BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi perangkat lunak berdasarkan hasil yang sudah diperoleh dari analisis kebutuhan dan proses perancangan perangkat lunak pada bab III. Bab ini membahas tentang penjelasan spesifikasi sistem, implementasi algoritma pada program, implementasi antarmuka dan implementasi metode.

4.1 Spesifikasi Sistem

Spesifikasi sistem mengacu pada hasil analisis kebutuhan dan perancangan perangkat lunak seperti yang terurai pada bab III. Dari spesifikasi sistem tersebut dapat dibangun sebuah aplikasi yang dapat berfungsi sesuai dengan kebutuhan dan tujuan pembuatan perangkat lunak. Spesifikasi sistem itu sendiri mampu diimplementasikan pada spesifikasi perangkat keras dan perangkat lunak.

4.1.1 Spesifikasi Perangkat Keras

Pengembangan aplikasi klasifikasi kategori bayar mahasiswa baru Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya ini menggunakan sebuah komputer dengan spesifikasi perangkat keras seperti yang telah diurai dalam Tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
Prosesor	Intel(R) Core(TM) i5 760 @ 2.80GHz 2.79GHz
Memori (RAM)	4 GB
Harddisk	ST3500418AS ATA Device
VGA	AMD Radeon HD 5700 Series

4.1.2 Spesifikasi Perangkat Lunak

Pengembangan aplikasi klasifikasi kategori bayar mahasiswa baru Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya ini menggunakan

sebuah komputer dengan spesifikasi perangkat lunak seperti yang telah diurai dalam Tabel 4.2.

Tabel 4.2 Spesifikasi Perangkat Lunak Komputer

Nama	Spesifikasi
Sistem Operasi	Windows 7 Ultimate
Bahasa Pemrograman	Java 1.7.0_9; Java HotSpot (TM) Client VM 23.5-b02
Tools pemrograman	NetBeans IDE 7.1 (Build 201112071828)
Server localhost	XAMPP Control Panel Version 2.5.8 XAMPP Windows Version 1.7.2
DBMS	MySQL versi 5.1.37

4.2 Implementasi Algoritma

Aplikasi klasifikasi ini terdiri dari tiga proses utama, yaitu normalisasi, proses *K-Nearest Neighbor* dan proses klasifikasi dengan *Fuzzy K-Nearest Neighbor*. Pada proses normalisasi dilakukan beberapa tahapan seperti perhitungan nilai maksimum dan minimum dari data latih, kemudian dilanjutkan dengan proses normalisasi menggunakan normalisasi min-max. Kemudian pada proses *K-Nearest Neighbor* sendiri juga terdiri dari beberapa tahapan, mulai dari perhitungan jarak menggunakan rumus jarak Euclidean, lalu pengurutan data dari yang terkecil hingga terbesar dan diambil berdasar nilai *k* yang terlibat. Selanjutnya merupakan proses klasifikasi *Fuzzy K-Nearest Neighbor* yang dilakukan dengan menghitung nilai keanggotaan pada masing-masing kelas terpilih dengan cara mengambil nilai keanggotaan terbesar dari perhitungan yang telah dilakukan. Daftar metode pada aplikasi klasifikasi ini dapat dilihat pada Tabel 4.3 berikut.

Tabel 4.3 Daftar Metode Aplikasi

No	Proses	Method	Keterangan
1	Normalisasi	`min (int [][] data)`	Metode min dipakai untuk mencari nilai minimal pada setiap fitur data uji yang keluarannya

			berupa *array* nilai minimum, yang kemudian dipakai dalam perhitungan metode normalisasi.
		`Max (int [][] data)`	metode max dipakai untuk mencari nilai maksimal pada setiap fitur data uji yang keluarannya berupa *array* nilai maksimum, yang kemudian dipakai dalam perhitungan metode normalisasi.
		`Normalisasi2D (int [][] data, int max[], int min[])`	Metode normalisasi dipakai untuk mendapatkan nilai normalisasi yang berkisar antara [0..1] pada masing-masing fitur
2	*K-Nearest Neighbor*	`euclid (float[][] dataLatih, float[] dataUji)`	Metode mendapatkan jarak antara data uji dengan data latih dengan menggunakan jarak Euclidean
		`jarakUrutDanKategori (float [] jaraknya, int [] kategori)`	Metode yang di dalamnya berisi cara mengurutkan jarak dari yang terkecil hingga terbesar menggunakan algoritma *Selection Sort.*

		`nilaiKa (int k, int [] kategori)`	Metode nilaiKa merupakan metode di mana data dari jarak yang telah terurut diambil sejumlah nilai k yang diinginkan.
		`distinct (int [ ] numbers)`	Metode yang di dalamnya memuat penyeleksian kelas yang akan dijadikan sebagai acuan keputusan pengklasifikasian
3	*Fuzzy K-Nearest Neighbor*	`fungsiKeanggotaan (int [] kategori, int k, double [] jarak, int [] kelas)`	Metode yang di dalamnya berisi algoritma pengambilan keputusan klasifikasi berdasar nilai keanggotaan terbesar
		`keanggotaan (double []data, int [] kelas)`	Metode yang di dalamnya berisi pembandingan nilai keanggotaan. Dan penentu keputusan akhir pengklasifikasian data uji.

4.2.1 Proses Normalisasi

Tahapan ini terdiri dari normalisasi semua data yang terlibat dalam proses perhitungan. Tujuan dari normalisasi adalah untuk merepresentasikan semua data set ke dalam skala yang sama, dalam hal ini memiliki interval antara 0 sampai 1 [BEM-06]. Normalisasi yang dipakai adalah normalisasi min-max. Metode normalisasi yang dipecah menjadi 3 metode secara keseluruhan ditunjukkan pada *Sourcecode* 4.1 hingga *Sourcecode* 4.3.

4.2.1.1 Pencarian Nilai Minimum

Sourcecode 4.1 merupakan implementasi pencarian nilai minimal dari keseluruhan data latih yang terlibat, yaitu untuk jalur SBNMPTN/SNMPTN dan

SPMK dengan 9 kategori. Baris ke-2 merupakan inisialisasi variabel min[] sebagai simpanan nilai minimum dari data. Proses pencarian nilai minimum itu sendiri terdapat pada proses perulangan. Perulangan terjadi sebanyak dua kali yaitu dengan batas i < panjang kolom data dan j < panjang baris data.

```
1   static int [] min (int [][] data){
2           int min[] = new int [data[0].length];
3
4           for (int i = 0; i < data[0].length; i++) {
5               int minTemp = data[0][i];
6               for (int j = 0; j < data.length; j++) {
7                   if (data[j][i] < minTemp) {
8                   minTemp = data[j][i];
9                   }
10              }
11              min[i] = minTemp;
13          }
14          return min;
15      }
```

Sourcecode 4.1 Implementasi Pencarian Nilai Minimum Data

Pada saat perulangan pada bagian j dilakukan, terdapat sebuah kondisi jika data[j][i] lebih kecil dari nilai minTemp yang sudah tersimpan sebelumnya, maka data[j][i]disimpan dalam minTemp, maka secara otomatis nilai minTemp akan berubah. Setelah perulangan terhadap j selesai, nilai minTemp tadi disimpan dalam min[] dan perulangan dilanjutkan dengan bertambahnya nilai i hingga i mencapai kurang dari panjang kolom data. Metode ini memiliki hasil akhir berupa nilai minimum yang disimpan dalam variabel min[], yang berjumlah sebanyak i.

4.2.1.2 Pencarian Nilai Maksimum

Sourcecode 4.2 menunjukkan implementasi pencarian nilai maksimal dari keseluruhan data latih yang terlibat, yaitu untuk jalur SBNMPTN/SNMPTN dan SPMK dengan 9 kategori.

```
1   static int [] max (int [][] data){
2           int max[] = new int [data[0].length];
3
4           for (int i = 0; i < data[0].length; i++) {
5               int maxTemp = 0;
```

```
6                 for (int j = 0; j < data.length; j++) {
7                     if (data[j][i] > maxTemp) {
8                     maxTemp = data[j][i];
9                     }
10                }
11                max[i] = maxTemp;
12            }
13            return max;
14        }
```

Sourcecode 4.2 Implementasi Pencarian Nilai Maksimum Data

Baris ke-2 merupakan inisialisasi variabel max[] sebagai simpanan nilai maksimum dari data. Proses pencarian nilai maksimum itu sendiri terdapat pada proses perulangan. Perulangan terjadi sebanyak dua kali yaitu dengan batas i < panjang kolom data dan j < panjang baris data.

Pada saat perulangan pada bagian j dilakukan, terdapat sebuah kondisi jika data[j][i] lebih besar dari nilai maxTemp yang sudah tersimpan sebelumnya, maka data[j][i] disimpan dalam maxTemp, maka secara otomatis nilai maxTemp akan berubah. Setelah perulangan terhadap j selesai, nilai maxTemp tadi disimpan dalam max[] dan perulangan dilanjutkan dengan bertambahnya nilai i hingga i mencapai kurang dari panjang kolom data. Metode ini memiliki hasil akhir berupa nilai maksimum yang disimpan dalam variabel max[], yang berjumlah sebanyak i.

4.2.1.3 Perhitungan Normalisasi

Sourcecode 4.3 merupakan implementasi dari algoritma normalisasi min-max. Proses normalisasi baru dilakukan setelah metode max() dan min() dijalankan, karena metode normalisasi() membutuhkan nilai yang diperoleh dari kedua metode tersebut di atas. Pada baris ke-2 merupakan inisialisasi variabel dataNorm[][] yang dipakai untuk menyimpan data hasil normalisasi. DataNorm bertipe array dua dimensi dengan panjang baris sebanyak panjang baris data dan panjang kolom sebanyak panjang kolom data.

```
1   static float[][] normalisasi2D (int [][] data, int max[],
    int min[]){
2           float dataNorm[][] = new float [data.length]
            [data[0].length];
3           float c;
```

```
4           float d;
5
6           for (int i = 0; i < data.length; i++) {
7               for (int j = 0; j < data[0].length; j++) {
8                   c = data[i][j] - min[j];
9                   d = max[j] - min[j];
10                  dataNorm[i][j] = c / d;
11
12              }
13          }
14          return dataNorm;
15      }
```

Sourcecode 4.3 Implementasi Normalisasi Min-max

Proses normalisasi terjadi di dalam proses perulangan yang dilakukan sebanyak i < jumlah baris data dan j < jumlah kolom data. Perulangan pada j terdapat pengurangan data[i][j] dengan nilai min[i] yang kemudian disimpan dalam variabel c, lalu pengurangan max[i] terhadap min[i] yang disimpan pada variabel d. Kemudian proses berlanjut dengan pembagian c dengan d yang disimpan pada variabel dataNorm[i][j]. Metode normalisasi2D() ini memiliki hasil akhir berupa nilai normalisasi data latih dan data uji yang disimpan dalam variabel dataNorm[][].

4.2.2 Proses *K-Nearest Neighbor*

Proses *K-Nearest Neighbor* mengimplementasikan proses perhitungan jarak Euclidean, pengurutan jarak dari yang terkecil hingga terbesar, pencarian indeks yang menyertai jarak terdekat, proses penyimpanan data berdasar nilai k dan proses pemampatan kelas terpilih sehingga tidak ada lagi kelas yang sama sebagai kelas acuan dalam perhitungan nilai keanggotaan di proses berikutnya.

4.2.2.1 Perhitungan Jarak Euclidean

Proses pertama diawali dengan perhitungan jarak Euclidean. Implementasi algoritma perhitungan jarak Euclidean seperti tersaji dalam *Sourcecode* 4.4.

```
1   static float [] euclid (float[][] dataLatih, float[] dataUji)
    {
2
3           float jarak[] = new float [dataUji.length];
4           float a = 0;
5           float b = 0;
```

```
6
7           for (int i = 0; i < dataLatih.length; i++) {
8               float c = 0;
9
10              for (int j = 0; j < dataUji.length; j++) {
11                  a = dataUji[j] - dataLatih[i][j];
12                  b = (float) Math.pow(a, 2);
13                  c = (float) (c + Math.pow(a, 2));
14              }
15
16              jarak[i] = (float) Math.sqrt(c);
17          }
18          return jarak;
19      }
```

Sourcecode 4.4 Implementasi Algoritma Jarak Euclidean

Metode euclid() merupakan metode yang di dalamnya terdapat proses perhitungan jarak Euclidean antara data uji dan data latih. Bermula pada inisialisasi variabel jarak[] yang berfungsi sebagai simpanan jaraknya nanti, metode ini juga terdapat dua kali proses perulangan yang melibatkan i < jumlah baris data latih dan j < jumlah baris data uji. Proses yang berlangsung pada perulangan kedua adalah pengurangan dataUji[j] terhadap dataLatih[i][j] yang kemudian disimpan dalam variabel a, dilanjutkan dengan pengkuadratan a yang disimpan dalam variabel b dan adanya pertambahan nilai variabel c dengan dirinya sendiri dan hasil dari b.

Proses dilanjutkan hingga j mencapai maksimum, kemudian dilanjutkan dengan mengakar nilai c yang disimpan dalam variabel jarak[i]. Metode ini memiliki hasil berupa nilai jarak Euclidean antara data latih dan data uji yang disimpan dalam variabel jarak[].

4.2.2.2 Pengurutan Jarak Secara *Ascending* Beserta Kategorinya

Setelah dilakukan perhitungan jarak, maka langkah selanjutnya adalah mengurutkan jarak dari yang terdekat hingga terjauh (*ascending*). Implementasi pengurutan jarak tersaji dalam *Sourcecode* 4.5.

```
1   static int [] jarakUrutDanKategori (float [] jaraknya,
    int[] kategori) {
2           int pjg = kategori.length;
```

```
3           float temp1[] = new float[pjg];
4           int temp2[] = new int[pjg];
5
6           for (int i = 0; i <pjg; i++) {
7               for (int j = i + 1; j <pjg; j++) {
8                   if (jaraknya[i] > jaraknya[j]) {
9                       temp1[i] = jaraknya[i];
10                      jaraknya[i] = jaraknya[j];
11                      jaraknya[j] = temp1[i];
12
13                      temp2[i] = kategori[i];
14                      kategori[i] = kategori[j];
15                      kategori[j] = temp2[i];
16                  }
17              }
18          }
29          return kategori;
20      }
```

Sourcecode 4.5 Implementasi Pengurutan Jarak dan Kategori

Metode jarakUrutDanKategori() merupakan metode yang di dalamnya terdapat proses pengurutan jarak beserta kategori yang menyertainya menggunakan *selection sort*. Pada baris ke-4 dan ke-5 terdapat inisialisasi variabel yang untuk menyimpan hasil sementara saat pengurutan, kedua variabel tersebut adalah temp1[] dan temp2[] yang keduanya bertipe array integer dengan panjang sejumlah baris dari kategori.

Terdapat dua kali pengulangan sebanyak i dan j, i dimulai dari 0 sedangkan j dimulai dari i+1. Hal ini karena iterasi j digunakan untuk membaca variabel array setelah indeks i sebagai bandingan variabel berindeks i. Saat iterasi j dijalankan, terdapat suatu kondisi yaitu jika jarak[i] bernilai lebih dari jarak[j] maka nilai dari jaraknya[i] masuk dalam variabel temp1[i], nilai dari jaraknya[j] disimpan dalam jaraknya[j] dan nilai temp1[] dimasukkan kembali dalam jaraknya[j]. Hal ini seperti menukar nilai berdasar posisinya (i dan j).

Bersamaan dengan proses penukaran nilai yang tersimpan pada variabel jaraknya[], terjadi juga proses penukaran nilai yang terjadi pada variabel kategori. Proses penukaran tersebut juga sama seperti penukaran nilai pada variabel

jaraknya[], hanya saja nilai ditampung sementara di variabel temp2[]. Metode ini memiliki hasil berupa nilai yang disimpan dalam variabel kategori.

4.2.2.3 Keterlibatan Nilai K

Nilai *k* salah satunya dipakai sebagai jumlah keterlibatan data dalam penyeleksian klasifikasi pada proses akhir. Selain itu nilai *k* juga dipakai sebagai parameter pengujian pada saat analisa yang dilakukan untuk mengetahui seberapa besar pengaruh nilai *k* dalam klasifikasi kategori bayar mahasiswa baru Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya Malang. Implementasi keterlibatan nilai *k* dapat dilihat pada *Sourcecode* 4.6.

```
1  static int [] nilaiKa (int k, int [] kategori) {
2          Scanner masukan = new Scanner (System.in);
3
4          int dataK [] = new int [k];
5
6          for (int i = 0; i < k; i++) {
7              dataK[i] = kategori[i];
8          }
9          return dataK;
```

Sourcecode 4.6 Implementasi Keterlibatan Nilai K

Metode nilaiKa() meminta masukan pengguna dengan memanggil variabel masukan yang bertipe Scanner. Scanner itu sendiri merupakan fungsi bawaan dari Java yang berguna untuk meminta masukan berupa nilai suatu variabel kepada pengguna. Terdapat inisialisasi variabel pada baris ke-4 yang bertipe array dengan panjang sejumlah *k*. Terdapat juga proses perulangan atau iterasi bagi kategori[] yang jika jumlahnya sudah sama dengan *k* maka akan disimpan ke variabel dataK[]. Metode ini memiliki hasil akhir berupa nilai data berjumlah *k* yang disimpan dalam variabel dataK.

4.2.2.4 Mencari *Distinct Value* Dari Kelas yang terpilih

Distinct value merupakan nilai dari sekumpulan data yang tidak memiliki duplikat. Pada implementasi metode *Fuzzy K-Nearest Neighbor* memerlukan adanya *distinct value* karena proses tersebut mampu mempengaruhi jumlah kelas yang terlibat pada saat perhitungan nilai keanggotaan. Penyeleksian *distinct value* seperti yang dijabarkan oleh Bear dalam website

http://www.dreamincode.net/forums/topic/138385-distinct-values-in-an-array/ seperti *Sourcecode* 4.7.

Fungsi distinct[] pada dasarnya menjabarkan mengenai proses penyeleksian angka kembar yang nantinya akan disimpan sebagai satu buah variabel untuk setiap angka kembar yang ada. Penulis menggunakan tipe data ArrayList karena penulis tidak tahu berapa jumlah data pasti dari hasil perhitungan *distinct value* tersebut.

```
1     public static boolean isIn(int num, int[] a){
2
3         for (int i = 0; i < a.length; i++) {
4             if (a[i] == num)
5                 return true;
6         }
7         return false;
8     }
9
10    static int [] dataFix (int [ ] numbers) {
11
12        ArrayList<Integer> fitur = new ArrayList<Integer>();
13        ArrayList<Integer> fitur1 = new ArrayList<Integer>();
14
15        int DistinctArray[] = new int[numbers.length];
16
17            for (int i = 0; i < numbers.length; i++) {
18                int temp = numbers[i];
19
20                // Pemanggilan metode isIn
21                if (!isIn(temp, DistinctArray)){
22
23                    DistinctArray[i] = temp;
24                    fitur.add(temp);
25                    fitur1.add(temp);
26                } else {
27                    fitur.add(temp);
28                }
29            }
30
31            int [] kelas = new int[fitur1.size()];
32
33            for (int j = 0; j < fitur1.size(); j++) {
```

```
34                  kelas[j] = fitur1.get(j);
35
36  //                  System.out.println("Distinct array nya
adalah: " + kelas[j]);
37              }
38              return kelas;
39      }
```

Sourcecode 4.7 Implementasi Pencarian *Distinct Value*

Proses pertama diawali dengan inisialisasi variabel bertipe ArrayList dan array sebagai simpanan hasil perhitungan. Terdapat satu kali perulangan atas i = 0 hingga i < panjang baris array. Terdapat inisialisasi variabel temp di dalam perulangan tersebut yang nilainya sama dengan numbers[i], yang artinya nilai dari variabel temp tersebut dapat berubah-ubah saat i-nya semakin bertambah.

Dalam proses iterasi tersebut juga terdapat sebuah kondisi yang memanggil metode isIn(). Jika isIn() bernilai *false* (salah) maka nilai yang disimpan dalam variabel temp dipindah ke dalam variabel DistinctArray(), juga disimpan dalam variabel fitur dan fitur1. Jika tidak (bernilai *true*) maka temp disimpan dalam variabel fitur.

Setelah iterasi terdapat inisialisasi variabel lagi yang bertipe array, yang panjangnya sejumlah fitur1.size(). Kemudian terdapat satu perulangan lagi yang di dalamnya terdapat proses penyimpanan nilai dari ArrayList fitur1 ke dalam array kelas. Dilakukan proses duplikat ke dalam variabel array karena proses selanjutnya memerlukan tipe data array, bukan ArrayList. Metode ini memiliki hasil akhir berupa nilai kelas yang hanya memiliki satu anggota saja yang disimpan dalam variabel kelas.

4.2.3 Proses *Fuzzy K-Nearest Neighbor*

4.2.3.1 Nilai Keanggotaan

Tahap selanjutnya dari serangkaian proses klasifikasi dengan *Fuzzy K-Nearest Neighbor* yaitu perhitungan nilai keanggotaan. Nilai keanggotaan merupakan suatu langkah yang dilakukan untuk menentukan data akan memasuki kelas yang mana. Implementasi algoritma penentuan nilai keanggotaan dapat dilihat pada *Sourcecode* 4.8.

```
1   static float [] fungsiKeanggotaan (int [] kategori, int k,
2   float [] jarak, int [] kelas) {
3
4           float [] keanggotaan = new float [k];
5           int[] jml = new int [k];
6           float Aa;
7           float B;
8
9           for (int i = 0; i < kelas.length; i++) {
10              int jmlData;
11              float A = 0;
12              float C = 0;
13
14              for (int j = 0; j < k; j++) {
15                  if (kelas[i] == kategori[j]) {
16                      jmlData = 1;
17                  } else {
18                      jmlData = 0;
19                  }
20
21                  Aa = (float) Math.pow(jarak[j], (-2));
22                  A = A + (jmlData*Aa);
23
24                  C = (float) (C + Math.pow(jarak[j], (-2)));
25              }
26
27              keanggotaan[i] = A / C ;
28          }
29          return keanggotaan;
30      }
```

Sourcecode 4.8 Implementasi Algoritma Nilai Keanggotaan

Metode fungsiKeanggotaan() berisi proses perhitungan nilai keanggotaan untuk setiap data yang diujikan. Proses tersebut dimulai dengan inisialisasi variabel keanggotaan[] yang dipakai sebagai penyimpan hasil perhitungan nilai keanggotaan yang bertipe array *double*. Selanjutnya inisialisasi variabel Aa untuk menyimpan pembobotan jarak data uji dengan data latih, variabel A untuk menyimpan hasil penjumlahan dirinya sendiri dengan hasil kali antara jmlData dengan Aa, B untuk menyimpan hasil akar dari A.

Setelah inisialisasi data terdapat proses perulangan atau iterasi yang melibatkan i = 0 hingga i < panjang kelas dan di dalam iterasi pertama terdapat

iterasi kedua yang melibatkan j = 0 hingga j < k. Terdapat kondisi di dalam iterasi j, yaitu jika kelas[i] nilainya sama dengan kategori[j] maka jmlData akan bernilai 1, jika tidak maka jmlData akan bernilai 0. Variabel jmlData nilainya dapat berubah-ubah saat j-nya mulai bertambah.

Setelah proses kondisi dilakukan, selanjutnya merupakan proses pembobotan jarak[j] yang disimpan dalam variabel Aa, lalu penjumlahan A dengan hasil kali antara jmlData dengan Aa disimpan dalam variabel A. Semua proses di atas dilakukan hingga j < k. Setelah j mencapai maksimum maka proses berlanjut dengan perhitungan keanggotaan dengan pembagian A dan C. Metode fungsiKeanggotaan() memiliki hasil akhir berupa nilai keanggotaan data uji yang disimpan dalam variabel keanggotaan.

4.2.3.2 Penentuan Kelas Akhir

Tahap akhir dari serangkaian proses klasifikasi dengan metode *Fuzzy K-Nearest Neighbor* adalah penentuan kelas. Kelas ditentukan dengan membandingkan nilai keanggotaan yang telah diperoleh dari tahap sebelumnya. Nilai keanggotaan terbesar akan dipilih menjadi keputusan akhir pengklasifikasian. Implementasi algoritma penempatan kelas dapat dilihat pada *Sourcecode* 4.9.

```
1   static int keanggotaan (double []data, int [] kelas){
2           int max = 0;
3
4           for (int i = 0; i < data.length; i++) {
5               int maxTemp = 0;
6               for (int j = 0; j < data.length; j++) {
7                   if (data[j] > maxTemp) {
8                   maxTemp = kelas[j];
9                   }
10              }
11              max = maxTemp;
12          }
13          return max;
14      }
```

Sourcecode 4.9 Implementasi Algoritma Penentuan Kelas Akhir

Pada tahap akhir perhitungan nilai keanggotaan terbesar ini memiliki proses yang sama dengan metode max(), di dalamnya juga terdapat pengambilan nilai maksimum dari nilai keanggotaan. Baris ke-2 merupakan inisialisasi variabel

max[] sebagai simpanan nilai maksimum dari data. Proses pencarian nilai maksimum itu sendiri terdapat pada proses perulangan. Perulangan terjadi sebanyak dua kali yaitu dengan batas i < panjang kolom data dan j < panjang baris data.

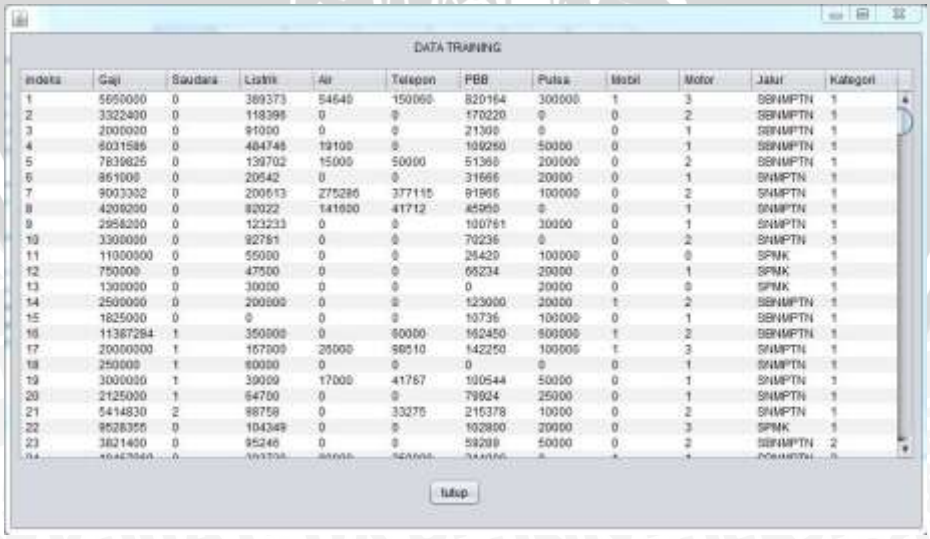
Pada saat perulangan pada bagian j dilakukan, terdapat sebuah kondisi jika data[j][i] lebih besar dari nilai maxTemp yang sudah tersimpan sebelumnya, maka data[j][i] disimpan dalam maxTemp, maka secara otomatis nilai maxTemp akan berubah. Setelah perulangan terhadap j selesai, nilai maxTemp tadi disimpan dalam max[] dan perulangan dilanjutkan dengan ditambahnya nilai i hingga i mencapai maksimum. Metode ini memiliki hasil akhir berupa nilai maksimum dari nilai keanggotaan yang disimpan dalam variabel max[] yang berjumlah sebanyak i.

4.3 Implementasi Antar Muka

Antar muka klasifikasi kategori bayar mahasiswa baru Universitas Brawijaya berguna untuk memudahkan pengguna dalam berinteraksi dengan program.

4.3.1 Tampilan Halaman Data Latih

Tampilan data latih berupa jendela yang di dalamnya berupa tabel yang berisi seluruh fitur yang terlibat beserta kategori bayarnya. Jendela data latih ditunjukkan pada gambar 4.1.



Gambar 4.1 Halaman Data Latih

Pada jendela data latih tersebut dapat dilihat bahwa terdapat barisan data yang dipakai dalam proses klasifikasi. Terdapat sebanyak 1.004 data yang dipakai

dalam proses klasifikasi untuk jalur SBNMPTN/SNMPTN dan 256 data untuk jalur SPMK, masing-masing memiliki 11 kategori, yaitu gaji orang tua, jumlah saudara kandung, biaya listrik, biaya air, biaya telepon, besar Pajak Bumi dan Bangunan (PBB), besar pulsa telepon genggam, jumlah mobil, jumlah motor, jalur masuk, serta kelas kategori.

4.3.2 Tampilan Halaman Analisa Klasifikasi Kategori

Tampilan halaman analisa klasifikasi terdiri atas beberapa *field* kosong yang meminta masukan dari pengguna. *Field* kosong tersebut harus diisi jika ingin mendapatkan hasil keluaran yang maksimal. Tampilan jendela analisa ditunjukkan pada Gambar 4.2.

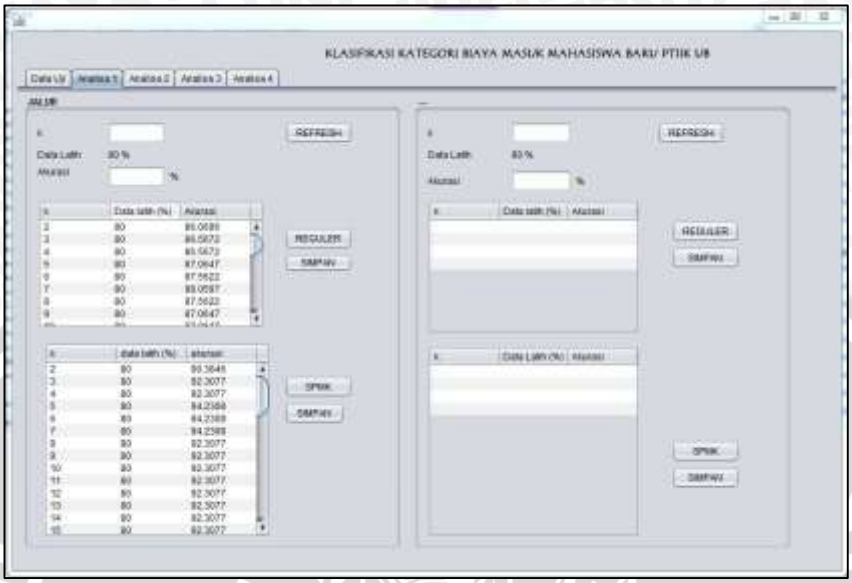
Gambar 4.2 Halaman Analisa Klasifikasi Bayar

Terdapat panel fitur yang berguna sebagai masukan parameter pengujian berupa lima fitur utama yang terlibat dalam perhitungan, yaitu gaji orang tua, biaya listrik, biaya air, biaya telepon dan besar PBB. Terdapat juga panel proses yang di dalamnya berisi *field* kosong yang menampilkan hasil perhitungan berupa kategori akhir dari data uji setelah pengguna memencet tombol proses, dan nominal sesuai dengan kategori yang ada. Sedangkan tombol *refresh* berguna untuk mengeset ulang semua *field* agar kosong kembali.

4.3.3 Tampilan Hasil Analisa 1

Panel analisa 1 berfungsi untuk menganalisa pengaruh nilai *k* terhadap akurasi yang didapat. Terbagi menjadi 2 jendela, jendela pertama digunakan untuk menguji pengaruh *k* terhadap akurasi pada jalur SBNMPTN/SNMPTN, sedangkan

jendela kedua digunakan untuk menguji pengaruh *k* terhadap akurasi pada jalur SPMK.



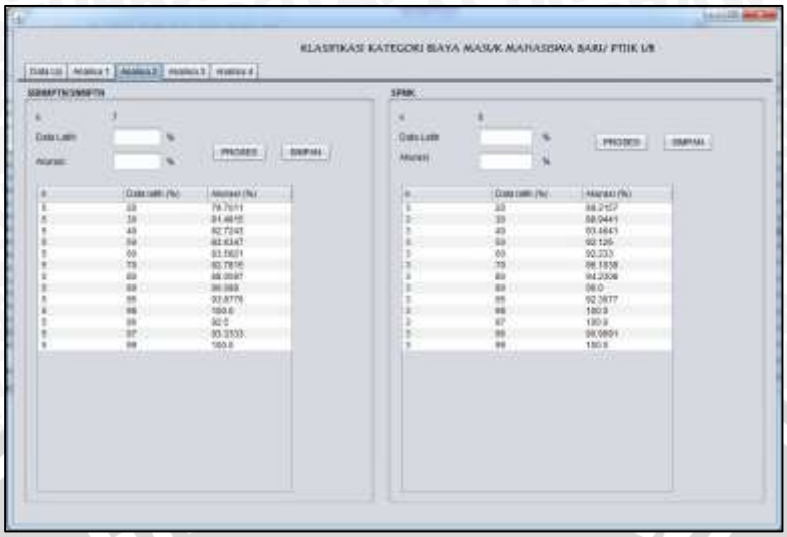
Gambar 4.3 Halaman Analisa 1

Pada jendela analisa 1 terdapat *field* masukan berupa nilai *k*, prosentase data latih dan tampilan hasil prosentase akurasi. Proses dilakukan dengan dua jenis data melalui jalur seleksi yang berbeda, yaitu SBNMPTN/SNMPTN dan SPMK, yang masing-masing memiliki tabel penyimpanan sendiri.

Terdapat tiga tombol, yaitu tombol proses yang berguna sebagai tombol pemrosesan data yang memiliki keluaran berupa akurasi, tombol simpan yang berfungsi untuk menyimpan data hasil perhitungan beserta *k* dan prosentase data latihnya, serta tombol *refresh* untuk mengeset jTextField menjadi kosong. Tampilan ditunjukkan pada Gambar 4.3.

4.3.4 Tampilan Hasil Analisa 2

Pada panel analisa 2 dibagi atas jalur SBNMPTN/SNMPTN dan SPMK, fungsinya yaitu untuk menganalisa pengaruh komposisi data latih dan data uji terhadap akurasi yang didapat. Jendela pertama digunakan untuk menguji pengaruh komposisi data latih dan data uji terhadap akurasi pada jumlah data 1.004 untuk jalur SNMPTN/SBNMPTN dan jendela kedua digunakan untuk menguji pengaruh komposisi data latih dan data uji terhadap akurasi pada jumlah data 256 untuk jalur SPMK.



Gambar 4.4 Halaman Analisa 2

Terdapat dua tombol, yaitu tombol proses yang berguna sebagai tombol pemrosesan data yang memiliki keluaran berupa akurasi, tombol simpan yang berfungsi untuk menyimpan data hasil perhitungan dan prosentase data latihnya. Keseluruhan tampilan ditunjukkan pada Gambar 4.4.

4.3.5 Tampilan Hasil Analisa 3

Jendela analisa 3 ini terbagi lagi menjadi 2 panel, yaitu panel perhitungan untuk jalur SNMPTN/SBNMPTN dan panel untuk perhitungan jalur SPMK. Tampilan halaman analisa 3 terdiri atas satu *field* kosong berfungsi untuk menampilkan hasil akurasi yang didapat. Pada analisa 3 ini nilai *k* dan jumlah data latih telah diatur konstan. Tampilan jendela analisa ditunjukkan pada Gambar 4.5.

Gambar 4.5 Halaman Analisa 3

Terdapat satu tombol proses yang berguna sebagai tombol pemrosesan data yang memiliki keluaran berupa akurasi. Keseluruhan tampilan ditunjukkan pada Gambar 4.5.

4.3.6 Tampilan Hasil Analisa 4

Jendela analisa 4 ini terbagi lagi menjadi 2 panel, yaitu panel perhitungan untuk jalur SBNMPTN/SNMPTN dan panel untuk jalur SPMK. Tampilan halaman analisa 3 terdiri atas satu *field* kosong berfungsi untuk menampilkan hasil akurasi yang didapat. Pada analisa 3 ini nilai *k* dan jumlah data latih telah diatur konstan.

Gambar 4.6 Halaman Analisa 4

Terdapat tiga tombol, yaitu tombol Manhattan yang berguna sebagai tombol pemrosesan data dengan menggunakan metode perhitungan jarak Manhattan, tombol Euclidean yang berguna sebagai tombol pemrosesan data dengan menggunakan metode perhitungan jarak Euclidean, tombol *refresh* untuk mengeset jTextField menjadi kosong. Keseluruhan tampilan ditunjukkan pada Gambar 4.6.