BAB 5 IMPLEMENTASI

Bab ini menjelaskan tentang implementasi sistem berdasarkan analisa kebutuhan dan proses perancangan yang telah dibuat. Pembahasan pada bab ini terdiri dari spesifikasi sistem, batasan implementasi, implementasi algoritma dan implementasi antarmuka.

5.1 Spesifikasi Sistem

Proses implementasi sistem membutuhkan spesifikasi yang sesuai dengan yang dibangun sehingga dapat berfungsi sesuai dengan kebutuhan. Spesifikasi perangkat keras maupun spesifikasi perangkat lunak yang dibutuhkan oleh sistem akan dijelaskan pada bagian di bawah ini.

5.1.1 Spesifikasi Perangkat Keras

Pengembangan dari Klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree - Support Vector Machine (BDT-SVM)* menggunakan perangkat keras yang sesuai dengan kebutuhan, yang ditunjukkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras.

No.	Komponen	Spesifikasi
1	*Processor*	*Intel (R) Core (TM) i3-3110M CPU @ 2.40GHz*
2	Memori (RAM)	*6.00 GB (5.88 GB usable)*
3	Monitor	16 bit
4	Hard disk	297.97 GB

Tabel 5.1 merupakan spesifikasi perangkat keras yang dibutuhkan untuk mendukung proses klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree - Support Vector Machine (BDT-SVM)*.

5.1.2 Spesifikasi Perangkat Lunak

Pengembangan dari Klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree - Support Vector Machine (BDT-SVM)* menggunakan perangkat lunak yang sesuai dengan kebutuhan, yang ditunjukkan pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak.

No.	Nama	Spesifikasi
1	Sistem Operasi	*Windows 10 Home Single Language*
2	Bahasa Pemrograman	*Java*
3	*Tools* Pemrograman	*Netbeans IDE 8.0.2*
4	*Database*	*MySQL*

Tabel 5.2 merupakan spesifikasi perangkat lunak yang dibutuhkan untuk mendukung proses klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree - Support Vector Machine (BDT-SVM)*.

5.2 Batasan Implementasi

Beberapa batasan dalam membangun sistem untuk klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree - Support Vector Machine (BDT-SVM)* yaitu sebagai berikut:

1. Sistem dibangun berupa aplikasi berbasis *desktop* dengan menggunakan bahasa pemrograman *java*.
2. Penyimpanan data dengan menggunakan *DBMS MySQL* dan operasi file berupa *file.csv*.
3. Metode yang digunakan dalam menyelesaikan masalah adalah metode *Binary Decision Tree - Support Vector Machine (BDT-SVM)*.
4. Data masukan yang digunakan dalam sistem adalah data hasil pemeriksaan Elektrokardiografi (EKG).
5. Hasil output dari hasil perhitungan dengan menggunakan metode *BDT* adalah menghasilkan atau membuat kelas baru yang digunakan pada perhitungan *SVM*, dan hasil output dari *SVM* yaitu hasil klasifikasi kondisi detak jantung yang terdiri dari empat klasifikasi yaitu Normal, *Atrial Fibrilation*, *PVC Bigeminy*, dan *Ventricular Tachycardia*.
6. Jumlah parameter sebanyak 7202 yang diambil pada waktu 20 detik yang terdiri dari 2 pemeriksaan yaitu *MLII* dab *VI*, 3601 diambil dari *MLII* dan *VI* selama 10 detik, dan 2161 dari *MLII* dan *VI* selama 6 detik.

5.3 Implementasi Algoritma

Sistem Klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree-Support Vector Machine (BDT-SVM)* memiliki beberapa algoritma yang terdapat pada sistem. Berikut ini penjelasan masing-masing dari setiap algoritma.

5.3.1 Implementasi Algoritma Perhitungan *Min-Max Normalization*

Algoritma dari perhitungan normalisasi adalah untuk mendapatkan nilai data ke dalam *range* yang kecil atau merupakan transformasi data atau mengubah data agar menjadi lebih kecil dengan rentang nilai antara 0 sampai dengan 1. Proses normalisasi menggunakan *min-max* normalisasi dengan mencari nilai *min* dan *max* dari setiap parameter yang ada. Pada perhitungan normalisasi menghasilkan keluaran berupa matriks dengan index *nxn*, dan *n* yaitu banyaknya jumlah data *training* dan *testing*. Kode program dari algoritma *BDT* dijelaskan pada Kode Program 5.1.

```
1   public float Normalisasi(int i, int j, float[][] b) {
        float normalisasi = ((b[i][j] - hitungMin(b, j)) /
```

```
2       (hitungMax(b, j) - hitungMin(b, j)));
            return normalisasi;
3       }
4
        public float hitungMin(float[][] data, int param) {
5        float tempr = data[0][param];
6        for (int i = 0; i < data.length; i++) {
7            if (data[i][param] < tempr) {
8                tempr = data[i][param];
9            }
10       }
11       return tempr;
12      }
13
        public float hitungMax(float[][] data, int param) {
14          float tempr = data[0][param];
15          for (int i = 0; i < data.length; i++) {
16              if (data=[i][param] > tempr) {
17                  tempr = data[i][param];
18              }
19          }
20          return tempr;
21      }
22
```

Kode Program 5.1 Implementasi Algoritma Perhitungan *Min-Max* Normalisasi.

Berikut penjelasan dari Kode Program 5.1:

1. Baris ke-1 sampai 4 merupakan fungsi dari proses perhitungan normalisasi. Pada baris ke-2 menunjukkan fungsi dari *min-max* normalisasi dengan memanggil fungsi *hitungMin()* dan *hitungMax()*.
2. Baris ke-5 sampai 13 merupakan fungsi dari proses perhitungan mencari nilai minimal. Baris ke-7 merupakan perulangan untuk data *training* dan *testing* pada baris ke-*i*. Kemudian pada baris ke-8 sampai 10 menjelaskan kondisi bahwa jika data lebih kecil dari *tempr*, maka akan menyimpan nilai min ke dalam variabel *tempr*.
3. Baris ke-14 sampai 22 merupakan fungsi dari proses perhitungan untuk mencari nilai maksimal. Baris ke-16 merupakan perulangan untuk data *training* dan *testing* pada baris ke-*i*. Kemudian pada baris ke-17 sampai 20 menjelaskan kondisi jika data lebih besar dari *tempr*, maka akan menyimpan nilai maksimal ke dalam variabel *tempr*.
4. Baris ke-6 dan 15 menjelaskan inisialisasi variabel *tempr* yang digunakan untuk menyimpan nilai maksimal dan minimal ke variabel *tempr*.

5.3.2 Implementasi Algoritma Perhitungan *Binary Decision Tree*

Algoritma dari perhitungan *binary decision tree* adalah untuk menemukan kelas baru dengan membentuk pohon menggunakan jarak dari hasil jarak *euclidean. BDT* ini memecahkan permasalahan *SVM* yang hanya bisa *binary class*, dimana akan membentuk node dalam membuat keputusan biner menggunakan klasifikasi *SVM*. Dalam pembentukan *tree*, akan mencari nilai maksimal dari jarak euclidean, kemudian dari nilai maksimal tersebut akan dicari nilai minimal yang

mendekati kelasnya. Pada perhitungan *BDT* menghasilkan keluaran berupa matriks dengan index *n*, dan *n* yaitu banyaknya jumlah baris data. Kode program dari algoritma *BDT* dijelaskan pada Kode Program 5.2.

```
1   //BDT level 1
2   hitungMax(0, 3, maxL1, indekKiri1, indekKanan1);
3   System.out.println(BinaryDT(indekKiri, indekKanan));
4   //kelas ke-1
5   if (indekKiri == 1) {
6      File file = new
       File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
       Dataset/Level 1/Kelas1.csv");
7      SimpanKelas1(file, indekKiri, DataTraining);
8   } else if (indekKiri == 2) {
9      File file = new
       File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
       Dataset/Level 1/Kelas2.csv");
       SimpanKelas1(file, indekKiri, DataTraining);
10     ....
11  }
12  //kelas ke-2
13  if (indekKanan == 1) {
14     File file1 = new
       File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
       Dataset/Level 1/Kelas1.csv");
15     SimpaneKelas1Neg(file1, indekKanan, DataTraining);
16  } else if (indekKanan == 2) {
17     File file1 = new
       File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
       Dataset/Level 1/Kelas2.csv");
18     SimpaneKelas1Neg(file1, indekKanan, DataTraining);
19     ....
20  }
21  //kelas ke-3
22  baris(indekKiri, indekKanan);
23  bandingKelasMin(baris);
24  mencariDenganLS(EuclideanDistance, bandingKelasMin(baris));
25  System.out.println("Bandingkan Nilai : " + (kelas1) + " dan
    " + (kelas2));
26  minL1.setText(String.valueOf(kelas1));
27  minL2.setText(String.valueOf(kelas2));
28  if (indekKiri == 1 && indekKanan == 4
       || indekKiri == 1 && indekKanan == 3
       || indekKiri == 1 && indekKanan == 2) {
29     System.out.println("Min ada di index " +
       mencariDenganLS2(EuclideanDistance,
       bandingKelasMin(baris), indekKiri));
30     int min;
31     //kondisi jika kelas yg dicari berapa pada indek ke-1 dan
       2, tetapi menunjuk kelas 2 maka di set ke-kelas 3
32     if (indekKiri == 1 && indekKanan == 2 &&
          mencariDenganLS2(EuclideanDistance,
          bandingKelasMin(baris), indekKiri) == 2) {
33        min = 3;
34        System.out.println("Min ada di index " + min);

35        min1.setText(String.valueOf(min));
36        System.out.println(alokasi2(baris, min));
```

```
37        //hasilnya pasti akan negatif
38        File file2 = new
      File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
      Dataset/Level 1/Kelas3.csv");
39        SimpaneKelas1Neg(file2, min, DataTraining);

40      } else {

41        min1.setText(String.valueOf(mencariDenganLS2(Euclidean
          Distance, bandingKelasMin(baris), indekKiri)));
42        System.out.println(alokasi2(baris,
      mencariDenganLS2(EuclideanDistance,
      bandingKelasMin(baris), indekKiri)));
43        if (mencariDenganLS2(EuclideanDistance,
      bandingKelasMin(baris), indekKiri) == 1) {
44          File file2 = new
      File("C:/Users/sony/Documents/NetBeansProjects/BacaText1/
      Dataset/Level 1/Kelas1.csv");
45          if (alokasi2(baris,
      mencariDenganLS2(EuclideanDistance,
      bandingKelasMin(baris), indekKiri)) == -1) { //kalau
      hasilx -1
46            SimpaneKelas1Neg(file2,
      mencariDenganLS2(EuclideanDistance,
      bandingKelasMin(baris), indekKiri), DataTraining);
47          } else { //kalau tidak
48            SimpanKelas1(file2,
      mencariDenganLS2(EuclideanDistance,
      bandingKelasMin(baris), indekKiri), DataTraining);
49          }
50      ....
51        }
```

Kode Program 5.2 Implementasi Algoritma Perhitungan *Binary Decision Tree*.

Berikut penjelasan dari Kode Program 5.2:

1. Baris ke-2 merupakan fungsi untuk memanggil *method* untuk mendapatkan nilai maksimal dari matriks *euclidean*. Parameter yang terdapat di dalam fungsi ini yaitu 0 yang berarti perulangan dimulai dari 0 sampai ke-3 untuk mendapatkan nilai maksimal dari kelas ke-1 sampai kelas ke-4. Kemudian terapat parameter lain seperti *maxLi* yaitu untuk mendapatkan nilai max yang ditampilkan di *GUI*, kemudian *indekKiri1*, dan *indekKanan1* juga untuk menampilkan di *GUI*.
2. Baris ke-3 untuk menampilkan di *console* kelas yang memasuki di *indekKiri* atau kelas positif dan kelas di *indekKanan* atau kelas negatif.
3. Baris ke-5 sampai baris ke-20 merupakan suatu percabangan untuk menyimpan hasil dari *indekKiri* dan *indekKanan* ke dalam file *.csv*. Pada baris ke-5 dan baris ke-6 menjelaskan bahwa jika *indekKiri* bernilai 1 maka akan menyimpan ke dalam *Kelas1.csv*, begitu juga seterusnya.
4. Baris ke-22 untuk memanggil fungsi *baris()* untuk mendapatkan nilai baris yang digunakan untuk mengetahui terletak dimanakah *indekKiri* maupun *indekKanan*.

5. Baris ke-23 untuk memanggil fungsi *bandingKelasMin()* untuk membandingkan nilai untuk mencari nilai minimal dari jarak maksimal yang telah terpilih.
6. Baris ke-24 yaitu fungsi untuk mencari nilai minimal terletak pada kelas ke berapa. Sehingga bisa menentukan node selanjutnya.
7. Baris ke-25 yaitu untuk menampilkan hasil minimal yang didapatkan untuk dicari nilai minimal.
8. Baris ke-26 sampai 27 untuk menampilkan hasil node yang menempati kelas positif dan negatif. kelas1 dan kelas2 adalah parameter yang menampung nilai yang digunakan untuk membandingkan agar dapat dicari nilai minimal dari kedua kelas tersebut.
9. Baris ke-28 adalah percabangan untuk membentuk pola *tree.* Untuk mengetahui dan menjalankan sesuai dengan nilai dari *indekKiri* dan *indekKanan.*
10. Baris ke-29 untuk menampilkan dan menjalankan fungsi untuk mendapatkan nilai minimal terdapat pada kela ke berapa.
11. Baris ke-32 sampai ke-39 adalah suatu kondisi jika kelas yang dicari berada pada indek ke-1 dan ke-2 tetapi menunjuk ke kelas-2 padahal kelas 2 sudah terdapat pada kelasnya, ini kemudian akan di set ke-3 (atau pada kelas ke-3).
12. Baris ke-40 sampai ke-50 adalah untuk menempatkan kelas terbaru jika indek terdapat pada kelas ke-1 dan seterusnya. Sehingga terciptalah satu gambaran *tree*.
13. Baris ke-51 adalah akhir dalam menentukan *tree* pada level ke-1. Untuk level selanjutnya sama saja tetapi nilai *max* nya akan hilang untuk satu kelas karena akan terdapat satu kelas sebagai kelas *fix*.

5.3.3 Implementasi Algoritma Perhitungan *Gravity Center*

Algoritma dari perhitungan *gravity center* adalah untuk menemukan titik pusat data dari setiap kelas, dapat dihitung dengan mencari nilai rata-rata setiap parameter atau fitur dari masing-masing data training pada setiap kelas. Pada perhitungan *gravity center* menghasilkan keluaran berupa matriks dengan *index nxn*, dan *n* yaitu banyaknya jumlah data *training*. Kode program dari algoritma *gravity center* dijelaskan pada Kode Program 5.3.

```
1   public float[][] gravityCenter1(float[][] data, int
      param, float[][] gravityCenter1, float [][] DataTraining)
2   {
3       float tempr = 0;
4       for (int h = 0; h < 1; h++) {
5          for (int i = 0; i < data.length; i++) {
6              tempr += data[i][param];
7           }
8           gravityCenter1[h][param]      =      tempr      /
frekuensi(DataTraining, 1);
```

```
9           }
10          for (int h = 0; h < 1; h++) {
11              System.out.print("\t"                                    +
    Float.valueOf(df.format(gravityCenter1[h][param])));
12          }
13          return data;
14      }
```

Kode Program 5.3 Implementasi Algoritma Perhitungan *Gravity Center*.

Berikut penjelasan dari Kode Program 5.3:

1. Baris ke-1 merupakan definisi fungsi *gravity center* untuk kelas ke-1 dengan tiga masukan yaitu data training kelas 1, parameter atau fitur, dan hasil perhitungan *gravity center*.
2. Baris ke-3 merupakan inisialisasi awal variabel *tempr* untuk penyimpanan data training.
3. Baris ke-4 sampai baris 7 menjelaskan tentang perulangan untuk data training untuk kelas ke-1 pada baris ke-*h* dan kolom ke-*i*. Kemudian disimpan di variabel *tempr*.
4. Baris ke-8 sampai 9 menjelaskan proses perhitungan untuk mencari nilai rata-rata dari setiap parameter yang disimpan di variabel *gravityCenter1*. Dengan cara variabel *tempr* dibagi dengan jumlah frekuensi data pada kelas ke-1.
5. Baris ke-10 sampai 14 menjelaskan untuk hasil tampilan dari matriks *gravity center*.

5.3.4 Implementasi Algoritma Perhitungan Jarak *Euclidean*.

Algoritma dari perhitungan jarak *euclidean* adalah untuk menemukan jarak dari titik pusat data yang telah didapatkan pada algoritma *gravity center* sehingga bisa mencari jarak terdekat dan terjauh dari setiap kelas. Dimana hasil dari nilai *euclidean* ini akan digunakan untuk proses *BDT*. Pada perhitungan jarak *euclidean* menghasilkan keluaran berupa matriks dengan indek *nxn*, dan *n* yaitu banyaknya jumlah kelas. Kode program dari algoritma jarak *euclidean* dijelaskan pada Kode Program 5.4.

```
1       public float [][] euclideanDistance(float[][] data) {
2           System.out.println("Matriks Euclidean Distance");
3           System.out.println();
4           float tempjarak;
5           float Hasil[][] = new
        float[data.length][data.length];
6           //Menampilkan Keterangan Parameter
7           for (int i = 0; i < data.length; i++) {
8               System.out.print("\tC" + (i + 1));
9           }
10          System.out.println();
11          for (int x = 0; x < data.length; x++) { //jumlah
        baris
12              //jumlah kolom atau parameter
13              for (int y = 0; y < data.length; y++) {
14                  tempjarak = 0;
```

```
15                  hasilTemp = 0;
16                  for (int z = 0; z < data[x].length; z++) {
17                      //rumus untuk euclidean
18                      tempjarak = (float) Math.pow((data[x][z]
    - data[y][z]), 2);
19                      hasilTemp += tempJarak;
20                      Hasil[x][y] = hasilTemp;
21                  }
22                  EuclideanDistance[x][y]       =       (float)
    Math.sqrt(Hasil[x][y]);
23              }
24          }
25          for (int x = 0; x < data.length; x++) {
26              //untuk membuat angka urutan data
27              System.out.print("C" + (x + 1) + "\t");
28              for (int y = 0; y < data.length; y++) {

29      System.out.print(Float.valueOf(df.format(EuclideanDistanc
        e[x][y])) + "\t");
30              }
31              System.out.println();
32          }
33          return EuclideanDistance;
34      }
```

Kode Program 5.4 Implementasi Algoritma Perhitungan *Gravity Center*.

Berikut penjelasan dari Kode Program 5.4:

1. Baris ke-1 merupakan definisi fungsi jarak *euclidean* dengan masukan yaitu data hasil *gravity center* semua kelas.
2. Baris ke-2 sampai 3 hanya menampilkan judul dari perhitungan ini.
3. Baris ke 4 sampai 5 menjelaskan inisialisasi awal variabel *tempjarak* dan h*asilTemp* yang digunakan untuk menyimpan hasil jarak.
4. Baris ke-6 sampai 10 menjelaskan untuk tampilan keterangan parameter.
5. Baris ke-11 sampai 24 menjelaskan perulangan untuk data *gravity center* ke-*x* dan ke-*y* untuk barisnya dan ke-*z* untuk mendapatkan nilai kolomnya. Kemudian pada baris ke- 16 sampai 19 merupakan fungsi untuk mendapatkan jarak euclidean.
6. Baris 25 sampai 32 menjelaskan perulangan untuk menampilkan matriks jarak euclidean yang disimpan di variabel *EuclideanDistance*.

5.3.5 Implementasi Algoritma Max Euclidean

Algoritma *max euclidean* adalah suatu proses untuk mencari nilai maksimal dari jarak euclidean. Ini merupakan tahap awal dalam pembentukan *tree.* Dibawah ini merupakan kode program untuk algoritma *max euclidean* yang dijelaskan pada Kode Program 5.5.

```
1   float maxBDT = 0;
2   for (int i = awalFor; i < akhirFor; i++) {
3       for (int y = awalFor; y < EuclideanDistance[i].length;
    y++) {
```

```
4           maxBDT = Math.max(maxBDT, EuclideanDistance[i][y]);
5       }
6   }
7   System.out.println("max = " + maxBDT);
8   indekKiri = mencariDenganLS(EuclideanDistance, maxBDT);
9   indekKanan = mencariDenganLS2(EuclideanDistance, maxBDT,
    indekKiri);
10  System.out.println("Nilai ditemukan pada indek : " +
    (indekKiri) + " dan " + (indekKanan));
11  maxL1.setText(String.valueOf(maxBDT));
12  indekKiri1.setText(String.valueOf(indekKiri));
13  indekKanan1.setText(String.valueOf(indekKanan));
14  return maxBDT;
```

Kode Program 5.5 Implementasi Algoritma *Max Euclidean*.

Berikut penjelasan dari Kode Program 5.5:

1. Baris ke-1 merupakan inisialisasi awal maxBDt = 0.
2. Baris ke-2 sampai 6 merupakan fungsi untuk mencari nilai *max* dari jarak euclidean.
3. Baris ke-7 untuk menampilkan hasil *max* dari jarak euclidean.
4. Baris ke-8 sampai 9 merupakan kode untuk mencari nilai *indekKiri* dan *indekKanan* berdasarkan *max euclidean* yang didapatkan. Sehingga bisa mengetahui *max euclidean* terdapat di kelas mana.
5. Baris ke-10 untuk menampilkan hasil *indekKiri* dan *indekKanan.*
6. Baris ke-11 sampai 13 untuk menampilkan di *GUI*.

5.3.6 Implementasi Algoritma Banding Kelas Minimal

Algoritma Banding Kelas Minimal adalah suatu proses untuk membandingkan 2 nilai jarak dengan membandingkan nilai yang paling terkecil dari jarak euclidean. Dibawah ini merupakan kode program untuk algoritma banding kelas minimal yang dijelaskan pada Kode Program 5.6.

```
1   float min = 0;
2   kelas1 = EuclideanDistance[kelasBaris1][indekKiri - 1];
3   kelas2 = EuclideanDistance[kelasBaris1][indekKanan - 1];

4   if (kelas1 < kelas2) {
5       min = kelas1;
6   } else if (kelas2 < kelas1) {
7       min = kelas2;
8   }
9   System.out.println("min " + min);
10  return min;
```

Kode Program 5.6 Implementasi Algoritma Banding Kelas Minimal.

Berikut penjelasan dari Kode Program 5.6:

1. Baris ke-1 merupakan inisialisasi awal *min = 0*.
2. Baris ke-2 sampai 3 untuk insialisasi variabel *kelas1* dan *kelas2*.

3. Baris ke-4 sampai 8 adalah proses untuk mencari nilai minimal dari *kelas1* dan *kelas2*.
4. Baris ke-9 untuk menampilkan hasil *min*.

5.3.7 Implementasi Algoritma Cari Kelas

Algoritma cari kelas adalah suatu proses untuk mencari kelas berdasarkan min yang telah didapatkan. Jadi min itu terletak di kelas berapa. Dibawah ini merupakan kode program untuk algoritma cari kelas yang dijelaskan pada Kode Program 5.7.

```
1  for (int j = 0; j < larikC.length; j++) {
2      for (int k = 0; k < larikC.length; k++) {
3          if (nilaiDicari == larikC[j][k])
4              return j + 1;
5          }
6      }
7  }
8  // Bila tidak ditemukan kecocokan
9  return -1;
```

Kode Program 5.7 Implementasi Algoritma Cari Kelas.

Berikut penjelasan dari Kode Program 5.7:

1. Baris ke-1 sampai 7 merupakan perulangan untuk mencari letak dimana min tersebut berada. Jika nilai yang dicari sama atau cocok maka *j+1* dan itu adalah letak dari kelasnya.
2. Baris ke-9 jika tidak ditemukan kecocokan maka *return* -1.

5.3.8 Implementasi Algoritma Alokasi Kelas

Algoritma alokasi kelas adalah suatu proses untuk menampatkan kelas ke kelas positif atau negatif. Di bawah ini merupakan kode program untuk algoritma alokasi kelas yang dijelaskan pada Kode Program 5.8.

```
1  if                              (mencariMin(EuclideanDistance,
   bandingKelasMin(kelasBaris1), indekKiri) == indekKanan) {
2      return -1;
3  } else {
4      return 1;
5  }
```

Kode Program 5.8 Implementasi Algoritma Alokasi Kelas.

Berikut penjelasan dari Kode Program 5.8:

1. Baris ke-1 sampai 5 merupakan perulangan untuk menempatkan kelas ke kelas positif atau negatif. Jika nilai *min* ditemukan dan sama dengan nilai pada *indekKanan* maka kelas akan masuk kelas negatif, tetapi jika tidak sama dengan *indekKanan* maka akan masuk ke kelas positif.

5.3.9 Implementasi Algoritma Perhitungan *Sequential Training* SVM

Proses algoritma *Sequential Training* adalah salah satu metode dalam *SVM* untuk mendapatkan nilai *hyperplane* optimal. Dimana pada tahap ini akan melakukan iterasi dengan melakukan perhitungan untuk mencari nilai *Ei*, nilai *delta alpha*, dan setelah mendapatkan nilai *delta alpha* kemudian dicari nilai *alpha*. Kode program dari algoritma perhitungan matriks *hessian* dijelaskan pada Kode Program 5.9.

```
1   public float[] Sequential_Training_SVM(float[] ai,
    float gamma, float complexity, int iterasiMax, float
    epsilon) {
2       ai = new float[HasilMatriksHessian.length];
3       System.out.println();
4       Learning_Rate(gamma);
5       //isi matriks alpha_i
6       for (int i = 0; i < HasilMatriksHessian.length;
    i++) {
7           ai[i] = 0;
8       }
9       //inisialisasi variabel stop
10      int iterasiStop = 0;
11      int x = 0;
12      float tempTotal = 0;

13      while (x <= iterasiMax) {
14          System.out.println();
15          System.out.println();
16          System.out.println("ITERASI " + (x));
17          //untuk mendapatkan nilai Ei
18          E_i(ai);
19          //untuk mendapatkan nilai δαi
20          δαi(ai);
21          //untuk mendapatkan nilai αi
22          αi(ai);

23          if (maxDeltaAlpha_i <= epsilon) {
24              iterasiStop = x;
25              x = iterasiMax;
26          } else {
27              iterasiStop = x;
28              x++;
29          }
30      }
31      System.out.println();

32  System.out.println("_____
    _");
33      System.out.println("Iterasi berhenti pada iterasi
    ke-" + iterasiStop);
        System.out.println("-----------------------------
34  -----");
```

```
35      return ai;
36  }
```

Kode Program 5.9 Implementasi Algoritma Perhitungan *Sequential Training SVM*.

Berikut penjelasan dari Kode Program 5.9.

1. Baris ke-1 menjelaskan fungsi dari *sequential training* dimana terdapat satu inputan nilai *alpha* ke-*i*.
2. Baris ke-2 menjelaskan inisialisasi variabel *alpha*.
3. Baris ke-4 yaitu memanggil fungsi *LearningRate()* untuk mendapatkan nilai *learning rate*.
4. Baris ke-6 sampai 8 untuk menginisialisasi variabel *alpha* menjadi 0.
5. Baris ke-9 sampai 12 untuk menginisialisasi variabel yang digunakan dalam *looping* jumlah iterasi.
6. Baris ke-13 menjelaskan perulangan dengan kondisi bahwa nilai *x* atau jumlah iterasi lebih kecil dari *iterasiMax* baru iterasi akan berhenti.
7. Baris ke-18 menjelaskan untuk memanggil *method Ei*.
8. Baris ke-20 menjelaskan untuk memanggil *method delta alpha*.
9. Baris ke-22 menjelaskan untuk memanggil *method alpha.*
10. Baris ke-23 sampai 30 menjelaskan kondisi agar iterasi menjadi berhenti dan menyimpan nilai *x* terakhir untuk menampilkan jumlah iterasi dan *x* akan bertambah sesuai dengan perulangan.
11. Baris ke-32 sampai 34 menjelaskan untuk menampilkan jumlah iterasi.

5.3.10 Implementasi Algoritma Perhitungan Kernel *Polynomial Degree d*

Algoritma dari perhitungan kernel *polynomial degree d* merupakan proses awal perhitungan dengan menggunakan metode *SVM* yang melibatkan *dot product* antar 2 data *training* pada ruang vektor yang berdimensi. Perhitungan kernel yang digunakan yaitu *Polynomial Degree d* yang nantinya keluaran yang dihasilkan berupa matriks kernel dengan index *nxn*, dengan *n* yaitu banyaknya jumlah data *training*. Kode program dari algoritma *kernel polynomial degree d* dijelaskan pada Kode Program 5.10.

```
1   for (int x = 0; x < data.length; x++) { //jumlah baris
2       for (int y = 0; y < data.length; y++) { //jumlah kolom
    atau parameter
3           hasil = 0;
4           for (int z = 0; z < data[x].length; z++) {
5               //rumus untuk kernel
6               hasil += (data[x][z] * data[y][z]);
7               HasilKernel[x][y] = hasil;
8           }
9           HasilAkhirKernel[x][y]          =          (float)
    Math.pow(HasilKernel[x][y], d);
```

```
10          }
11      }
```

Kode Program 5.10 Implementasi Algoritma Perhitungan Kernel *Polynomial Degree d*.

Berikut penjelasan dari Kode Program 5.10.

1. Baris ke-1 sampai 4 merupakan perulangan untuk inputan sebanyak data *training* ke-*x*, *y*, dan *z*. Perulangan ke-*x* dan *y* sebanyak jumlah baris dari data *training*. Karena hasil matriks kernel ini memiliki dimensi data sesuai dengan jumlah baris pada data *training*, sedangkan perulangan ke-*z* untuk menelusuri kolom dari data *training*.
2. Baris ke-5 sampai 10 merupakan proses perhitungan dari kernel ini. Dimana akan mengalikan data *training* dikali dengan data *training* sesuai dengan jumlah ordo baris dan kolom yang sesuai kemudian hasilnya akan dijumlahkan dan disimpan ke variabel *HasilKernel*. Pada baris ke-9, hasil dari *HasilKernel* akan dipangkatkan sejumlah d yang diinputkan dan hasil akhir disimpan di variabel *HasilAkhirKernel*.

5.3.11 Implementasi Algoritma Perhitungan Matriks *Hessian*

Proses perhitungan matriks hessian dilakukan untuk mencari nilai hessian yang nanti akan digunakan untuk perhitungan yang ada pada proses algoritma *Sequential Training SVM*. Variabel dari perhitungan matriks hessian didapatkan dari hasil perhitungan fungsi kernel *Polynomial Degree d* dan nilai dari konstanta *lambda*. Kode program dari algoritma perhitungan matriks hessian dijelaskan pada Kode Program 5.11.

```
1   public float [][] Matriks_Hessian(float[] kelasbaru,
    float[][] HasilAkhirKernel, float lamda) {
2       HasilMatriksHessian = new
    float[HasilAkhirKernel.length][HasilAkhirKernel.length];
3       System.out.println();
4       System.out.println("Matriks Hessian:");

5       System.out.print("\t");
6       System.out.println();
7       //Menampilkan Keterangan Parameter
8       for (int i = 0; i < HasilAkhirKernel.length; i++) {
9           System.out.print("    \t \t    X" + (i + 1));
10      }
11      System.out.print("\t\tKELAS BARU");
12      System.out.println();
13      for (int x = 0; x < HasilAkhirKernel.length; x++) {
    //jumlah baris
14          for (int y = 0; y < HasilAkhirKernel.length; y++) {
    //jumlah kolom atau parameter
15              HasilMatriksHessian[x][y] = kelasbaru[y] *
    kelasbaru[x] * (HasilAkhirKernel[x][y] + ((float)
    Math.pow(lamda, 2)));
16          }
17      }
```

```
18      for (int x = 0; x < HasilAkhirKernel.length; x++) {
19          //untuk membuat angka urutan data
20          System.out.print(" " + (x + 1) + "\t");
21          for (int y = 0; y < HasilAkhirKernel.length; y++) {

22  System.out.print((HasilMatriksHessian[x][y]) + "\t");
23          }
24          System.out.print("\t" + kelasbaru[x]);
25          System.out.println();
26      }
27      return HasilMatriksHessian;
28  }
```

Kode Program 5.11 Implementasi Algoritma Perhitungan Matriks *Hessian*.

Berikut penjelasan dari Kode Program 5.11.

1. Baris ke-1 merupakan fungsi dari matriks *hessian* dimana terdapat dua inputan yaitu inputan kelas baru dari hasil *BDT* dan hasil kernel.
2. Baris ke-2 menjelaskan inisialisasi variabel *HasilMatriksHessian* untuk menampung nilai dari hasil matriks *hessian*.
3. Baris ke-3 sampai 6 menjelaskan untuk menampilkan tulisan "Matriks Hessian".
4. Baris ke-13 sampai 14 menjelaskan perulangan berdasarkan jumlah hasil kernel ke-*x* dan *y*.
5. Baris ke-15 menjelaskan proses perhitungan dari matriks hessian. Dimana akan mengalikan kelas baru ke-*x* dikali dengan kelas baru ke-*y* dan kemudian dikalikan dengan hasil kernel ditambah dengan nilai *lambda* yang dipangkatkan. Hasil tersebut disimpan di variabel *HasilMatriksHessian*.
6. Baris ke-18 sampai 26 menjelaskan proses untuk menampilkan hasil matriks hessian.

5.3.12 Implementasi Algoritma Perhitungan *Learning Rate*

Proses algoritma *Learning Rate* adalah untuk mendapatkan nilai *learning rate* yang akan digunakan untuk perhitungan mencari nilai *delta alpha*. Kode program dari algoritma perhitungan *learning rate* dijelaskan pada Kode Program 5.12.

```
1   public void Learning_Rate(float gamma) {
2       System.out.println();
3       System.out.println("Learning rate:");
4       float MaxHessian = 0;
5       //cari nilai maksimal dari matriks hessian
6       for (int x = 0; x < HasilMatriksHessian.length; x++) {
7   MaxHessian                  =                  Math.max(MaxHessian,
    HasilMatriksHessian[x][x]);
8       }
9       //fungsi untuk mendapatkan nilai learning rate atau
    gamma
10      gammaLearningRate = gamma / MaxHessian;
11      System.out.println("Max Dij = " + MaxHessian);
```

```
12      System.out.println("Learning rate = " +
        (gammaLearningRate));
13  }
```

Kode Program 5.12 Implementasi Algoritma Perhitungan *Learning Rate*.

Berikut penjelasan dari Kode Program 5.12.

1. Baris ke-1 menjelaskan *method* untuk *learning rate*.
2. Baris ke-4 menjelaskan inisialisasi variabel *MaxHessian*.
3. Baris ke-6 sampai 8 menjelaskan perulangan untuk mencari nilai maksimal dari matriks hessian pada data ke-x.
4. Baris ke-10 menjelaskan untuk mendapatkan hasil akhir dari *learning rate* dengan membagi variabel *gamma* dengan hasil maksimal dari matriks hessian.

5.3.13 Implementasi Algoritma Perhitungan *Ei*

Proses algoritma *Ei* adalah proses yang pertama kali dilakukan pada tahap iterasi. Nilai *Ei* didapatkan dengan menjumlahkan hasil perkalian dari matriks *hessian* dengan nilai *alpha*. Kode program dari algoritma perhitungan *Ei* dijelaskan pada Kode Program 5.13.

```
1   for (int x = 0; x < HasilMatriksHessian.length; x++) {
2       for (int y = 0; y < HasilMatriksHessian.length; y++) {
3           Ei[x][y] = (HasilMatriksHessian[x][y] * ai[x]);
4       }
5   }
6   for (int x = 0; x < HasilMatriksHessian.length; x++) {
7       for (int y = 0; y < HasilMatriksHessian.length; y++) {
8           HasilEi[x] += Ei[x][y];
9       }
10  }
```

Kode Program 5.13 Implementasi Algoritma Perhitungan *Ei*.

Berikut penjelasan dari Kode Program 5.13.

1. Baris ke-1 sampai 5 menjelaskan perulangan untuk mendapatkan nilai *Ei* sesuai dengan jumlah ke-*x* dan *y*. Pada baris ke-3 melakukan perhitungan dengan mengalikan hasil matriks hessian dengan nilai *alpha*.
2. Baris ke-6 sampai 10 menjelaskan perulangan mendapatkan hasil akhir dari *Ei* yaitu hasil dari baris ke-3 akan dijumlahkan sehingga mendapatkan nilai *Ei* yang disimpan di variabel *HasilEi*.

5.3.14 Implementasi Algoritma Perhitungan *Delta Alpha*

Proses algoritma *Ei* adalah proses kedua pada tahap iterasi. Nilai *delta alpha* yaitu perkalian antara gamma dengan hasil *(1-Ei)* dan dibandingkan dengan nilai *alpha* ke-*i* untuk dicari nilai paling maksimum. Hasil dari nilai maksimum nantinya akan dibandingkan dengan hasil dari variabel *C* yang

dikurangi dengan alpha ke-*i*. Kode program dari algoritma perhitungan *delta alpha* dijelaskan pada Kode Program 5.14.

```
1   for (int i = 0; i < HasilMatriksHessian.length; i++) {
2               δαi[i] = Math.min(Math.max(gammaLearningRate *
    (1 - HasilEi[i]), -ai[i]), complexity - ai[i]);
3   }

4   for (int i = 0; i < HasilMatriksHessian.length; i++) {
5   maxDeltaAlpha_i = Math.max(maxDeltaAlpha_i, δαi[i]);
6   }
7   for (int i = 0; i < HasilMatriksHessian.length; i++) {
8       System.out.print("\t" + (δαi[i]));
9   }
10  System.out.println();
11  System.out.println();
12  System.out.println("Max δαi=" + (maxDeltaAlpha_i));
```

Kode Program 5.14 Implementasi Algoritma Perhitungan *Delta Alpha*.

Berikut penjelasan dari Kode Program 5.14.

1. Baris ke-1 sampai 3 menjelaskan perulangan untuk mendapatkan nilai *delta alpha* sebanyak *i*.
2. Baris ke-4 sampai 6 untuk mencari nilai maksimum dari hasil *delta alpha*.
3. Baris ke-7 sampai 9 menampilkan hasil *delta alpha*.
4. Baris ke-12 menampilkan hasil maksimum dari *delta alpha*.

5.3.15 Implementasi Algoritma Perhitungan *Alpha*

Proses algoritma *alpha* adalah proses ketiga pada tahap iterasi. Perhitungan nilai *delta alpha* merupakan proses penambahan nilai dari delta alpha ke-*i* pada proses sebelumnya. Kode program dari algoritma perhitungan *alpha* dijelaskan pada Kode Program 5.15.

```
1   for (int i = 0; i < HasilMatriksHessian.length; i++) {
2       ai[i] = ai[i] + δαi[i];
3   }
```

Kode Program 5.15 Implementasi Algoritma Perhitungan *Alpha*.

Berikut penjelasan dari Kode Program 5.15.

1. Baris ke-1 sampai 3 menjelaskan proses perulangan sampai ke-i untuk mendapatkan nilai dari *alpha* dengan menjumlahkan nilai *alpha* awal dengan hasil *delta alpha* dan hasilnya disimpan di variabel *ai*.

5.3.16 Implementasi Algoritma *Support Vector*

Proses algoritma *support vector* adalah nilai *alpha* pada iterasi terakhir. Kode program dari algoritma perhitungan *support vector* dijelaskan pada Kode Program 5.16.

```
1   public float[] supportVectorL1(float[] kelasbaru, float[]
    ai) {
2       System.out.println();
```

```
3       System.out.println("Support Vector:");
4       //Menampilkan Keterangan Parameter
5       for (int i = 0; i < HasilMatriksHessian.length; i++) {
6           System.out.print("\t\t\tX" + (i + 1));
7       }
8        System.out.println();
9        for (int i = 0; i < HasilMatriksHessian.length; i++) {
10           System.out.print("\t" + (ai[i]));
11      }
12       System.out.println();
13       for (int i = 0; i < HasilMatriksHessian.length; i++) {
14          System.out.print("\t\t\t" + kelasbaru[i]);
15      }
16      //mengosongkan isi tabel
17      try {
18          String trunclt = "TRUNCATE TABLE support_vector";
19          int truclts = Koneksi.execute(trunclt);
20          //insert data ai ke dalam tabel
21          for (int i = 0; i < HasilMatriksHessian.length; i++)
22  {
23              String query = "INSERT INTO
    support_vector(id_pasien,ai,kelas) VALUES(" + (i + 1) + ","
    + ai[i] + "," + kelasbaru[i] + ")";
24              int status_ins = Koneksi.execute(query);
25          }
26      } catch (Exception truclts) {
27          JOptionPane.showMessageDialog(null, "Gagal, Koneksi
    Database Bermasalah atau Penulisan Perintah Salah", "Pesan",
28          JOptionPane.WARNING_MESSAGE, new
    ImageIcon("C:\\Users\\sony\\Documents\\NetBeansProjects\\Bac
    aText1\\Gambar\\x.png"));
29      }

30      return ai;
31  }
```

Kode Program 5.16 Implementasi Algoritma Perhitungan *Alpha*.

Berikut penjelasan dari Kode Program 5.16.

1. Baris ke-1 merupakan fungsi *support vector machine*.
2. Baris ke-2 sampai 3 untuk menampilkan judul *support vector* di *console.*
3. Baris ke-4 sampai 8 untuk menampilkan parameter di *console*.
4. Baris ke- 9 sampai 12 untuk menampilkan nilai *alpha* pada iterasi terakhir di *console*.
5. Baris ke-13 sampai 15 untuk menampilkan kelas dari hasil BDT.
6. Baris ke-17 sampai 19 untuk fungsi *truncate* di database sehingga isi *database* akan selalu kosong saat program dijalankan.
7. Baris ke-21 sampai 25 untuk fungsi memasukkan data *alpha* ke dalam database.
8. Baris ke-26 sampai 29 untuk menampilkan pesan *error* saat proses *insert* gagal.

5.3.17 Implementasi Algoritma Perhitungan *W.X*$^{+}$ *dan W.X*$^{-}$

Proses algoritma mendapatkan nilai bobot bernilai positif dengan menggunakan hasil dari perhitungan kernel yang memiliki kelas positif dengan nilai *alpha* tertinggi di kelas positif, sedangkan bobot negatif mencari nilai kernel dari nilai *alpha* tertinggi dari kelas negatif. Kode program dari algoritma perhitungan *w.x*$^{+}$ dan *w.x*$^{-}$ dijelaskan pada Kode Program 5.17.

```
1   for (int i = 0; i < HasilAkhirKernel.length; i++) {
2       for (int x = indekPos; x < indekPos + 1; x++) {
3           xixplus[i][x] = HasilAkhirKernel[i][x];
4       }
5   }

6   for (int i = 0; i < HasilAkhirKernel.length; i++) {
7       for (int x = indekNeg; x < indekNeg+1; x++) {
8           xixminus[i][x] = HasilAkhirKernel[i][x]; //
9       }
10  }

11  //-----------------------------------------------------
12  public int maxAlphaPos(int id_pasien, String SQL) {
13      System.out.println();
14      int kelas =0;
15      float MaxAlphapos = 0;
16      id_pasien = 0;
17      System.out.println("Maksimal αi+:");
18      try {
19          ResultSet kr = Koneksi.executeQuery(SQL);
20          while (kr.next()) {
21              id_pasien                                    =
    Integer.parseInt(kr.getString("id_pasien"));
22              MaxAlphapos                                  =
    Float.parseFloat(kr.getString("ai"));
23              kelas = Integer.parseInt(kr.getString("kelas"));
24          }
25       } catch (Exception kr) {
26            JOptionPane.showMessageDialog(null,  "Gagal,  Tidak
    bisa            Membaca            Data",            "Pesan",
    JOptionPane.WARNING_MESSAGE,                              new
    ImageIcon("C:\\Users\\sony\\Documents\\NetBeansProjects\\Bac
    aText1\\Gambar\\x.png"));
27          }//ctch
28          System.out.println("Max Alpha di Indek " + id_pasien
    + " = " + MaxAlphapos+ "pada Kelas "+kelas);

29       return id_pasien;
30  }
```

Kode Program 5.17 Implementasi Algoritma Perhitungan *W.X*$^{+}$ *dan W.X*$^{-}$.

Berikut penjelasan dari Kode Program 5.17.

1. Baris ke-1 sampai 5 untuk mendapatkan nilai bobot positif yang disimpan di variabel *xixplus* sesuai perulangan ke-*indekPos*.
2. Baris ke-6 sampai 10 untuk mendapatkan nilai bobot negatif yang disimpan di variabel *xixminus* sesuai perulangan ke-*indekNeg*.

3. Baris ke-11 merupakan method untuk mendapatnilai nilai *indekPos* maupun *indekNeg* dengan nilai pengembalian id_pasien.
4. Baris ke-14 sampai 16 menjelaskan inisialisasi awal untuk variabel yang berguna untuk membaca data di *database*.
5. Baris ke-18 sampai 27 menjelaskan proses menampilkan data dari *database* untuk diambil nilai *maxAlpha* dari *id_pasien* yang nantinya akan digunakan untuk mengetahui indek dari hasil kernel.
6. Baris ke-28 menjelaskan untuk tampilan nilai indek *maxAlpha*.

5.3.18 Implementasi Algoritma *alphakiplus* atau *alphakiminus*

Proses algoritma *jumlahalphakminus* atau *jumlahalphakplus* adalah proses untuk mendapatkan nilai *jumlahalphakminus* atau *jumlahalphakplus*. Kode program dari algoritma perhitungan *jumlahalphakminus* atau *jumlahalphakplus* dijelaskan pada Kode Program 5.18.

```
1   float[][] αiyikminus = new
2   float[HasilAkhirKernel.length][HasilAkhirKernel.length];
3   float tempr = 0;
4   System.out.println();
5   System.out.println("\tαiyikminus");
6   for (int i = 0; i < HasilAkhirKernel.length; i++) {
7       for (int x = indekNeg; x < indekNeg + 1; x++) {
8           αiyikminus[i][x] = ai[i] * kelasbaru[i] *
    xixminus[i][x];
9           tempr += αiyikminus[i][x];
10          jumlahαiyikminus = tempr;
11      }
12  }
13  System.out.println("\t--------------------+");
14  System.out.println("\tJumlah = " + (jumlahαiyikminus));

15  return jumlahαiyikminus;
```

Kode Program 5.18 Implementasi Algoritma Perhitungan *Jumlahalphakminus* atau *Jumlahalphakplus*.

Berikut penjelasan dari Kode Program 5.18.

1. Baris ke-1 sampai ke 3 merupakan inisialisasi variabel.
2. Baris ke-4 sampai 5 untuk menampilkan judul di *console*.
3. Baris ke-6 sampai 12 merupakan perulangan untuk melakukan proses perhitungan sehingga mendapatkan nilai *jumlahalphaikminus*. Ini juga berlaku untuk *jumlahalphaikplus*.
4. Baris ke-13 sampai 14 untuk menampilkan hasil.

5.3.19 Implementasi Algoritma Perhitungan *b* atau bias

Perhitungan nilai bias membutuhkan total jumlah bobot yang didapatkan dari perkalian antara nilai *alpha* ke-*i*, nilai kelas baru, dan hasil bobot positif dan

negatif. Kedua bobot yang didapat yaitu bobot posisif ($W.X^{+}$) dan bobot negatif ($W.X^{-}$) kemudian dijumlahkan dan dikalikan dengan minus setengah sehingga menghasilkan nilai bias. Kode program dari algoritma perhitungan bias dijelaskan pada Kode Program 5.19.

```
1   //untuk mendapat nilai b atau bias
2   public float b(float b, float jumlahαiyikminus, float
    jumlahαiyikplus) {
3       float jumlah;
4       //hitung b
5       jumlah = (jumlahαiyikplus) + (jumlahαiyikminus);
6       System.out.println("");
7       System.out.println("\tJumlah = (" + (jumlahαiyikplus) +
    ") + (" + (jumlahαiyikminus) + ") = " + (jumlah));

8       b = (float) -0.5 * jumlah;
9       System.out.println();
10      System.out.println("\tb=" + (b));

11      return b;
12  }
```

Kode Program 5.19 Implementasi Algoritma Perhitungan bias atau *b.*

Berikut penjelasan dari Kode Program 5.19.

1. Baris ke-2 menjelaskan method untuk nilai bias dengan dua inputan yaitu jumlah nilai perkalian antara *alpha*, nilai kelas dan bobot positif maupun negatif dan nilai *b*.
2. Baris ke-3 menjelaskan inisialisasi awal variabel *jumlah*.
3. Baris ke-5 menjelaskan perhitungan dengan menambahkan nilai perkalian antara *alpha*, nilai kelas dan bobot positif maupun negatif dan disimpan di variabel jumlah.
4. Baris ke-8 menjelaskan perhitungan nilai *b*.
5. Baris ke-9 sampai 10 menampilkan hasil *b*.

5.3.20 Implementasi Algoritma Perhitungan *f(x) Testing*

Perhitungan nilai *f(x) testing* pada tahap pengujian dengan menggunakan data testing dengan parameter nilai *bias* dan nilai *alpha*. Kemudian hasil dari fungsi ini adalah bernilai negatif atau posifif. Jika bernilai negatif maka akan masuk ke kelas negatif begitupun sebaliknya hingga mencapai level terakhir dan mencapai semua data *testing* telah mendapatkan kelas. Kode program dari algoritma perhitungan *f(x) testing* dijelaskan pada Kode Program 5.20.

```
    //method kernel testing
1   ....
2   for (int x = 0; x < data.length; x++) { //jumlah baris
3       for (int y = 0; y < dataUji.length; y++) { //jumlah
    kolom atau parameter
5           hasil = 0;
6           for (int z = 0; z < data[x].length; z++) {
7               //rumus untuk kernel
```

```
8               hasil += (data[x][z] * dataUji[y][z]);
9               HasilKernel[x][y] = hasil;
10          }
11          HasilAkhirKernelTesting[x][y]       =       (float)
    Math.pow(HasilKernel[x][y], d);
12      }
13  }

    //method dot product alpha, kelas, dan kernel testing
14  ....
15  for (int i = 0; i < dataTr.length; i++) {
16      for (int j = 0; j < dataUji.length; j++) {
17          nilai_alphaiyk_1[i][j]  =  (ai[i]  *  kelasbaru1[i]  *
    HasilAkhirKernelTesting[i][j]);
18          sigmaTotal[j] += nilai_alphaiyk_1[i][j];
19      }
20  }

    //method fxTesting
21  ....
22  for (int j = 0; j < dataUji.length; j++) {
23  .               nilai_fx_1[j] = sigmaTotal[j] + b;
24          }
    ....
```

Kode Program 5.20 Implementasi Algoritma Perhitungan *f(x) testing.*

Berikut penjelasan dari Kode Program 5.20.

1. Baris ke-2 sampai 13 menjelaskan proses mendapatkan nilai kernel *testing* menggunakan kernel *polynomial degree d*. Proses nya sama dengan perhitungan kernel, yang berbeda adalah *dot product* pada kernel saat tahap *training* adalah dengan data *training*, sedangkan tahap tetsing *dot product* dengan data *training* dan data *testing*. Sehingga mendapatkan nilai kernel testing yang disimpan di variabel *HasilAkhirKernelTesting*.
2. Baris ke-14 sampai 20 merupakan perulangan untuk menghitung *dot product* antara nilai *alpha*, kelas, dan kernel *testing*. Baris ke-17 menghitung *dot product* dan disimpan di variabel *nilai_alphaiyk_1* kemudian pada baris ke-18 hasil dari perkalian tersebut dijumlahkan dan disimpan di variabel *sigmaTotal*.
3. Baris ke-21 sampai 24 merupakan method *fxTesting* dan menjelaskan perulangan untuk mendapatkan hasil akhir *f(x)* dengan menjumlahkan hasil *sigmaTotal* dengan nilai *bias*.

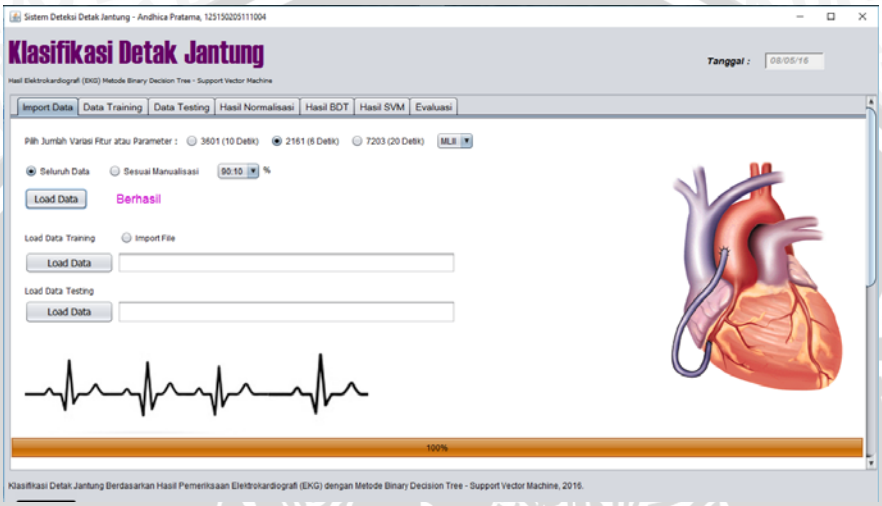
5.4 Implementasi Antarmuka

Implementasi antarmuka atau *interface* merupakan penghubung antara *user* dan sistem secara langsung. *Interface* dari sistem Klasifikasi Kondisi Detak Jantung berdasarkan Hasil Pemeriksaan Elektrokardiografi (EKG) dengan menggunakan *Binary Decision Tree-Support Vector Machine (BDT-SVM)* terdiri dari halaman *import* data, halaman liat data *training* dan data *testing*, halaman

normalisasi, halaman hasil *generate tree* untuk pembentukan kelas baru, halaman hasil *SVM*, dan halaman evaluasi.

5.4.1 Halaman *Import* Data

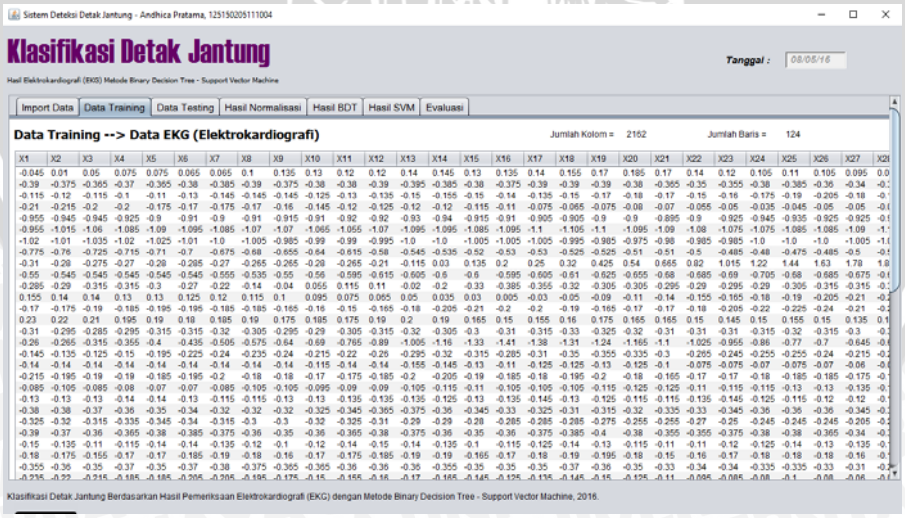
Implementasi halaman *import* data merupakan antarmuka untuk mendukung proses perhitungan *SVM*, yang mana *import* data untuk mengambil data dari *file csv* dan data yang diambil adalah data *training* dan data *testing*. Implementasi halaman *import* data terdapat pada Gambar 5.1 di bawah ini.



Gambar 5.1 Perancangan Halaman *Import* Data.

Gambar 5.1 menjelaskan antarmuka *import* data dimana mengambil data dari *file csv* untuk di gunakan pada proses klasifikasi ini.

5.4.2 Halaman Lihat Data *Training* dan Data *Testing*



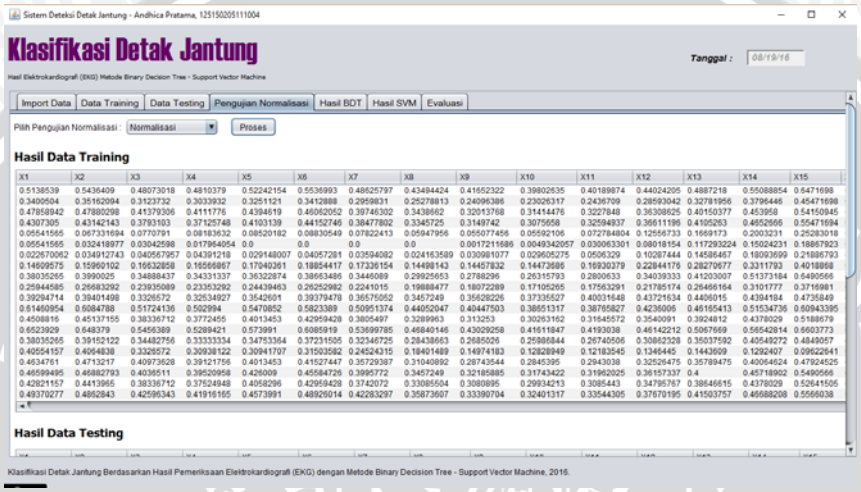
Gambar 5.2 Halaman Data Training.

Implementasi halaman lihat data *training* dan *testing* merupakan antarmuka untuk menampilkan hasil data *training* dan data *testing* yang telah

berhasil di *import* di sistem. Pada Gambar 5.2 menjelaskan antarmuka *load* data dimana mengambil data dari *file csv* untuk di gunakan pada proses klasifikasi ini. Untuk tampilan halaman data *testing* yaitu sama seperti halaman data *training* tetapi dengan *tap* yang berbeda.

5.4.3 Halaman Normalisasi

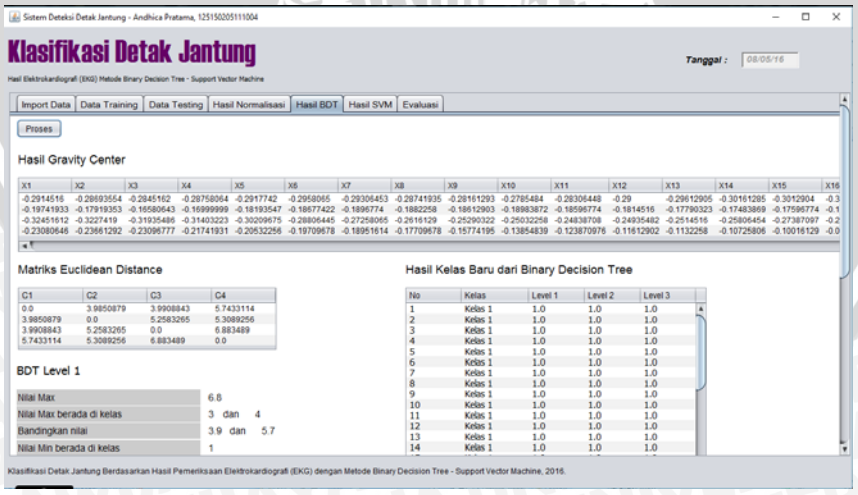
Implementasi halaman normalisasi merupakan antarmuka untuk menampilkan hasil perhitungan normalisasi data *training* dan data *testing*. Implementasi halaman normalisasi terdapat pada Gambar 5.3 di bawah ini.



Gambar 5.3 Halaman Normalisasi.

Gambar 5.3 menjelaskan antarmuka hasil normalisasi data dari data yang telah berhasil di *import* dari *csv* untuk di gunakan pada proses klasifikasi ini. Halaman ini bisa memilih untuk melakukan normalisasi atau tidak.

5.4.4 Halaman Hasil *Generate Tree* (*Binary Decision Tree*)

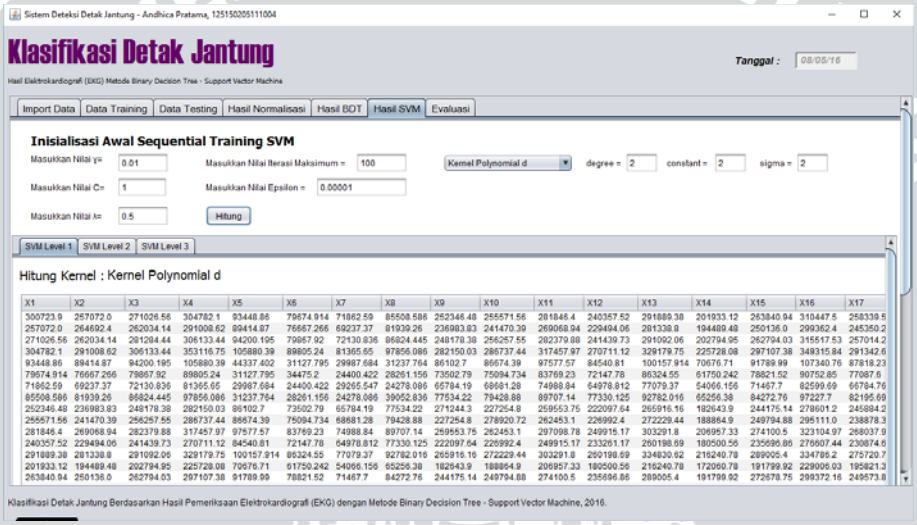


Gambar 5.4 Halaman Hasil BDT.

Gambar 5.4 menjelaskan antarmuka dari proses *generate tree* untuk menghasilkan kelas baru agar bisa digunakan pada perhitungan SVM. Halaman ini akan menampilkan hasil perhitungan *gravity center*, matriks jarak *euclidean*, dan kelas baru yang terbentuk. Implementasi pada halaman *binary decision tree* ini merupakan antarmuka atau *interface* untuk menampilkan hasil dari pembentukan *tree* untuk mendapatkan kelas baru.

5.4.5 Halaman Hasil SVM

Implementasi pada halaman *SVM* ini merupakan antarmuka atau *interface* untuk menampilkan hasil perhitungan dari *SVM* untuk level 1, level 2, dan level 3. Pada halaman ini akan mengisi inisialisasi awal untuk perhitungan dan menampilkan tabel hasil perhitungan seperti kernel, matriks *hessian*, nilai *support vector*, dan nilai bias. Implementasi halaman hasil *SVM* terdapat pada Gambar 5.5.

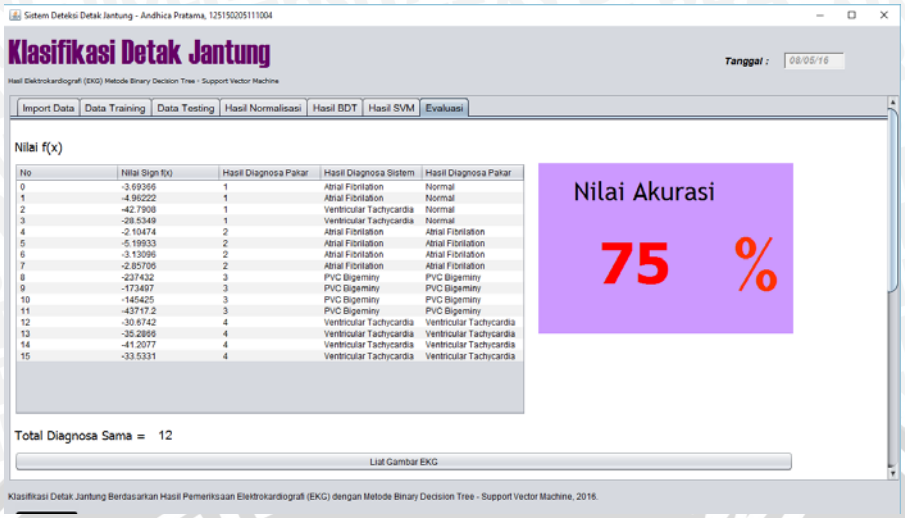


Gambar 5.5 Halaman Hasil SVM.

Gambar 5.5 menjelaskan antarmuka dari proses hasil *SVM* untuk mendapatkan hasil nilai *alpha* dan nilai bias agar bisa mendapatkan hasil klasifikasi. Antarmuka tersebut terdapat inisialisasi yang harus diimplementasikan. Kemudian terdapat pemilihan jenis kernel dan menampilkan hasil *SVM* level 1 hingga level 3.

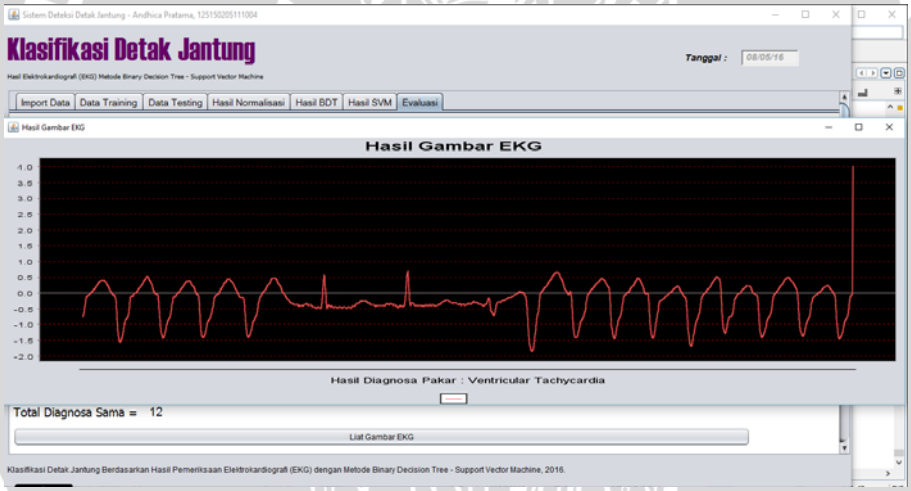
5.4.6 Halaman Evaluasi

Implementasi halaman evaluasi adalah antarmuka atau *interface* untuk menampilkan hasil perhitungan dari *f(x) testing* untuk menentukan sebuah kelas dari metode *SVM* dan menampilkan hasil akurasi dengan membandingkan hasil yang didapatkan oleh sistem serta hasil kelas awal yang telah ditetapkan oleh pakar. Selain itu pada halaman ini juga bisa menampilkan hasil gambar EKG dari setiap data *testing* yang dilipih. Implementasi antarmuka halaman evaluasi terdapat pada Gambar 5.6.



Gambar 5.6 Halaman Hasil Evaluasi.

Pada halaman evaluasi ini juga terdapat tombol untuk menampilkan gambar EKG berdasarkan data *testing* yang dimasukkan. Berikut ini merupakan Gambar 5.7 untuk menampilkan gambar EKG.



Gambar 5.7 Halaman Hasil Gambar EKG.