan.

Sumber : Implementasi

1. Implementasi Baca Data Latih

Proses pertama pada *FCM clustering* adalah membaca data latih sebanyak *n* buah data. Proses masukan data latih ini diterapkan kedalam sebuah *method* `BacaDataLatih()`. Untuk dapat membaca data latih yang telah disimpan sebelumnya pada sebuah *database*, maka diperlukan sebuah *method* `KoneksiDB()` untuk mengoneksikan antara program dengan *database*. Penerapan untuk proses baca data latih ditunjukkan pada Gambar (5.2).

```
private static void BacaDataLatih(){
 KoneksiDB();
 try {
     String sql="";
     ResultSet rs=null;

     double [][] matrixGanas = new double[250][jumlahAtribut];
```

```
double [][] matrixJinak = new double[250][jumlahAtribut];

int [] idMatrixDataGanas = new int[250];
int [] idMatrixDataJinak = new int[250];

sql = "select * from datalatih where severity='1'";
rs = statement.executeQuery(sql);
int counter=0;
while(rs.next()&&counter<250){
      matrixGanas[counter][0]=rs.getDouble("birads");
      matrixGanas[counter][1]=rs.getDouble("age");
      matrixGanas[counter][2]=rs.getDouble("shape");
      matrixGanas[counter][3]=rs.getDouble("margin");
      matrixGanas[counter][4]=rs.getDouble("density");
      matrixGanas[counter][5]=rs.getDouble("severity");
      idMatrixDataGanas[counter]=rs.getInt("id");
      counter++;
 }
 sql = "select * from datalatih where severity='0'";
 rs = statement.executeQuery(sql);
 counter=0;
 while(rs.next()&&counter<250){
      matrixJinak[counter][0]=rs.getDouble("birads");
      matrixJinak[counter][1]=rs.getDouble("age");
      matrixJinak[counter][2]=rs.getDouble("shape");
      matrixJinak[counter][3]=rs.getDouble("margin");
      matrixJinak[counter][4]=rs.getDouble("density");
      matrixJinak[counter][5]=rs.getDouble("severity");
      idMatrixDataJinak[counter]=rs.getInt("id");
      counter++;
  }
  rs.close();
   statement.close();

 int jumlahDataGanas = (jumlahData * PersenGanas) / 100;
 int jumlahDataJinak = (jumlahData * PersenJinak) / 100;

 matriksData = new double[jumlahData][jumlahAtribut];
 idMatrixData = new int[jumlahData];

 for(int i=0; i<jumlahData; i++){
     numbers.add(i);
 }
 int index;
 int counterDataLatih=0;
 for(int i=0; i<jumlahDataGanas; i++){
 index = (Integer)numbers.get(i);
 matriksData[counterDataLatih][1]=matrixGanas[index][0];
 matriksData[counterDataLatih][1]=matrixGanas[index][1];
 matriksData[counterDataLatih][1]=matrixGanas[index][2];
```

```
        matriksData[counterDataLatih][1]=matrixGanas[index][3];
        matriksData[counterDataLatih][1]=matrixGanas[index][4];
        matriksData[counterDataLatih][1]=matrixGanas[index][5];
        idMatrixData[counterDataLatih]=idMatrixDataGanas[index];
        counterDataLatih++;
        }

        for(int j=0; j<jumlahDataJinak; j++){
        index = (Integer)numbers.get(j);
        matriksData[counterDataLatih][1]=matrixJinak[index][0];
        matriksData[counterDataLatih][1]=matrixJinak [index][1];
        matriksData[counterDataLatih][1]=matrixJinak [index][2];
        matriksData[counterDataLatih][1]=matrixJinak  [index][3];
        matriksData[counterDataLatih][1]=matrixJinak [index][4];
        matriksData[counterDataLatih][1]=matrixJinak [index][5];
        idMatrixData[counterDataLatih]=idMatrixDataJinak[index];
        counterDataLatih++;
        }
        }
        catch(Exception e){
        JOptionPane.showMessageDialog(null, e, "Error Baca Data
Latih", JOptionPane.ERROR_MESSAGE);
            }
}
```

Gambar 5.2 Implementasi Baca Data Latih
Sumber : Implementasi

2. Implementasi Pembentukan Matriks Partisi Awal

Proses pembentukan matriks awal dimulai dengan pembangkitan bilangan random yang diikuti dengan penjumlahan elemen setiap kolom sejumlah data. Jumlah bilangan yang dibangkitkan yaitu jumlah data x jumlah *cluster*. Kemudian dilakukan perhitungan nilai elemen matriks dengan membagi nilai μik dengan Qi yang bersesuaian.

Kode program untuk proses pembentukan matriks partisi awal merujuk pada persamaan (2.1) dan (2.2). Implementasi untuk proses pembentukan matriks partisi awal adalah dengan membuat *method* `matriksPartisiAwal()` seperti yang terlihat pada Gambar (5.3).

```
public static void matriksPartisiAwal(){
matriksAwal=new double [jumlahData][jumlahCluster];
double []Qi= new double[jumlahData];
     for(int i=0; i<jumlahData; i++){
          for (byte k=0; k<jumlahCluster; k++){
               matriksAwal[i][k]=Math.random();
               Qi[i]+=matriksAwal[i][k];
```

```
            }
        }
        for(int i=0; i<jumlahData; i++){
             for (byte k=0; k<jumlahCluster; k++){
                  if(Qi[i]==0.0) matriksAwal[i][k]=0.0;
                  else matriksAwal[i][k]=matriksAwal[i][k]/Qi[i];
                  System.out.println(matriksAwal[i][k]);
              }
         }

}
```

Gambar 5.3 Implementasi Pembentukan Matriks Partisi Awal
Sumber : Implementasi

3. Implementasi Penentuan Pusat Cluster

Perhitungan pusat *cluster* dilakukan sesuai dengan persamaan (2.3) sehingga menghasilkan pusat *cluster* sejumlah *cluster* x jumlah atribut. Struktur penulisan kode program untuk implementasi penentuan pusat cluster ditunjukkan pada Gambar (5.4).

```
public static double[][] cariPusatCluster (double[][] matriksU,
double[][] matriksData){
double[][]matriksV = new double[jumlahCluster][jumlahAtribut];
     for (byte k=0; k<jumlahCluster; k++){
          for (byte j=0; j<jumlahAtribut; j++){
    double a=0, b=0;
       for (int i=0; i<jumlahData; i++){
           a +=Math.pow(matriksAwal[i][k], w)*matriksData[i][j];
           b +=Math.pow(matriksAwal[i][k], w);
       }
            if(b==0.0) matriksV[k][j]=0;
            else matriksV[k][j]=a/b;
            System.out.println(matriksV[k][j]);
     }
}
  return matriksV;
}
```

Gambar 5.4 Implementasi Penentuan Pusat *Cluster*
Sumber : Implementasi

4. Implementasi Perhitungan Fungsi Objektif

Proses perhitungan fungsi objektif akan menghasilkan *output* berupa nilai fungsi objektif iterasi ke-t (Pt). Perhitungan fungsi objektif dilakukan dengan menggunakan persamaan (2.4). Kode program implementasi perhitungan nilai

fungsi objektif pada metode *clustering* FCM ditunjukkan oleh kode program pada Gambar (5.5).

```
public     static     double     hitungFungsiObyektif(double[][]
matriksAwal, double[][] matriksV, double[][] matriksX){
double sigma1=0,sigma2=0,sigma3=0;
     for(int i=0;i<jumlahData;i++){
          for(int k=0;k<jumlahCluster;k++){
               for(int j=0;j<jumlahAtribut;j++){
                  sigma3=sigma3+Math.pow((matriksX[i][j]-
matriksV[k][j]),2);
                  sigma3=sigma3*Math.pow((matriksAwal[i][k]),w);
               }
               sigma2=sigma2+sigma3;
               sigma3=0;
           }
            sigma1=sigma1+sigma2;
            sigma2=0;
      }
      return sigma1;
}
```

Gambar 5.5 Implementasi Perhitungan Fungsi Objektif
Sumber : Implementasi

5. Implementasi Perhitungan Perubahan Matriks Partisi

Perhitungan perubahan matriks partisi dilakukan untuk memperbarui nilai matriks awal berdasarkan pusat *cluster* yang terbentuk sesuai dengan persamaan (2.5). Implementasi perhitungan perubahan matriks partisi ditunjukkan pada Gambar (5.6).

```
public   static   void   hitungPerubahanMatriksPartisi(double[][]
matriksAwal, double[][] matriksData, double[][] matriksV){
double a=0,b=0,sigma3=0;
for(int i=0;i<jumlahData;i++){
     for(int k=0;k<jumlahCluster;k++){
          for(int j=0;j<jumlahAtribut;j++)
             a=a+Math.pow((matriksData[i][j]-matriksV[k][j]),2);
             a=Math.pow(a,(-1/(w-1)));
               for(int l=0;l<jumlahCluster;l++){
                    for(int j=0;j<jumlahAtribut;j++)
                      sigma3=sigma3+Math.pow((matriksData[i][j]-
matriksV[l][j]),2);
                     b=b+Math.pow(sigma3,(-1/(w-1)));
                             sigma3=0;
                    }
                    matriksAwal[i][k]=a/b;
                   a=0;b=0;
      }
```

```
}
}
```

Gambar 5.6 Implementasi Perhitungan perubahan Matriks Partisi
Sumber : Implementasi

6. Implementasi Pengecekan Kondisi Berhenti

Proses pengecekan kondisi berhenti dilakukan dengan cara menghitung selisih Pt dengan nilai Pt-1 dihitung kemudian dibandingkan dengan nilai kesalahan minimum (ξ) yang telah ditentukan untuk pemeriksaan kondisi berhenti pada proses *clustering* data. Jika |Pt – Pt-1| < ξ, maka perulangan pada proses *clustering* dihentikan. Penerapan untuk proses pengecekan kondisi berhenti adalah dengan membuat *method* `isBerhenti(double Pt, double PtMin1, int iterasiKe)`. Implementasi pengecekan kondisi berhenti ditunjukkan pada Gambar (5.7).

```
public static boolean isBerhenti(double Pt, double PtMin1, int
iterasiKe){
     if(Math.abs(Pt-PtMin1)<error || iterasiKe>maxIterasi)
     return true;
          else
          return false;
}
```

Gambar 5.7 Implementasi Pengecekan Kondisi Berhenti
Sumber : Implementasi

7. Implementasi Pengelompokan Data

Proses pengelompokan data melibatkan *input* berupa data latih dan derajat keanggotaan. Proses ini menghasilkan *output* berupa kelompok data yang menyatakan *cluster* dari setiap data. Pada implementasinya, proses ini diterapkan kedalam sebuah *method* `tentukanIdxCluster(double[][] matriksU)`. Implementasi pengelompokan data dapat dilihat pada Gambar (5.8).

```
public static int[] tentukanIdxCluster(double[][] matriksU){
int[] idxCluster=new int[jumlahData];
     for(int i=0;i<jumlahData;i++){
          double max=0;
          int idxMax=0;
          for(int j=0;j<jumlahCluster;j++)
               if(matriksU[i][j]>max){
                  max=matriksU[i][j];
```

```
                idxMax=j;
            }
            idxCluster[i]=idxMax;
        }
        return idxCluster;
}
```

Gambar 5.8 Implementasi Pengelompokan Data
Sumber : Implementasi

5.2.2 Implementasi Kelas Varian

Kelas varian merupakan kelas untuk perhitungan nilai varian *cluster*. Perhitungan varian terbagi menjadi 3 jenis, yaitu proses perhitungan varian tiap *cluster*, proses perhitungan *varian within cluster*, dan proses perhitungan *varian between cluster*. Proses dari perhitungan varian adalah untuk mendapatkan nilai batasan varian dari hasil *cluster* yang terbentuk. Lewat nilai batasan varian inilah yang nantinya bisa dijadikan acuan untuk memilih hasil *cluster* mana yang efektif guna dijadikan bahan bagi proses ekstraksi aturan *fuzzy*. Pada implementasinya, kelas `varian.java` dibagi menjadi 4 *method* utama, yaitu `varianTiapCluster()`,`varianWtihinCluster()`, `varianBetweenCluster()`, dan `batasanVarian()` dan 6 method pendukung. Penjelasan *method* pembentuk kelas `varian.java` dijelaskan pada Tabel (5.2).

Tabel 5.2 *Method – method* pembentuk kelas `varian.java`

Nama *method*	Deskripsi
`jumlahDataTiapCluster(int indexCluster)`	*Method* yang digunakan untuk menghitung jumlah data pada tiap *cluster* yang terbentuk.
`rataRataAtribut(int indexCluster, int indexAtribut)`	Sebuah *method* yang digunakan menghitung nilai rata – rata pada tiap atribut data pada suatu *cluster* yang terbentuk.
`standarDeviasi()`	*Method* yang digunakan untuk menghitung nilai standar deviasi (sigma) dari data
`varianTiapAtribut(int x, int y)`	*Method* yang digunakan untuk mencari nilai varian tiap atribut pada suatu *cluster*.
`varianTiapCluster(int x)`	*Method* yang digunakan untuk proses perhitungan varian tiap *cluster*.

`varianWithinCluster()`	Sebuah *method* yang digunakan untuk proses perhitungan *variance within cluster*.
`varianAntarAtribut(int x, int y)`	Sebuah *method* yang digunakan untuk menghitung nilai varian antar atribut pada suatu *cluster*.
`varianAntarCluster(int x)`	*Method* untuk menghitung nilai varian antar *cluster*.
`varianBetweenCluster()`	Sebuah *method* yang digunakan untuk menghitung nilai *variance between cluster*.
`batasanVarian()`	*Method* yang diperuntukan untuk mengitung nilai batasan varian.

Sumber : Implementasi

1. Implementasi Proses Perhitungan Varian Tiap *Cluster*

Proses perhitungan varian tiap *cluster* diterapkan dengan membuat *method* `varianTiapCluster()`, dimana dalam *method* tersebut perlu untuk memanggil *method* tambahan lainnya untuk membantu proses perhitungan, diantaranya adalah method `jumlahDataTiapCluster()`, `rataRataAtribut()`, `standarDeviasi()` dan `varianTiapAtribut()`. Alur proses untuk *method* `varianTiapCluster()` mengacu pada Persamaan (2.8). Implementasi proses perhitungan varian tiap *cluster* ditunjukkan pada Gambar (5.9).

```
private static int jumlahDataTiapCluster(int indexCluster){
int jumlahDataCluster=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
     for(int i=0;i<sc.jumlahData;i++){
          if(idxHasilCluster[i]==indexCluster){
            jumlahDataCluster++;
          }
     }
     return jumlahDataCluster;
}
private static double rataRataAtribut(int indexCluster, int
indexAtribut){
double rataAtribut=0, totalData=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
int jumlahDataCluster=0;
     for(int i=0;i<sc.jumlahData;i++){
          if(idxHasilCluster[i]==indexCluster){
   totalData=totalData+sc.matriksData[i][indexAtribut];
   jumlahDataCluster++;
          }
```

```
      }
      rataAtribut=totalData/jumlahDataCluster;
      return rataAtribut;
}
public static void standarDeviasi(){
standarDeviasi              =              new              double
[sc.jumlahCluster][sc.jumlahAtribut];
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
      for(int i=0;i<sc.jumlahCluster;i++){
            for(int j=0;j<sc.jumlahAtribut;j++){
                double temp=0.0, temp1=0;
                for(int k=0;k<sc.jumlahData;k++){
                     if(idxHasilCluster[k]==i){
temp=temp+Math.pow((sc.matriksData[k][j]-
rataRataAtribut(i,j)),2);
                     }
                }
                     if ((jumlahDataTiapCluster(i)-1)==0){
                         standarDeviasi[i][j]=0;
                     }
                     else{
standarDeviasi[i][j]=Math.sqrt(temp/(jumlahDataTiapCluster(i)-
1));
                     }
            }
      }
}
private static double varianTiapAtribut(int x, int y){
int temp=0;
double varianAtribut=0,jumDataCluster=0,rataAtribut=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
rataAtribut=rataRataAtribut(x,y);
      for(int k=0;k<sc.jumlahData;k++){
            if(idxHasilCluster[k]==x){
varianAtribut=varianAtribut+Math.pow(sc.matriksData[k][y]-
rataAtribut,2);
             }
      }
            if(jumlahDataTiapCluster(x)==1){
                varianAtribut=0;
            }
            else{
varianAtribut=varianAtribut*1/((jumlahDataTiapCluster(x))-1);
            }
      return varianAtribut;
}
private static double varianTiapCluster(int x){
double varian=0, temp=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
      for(int j=0;j<sc.jumlahAtribut;j++){
```

```
            temp=varianTiapAtribut(x,j);
            varian=varian+temp;
        }
        return varian;
}
```

Gambar 5.9 Implementasi Perhitungan Varian tiap *Cluster*
Sumber : Implementasi

2. Implementasi Proses Perhitungan *Varian Within Cluster*

Proses perhitungan *varian within cluster* nantinya melibatkan hasil dari *method* `varianTiapCluster()`. Dalam implementasinya, proses perhitungan *varian within cluster* diterapkan dengan membuat *method* `varianWithinCluster()`. Proses pada *method* ini mengacu pada Persamaan (2.9). Variabel *temp* digunakan untuk menampung jumlah total nilai varian tiap cluster yang terbentuk. Implementasi untuk proses perhitungan *varian within cluster* ditunjukkan pada Gambar (5.10).

```
private static double varianWithinCluster(){
double VW=0,temp=0;
  for(int i=0;i<sc.jumlahCluster;i++){
  temp=temp+((jumlahDataTiapCluster(i)-1)*varianTiapCluster(i));
  }
  double temp2=sc.jumlahData-sc.jumlahCluster;
  double temp3=1/temp2;
  VW=temp3*temp;
  return VW;
}
```

Gambar 5.10 Implementasi Perhitungan *Varian Within Cluster*
Sumber : Implementasi

3. Implementasi Proses Perhitungan *Varian Between Cluster*

Proses perhitungan varian tiap *cluster* diterapkan dengan membuat *method* `varianBetweenCluster()`, dimana dalam *method* tersebut perlu untuk memanggil *method* tambahan lainnya untuk membantu proses perhitungan, diantaranya adalah method `jumlahDataTiapCluster()`, `rataRataAtribut()`, dan `varianAntarAtribut()`. Implementasi untuk proses perhitungan varian *between cluster* ditunjukkan pada Gambar (5.11).

```
private static double varianAntarAtribut(int x, int y){
double varianAntarAtribut=0,jumDataCluster=0,rataAtribut=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
rataAtribut=rataRataAtribut(x,y);
```

```
      for(int k=0;k<sc.jumlahData;k++){
           if(idxHasilCluster[k]==x){
       varianAntarAtribut=varianAntarAtribut+(Math.pow(sc.matriksD
       ata[k][y]-rataAtribut,2))*jumlahDataTiapCluster(x);
           }
      }
      return varianAntarAtribut;
}

private static double varianAntarCluster(int x){
double varian=0, temp=0;
int[] idxHasilCluster=sc.tentukanIdxCluster(sc.matriksAwal);
      for(int j=0;j<sc.jumlahAtribut;j++){
          temp=varianAntarAtribut(x,j);
          varian=varian+temp;
      }
      return varian
}

private static double varianBetweenCluster(){
double VB=0,temp=0;
      for(int i=0;i<sc.jumlahCluster;i++){
           temp=temp+varianAntarCluster(i);
      }
      double temp2=sc.jumlahCluster-1;
      double temp3=1/temp2;
      VB=temp3*temp;
      return VB;
}
```

Gambar 5.11 Implementasi Perhitungan *Varian Between Cluster*
Sumber : Implementasi

4. Implementasi Proses Perhitungan Batasan Varian

Perhitungan nilai batasan varian merupakan proses terakhir pada proses perhitungan varian. Nilai batasan varian didapatkan dari hasil bagi antara nilai *varian within cluster* dengan *varian between cluster* sesuai dengan Persamaan (2.11). Pada implementasinya, proses perhitungan batasan varian diterapkan pada sebuah *method* `batasanVarian()` yan ditunjukkan pada Gambar (5.12).

```
public static double batasanVarian()
{
double batasanVarian=0;
batasanVarian=varianWithinCluster()/varianBetweenCluster();
return batasanVarian;
}
```

Gambar 5.12 Implementasi Perhitungan Batasan Varian
Sumber : Implementasi

5.2.2 Implementasi kelas Ekstraksi Aturan *Fuzzy*

Kelas ekstraksi aturan fuzzy merupakan kelas untuk mendapatkan nilai koefisien output dan aturan *fuzzy*. Pada kelas `ekstraksiAturanFuzzy()` terdapat dua *method* utama, yaitu *method* `hitungDerajatKeanggotaan()` dan `perhitunganKoefisienOutput()`.

1. Implementasi Perhitungan Nilai Derajat Keanggotaan

Proses perhitungan nilai derajat keanggotaan untuk tiap data terhadap pusat *cluster* dan sigma dengan menggunakan fungsi *gauss* sama seperti pada Persamaan (2.21). Pada implementasinya, proses ini diterapkan ke dalam *method* `hitungDerajatKeanggotaan()`. Implementasi Perhitungan Nilai Derajat Keanggotaan dalam *method* `hitungDerajatKeanggotaan()` ditunjukkan pada Gambar (5.13).

```
public static void hitungDerajatKeanggotaan()
{
tabelMyu=new double[fcm.jumlahData][fcm.jumlahCluster];
     for(int i=0;i<fcm.jumlahData;i++){
          for(int k=0;k<fcm.jumlahCluster;k++){
               double sigma=1;
               for(int j=0;j<fcm.jumlahAtribut;j++){
                   if(sd.standarDeviasi[k][j]!=0){
                  sigma=sigma+((Math.pow((fcm.matriksData[i][j]-
fcm.V[k][j]),2))/(2*(Math.pow(sd.standarDeviasi[k][j],2))));
                    }
                   else if(sd.standarDeviasi[k][j]==0){
                         sigma=sigma+0.0;
                   }
               }
               tabelMyu[i][k]=Math.exp((-1)*sigma);
          }
     }
}
```

Gambar 5.13 Implementasi Perhitungan Nilai Derajat Keanggotaan
Sumber : Implementasi

2. Implementasi Perhitungan Koefisien *Output*

Implementasi proses perhitungan koefisien *output* diterapkan pada *method* `perhitunganKoefisienOutput()`. Proses perhitungan koefisien *output* memerlukan operasi matriks yang cukup kompleks, diantaranya seperti perkalian matriks, *transpose*, dan *inverse*. Oleh karena itulah perlu adanya sebuah *library*

tambahan, yaitu `org.ejml.simple.SimpleMatrix` untuk memudahkan operasi matriks dengan dimensi yang cukup panjang. Implementasi perhitungan koefisien *ouput* pada *method* `perhitunganKoefisienOutput()` ditunjukkan pada Gambar (5.14).

```
private static double[][] hitungMatriksD(){
double [][]d= new
double[fcm.jumlahData][fcm.jumlahAtribut*fcm.jumlahCluster];
   for(int i=0;i<fcm.jumlahData;i++)
   {
      int k=0,l=0,cekAtribut=1;
      for(int j=0;j<fcm.jumlahAtribut*fcm.jumlahCluster;j++){
         if(cekAtribut==fcm.jumlahAtribut)
        {
            d[i][j]=tabelMyu[i][l];
            k=0;
            l++;
            cekAtribut=0;
         }
         else{

d[i][j]=fcm.matriksData[i][k]*tabelMyu[i][l];
                    k++;
                }
                cekAtribut++;
            }
        }
        return d;
    }
private static double[][] hitungMatriksU(){
double [][] u = new
double[fcm.jumlahData][fcm.jumlahAtribut*fcm.jumlahCluster];
double [] totalMiu = new double[fcm.jumlahData];
double [][] d = hitungMatriksD();
      for(int i=0;i<fcm.jumlahData;i++){
            totalMiu[i] = 0;
            for(int k=0;k<fcm.jumlahCluster;k++){
                totalMiu[i] = totalMiu[i]+tabelMyu[i][k];
            }
         for(int j=0;j<fcm.jumlahAtribut*fcm.jumlahCluster;j++){
               u[i][j] = d[i][j]/totalMiu[i];

            }
        }
        return u;
    }
private static double[][] hitungMatriskUT(){
SimpleMatrix u = new SimpleMatrix(hitungMatriksU());
```

```
SimpleMatrix uT = u.transpose();
double uTranspose[][]=new double[uT.numRows()][uT.numCols()];
     for(int i=0;i<uT.numRows();i++){
        for(int j=0;j<uT.numCols();j++){
            uTranspose[i][j]=uT.get(i, j);
            }
        }
        return uTranspose;
    }
private static double[][] operasiMatriksUTxU(){
SimpleMatrix uT = new SimpleMatrix(hitungMatriskUT());
SimpleMatrix u = new SimpleMatrix(hitungMatriksU());
SimpleMatrix uTU = uT.mult(u);
        double uTransposeU[][]=new
double[uTU.numRows()][uTU.numCols()];
        for(int i=0;i<uTU.numRows();i++){
            for(int j=0;j<uTU.numCols();j++){
                uTransposeU[i][j]=uTU.get(i, j);
            }
        }
        return uTransposeU;
    }

private static double [][] operasiInverseU(){
SimpleMatrix uTUI = new SimpleMatrix(operasiMatriksUTxU());
SimpleMatrix uTUInverse = uTUI.pseudoInverse();
double inverseU[][]=new
double[uTUInverse.numRows()][uTUInverse.numCols()];
      for(int i=0;i<uTUInverse.numRows();i++){
         for(int j=0;j<uTUInverse.numCols();j++){
            inverseU[i][j]=uTUInverse.get(i, j);
            }
         }
         return inverseU;
}
public static double [][] perhitunganKoefisienOutput(){
SimpleMatrix u = new SimpleMatrix(hitungMatriksU());
SimpleMatrix uT = u.transpose();
double uTranspose[][]=new double[uT.numRows()][uT.numCols()];
        for(int i=0;i<uT.numRows();i++){
            for(int j=0;j<uT.numCols();j++){
                uTranspose[i][j]=uT.get(i, j);
            }
        }
SimpleMatrix uTU = uT.mult(u);
double uTransposeU[][]=new double[uTU.numRows()][uTU.numCols()];
        for(int i=0;i<uTU.numRows();i++){
            for(int j=0;j<uTU.numCols();j++){
                uTransposeU[i][j]=uTU.get(i, j);
            }
```

```
        }
SimpleMatrix ui = new SimpleMatrix(uTransposeU);
SimpleMatrix uTUInverse = ui.invert();
double inverseU[][]=new
double[uTUInverse.numRows()][uTUInverse.numCols()];
         for(int i=0;i<uTUInverse.numRows();i++){
            for(int j=0;j<uTUInverse.numCols();j++){
                inverseU[i][j]=uTUInverse.get(i, j);
            }
         }
SimpleMatrix inverseUxUT = uTUInverse.mult(uT);
double nk[][]=new double[fcm.jumlahData][1];
         for(int i=0;i<fcm.jumlahData;i++){
             nk[i][0]=fcm.matriksData[i][fcm.jumlahAtribut-1];
         }
         SimpleMatrix targetOutput = new SimpleMatrix(nk);
         SimpleMatrix ko = inverseUxUT.mult(targetOutput);
         double koefisienOutput[][] = new double
[fcm.jumlahCluster][fcm.jumlahAtribut];
         int k=0;
         for(int i=0;i<fcm.jumlahCluster;i++){
             for(int j=0;j<fcm.jumlahAtribut;j++){
                 if(ko.get(k,0)>-1000&&ko.get(k, 0)<1000){
                     koefisienOutput[i][j]=ko.get(k,0);
                 }
                 else{
                     koefisienOutput[i][j]=0;
                 }
                 k++;
             }
         }
              return koefisienOutput;
     }
```

Gambar 5.14 Implementasi Perhitungan Koefisien *Output*
Sumber : Implementasi

5.2.4 Implementasi Kelas FIS Sugeno

Kelas FIS Sugeno merupakan kelas untuk pengujian akurasi terhadap data uji menggunakan *fuzzy inference system* model sugeno orde-satu dengan aturan yang telah dibangkitkan pada proses *clustering* dan ekstraksi aturan *fuzzy*. Implementasi pada proses ini diterapkan ke dalam kelas `fisSugeno.java`, dimana di dalamnya terdapat 6 buah *method* diantaranya adalah `BacaDataUji()`, `getCenter()`, `hitungMiu()`, `hitungAlphaPredikat()`, `hitungZ()`, dan `defuzzy()`.

1. Implementasi Proses Pembacaan Data Uji

Langkah pertama dalam proses pengujian adalah memasukkan data uji. Dalam implementasinya, data uji disimpan ke dalam sebuah database yang nantinya dipanggil menggunakan *method* `BacaDataUji()`. Di dalam *method* `BacaDataUji()` terdapat *method* `KoneksiDB()` yang fungsinya sebagai konektor antara aplikasi dengan *database*. Implementasi proses baca data uji ditunjukkan pada Gambar (5.15).

```
private static void BacaDataUji()
{
KoneksiDB();
matriksDataUji = new double[100][9];
try {
    String sql = "select * from datauji";
    statement.executeQuery(sql);
    ResultSet rs = statement.executeQuery(sql);
    double id=0, birads=0,age=0,shape=0,margin=0,density=0,nr=0;
    int cek=0;
    while(rs.next()){
         birads=rs.getDouble("birads");
         age=rs.getDouble("age");
         shape=rs.getDouble("shape");
         margin=rs.getDouble("margin");
         density=rs.getDouble("density");
         id=rs.getDouble("id");
         nr=rs.getDouble("severity");

         matriksDataUji[cek][0]=birads;
         matriksDataUji[cek][1]=age;
         matriksDataUji[cek][2]=shape;
         matriksDataUji[cek][3]=margin;
         matriksDataUji[cek][4]=density;
         matriksDataUji[cek][6]=id;
         matriksDataUji[cek][7]=nr;
         cek++;
         }
         rs.close();
         statement.close();
         }
         catch(Exception e){
         JOptionPane.showMessageDialog(null,   e,   "Kesalahan",
JOptionPane.ERROR_MESSAGE);
            }
}
```

Gambar 5.15 Implementasi Baca Data Uji
Sumber : Implementasi

2. Implementasi Inisialisasi Pusat *Cluster*

Proses inisialisasi pusat *cluster* merupakan proses untuk mendapatkan nilai pusat *cluster* yang sebelumnya telah dihitung pada proses *clustering*. Proses ini diimplementasikan dengan membuat *method* `getCenter()`. Implementasi inisialisasi pusat *cluster* ditunjukkan pada Gambar (5.16).

```
private static double[][] getCenter()
{
double [][] pusatCluster = new double[fcm.jumlahCluster]
[fcm.jumlahAtribut];
   for(int i=0;i<fcm.jumlahCluster;i++){
       for(int j=0;j<fcm.jumlahAtribut;j++){
       pusatCluster[i][j]=fcm.V[i][j];
       }
   }
   return pusatCluster;
}
```

Gambar 5.16 Implementasi Inisialisasi Pusat *Cluster*
Sumber : Implementasi

3. Implementasi Proses Perhitungan Derajat Keanggotaan

Proses perhitungan derajat keanggotaan dilakukan terhadap data uji yang memiliki 6 parameter. Nantinya 6 parameter tersebut dicari derajat keanggotaannya menggunakan fungsi *gauss*. Implementasi untuk proses perhitungan derajat keanggotaan diterapkan pada *method* `hitungMiu()` yang ditunjukkan pada Gambar (5.17).

```
private static double[][] hitungMiu(double [][] dataUji, int x)
{
double[][] miuDataUji = new
double[fcm.jumlahCluster][fcm.jumlahAtribut-1];
double [][] center = getCenter();
     for(int i=0;i<fcm.jumlahCluster;i++){
        double tempSigma=0;
        for(int j=0;j<fcm.jumlahAtribut-1;j++){
            tempSigma=((Math.pow((dataUji[x][j]-
center[i][j]),2))/(2*(Math.pow(sd.standarDeviasi[i][j],2))));
            miuDataUji[i][j]=Math.exp((-1)*tempSigma);
        }
      }
      return miuDataUji;
}
```

Gambar 5.17 Implementasi Perhitungan Derajat Keanggotaan
Sumber : Implementasi

4. Implementasi Perhitungan *Alpha Predikat*

Implementasi proses perhitungan *alpha predikat* diterapkan pada sebuah *method* `hitungAlphaPredikat()`. Implementasi proses perhitungan *alpha predikat* ditunjukkan pada Gambar (5.18).

```
private static double [] hitungAlphaPredikat(double [][]
dataUji, int x){
double []alphaPredikat = new double [fcm.jumlahCluster];
double [][]miuDataUji = hitungMiu(dataUji, x);
     for(int i=0;i<fcm.jumlahCluster;i++)
     {
        double temp=1;
        for(int j=0;j<fcm.jumlahAtribut-1;j++)
        {
        temp=temp*miuDataUji[i][j];
        }
        alphaPredikat[i]=temp;
      }
      return alphaPredikat;
}
```

Gambar 5.18 Implementasi Perhitungan *Alpha Predikat*
Sumber : Implementasi

5. Implementasi Perhitungan Nilai *z*

Implementasi proses perhitungan nilai *z* untuk tiap aturan diterapkan pada sebuah *method* `hitungZ()` yang ditunjukkan pada Gambar (5.19).

```
private static double [] hitungZ(double [][] dataUji, int x)
{
   double []z = new double [fcm.jumlahCluster];
   double [][]ko = aturan.perhitunganKoefisienOutput();
        for(int i=0;i<fcm.jumlahCluster;i++)
        {
           double temp=0;
           for(int j=0;j<fcm.jumlahAtribut-1;j++)
           {
               double temp2=0;
               temp2=dataUji[x][j]*ko[i][j];
               temp=temp+temp2;
           }
          z[i]=temp+ko[i][fcm.jumlahAtribut-1];
         }
         return z;
}
```

Gambar 5.19 Implementasi Perhitungan Nilai *z*
Sumber : Implementasi

6. Implementasi Proses *Defuzzy*

Proses *defuzzy* merupakan proses terakhir pada fase pengujian. Proses *defuzzy* merupakan proses merubah nilai *fuzzy* menjadi bentuk *crisp*. Pada implementasinya, proses *defuzzy* diterapkan ke dalam *method* `defuzzy()`. Untuk memudahkan alur penulisan kode program, maka diperlukan variabel *temp* untuk menampung jumlah total antara perkalian *aplha predikat* dengan nilai *z*, sedangkan variabel *temp2* digunakan untuk menampung nilai total dari hasil penjumlahan setiap *alpha predikat* pada masing – masing aturan. Implementasi proses *defuzzy* ditunjukkan pada Gambar (5.20).

```
private static double defuzzy(double [][] dataUji, int x)
{
double zAkhir=0;
double []a=hitungAlphaPredikat(dataUji,x);
double []b=hitungZ(dataUji,x);
double temp=0, temp2=0;
     for(int i=0;i<fcm.jumlahCluster;i++)
     {
          temp=temp+(a[i]*b[i]);
          temp2=temp2+a[i];
     }
     zAkhir=temp/temp2;
     return zAkhir;
}
```

Gambar 5.20 Implementasi Proses *Defuzzy*
Sumber : Implementasi

5.3 Implementasi Antarmuka

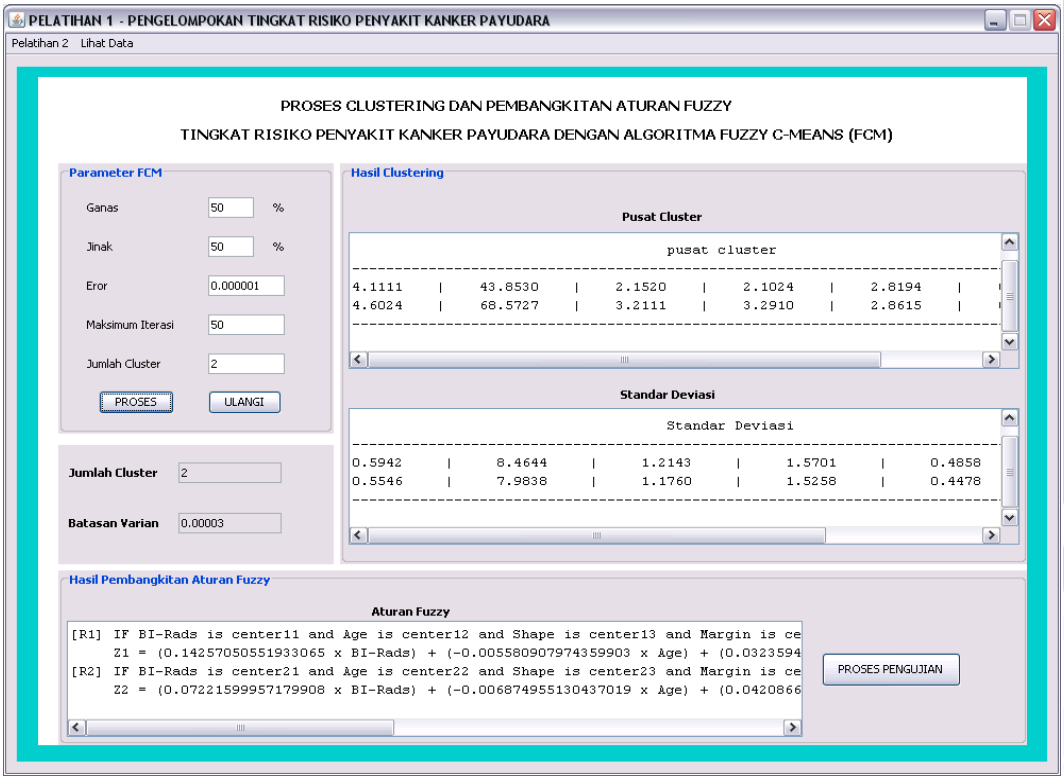
Implementasi antarmuka diterapkan berdasarkan pada bab perancangan. Aplikasi sistem pembangkitan aturan secara otomatis untuk pengelompokan tingkat risiko penyakit kanker payudara ini menggunakan bahasa pemrograman Java. Berikut adalah implementasi antarmuka yang digunakan untuk proses *clustering*, pemilihan jumlah *cluster* ideal, pembangkitan aturan, dan proses inferensi *fuzzy* untuk pengelompokan tingkat risiko penyakit kanker payudara.

5.3.1 Implementasi Antarmuka Pelatihan 1

Antarmuka pelatihan 1 adalah antarmuka yang akan tampil pertama kali saat program dijalankan. Pelatihan 1 mencakup menu-menu yang ada pada aplikasi. Menu-menu pelatihan 2 dan lihat data. Pada pelatihan 1 dilakukan proses

pelatihan. Pelatihan 1 digunakan untuk proses *clustering* terhadap data latih dan proses pembangkitan aturan *fuzzy*.

Ada beberapa *field* dan tombol yang digunakan, diantaranya adalah *field* untuk memasukkan data latih yang terdiri dari kelas ganas dan kelas jinak dalam bentuk persen yang sebelumnya telah disimpan pada *database*, *field* untuk memasukkan nilai eror terkecil, iterasi maksimum dan jumlah *cluster*. Tombol "Proses" merupakan tombol untuk memproses data latih berdasarkan parameter yang ditentukan sebelumnya. Tombol "Ulangi" merupakan tombol untuk mengosongkan seluruh *field* masukan agar dapat memulai proses *clustering* kembali. Sedangkan tombol proses pengujian digunakan untuk pengujian aturan *fuzzy* yang telah terbentuk dengan sistem inferensi *fuzzy* sugeno orde-satu. Implementasi antarmuka pelatihan 1 ditunjukkan pada Gambar (5.21).

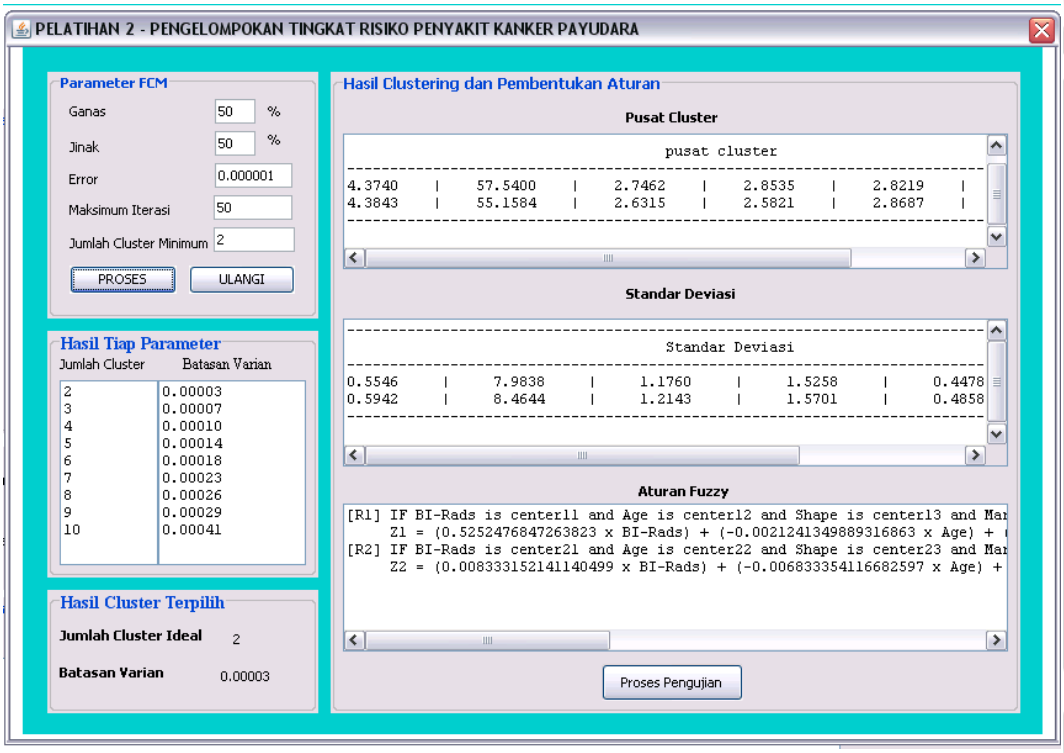


Gambar 5.21 Implementasi Antarmuka Pelatihan 1
Sumber : Implementasi

5.3.2 Implementasi Antarmuka Pelatihan 2

Antarmuka Pelatihan 2 merupakan antarmuka yang digunakan untuk menjembatani *user* dengan sistem dalam melakukan proses pelatihan

menggunakan *fuzzy c-means clustering* terhadap data latih. Ada beberapa *field* dan tombol yang digunakan, diantaranya adalah *field* untuk memasukkan data latih yang terdiri dari kelas ganas dan kelas jinak dalam bentuk persen yang sebelumnya telah disimpan pada *database*, *field* untuk memasukkan nilai eror terkecil, maksimum iterasi, dan jumlah *cluster* minimum. Tombol "Proses" merupakan tombol untuk memproses data latih berdasarka parameter yang ditentukan sebelumnya. Sedangkan tombol "Ulangi" merupakan tombol untuk mengosongkan seluruh *field* masukan agar dapat memulai proses *clustering* kembali. Implementasi antarmuka pelatihan 2 dapat dilihat pada Gambar (5.22).

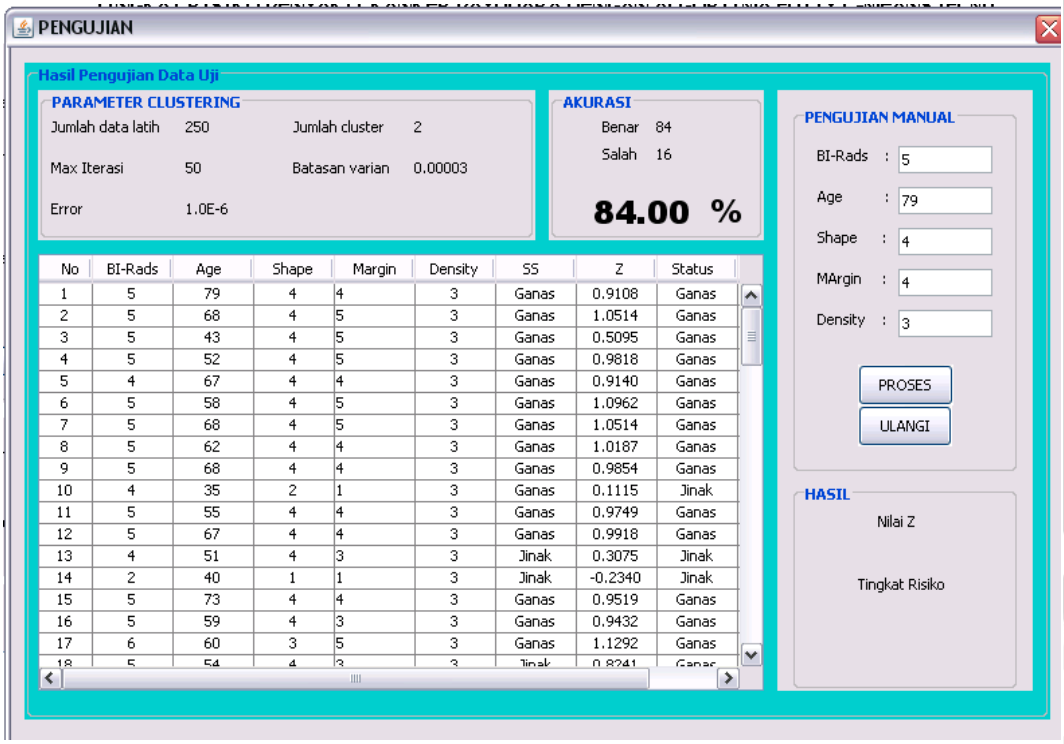


Gambar 5.22 Implementasi Antarmuka Pelatihan 2
Sumber : Implementasi

Pada antarmuka ini disediakan informasi hasil dari proses setiap pengujian sebanyak 9 kali, diantaranya adalah nilai batasan varian dan jumlah *cluster*. Sistem juga akan memilih secara otomatis hasil *cluster* mana yang akan dipakai pada proses ekstraksi aturan *fuzzy* berdasarkan analisa varian. Dari proses analisa varian, diberikan informasi terkait jumlah *cluster* terpilih, batasan varian terpilih, pusat *cluster* terpilih, standar deviasi *cluster* terpilih, dan aturan *fuzzy* beserta koefisien *output* yang terbentuk.

5.3.3 Implementasi Antarmuka Pengujian

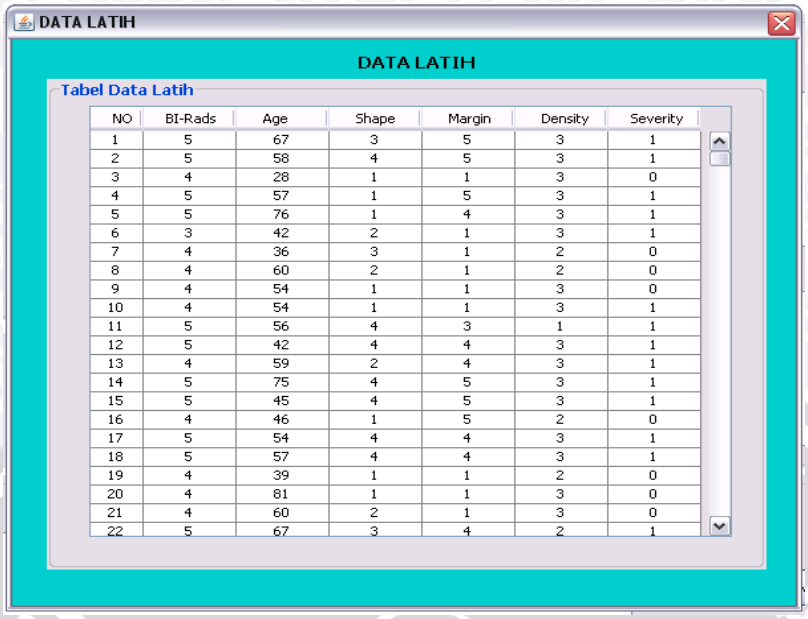
Implementasi pengujian data uji merupakan antarmuka untuk menampilkan pengelompokan tingkat risiko penyakit kanker payudara berdasarkan ekstraksi aturan *fuzzy* pada proses pelatihan sebelumnya. Dalam form ini juga ditampilkan parameter – parameter yang digunakan pada proses pelatihan dengan FCM *clustering*. Pada implementasi pengujian ini juga ditampilkan akurasi hasil perhitungan data uji dengan menggunakan aturan *fuzzy* yang telah terbentuk dengan algoritma sugeno orde-satu. Pada antarmuka pengujian ini juga dapat dilakukan perhitungan untuk pengelompokan tingkat risiko kanker payudara pada data yang baru.Tampilan antarmuka untuk implementasi pengujian ditunjukkan pada Gambar (5.23).



Gambar 5.23 Implementasi Antarmuka Pengujian
Sumber : Implementasi

5.3.4 Implementasi Antarmuka Data Latih

Implementasi antarmuka data latih digunakan untuk melihat data latih yang akan digunakan pada proses pelatihan. Implementasi antarmuka data ditunjukkan pada Gambar (5.24).

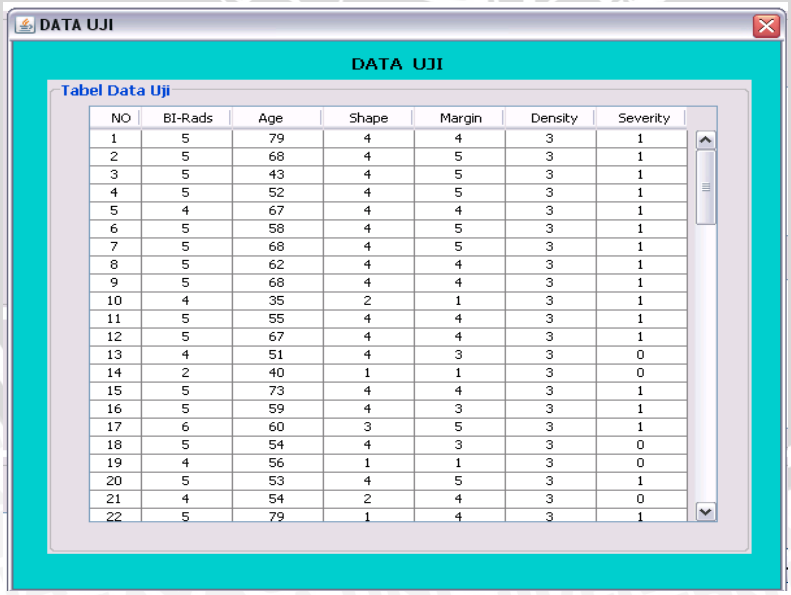


NO	BI-Rads	Age	Shape	Margin	Density	Severity
1	5	67	3	5	3	1
2	5	58	4	5	3	1
3	4	28	1	1	3	0
4	5	57	1	5	3	1
5	5	76	1	4	3	1
6	3	42	2	1	3	1
7	4	36	3	1	2	0
8	4	60	2	1	2	0
9	4	54	1	1	3	0
10	4	54	1	1	3	1
11	5	56	4	3	1	1
12	5	42	4	4	3	1
13	4	59	2	4	3	1
14	5	75	4	5	3	1
15	5	45	4	5	3	1
16	4	46	1	5	2	0
17	5	54	4	4	3	1
18	5	57	4	4	3	1
19	4	39	1	1	2	0
20	4	81	1	1	3	0
21	4	60	2	1	3	0
22	5	67	3	4	2	1

Gambar 5.24 Implementasi Antarmuka Data Latih
Sumber : Implementasi

5.3.5 Implementasi Antarmuka Data Uji

Implementasi antarmuka data uji digunakan untuk melihat data uji yang digunakan pada proses pengujian. Implementasi antarmuka data uji ditunjukkan pada Gambar (5.25).



NO	BI-Rads	Age	Shape	Margin	Density	Severity
1	5	79	4	4	3	1
2	5	68	4	5	3	1
3	5	43	4	5	3	1
4	5	52	4	5	3	1
5	4	67	4	4	3	1
6	5	58	4	5	3	1
7	5	68	4	5	3	1
8	5	62	4	4	3	1
9	5	68	4	4	3	1
10	4	35	2	1	3	1
11	5	55	4	4	3	1
12	5	67	4	4	3	1
13	4	51	4	3	3	0
14	2	40	1	1	3	0
15	5	73	4	4	3	1
16	5	59	4	3	3	1
17	6	60	3	5	3	1
18	5	54	4	3	3	0
19	4	56	1	1	3	0
20	5	53	4	5	3	1
21	4	54	2	4	3	0
22	5	79	1	4	3	1

Gambar 5.25 Implementasi Antarmuka Data Uji
Sumber : Implementasi