LAMPIRAN 1 *SOURCE CODE*

Lampiran ini berisi kode program sistem prakiraan cuaca jangka pendek. Kode program untuk tampilan sistem tidak dicantumkan sehingga hanya kode program yang berhubungan dengan proses analisis saja yang dicantumkan pada bagian ini

- **Fungsi Pembuatan Pohon Keputusan**

Tabel Lampiran 1. 1 Kode Program Pembuatan Pohon Keputusan

```
if(GUICBR.jmlKelas == 0)
   jmlKelas = new double[2];

else
   jmlKelas = new double[6];

if(depth == 1)
   rules = "IF ";

else
   rules += " AND ";

temp = sameClass(data);

if ((( temp.equals("HUJAN"))||( temp.equals("TIDAKHUJAN")) || ( temp.equals("HUJANSANGATRINGAN"))
|| ( temp.equals("HUJANRINGAN")) || ( temp.equals("HUJANSEDANG")) || ( temp.equals("HUJANLEBAT"))
|| ( temp.equals("HUJANSANGATLEBAT"))) && (atribut.length != 0))
 {
   probReachLeaf = sumWeight(data, indukToString(induk), cabang);
   otherClass = sumWeightNotInAClass(data, temp, indukToString(induk), cabang);
   jmlKelas = hitungJumlahKelas(data, temp, indukToString(induk), cabang);
   if(GUICBR.jmlKelas == 1)
      InsertIntoTree(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], jmlKelas[2], jmlKelas[3],
       jmlKelas[4], jmlKelas[5], probReachLeaf, otherClass);
   else
      InsertIntoTree2(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], probReachLeaf, otherClass);
   rules += indukToString(induk)+" "+cabang+" THEN Kondisi = "+temp+"      "+probReachLeaf+"/"+otherClass;
   System.out.println(ruleCounter +" "+rules);
   ruleCounter++;
}

else if (atribut.length == 0)
{
   temp = commonClass(data);
   probReachLeaf = sumWeight(data, indukToString(induk), cabang);
   otherClass = sumWeightNotInAClass(data, temp, indukToString(induk), cabang);
   jmlKelas = hitungJumlahKelas(data, temp, indukToString(induk), cabang);
   if(GUICBR.jmlKelas == 1)
      InsertIntoTree(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], jmlKelas[2], jmlKelas[3],
      jmlKelas[4], jmlKelas[5], probReachLeaf, otherClass);
   else
      InsertIntoTree2(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], probReachLeaf, otherClass);

   rules += indukToString(induk)+" "+cabang+" THEN Kondisi = "+temp+"      "+probReachLeaf+"/"+otherClass;
   System.out.println(ruleCounter + " "+rules);
   ruleCounter++;
}
```

Tabel Lampiran 1. 1 Tabel Kode Program Pembuatan Pohon Keputusan (lanjutan)

```
else
 {
  gain = new double[atribut.length];

  String[] valueAtribut = new String[0];
  for (i=0 ; i < atribut.length ; i++) {
     samples = getSamples(atribut[i],data);
     gain[i] = setGain(atribut[i], samples);
     System.out.println("GAIN "+atribut[i].getNama()+" ADALAH "+gain[i]);
  }

  if (!(sameGain(gain))) {
     int indexHighestGain = biggest(gain);
     if(GUICBR.jmlKelas == 1)
        InsertIntoTree(atribut[indexHighestGain].getNama(), cabang, induk, true, depth,Double.NaN,
        Double.NaN,
        Double.NaN, Double.NaN, Double.NaN, Double.NaN, Double.NaN, Double.NaN);
     else
        InsertIntoTree2(atribut[indexHighestGain].getNama(), cabang, induk, true, depth,Double.NaN,
        Double.NaN,
        Double.NaN, Double.NaN);

     rules += indukToString(induk)+" "+cabang;
     induk = lastIDInserted();
     valueAtribut = atribut[indexHighestGain].getValue();
     depth++;
     for(int ii = 0; ii < valueAtribut.length; ii++) {
        cabang = valueAtribut[ii];
        newSamples = setSamples(atribut[indexHighestGain].getNama(), valueAtribut[ii], data);
        if (newSamples.length == 0) {
           temp = commonClass(data);
           probReachLeaf = sumWeight(data, indukToString(induk), cabang);
           otherClass = sumWeightNotInAClass(data, temp, indukToString(induk), cabang);
           jmlKelas = hitungJumlahKelas(data, temp, indukToString(induk), cabang);
           if(GUICBR.jmlKelas == 1)
              InsertIntoTree(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], jmlKelas[2],
              jmlKelas[3], jmlKelas[4], jmlKelas[5], probReachLeaf, otherClass);
           else
              InsertIntoTree2(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], probReachLeaf,
              otherClass);
           tempRule = rules+" AND "+indukToString(induk)+" "+cabang+" THEN Kondisi = "+temp+"
           "+probReachLeaf+"/"+otherClass;
           System.out.println(ruleCounter + " "+tempRule);
           tempRule = "";
           ruleCounter++;
        }

        else {
           int haha = induk;
           newAtribut = eliminasiAtribut(atribut[indexHighestGain], atribut);
           generateDecisionTree(newAtribut, newSamples, depth, rules, valueAtribut[ii]);
           induk = haha;
        }
     }

  }
```

Tabel Lampiran 1. 1 Tabel Kode Program Pembuatan Pohon Keputusan (lanjutan)

```
    else {
      temp = commonClass(data);
      probReachLeaf = sumWeight(data, indukToString(induk), cabang);
      otherClass = sumWeightNotInAClass(data, temp, indukToString(induk), cabang);
      jmlKelas = hitungJumlahKelas(data, temp, indukToString(induk), cabang);
      if(GUICBR.jmlKelas == 1)
        InsertIntoTree(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], jmlKelas[2], jmlKelas[3],
        jmlKelas[4], jmlKelas[5], probReachLeaf, otherClass);
      else
        InsertIntoTree2(temp, cabang, induk, false, depth, jmlKelas[0], jmlKelas[1], probReachLeaf, otherClass);
      rules += indukToString(induk)+" "+cabang+" THEN Kondisi = "+temp+"
      "+probReachLeaf+"/"+otherClass;
      System.out.println(ruleCounter + " "+rules);
      ruleCounter++;
    }
  }
```

Kode program ini berfungsi untuk proses pembuatan pohon keputusan. Diawali dengan pemeriksaan jumlah kategori yang dipilih pengguna, kode program ini kemudian memeriksa apakah *training data* berasal dari kelas atau kategori yang sama. Jika data memiliki kelas yang sama maka sebuah *node* dapat langsung dimasukkan ke dalam pohon keputusan dengan terlebih dahulu menghitung berapa banyak kasus yang tergolong ke dalam *node* tersebut melalui fungsi sumWeight() dan berapa banyak kasus yang tergolong ke dalam kelas yang salah melalui fungsi sumWeightNotInAClass(). Jika atribut yang tersisa hanya satu maka dicari kelas paling umum dari himpunan *training data* tersebut dan dimasukkan ke dalam pohon keputusan.

Jika atribut masih lebih dari satu dan *training data* tidak tergolong hanya ke dalam 1 kelas spesifik saja, maka dilakukan perhitungan *GainRatio* untuk setiap atribut. Setelah menemukan atribut dengan *GainRatio* terbesar langkah berikutnya adalah menyaring *training data* yang memenuhi syarat atribut tersebut sehingga diperoleh sampel baru untuk *trainig data* pada iterasi berikutnya. Proses rekursif ini dilakukan terus menerus hingga dicapai *base case* yaitu pada saat seluruh *training data* tergolong ke dalam kelas atau kategori yang sama atau pada saat atribut sudah habis.

- **Fungsi Pengujian Pohon Keputusan**

Tabel Lampiran 1. 2 Kode Program Pengujian Pohon Keputusan

```
for(i=0;i<td.length;i++) {
 mv = false;
 parent = getNode(depth[0],0,null);
 for (j=1;j<depth.length;j++) {
    if (parent.getAtribut()) {

       if (parent.getNode().equals("ArahAngin")) {
          child = getNode(depth[j], parent.getID(), td[i].getArahAngin());
       }

       else{
          trData = getTrainingData(parent.getNode(),td[i]);
          cabangKontinu = getCabang(depth[j], parent.getID());

          if(cabangKontinu[0].contains("<=")){
          threshold = Double.parseDouble(cabangKontinu[0].substring(3));
          if (trData <= threshold)
             cabangTerpilih = "<= "+threshold;
          else
             cabangTerpilih = "> "+threshold;
       }

       else if(cabangKontinu[0].contains(">")){
          threshold = Double.parseDouble(cabangKontinu[0].substring(2));
          if (trData > threshold)
             cabangTerpilih = "> "+threshold;
          else
             cabangTerpilih = "<= "+threshold;
       }

       child = getNode(depth[j], parent.getID(), cabangTerpilih);

    }

    parent = child;
 }

 else {
    break;
 }
}

  this.realCondition[counter] = td[i].getCurahHujanBesok();
  if(!mv)
     this.predictedCondition[counter] = parent.getNode();
  else
     this.predictedCondition[counter] = kondisiProbabilitas;
  if (this.predictedCondition[counter].equals(realCondition[counter])
     this.benar++;
  else this.salah++;

  counter++;

 }

 this.akurasi = Double.parseDouble(Integer.toString(this.benar))/Double.
parseDouble(Integer.toString(this.benar+this.salah))*100;
```

Kode program ini berfungsi untuk menjalankan proses pengujian akurasi pohon keputusan. Untuk setiap iterasi pada *training data*, langkah pertama yang dilakukan adalah mencari *parent* dari pohon keputusan yang telah terbentuk.

Setelah *parent* diperoleh kemudian lakukan iterasi lagi namun kali ini yang bertindak sebagai *iterator* adalah kedalaman dari pohon. Untuk setiap kedalaman pohon, program ini akan mengambil nilai atribut *training data* yang relevan pada setiap *node* pohon keputusan mulai dari *root* hingga *leaf* sehingga pada akhirnya kelas atau kategori dapat diperoleh. Kategori inilah yang akan menjadi prediksi sifat hujan suatu *training data*. Setelah setiap *training data* mendapatkan prediksi sifat hujan, hasil prediksi ini kemudian dibandingkan dengan keadaan yang sebenarnya sehingga persentase akurasi dapat diperoleh pada akhir kode program ini.

- **Fungsi Klasifikasi Data Baru**

Tabel Lampiran 1. 3 Kode Program Klasifikasi Data Baru

```
parent = getNode(depth[0],0,null);
for (j=1;j<depth.length;j++) {
   if (parent.getAtribut()) {

      if (parent.getNode().equals("ArahAngin")) {
         child = getNode(depth[j], parent.getID(), dataBaru.getArahAngin());
      }

      else{
         trData = getTrainingData(parent.getNode(), dataBaru);
         cabangKontinu = getCabang(depth[j], parent.getID());

         if(cabangKontinu[0].contains("<=")){
            threshold = Double.parseDouble(cabangKontinu[0].substring(3));
            if (trData <= threshold)
               cabangTerpilih = "<= "+threshold;
            else
               cabangTerpilih = "> "+threshold;
         }

         else if(cabangKontinu[0].contains(">")){
            threshold = Double.parseDouble(cabangKontinu[0].substring(2));
            if (trData > threshold)

               cabangTerpilih = "> "+threshold;
            else
               cabangTerpilih = "<= "+threshold;
         }
         child = getNode(depth[j], parent.getID(), cabangTerpilih);

      }

      parent = child;
   }
```

Tabel Lampiran 1. 3 Kode Program Klasifikasi Data Baru

```
        else {
            break;
        }
    }

    if(!mv)
        returnedValue = parent.getNode();
    else
        returnedValue = kondisiProbabilitas;

    return returnedValue;
```

Kode program ini tidak jauh berbeda dengan kode program Fungsi Pengujian Pohon Keputusan karena pada dasarnya apa yang dilakukan pada kedua fungsi ini adalah sama, perbedaannya hanya terletak pada jumlah data yang akan diuji. Pada pengujian pohon keputusan, data yang akan diiterasi adalah seluruh *training data* sedangkan pada Fungsi Klasifikasi Data Baru, data yang diuji hanya data yang berasal dari pengguna setelah memasukkan nilai unsur-unsur cuaca.

- **Fungsi Pemangkasan Pohon (*Pruning Tree*)**

Tabel Lampiran 1. 4 Kode Program Fungsi Pemangkasan Pohon Keputusan

```
if(GUICBR.jmlKelas == 1){
 nodes = new String[6];
 for(i=0;i<this.parents.length;i++){
    node = sameNode(this.depth, this.parents[i]);
    sumVariabel = getTotalKelas(this.depth,this.parents[i]);
    if(!(node.equals("not the same"))){
        stoppingCriteria[i] = true;
        updateInduk(this.parents[i], sumVariabel, node);
        deleteChildren(this.depth,this.parents[i]);
    }

    else{
        nodes[0] = "TIDAKHUJAN";
        nodes[1] = "HUJANSANGATRINGAN";
        nodes[2] = "HUJANRINGAN";
        nodes[3] = "HUJANSEDANG";
        nodes[4] = "HUJANLEBAT";
        nodes[5] = "HUJANSANGATLEBAT";

        jumlahVariabel = getKelasAnak(this.depth,this.parents[i]);
        eec = estimateErrorChildren(jumlahVariabel);

        eeTiapKelas = new double[7];
```

Tabel Lampiran 1. 4 Kode Program Fungsi Pemangkasan Pohon Keputusan (Lanjutan)

```
            for(j=0;j<(eeTiapKelas.length-1);j++){
               eeTiapKelas[j] = sumVariabel[6] * getUCL(sumVariabel[6],(sumVariabel[6] - sumVariabel[j]));
               System.out.println("EE "+(j+1)+" = "+eeTiapKelas[j]);
            }
            eeTiapKelas[6] = eec;
            bestClass = bestClass(eeTiapKelas);

            if(!(bestClass == 6)){
               updateInduk(this.parents[i], sumVariabel, nodes[bestClass]);
               deleteChildren(this.depth,this.parents[i]);
            }
         }
      }
   } else{
      nodes = new String[2];
      for(i=0;i<this.parents.length;i++){
         node = sameNode(this.depth, this.parents[i]);
         sumVariabel = getTotalKelas(this.depth,this.parents[i]);
         if(!(node.equals("not the same"))){
            stoppingCriteria[i] = true;
            updateInduk(this.parents[i], sumVariabel, node);
            deleteChildren(this.depth,this.parents[i]);
         }

          else{
             nodes[0] = "HUJAN";
             nodes[1] = "TIDAKHUJAN";

             jumlahVariabel = getKelasAnak(this.depth,this.parents[i]);

             eec = estimateErrorChildren(jumlahVariabel);

             eeTiapKelas = new double[3];

             for(j=0;j<(eeTiapKelas.length-1);j++){
                eeTiapKelas[j] = sumVariabel[2] * getUCL(sumVariabel[2],(sumVariabel[2] - sumVariabel[j]));
                System.out.println("EE "+(j+1)+" = "+eeTiapKelas[j]);
             }
             eeTiapKelas[2] = eec;
             bestClass = bestClass(eeTiapKelas);

             if(!(bestClass == 2)){
                updateInduk(this.parents[i], sumVariabel, nodes[bestClass]);
                deleteChildren(this.depth,this.parents[i]);
             }
          }
       }
    }
    this.lastDepth = this.depth;
    this.lastParents = this.parents;
```

Kode program ini merupakan isi dari proses pemangkasan pohon keputusan dan tidak mengandung proses rekursif karena proses rekursifnya dilakukan dengan memanggil fungsi di atas pada bagian lain dari program. Langkah yang dilakukan pada fungsi ini dimulai dengan memeriksa apakah kategori atau kelas yang ada pada setiap *leaf* memiliki nilai yang sama. Jika iya maka subpohon yang mengandung *leaves* tersebut akan langsung dipangkas. Sedangkan jika kategori atau kelas yang ada pada *leaves* tersebut berbeda-beda, maka untuk setiap *leaf*

perlu dihitung nilai *upper confidence limit* (UCL). Masing-masing UCL akan dikalikan dengan berapa banyak jumlah kasus yang tergolong ke dalam *leaf* tersebut. Hasil akumulasi ini akan dibandingkan dengan jumlah prediksi *error* pada *leaf* baru yang akan menggantikan subpohon ini. Jika jumlah prediksi *error* pada *leaf* baru lebih kecil maka subpohon tersebut akan dipangkas dan digantikan dengan *leaf* yang bersangkutan.

- **Fungsi Perhitungan Entropi**

Tabel Lampiran 1. 5 Kode Program Fungsi Perhitungan Entropi

```
public double HitungEntropi(double[] jk, double tk) {
x = 0;
for (i=0; i<jk.length ; i++) {
   if (jk[i] == 0) {
      x+=0;

   }

   else {
      try {
         temp = jk[i]/tk;
         x += temp * (log2(temp));
      }

      catch (InputMismatchException inputMismatchException) {
         System.out.println("Error input");

      }

      catch(ArithmeticException arithmeticException) {
         System.out.println("Error aritmetik");
      }
   }
}
if (x!=0) {
   x = -1 * x;
}

return x;
}
```

Kode program ini berfungsi untuk menghitung nilai entropi secara umum sehingga bisa diterapkan untuk menghitung entropi sebelum maupun sesudah proses partisi. Proses perhitungan entropi ini dilakukan dengan menerapkan rumus yang ada pada Persamaan 2.2 dan 2.3.

- **Fungsi Perhitungan *Gain* dan *SplitInfo***

Tabel Lampiran 1. 6 Kode Program Fungsi Perhitungan *Gain* dan *SplitInfo*

```
public double HitungGain(double tk, double et, double[] jk, double[] ek) {
    x = 0;
    for (i=0; i<ek.length; i++) {
        try {
            temp = jk[i]/tk;
            x += temp * ek[i];
        }catch (InputMismatchException inputMismatchException) {
            System.out.println("Error input");
        }

        catch(ArithmeticException arithmeticException) {
            System.out.println("Error aritmetik");
        }
    }
    return (et-x);
}

public double hitungSplitInfo(double[] jumlahKasus, double totalKasus) {
    int i;
    double splitInfo = 0.0, temp = 0.0;

    for(i=0;i<jumlahKasus.length;i++) {
        temp = jumlahKasus[i]/totalKasus;
        if(temp == 0)
            splitInfo -= 0;
        else
            splitInfo -= temp * log2(temp);
    }
    return splitInfo;
}
```

Kode program ini berfungsi untuk menghitung *Gain* dan *SplitInfo* dengan menerapkan rumus yang ada pada Persamaan 2.4 dan 2.6

- **Fungsi Perhitungan *Upper Confidence Limit***

Tabel Lampiran 1. 7 Kode Program Fungsi Perhitungan *Upper Confidence Limit*

```
public double getUCL(int l, int m){
  double x = 0.0;
  double f = Double.parseDouble(Integer.toString(m))/Double.parseDouble(Integer.toString(l));
  double z = 0.69;
  double fl = f/Double.parseDouble(Integer.toString(l));
  double y = fl - (f*fl) + (z*z/(4*Double.parseDouble(Integer.toString(l))*Double.parseDouble(Integer.toString(l))));
  double dual = (2*Double.parseDouble(Integer.toString(l)));
  double sqrtY = Math.sqrt(y);

  double a = (z*z)/dual;
  double b = (z * sqrtY);
  double c = (1 + (z*z/Double.parseDouble(Integer.toString(l))));

  x = (f + a + b) / c;

  return x;
}
```

Kode program ini berfungsi untuk menghitung *Upper Confidence Limit* dengan menerapkan rumus yang ada pada Persamaan 2.8