# IMPLEMENTASI



## Implementasi Sistem

Pada bagian ini dilakukan penerapan dari perancangan sesuai dengan yang telah dibuat sebelumnya di rancangan algoritma *Neighbor Weight K-Nearest Neighbor* (NWKNN) dan seleksi fitur *Particle Swarm Optimization* (PSO). Dalam implementasi ini dibagi menjadi beberapa proses implementasi menjadi kode program untuk *pre-processing*, kode program normalisasi data, kode program penyeleksian situr, kode program perhitungan jarak, kode program perhitungan bobot, kode program perhitungan skor. Pada setiap proses memiliki sub proses kode program sesuai dengan perancangan algoritma.

### Implementasi *Pre-Processing Data*

Pada bagian implementasi pre-processing data merupakan proses untuk membaca data berupa *Excel* dengan extensi (.xls) dan mengubah datanya kedalam bentuk array *String* 2 Dimensi. Kode program untuk implementasi fungsi *pre-processing* data terdapat pada Kode Program 5.1.

|  |  |
| --- | --- |
| Kode Program : Fungsi *Pre-Processing* Data | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static String[][] data(File f) throws Exception {  Workbook wb = Workbook.getWorkbook(f);  Sheet s = wb.getSheet(0);  int row = s.getRows();  int col = s.getColumns();  String[][] hsl = new String[row][col];  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  Cell c = s.getCell(j, i);  hsl[i][j] = c.getContents();  }  }  return hsl;  } |

Kode Program 5.1 *Pre-Processing* Data

Penjelasan untuk implementasi Kode Program 5.1 dijabarkan sebagai berikut:

1. Baris 1 menjelaskan deklarasi fungsi untuk mengambil data dengan kembalian fungsi berupa array *String* 2 dimensi dengan parameter tipe data *File*.
2. Baris 2-5 menjelaskan inisialisasi variabel untuk mengambil *file excel* dari alamat yang tersimpan di variabel f, selanjutnya mengambil sheet, baris dan kolom yang digunakan.
3. Baris 6 menjelaskan pembuatan *array String* 2 dimensi sebagai tempat pengisian nilai dari *file excel* yang diambil.
4. Baris 8-11 menjelaskan proses pemindahan isi *file excel* kedalam array hsl untuk dilakukan perhitungan.
5. Baris 14 menjelaskan keluaran dari fungsi berupa *array* hsl.

### Implementasi Normalisasi data

Pada bagian implementasi normalisasi data bertujuan untuk mengubah data menggunakan nilai minimum dan maksimum untuk diubah dalam rentang nilai antara [0, 1]. Untuk implementasi fungsi terdapat pada Kode Program 5.2.

|  |  |
| --- | --- |
| Kode Program : Fungsi Normalisasi Data | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | public static String[][] normalisasi(String[][] x) {  String[][] hsl = new String[x.length][x[0].length];  double max = 0, min = 4;  for (int i = 0; i < x.length; i++) {  for (int j = 0; j < hsl[i].length - 2; j++) {  if (Double.parseDouble(x[i][j]) > max) {  max = Double.parseDouble(x[i][j]);  }  if (Double.parseDouble(x[i][j]) < min) {  min = Double.parseDouble(x[i][j]);  }  }  }  for (int i = 0; i < x.length; i++) {  for (int j = 0; j < hsl[i].length; j++) {  if (j < hsl[i].length - 2) {  double a = (Double.parseDouble(x[i][j]) - min)   / (max - min);  DecimalFormat df = new DecimalFormat("#.###");  hsl[i][j] = df.format(a);  } else {  hsl[i][j] = x[i][j];  }  }  }  return hsl; |

Kode Program 5.2 Normalisasi Data

Penjelasan untuk implementasi Kode Program 5.2 dijabarkan sebagai berikut:

1. Baris 1 menjelaskan deklarasi fungsi untuk normalisasi data dengan kembalian fungsi berupa array *String* 2 dimensi dengan parameter tipe data *array string* 2 dimensi.
2. Baris 2-3 menjelaskan inisialisasi variable *array string* 2 dimensi hsl sebagai tempat penyimpanan hasil normalisasi dan inisialisasi awal variable min dan max untuk tempat penyimpanan nilai minimum dan maximum.
3. Baris 4-13 menjelaskan proses pencarian nilai minimum dan maximum dari seluruh data yang ada.
4. Baris 15-25 menjelaskan proses normalisasi data mentah sehingga menghasilkan nilai antara [0, 1] dengan format 3 angka di belakang koma.
5. Baris 26 menjelaskan keluaran dari fungsi berupa *array* hsl.

### Implementasi Penyeleksian Fitur (PSO)

Pada bagian implementasi penyeleksian fitur (PSO) bertujuan untuk menyeleksi fitur-fitur yang nantinya digunakan sebagai bahan perhitungan sehingga menghasilkan nilai optimum dengan fitur yang minimum. Untuk implementasi dari fungsi implementasi penyeleksian fitur terdapat pada Kode Program 5.3.

|  |  |
| --- | --- |
| Kode Program : Fungsi Penyeleksian Fitur (PSO) | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | public static int[] pso(String[][] data, int nPartikel, int k, int e, double persen) {  int[][] partikel;  int generasi = 10;  double w = 0.5;  double c1 = 1;  double c2 = 1;  double[] fit = new double[nPartikel];  partikel = inisialAwal(data, nPartikel);  double[][] v = new double[nPartikel][data[0].length - 2];  int[][] pbest = new int[nPartikel][data.length - 2];  int[] gbest = new int[data.length - 2];  for (int i = 0; i < generasi; i++) {  for (int j = 0; j < nPartikel; j++) {  fit[j] = fitness(data, k, e, partikel[j], persen);  }  if (i == 0) {  Pbest = partikel;  pbest = Pbest;  fitPbest = fit;  gbest(partikel, fit);  gbest = Gbest;  } else {  updatePbest(partikel, pbest, fit, fitPbest);  pbest = Pbest;  gbest(Pbest, fitPbest);  gbest = Gbest;  double[][] temp = kecepatan(w, c1, c2, partikel,    Pbest, Gbest, v);  v = temp;  int[][] tempPosisi = updatePosisi(partikel, v);  partikel = tempPosisi;  }  }  return gbest;  } |

Kode Program 5.3 Penyeleksian Fitur (PSO)

Penjelasan untuk implementasi Kode Program 5.3 dijabarkan sebagai berikut:

1. Baris 1 menjelaskan deklarasi fungsi untuk penyeleksian fitur dengan kembalian fungsi berupa array *int* 1 dimensi dengan parameter tipe data *array string* 2 dimensi, banyak partikel tipe data integer, nilai k dan e bertipe data integer dan nilai persentase data yang digunakan bertipe data double.
2. Baris 3-12 menjelaskan inisialisasi variabel partikel, banyak generasi, nilai w, nilai c1, nilai c2, nilai fitness partikel, inisilisasi awal pembuatan partikel secara random, inisialisasi kecepan partikel, inisialisasi partikel Pbest, inisialisasi partikel Gbest.
3. Baris 14-17 menjelaskan proses pencarian nilai *fitness* dari setiap pertikel yang ada yang disimpan kedalam variabel fit.
4. Baris 19-23 menjelaskan proses insialisasi awal untuk partikel pbest, fitness dari Pbest, dan nilai partikel gbest.
5. Baris 24-35 menjelaskan proses pengupdetan nilai pbest, nilai gbest, dan perubahan posisi nilai partikel.
6. Baris 36 menjelaskan keluaran dari fungsi berupa *array* gbest.

### Implementasi Perhitungan Jarak

Pada bagian implementasi perhitungan jarak bertujuan untuk menghitung jarak antara data latih dengan data test yang nantinya digunakan sebagai bahan perhitungan selanjutnya. Untuk implementasi dari fungsi implementasi penyeleksian fitur terdapat pada Kode Program 5.4.

|  |  |
| --- | --- |
| Kode Program : Fungsi Perhitungan Jarak | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public static double[][] jarak(String[][] x, String[] y) {  double[][] jarak = new double[x.length][2];  DecimalFormat df = new DecimalFormat("#.###");  int kelas = x[0].length - 1;  for (int i = 0; i < jarak.length; i++) {  double htng = 0;  for (int j = 0; j < x[i].length - 2; j++) {  double a = Math.pow(Double.parseDouble(x[i][j]) -   Double.parseDouble(y[j]), 2);  htng += a;  }  String a = df.format(Math.sqrt(htng));  jarak[i][0] = Double.parseDouble(a);  jarak[i][1] = Double.parseDouble(x[i][kelas]);  }  return jarak;  } |

Kode Program 5.4 Perhitungan Jarak

Penjelasan untuk implementasi Kode Program 5.4 dijabarkan sebagai berikut:

1. Baris 1 menjelaskan deklarasi fungsi untuk perhitungan jarak dengan kembalian fungsi berupa array *double* 2 dimensi dengan parameter tipe data *array string* 2 dimensi dan *array string* 1 dimensi.
2. Baris 2-4 menjelaskan menjelaskan inisialisasi variabel jarak yang berisi nilai jarak tiap-tiap data latih dengan data test, inisialisasi buat format nilai jarak dengan format 3 angka dibelakang koma dan inisialisasi kelas dari tiap jarak yang didapat.
3. Baris 5-13 menjelaskan proses perhitungan jarak dari data tes dengan tiap-tiap data latih dengan format 3 angka di belakang koma.
4. Baris 14-15 menjelaskan proses pengisian nilai *array* jarak yang berisi nilai jarak dengan kelasnya.
5. Baris 17 menjelaskan keluaran dari fungsi berupa *array* gbest.

### Implementasi Perhitungan Bobot

Pada bagian implementasi perhitungan bobot bertujuan untuk menghitung bobot untuk tiap kelas yang tersedia. Untuk implementasi dari fungsi implementasi penyeleksian fitur terdapat pada Kode Program 5.5.

|  |  |
| --- | --- |
| Kode Program : Fungsi Perhitungan Bobot | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | public static double[][] bobot(double[][] jarak, int k, int e) {  double[][] hsl = new double[3][2];  double[] kumpulan = new double[3];  double min = k;  for (int j = 0; j < k; j++) {  if (jarak[j][1] == 1.0) {  kumpulan[0] += 1;  } else if (jarak[j][1] == 2.0) {  kumpulan[1] += 1;  } else if (jarak[j][1] == 3.0) {  kumpulan[2] += 1;  }  }  for (int i = 0; i < kumpulan.length; i++) {  if (kumpulan[i] != 0) {  if (kumpulan[i] < min) {  min = kumpulan[i];  }  }  }  for (int i = 0; i < hsl.length; i++) {  if (kumpulan[i] > 0) {  double b = e;  double a = Math.pow(kumpulan[i] / min, (1 / b));  hsl[i][0] = (1 / a);  hsl[i][1] = kumpulan[i];  } else {  hsl[i][0] = 0;  hsl[i][1] = kumpulan[i];  }  }  return hsl;  } |

Kode Program 5.5 Perhitungan Bobot

Penjelasan untuk implementasi Kode Program 5.5 dijabarkan berikut:

Baris 1 menjelaskan deklarasi fungsi untuk perhitungan bobot dengan kembalian fungsi berupa array *double* 2 dimensi dengan parameter tipe data *array double* 2 dimensi, nilai k dan e bertipe data integer.

1. Baris 2-4 menjelaskan proses inisialisasi variabel hsl sebagai tempat penyimpanan bobot, variabel kumpulan sebagai tempat penyimpanan banyak anggota dari tiap kelas dan variabel min sebagai nilai k untuk mengambil k data dari urutan paling atas.
2. Baris 5-11 menjelaskan proses penghitungan jumlah anggota dari tiap kelas yang tersaring dari nilai k.
3. Baris 15-18 menjelaskan proses mencari anggota paling sedikit dari keseluruhan kelas yang bukan bernilai 0.
4. Baris 23-31 menjelaskan proses menghitung nilai bobot untuk tiap kelas.
5. Baris 34 menjelaskan keluaran dari fungsi berupa *array* hsl.

### Implementasi Perhitungan Skor

Pada bagian implementasi perhitungan skor bertujuan untuk menghitung skor akhir untuk tiap kelas yang tersedia. Untuk implementasi dari fungsi implementasi penyeleksian fitur terdapat pada Kode Program 5.6.

|  |  |
| --- | --- |
| Kode Program : Fungsi Perhitungan Skor | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public static int skor(double[][] jarak, double[][] bobot, int k) {  double[][] hslSkor = new double[bobot.length][2];  int kelas;  for (int i = 0; i < hslSkor.length; i++) {  double count = 0;  for (int j = 0; j < k; j++) {  if (i + 1 == jarak[j][1]) {  if (bobot[i][1] > 0) {  double a = jarak[j][0];  count += a;  } else {  count = 0;  }  }  }  hslSkor[i][0] = bobot[i][0] \* count;  hslSkor[i][1] = i + 1;  }  kelas = Max(hslSkor);  return kelas;  } |

Kode Program 5.6 Perhitungan Skor

Penjelasan untuk implementasi Kode Program 5.6 dijabarkan sebagai berikut:

1. Baris 1 menjelaskan deklarasi fungsi untuk perhitungan skor dengan kembalian fungsi berupa integer dengan parameter tipe data *array double* 2 dimensi untuk jarak dan tipe data *array double* 2 dimensi untuk bobot, nilai k.
2. Baris 2-3 menjelaskan inisialisasi variabel hslSkor sebagai tempat penampungan nilai dari skor dan inisialisasi variabel kelas sebagai tempat penyimpanan hasil prediksi kelas dari sistem.
3. Baris 4-17 menjelaskan proses pencarian nilai skor untuk tiap-tiap kelas dan menyimpanya kedalam *array* hslSkor.
4. Baris 19 menjelaskan proses pencarian prediksi kelas dengan mencari nilai skor tertinggi dari keseluruhan kelas.
5. Baris 20 menjelaskan keluaran dari fungsi berupa *integer* kelas.