BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Implementasi dilakukan untuk proses transformasi representasi rancangan ke dalam bahasa pemrograman sehingga dapat dimengerti oleh komputer. Lingkungan implementasi yang akan dijelaskan meliputi lingkungan implementasi perangkat keras dan lingkungan implementasi perangkat lunak.

4.1.1. Lingkungan Perangkat Keras

Perangkat keras yang digunakan pada penelitian ini adalah sebuah PC (*Personal Computer*) dengan spesifikasi sebagai berikut :

1. *Processor* Intel® Core™ 2 Duo Th800 2.00 GHz
2. RAM 2GB
3. *Harddisk* dengan kapasitas 250 GB
4. Monitor
5. *Keyboard*
6. *Mouse*

4.1.2. Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan pada penelitian ini adalah sebagai berikut :

1. Sistem operasi Windows 7 *Home Basic*
2. *Netbeans IDE* 7.2 dalam mengembangkan aplikasi
3. MySQL, yaitu perangkat lunak untuk manajemen basis data SQL.
4. PhpMyAdmin untuk mengelola administrasi MySQL.
5. JDK 1.7, yaitu *development kit* untuk menulis program java.

4.2. Implementasi Program

Berdasarkan rancangan pembuatan sistem pada bab 3, maka pada subbab ini akan dijelaskan implementasi proses – proses tersebut. Secara garis besar

proses dikelompokkan menjadi enam tahap, yaitu tahap *preprocessing*, tahap *frequent 1-itemset*, tahap *frequent itemset*, tahap *generate rule*, tahap *akurasi rule* dan tahap klasifikasi. Implementasi program terdiri dari 15 *class* yang dijelaskan pada Tabel 4.1.

Tabel 4. 1. *Class* pada Implementasi Program dan Fungsinya

Nama *Class*	*Method*	Fungsi
Class Kolom	1. *method* getID	Menginisialisasi parameter ID
	2. *method* getNIP	Menginisialisasi parameter NIP
	3. *method* getNUPTK	Menginisialisasi parameter NUPTK
	4. *method* getTP	Menginisialisasi parameter tingkat pendidikan
	5. *method* getSermapel	Menginisialisasi parameter sertifikasi mata pelajaran
	6. *method* getUsia	Menginisialisasi parameter usia
	7. *method* getPola	Menginisialisasi parameter pola sertifikasi
	8. *method* getMK	Menginisialisasi parameter masa kerja
	9. *method* getLulus	Menginisialisasi parameter lulus
Class Koneksi	1. *method* dbConnection	Mengatur koneksi atau pemanggilan *database*
	2. *method* getData	Mengecek data sesuai kolom
	1. *method* insert	Proses pemasukan data ke dalam *tree*
	2. *method* cari	Proses pencarian data di dalam *tree*
	3. *method* simpanPath	Proses penyimpanan data di dalam *tree*
	4. *method* CPB	Proses pencarian *prefix* dari data di dalam *tree* dan pembentukan *conditional pattern base*
	5. *method* SupportPref	Menyimpan nilai *prefix* dari proses CPB
	6. *method* CFT	Proses pembentukan *conditional FP Tree*
	7. *method* urutFI	Mengurutkan *frequent itemset* yang didapat
	8. *method* koleksiFI	Menyimpan *frequent itemset* yang didapat

	9. *method* FreqIt	Menghasilkan *ArrayList* dengan tipe *String* agar dapat dicetak dengan pemisah (->) antara *antecedent* dan *consequent* dengan *parameter* bertipe *ArrayList<FrequentItemset>*
	10. *method* PrintIt	Menghasilkan *ArrayList* dengan tipe *String* agar dapat dicetak dengan pemisah (->) antara *antecedent* dan *consequent* dengan *parameter* bertipe *Object*
	11. *method* dataFreq	Menghasilkan *ArrayList* yang berisi seluruh *frequent itemset*
	12. *method* antecedent	Menghasilkan *ArrayList* yang berisi *antecedent* dari *frequent itemset*
Class Path	*constructor* Path	Menyimpan data lintasan *Tree*
Class Prefix	*constructor* Prefix	Menyimpan data *prefix* dari seluruh *suffix*
Class FrequentItemset	*constructor* FrequentItemset	Menyimpan data *frequent itemset*
Class Confidence	*constructor* Confidence	Menyimpan data nilai *confidence*
Class FPTreeNode	1. *constructor* FPTreeNode	Inisialisasi variable *tree*
	2. method FPTreeNode	
Class FPTreeHeaderEntry	1. *constructor* FPTreeHeaderEntry	Sebagai *header table* yang menyimpan informasi total bobot tiap *parameter* yang menjadi *frequent 1-itemset* yang telah masuk ke dalam *FP-Tree*
	2. *method* FPTreeHeaderEntry	
Class GUI	1. *method* GUI	Mengatur tampilan GUI dan menjalankan proses sistem keseluruhan. *Class* ini merupakan kelas utama untuk melakukan proses penentuan kandidat awal, *frequent itemset, rule* yang dihasilkan dan juga proses pengujian. Memiliki *constructor* GUI
	2. *method* bacaData	Membaca *file* hasil pengujian dan ditampilkan pada tabel
	3. *method* cek	Menjalankan perhitungan r.wk, penghapusan *rule* yang memiliki r.wk terkecil, perhitungan rata-rata akurasi

		dan rata-rata *lift ratio*
	4. *method* frequentItemset	Menentukan *frequent itemset*
	5. *method* kandidatAwal	Menjalankan proses pencarian kandidat awal *frequent itemset*
	6. method getMinConfidence	Mendapatkan nilai *minimum confidence*
	7. *method* setMinConfidence	Inisialisasi *minimum confidence*
	8. method getMinSupport	Mendapatkan nilai *minimum support*
	9. *method* setMinSupport	Inisialisasi *minimum support*
	10. *method* urutDF	Mengurutkan data hasil perhitungan *discriminant function* (DF)
Class Hitung	1. *method* setTrans	Inisilisasi data transaksi
	2. *method* hitungDF	Menjalankan proses perhitungan nilai DF
	3. *method* hitungLift	Menjalankan proses perhitungan *lift ratio*
	4. *method* hitungConf	Menjalankan proses perhitungan nilai *confidence*
	5. *method* hitungSupport2	Menjalankan proses perhitungan nilai *support* dengan parameter *List*
	6. *method* hitungSupport	Menjalankan proses perhitungan *support* dengan parameter *Object*
	7. *method* hitungAkurasi	Menjalankan proses perhitungan nilai DF
Class Normalisasi	1. *method* Preprocessing	Menjalankan proses *preprocessing*
	2. *method* getTrans	*Method* untuk mendapatkan Trans atau transaksi
Class Pengujian	1. *method* setTrans	Inisialisasi data transaksi
	2. *method* hasilUji	Menjalankan proses pengujian data sehingga dihasilkan kecenderungan kelas dan hasil akurasinya
Class Rule	1. *method* Rule2	Menjalankan proses pengubahan bentuk *rule* hasil proses menjadi kalimat
	2. *method* hapusRWK	Mengurutkan data berdasarkan nilai r.wk dan menghapus data yang memiliki nilai r.wk terkecil

Class Tree	1. *method* setTrans	Inisialisasi data transaksi
	2. *method* setHitungAntece	Inisialisasi hasil perhitungan *antecedent* data
	3. *method* getHitungAntece	Mendapatkan hasil perhitungan *antecedent* data
	4. *method* fpTree	Menjalankan proses pembentukan FP-Growth

4.2.1. Implementasi Tahap Preprocessing

Pada tahap ini dilakukan penginisialisasian data, agar memudahkan dalam memproses algoritma *FP-Growth*. Setiap data yang ditemukan di dalam *database* akan diubah menjadi angka sesuai *flowchart preprocessing* pada gambar 3.2. Source *code* preprocessing dapat dilihat pada *source code* 4.1.

```
1   DefaultTableModel Preprocessing(String JumRecord, JTable
    tabel_hasil) throws Exception {
2           Object[] klr = {"No", "NIP", "NUPTK", "Tingkat
    Pend", "Matapel Sertifikasi", "Usia", "Jenis Sertifikasi",
    "Masa Kerja", "Status Sertifikasi"};
3           Prepro = new DefaultTableModel(null, klr);
4           String query = "SELECT ID,RIGHT
    (NIP,3),NUPTK,Tingkat_Pend,Matapel_Sertifikasi,Usia,Jenis_Se
    rtifikasi,Masa_Kerja,Status_Sertifikasi FROM data2 LIMIT " +
    JumRecord + ";";
5           tabel_hasil.setModel(Prepro);
6           ResultSet rs = null;
7           Kolom field2;
8           Trans = new ArrayList<Kolom>();
9           Statement stmt;
10          Connection conn = null;
11          try {
12              String url = "jdbc:mysql://localhost/ser";
13              String id = "root";
14              String pw = "";
15  conn = (Connection) DriverManager.getConnection(url, id,
    pw);
16              stmt = (Statement) conn.createStatement();
17              rs = stmt.executeQuery(query);
18  while (rs.next()) {
19                  if (rs.getString(2).isEmpty()) {
20                      NIP = 1;
21                  } else {
```

```
22                      NIP = 2;
23                  }
24                  if (rs.getString(3).isEmpty()) {
25                      NUPTK = 3;
26                  } else {
27                      NUPTK = 4;
28                  }
29                  if (rs.getString(4).equals("SMA")) {
30                      TP = 5;
31                  } else {
32                      TP = 6;
33                  }
34                  if (rs.getString(5).isEmpty()) {
35                      SerMapel = 7;
36                  } else {
37                      SerMapel = 8;
38                  }
39                  if (rs.getInt(6) <= 35) {
40                      usia = 9;
41                  } else {
42                      usia = 10;
43                  }
44                  if (rs.getString(7).equals("Portofolio")) {
45                      pola = 11;
46                  } else {
47                      pola = 12;
48                  }
49                  if (rs.getInt(8) <= 20) {
50                      MK = 13;
51                  } else {
52                      MK = 14;
53                  }
54                  if (rs.getString(9).equals("Lulus")) {
55                      lulus = 15;
56                  } else {
57                      lulus = 16;
58                  }
59      field2=new Kolom(ID, NIP, NUPTK, TP, SerMapel, usia,
pola, MK, lulus);
```

```
60                  Trans.add(field2);
61                  Object[] field = {rs.getString(1), NIP,
NUPTK, TP, SerMapel, usia, pola, MK, lulus};
62                  Prepro.addRow(field);
63              }
64          } catch (SQLException e) {
65              System.err.println("koneksi gagal");
66          }
67          return Prepro;
68  }
```

Source Code* 4.1. Proses *Preprocessing

Awalnya dilakukan pembacaan data pada database (baris 6-17). Selanjutnya masing-masing *record* akan diinisialisasi sesuai dengan kolomnya (baris 19-58). Selanjutnya data yang sudah terinisialisasi akan ditambahkan ke *field2* dan ditambahkan ke List *Trans*, yang merupakan kumpulan data terinisialisasi (baris 59-60).

4.2.2. Implementasi Tahap *Frequent 1-Itemset*

Pada tahap ini, dilakukan proses seleksi *parameter* sertifikasi guru yang frekuensi kemunculannya pada data transaksi lebih besar atau sama dengan minimum *support*. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.2 dan 4.3.

```
1   public List kandidatAwal() {
2           ArrayList temp = new ArrayList();
3           List list = new ArrayList();
4           List list1 = new ArrayList();
5           Set a = null;
6
7           for (int i = 0; i < Trans.size(); i++) {
8               list.add(Trans.get(i).getNIP());
9               list.add(Trans.get(i).getNUPTK());
10              list.add(Trans.get(i).getLulus());
11              list.add(Trans.get(i).getMK());
12              list.add(Trans.get(i).getPola());
13              list.add(Trans.get(i).getSermapel());
14              list.add(Trans.get(i).getTP());
```

```
15              list.add(Trans.get(i).getUsia());
16              a = new HashSet(list);
1           }
2           for (Object obj : a) {
3               list1.add(obj);
4               Collections.sort(list1);
5           }
6           Iterator item = list1.iterator();
7           while (item.hasNext()) {
8               temp.add(item.next());
9           }
10          kandidat.add(temp);
11          return kandidat;
12      }
```

Source Code* 4.2. Proses Pembentukan Kandidat Awal *Frequent 1-Itemset

Proses pada fungsi ini dilakukan dengan pembacaan setiap data transaksi dan diambil *parameter* sertifikasi guru pada *index* data transaksi yang sedang dibaca pada *ArrayList list.* Jika data tersebut terdapat pada *list,* maka akan disimpan pada *ArrayList* kandidat (baris 3-11).

```
1   public void frequentItemset() {
2
3           Object[] freq = {"Frequent Itemset", "Support"};
4           satuItem = new DefaultTableModel(null, freq);
5           oneitemset.setModel(satuItem);
6           List frequentList = new ArrayList();
7           List candidat = (List) kandidat.get(0);
8           for (int j = 0; j < candidat.size(); j++) {
9               if (hitung.hitungSupport(candidat.get(j)) >=
getMinSupport() / 100 * Trans.size()) {
10                  FrequentList.add(candidat.get(j));
11  Object[] one =
{candidat.get(j),hitung.hitungSupport(candidat.get(j))};
12                  satuItem.addRow(one);
13              }
14          }
15      }
```

Source Code* 4.3. Proses Seleksi *Parameter* untuk Pembentukan *Frequent 1-Itemset

Pada proses ini, dilakukan perhitungan nilai *support* tiap *parameter* sertifikasi guru dengan cara nilai frekuensi kemunculannya dibagi dengan total transaksi. Kemudian nilai *support* tiap *parameter* dibandingkan dengan nilai minimum *support* yang telah ditentukan. *Parameter* yang memiliki nilai *support* lebih dari atau sama dengan minimum *support* yang akan menjadi *frequent 1-itemset*.

Untuk menghitung nilai *support* dibutuhkan total transaksi sehingga dilakukan pemanggilan fungsi Trans.size(). Pembacaan data Frek asli dilakukan setiap data yang ada yang telah tersimpan pada *ArrayList* kandidat. Setiap index selanjutnya dilakukan perhitungan nilai frekuensinya pada getMinSupport() dibagi dengan total transaksi jml (baris 7-9). Hasil dari perhitungan nilai *support* tersebut disimpan dalam *FrequentList* (baris 10-24).

4.2.3. Implementasi Tahap *Frequent Itemset*

Pada tahap ini, dilakukan pembentukan kombinasi *parameter* yang menjadi *frequent 1-itemset* dan kemudian dilakukan perhitungan *support* untuk masing – masing kombinasi. Kombinasi yang memiliki nilai *support* lebih besar atau sama dengan nilai minimum *support* yang telah ditentukan, akan menjadi *frequent itemset*. Tahap *frequent itemset* terdiri dari 4 proses, yaitu pembentukan *FP-tree*, proses *conditional pattern base*, proses *conditional FP-tree*, dan proses koleksi *frequent itemset*.

4.2.3.1. Implementasi Proses Pembentukan *FP-Tree*

Proses pembentukan *FP-Tree* ini bertujuan untuk menyimpan data transaksi sehingga tidak perlu dilakukan pembacaan data transaksi berulang kali. Pada proses ini, *parameter* setiap transaksi dimasukkan ke dalam *tree* sebagai satu lintasan. Node dari *FP-tree* menyimpan informasi kode *parameter*, bobot, kedalaman, indeks pada *header table*, indeks *node*, *node parent*, *node sibling*, dan *node link*. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.4.

Pembentukan *FP-tree* diimplementasikan pada *class FPTree* yaitu pada fungsi *insert*. Pada fungsi ini, parameter inputan berupa ArrayList<String> items yang merupakan semua *parameter* yang dimiliki suatu transaksi dan *int count*

yang merupakan jumlah bobot *parameter*.

Proses awal adalah inisialisasi *node root*. Kemudian dilakukan pembacaan tiap *index* data pada items, yang berarti data *parameter* pada *index* tersebut dimasukkan ke dalam struktur *FP-tree* (baris 1-2).

Parameter yang dimasukkan pada struktur *FPtree*, diambil *index*nya pada item2index yang menyimpan *index parameter* pada *header table*. *Header table* merupakan tabel informasi mengenai *node parameter* yang sama yaitu yang menyimpan *index node* tiap *parameter*, kode *parameter*nya, total frekuensi yang telah masuk ke dalam *FP-tree* tiap *parameter*, dan *head node* dari tiap *parameter*. Setelah *index* dari kode *parameter* didapatkan, maka pada *header table* data pada *index* tersebut ditambahkan 1 nilai *count*-nya (baris 3-5).

Proses selanjutnya adalah inisialisasi *node pointer* dengan nama walker, dimana walker merupakan anak dari *node* sekarang. Selama *node* sekarang memiliki anak, maka *pointer* akan menulusuri semua saudara dari anak *node* sekarang. Selama penelusuran, dilakukan pengecekan kode *parameter* pada *node* tersebut sama atau tidak dengan kode *parameter* yang dimasukkan ke dalam *FP-tree*. Jika ada yang sama maka posisi *pointer* berada pada *node* tersebut, sedangkan bila tidak ada yang sama berarti belum pernah terbentuk *node* dengan kode *parameter* yang dimasukkan ke dalam *FP-tree* yang menjadi anak dari *node* sekarang atau walker bernilai *null*. Jika walker bernilai *null* maka diciptakan *node* baru yang dideklarasikan dengan nama new_node yang menjadi anak dari *node* sekarang dan *head node* pada *header table* untuk kode *parameter* tersebut (baris 6-16). Kemudian *node* sekarang adalah *node* baru. Sedangkan, untuk walker yang tidak bernilai *null* maka bobot pada *node* yang dirujuk oleh *pointer* walker ditambahkan 1 dan *node* sekarang adalah *node* yang dirujuk oleh *pointer* walker (baris 17-28).

```
1   void insert(ArrayList<Integer> items, int count) {
2         FPTreeNode current_node = root;
3         for (int index = 0; index < items.size(); index++) {
4             int entry_index = ((Integer) item2index.get(new
Integer(items.get(index)))).intValue();
5             header[entry_index].count += count;
6             FPTreeNode walker = current_node.child;
7     for (; walker != null; walker = walker.sibling) {
```

```
8                   if (walker.item == items.get(index)) {
9                       break;
10                  }
11              }
12              if (walker == null) {
13                  if (current_node.child != null) {
14                      hasMultiplePaths = true;
15                  }
16                  count_nodes++;
17                  FPTreeNode new_node = new
FPTreeNode(items.get(index), count,
18                          index + 1, entry_index, count_nodes,
19                          current_node,
20                          current_node.child,
21                          header[entry_index].head);
22                  header[entry_index].head = new_node;
23                  current_node.child = new_node;
24                  current_node = new_node;
25              }
26              else {
27                  walker.count += count;
28                  current_node = walker;
29              }
30  }
31  }
```

Source Code* 4.4. Proses Pembentukan *FP-Tree

4.2.3.2. Implementasi Proses *Conditional Pattern Base*

Pada proses ini, dilakukan pengkoleksian lintasan yang dimiliki tiap *suffix*, dimana *suffix* merupakan *parameter* yang menjadi *frequent 1-itemset*. Selain itu, proses ini juga menyimpan bobot tiap *node* pada lintasan tersebut dengan tujuan untuk digunakan dalam menghitung nilai *support* tiap *parameter* yang menjadi *prefix* pada tiap lintasan. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.5.

Fungsi CPB memiliki parameter inputan berupa *parameter* yang menjadi *suffix* int suffix dan bertipe data *String* serta *arraylist* yang menyimpan data lintasan p. Proses dilakukan dengan melakukan pembacaan tiap baris data lintasan

p. Untuk setiap *index* data lintasan p dilakukan pengecekan apakah *item* terakhir dari lintasan p *index* tersebut sama dengan *suffix* (baris 1-7). Apabila sama maka *suffix* disimpan ke dalam *ArrayList* itemSuf dan item selain *suffix* disimpan ke dalam A*rrayList* itemPref yang berarti merupakan *prefix* dari lintasan *index* tersebut dan bobot dari lintasan tersebut disimpan ke dalam *ArrayList* bobotPref (baris 8-16).

```
1  Prefix CPB(int suffix, ArrayList<Path> p) {
2          Prefix hasil = new Prefix();
3          ArrayList itemPref = new ArrayList<Integer>();
4          ArrayList itemSuf = new ArrayList<Integer>();
5          ArrayList bobotPref = new ArrayList<Integer>();
6          for (int i = 0; i < (p.size()); i++) {
7                 itemSuf.add(suffix);
8  if ((p.get(i).kd_jb.get(p.get(i).kd_jb.size() - 1)) ==
   suffix) {
9    for (int j = 0; j < p.get(i).kd_jb.size() - 1; j++) {
10              itemPref.add(p.get(i).kd_jb.get(j));
11             bobotPref.add(p.get(i).bobot.get(j));
12                }
13   hasil = new Prefix(itemPref, itemSuf, bobotPref);
14             }
15         }
16         return hasil;
17     }
```

Source Code* 4.5. Proses *Conditional Pattern Base

4.2.3.3. Implementasi *Conditional FP-Tree*

Proses ini bertujuan untuk mengetahui nilai *support* tiap lintasan *prefix* tiap *suffix*. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.6.

Fungsi CFT memiliki parameter inputan berupa kombinasi *parameter* yang dideklarasikan sebagai ArrayList<Integer> dengan nama variabel items dan data lintasan ArrayList<Path> pa. Proses dilakukan dengan mengecek setiap data lintasan sampai ditemukan data lintasan yang terdiri dari kombinasi *parameter* yang sama pada items. Setiap data lintasan yang sesuai maka nilai bobotnya dijumlahkan dan disimpan pada variabel bobot yang bertipe *double* (baris 1-6). Setelah pengecekan dilakukan pada semua data lintasan, maka dihitung nilai

support hubungan antara kombinasi *parameter* tersebut dengan cara total bobot yang disimpan dalam variabel bobot dibagi dengan jumlah transaksi yang disimpan dalam variabel totData. Kemudian kombinasi tersebut disimpan dalam ArrayList<String> dan nilai *support* kombinasi *parameter* disimpan dalam ArrayList<Double> FIbobotnya (baris 7-25).

```
1  FrequentItemset CFT(ArrayList<Integer> items,
   ArrayList<Path> pa) {
2          int nilai;
3          double bobot = 0;
4  ArrayList<Integer> fItemset = new ArrayList<Integer>();
5   ArrayList<Double> FIbobotnya = new ArrayList<Double>();
6          FrequentItemset hasil = new FrequentItemset();
7          for (int i = 0; i < pa.size(); i++) {
8              nilai = 0;
9              if (pa.get(i).kd_jb.get(pa.get(i).kd_jb.size() -
   1) == items.get(items.size() - 1)) {
10                 for (int x = 0; x < items.size() - 1; x++) {
11           if (pa.get(i).kd_jb.contains(items.get(x))) {
12                         nilai += 1;
13                     }
14                 }
15                 if (nilai == items.size() - 1) {
16                     bobot = bobot + pa.get(i).bobot.get(0);
17                 }
18             }
19         }
20         bobot = (bobot / total);
21         for (int j = 0; j < items.size(); j++) {
22             fItemset.add(items.get(j));
23             FIbobotnya.add(bobot);
24         }
25         hasil = new FrequentItemset(fItemset, FIbobotnya);
26         return hasil;
27 }
```

Source Code* 4.6. Proses *Conditional FP-Tree

4.2.3.4. Implementasi Koleksi *Frequent Itemset*

Pada proses ini, dilakukan pengecekan nilai *support* setiap lintasan *prefix*

tiap *suffix* terhadap nilai minimum *support* yang telah ditentukan. Lintasan *prefix* yang memiliki nilai *support* lebih besar atau sama dengan nilai minimum *support* akan menjadi *frequent itemset*. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.7.

Fungsi koleksiFI berfungsi untuk mengkoleksi hasil dari proses *Conditional FP-tree* yang memiliki nilai *support* lebih besar atau sama dengan nilai *minimum support* yang telah ditentukan. Proses dilakukan dengan menghapus data *frequent 1-itemset* yang dideklarasikan dengan ArrayList<FrequentItemset> hasil untuk kandidat *frequent itemset* yang nilai *minimum support*nya tidak memenuhi *minimum support* yang telah ditentukan.

```
1  ArrayList<FrequentItemset>
   koleksiFI(ArrayList<FrequentItemset> hasil) {
2          for (int i = 0; i < hasil.size(); i++) {
3          if ((double) hasil.get(i).bobot.get(0) < minsup) {
4                  hasil.remove(hasil.get(i));
5                  i = i - 1;
6              }
7          }
8          return hasil;
9      }
```

Source Code* 4.7. Proses Koleksi *Frequent Itemset

4.2.4. Implementasi Proses *Generate Rule*

Pada tahap ini, dilakukan perhitungan nilai *confidence* pada setiap *frequent itemset*. Nilai *confidence* didapatkan dari nilai *support frequent itemset* dibagi dengan nilai *support* bagian *antecedent*. *Frequent Itemset* yang memiliki nilai *confidence* lebih besar atau sama dengan nilai *minimum confidence* yang telah ditentukan akan menjadi *rule*. Tahap *generate rule* terdiri dari 2 proses, yaitu perhitungan *confidence* dan proses seleksi *rule*.

4.2.4.1. Implementasi Proses Hitung *Confidence*

Proses ini merupakan proses awal dari tahap *generate rule*. Pada proses ini, tiap *frequent itemset* dihitung nilai *confidence*-nya. Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.8.

Fungsi hitungConf berfungsi untuk menghitung nilai *confidence* untuk setiap data *frequent itemset*. Parameter inputannya bertipe *ArrayList*. Perhitungan nilai *confidence* didapat dari hasil bagi *consequent* yang diimplementasikan pada *hitungSupport2(d)* dengan nilai *antecedent* yang diimplementasikan pada *hitungSupport2(a)*.

```
1   public double hitungConf(ArrayList d) {
2           List a = new ArrayList();
3           double confidence;
4           double antecedent;
5           double consequent;
6           for (int i = 0; i < d.size(); i++) {
7               if (i <= d.size() - 2) {
8                   a.add(d.get(i));
9               }
10          }
11          consequent = hitungSupport2(d);
12          antecedent = hitungSupport2(a);
13          confidence = consequent / antecedent;
14          a.clear();
15          return confidence;
16      }
```

Source Code* 4.8. Proses Hitung *Confidence

4.2.4.2. Implementasi Proses Seleksi Rule

Pada proses ini, *frequent itemset* yang memiliki nilai *confidence* lebih besar atau sama dengan *minimum confidence* dibangkitkan sebagai *rule* (baris16-28). Implementasi dari proses ini dapat ditunjukkan pada *source code* 4.9.

```
1   ArrayList d = new ArrayList();
2           List e = new ArrayList();
3           ArrayList<Object> hasilFreq1 = p.antecedent(hasil);
4           ArrayList<String> dataFreq = p.dataFreq(hasil);
5           hasilHitung = new double[hasilFreq.size()];
6           for (int i = 0; i < hasilFreq1.size(); i++) {
7     hasilHitung[i] = (hitung.hitungDF(hasilFreq1.get(i)));
8           }
```

```
9           for (int i = 0; i < hasilFreq.size(); i++) {
10              b = hasilFreq.get(i).toString();
11          StringTokenizer st1 = new StringTokenizer(b, ",->[] ");
12              while (st1.hasMoreTokens()) {
13                  word = st1.nextToken();
14                  d.add(word);
15              }
16  String[] isi = {hasilFreq.get(i),
    String.valueOf(hitung.hitungSupport2(d)),
    String.valueOf(hitung.hitungConf(d))};
17      Tmod.addRow(isi);
18              ArrayList kata = new ArrayList();
19              ArrayList list = new ArrayList();
20
21              if ((hitung.hitungConf(d) >= minConf / 100) &&
    (hitung.hitungLift(d) > 1)) {
22                  Object[] field = {hasilFreq.get(i),
    rules.Rule2(hasilFreq.get(i)), hitung.hitungConf(d),
    hitung.hitungLift(d), hitung.hitungAkurasi(d)};
23
24                  rule.addRow(field);
25              } else {
26                  d.clear();
27                  continue;
28              }
```

Source Code* 4.9. Proses Seleksi *Rule

Fungsi koleksiRule memiliki inputan data *frequent itemset* yang memenuhi nilai *minimum confidence* yaitu ArrayList<Rule> rul. Proses dalam fungsi ini adalah mengubah format penulisan *rule* dari format kode menjadi nama dari *rule* tersebut. *Rule* disimpan pada *ArrayList<Object>hasilFreq1*.

4.2.5. Implementasi Proses *Lift Ratio*

Pada tahap ini dilakukan perhitungan nilai *lift ratio* pada *frequent itemset* yang menjadi *rule*. Nilai *lift ratio* didapatkan dari hasil pembagian dari nilai *confidence* terhadap nilai *support* item yang menjadi *consequent.* Nilai *lift ratio* ini diperoleh dari nilai *confidence* yang diimplementasikan pada *hitungConf* dengan *expectConf* (baris 8-13).Implementasi dari proses ini dapat ditunjukkan

pada *source code* 4.12.

```
1   public double hitungLift(ArrayList d) {
2           double expectConf = 0;
3           double confidence = 0;
4           double lift = 0;
5           double jumLift = 0;
6           ArrayList cons = new ArrayList();
7           double totalAccu = 0;
8           confidence = hitungConf(d);
9           cons.add(d.get(d.size() - 1));
10          jumLift = hitungSupport2(cons);
11          expectConf = jumLift / Trans.size();
12          lift = confidence / expectConf;
13          return lift;
14      }
```

Source Code* 4.11. Proses *Lift Ratio

4.2.6. Implementasi Proses Penentuan Kelayakan Sertifikasi Guru

Pada tahap ini, dilakukan tahap klasifikasi dengan proses *Associative Classification* untuk menentukan layak atau tidaknya seorang PTK lulus dalam sertifikasi guru. Proses ini menggunakan *rule-rule* yang telah dibentuk pada proses pelatihan.

4.2.6.1. Implementasi Proses *Associative Classification*

Dari *rule-rule* yang telah didapat akan dilatih menggunakan data latih dengan cara menghitung nilai DF setiap *rule* Nilai DF *rule* kemudian diurutkan dari nilai terbesar ke nilai terkecil. Jika *rule* pertama mengklasifikasikan data latih dengan benar, maka nilai *rightN*-nya bertambah. Jika tidak maka *rule* diletakkan di urutan terakhir dan nilai *wrongN*-nya bertambah (baris 18-30).

```
1   public double hitungDF(Object rules) {
2           double df = 1;
3           double jum = 0;
4           double rul = 1;
5           ArrayList a = new ArrayList();
6           String b = null, word = null;
```

```
7           ArrayList list = new ArrayList();
8           list.add(String.valueOf(Trans.get(0).getNIP()));
9           list.add(String.valueOf(Trans.get(0).getNUPTK()));
10          list.add(String.valueOf(Trans.get(0).getLulus()));
11          list.add(String.valueOf(Trans.get(0).getMK()));
12          list.add(String.valueOf(Trans.get(0).getPola()));
13  list.add(String.valueOf(Trans.get(0).getSermapel()));
14          list.add(String.valueOf(Trans.get(0).getTP()));
15          list.add(String.valueOf(Trans.get(0).getUsia()));
16          b = rules.toString();
17          StringTokenizer st1 = new StringTokenizer(b, ",[]
");
18          while (st1.hasMoreTokens()) {
19              word = st1.nextToken();
20              if (list.contains(word)) {
21                  jum = 1;
22              } else {
23                  jum = 0;
24              }
25              rul = rul * jum;
26          }
27          df = hitungConf(list) * rul;
28          rul = 1;
29          return df;
30      }
```

***Source Code* 4.12. Proses Perhitungan Nilai DF**

Nilai DF yang telah didapat kemudian diurutkan secara menurun.

```
1   public void urutDF(ArrayList list, double[] hasilDF) {
2           int index = 0;
3           double temp = 0;;
4
5           for (int i = 0; i < list.size(); i++) {
6               for (int j = i + 1; j < list.size(); j++) {
7                   if (hasilDF[i] <= hasilDF[j]) {
8                       temp = hasilDF[i];
9                       hasilDF[i] = hasilDF[j];
10                      hasilDF[j] = temp;
```

```
11                      index = j;
12                      Collections.swap(list, i, index);
13                  } }
14                  } }
```

***Source Code* 4.13. Proses Pengurutan Nilai DF**

Jika semua *rule* telah dilatih pada semua data latih, maka dilakukan pengecekan pada setiap *rule*. Jika nilai *rightN rule* = 0 maka *rule* akan dibuang. Jika lebih dari 0 maka akan dimasukkan dalam *classifier* kemudian dihitung nilai *r.wk*-nya dan nilai *r.wk* terkecil akan dibuang (barsi 6-22).

```
1   ArrayList hapusRWK(ArrayList list, double[] hasilRWK) {
2           int index = 0;
3           double temp = 0, hasilConf = 0;
4
5           ArrayList ruleDF = new ArrayList();
6           for (int i = 0; i < list.size(); i++) {
7               for (int j = i + 1; j < list.size(); j++) {
8                   if (hasilRWK[i] <= hasilRWK[j]) {
9                       temp = hasilRWK[i];
10                      hasilRWK[i] = hasilRWK[j];
11                      hasilRWK[j] = temp;
12                      index = j;
13                      Collections.swap(list, i, index);
14                  }
15              }
16          }
17          for (int i = 0; i < list.size(); i++) {
18              if (hasilRWK[i] != hasilRWK[list.size() - 1]) {
19                  ruleDF.add(list.get(i));
20              }
21          }
22  return ruleDF;
23           }   }
```

***Source Code* 4.14. Proses Hapus Nilai RWK**

Klasifikasi terhadap data baru atau data uji dapat dilakukan jika terdapat

rule yang dibangkitkan. Langkah pertama adalah memilih dan melakukan pembacaan data uji. Selanjutnya data uji ditransformasi seperti pada proses pelatihan. Kemudian dilakukan perhitungan nilai DF untuk setiap *rule* pada setiap *record*.

Hasil perhitungan DF pada *record* ke-n dijumlahkan sesuai dengan masing-masing kelas. Jumlah DF masing-masing kelas ini kemudian dibagi dengan jumlah seluruh nilai DF untuk menentukan hasil klasifikasi data uji (baris 10-31). Jika jumlah nilai DF pada kelas C1 lebih besar daripada kelas C2, maka data uji tersebut diklasifikasikan sebagai kelas C1. Sebaliknya, jika jumlah nilai DF pada kelas C2 lebih besar daripada kelas C1, maka data uji diklasifikasikan sebagai kelas C2. Jika jumlah nilai DF kelas C1 sama dengan kelas C2, maka dilihat jumlah *rule* kelas mana yang paling banyak dibangkitkan (baris 32-38). Proses ini dapat dilihat pada *source code* 4.15.

```
1           for (int i = 0; i < list1.size(); i++) {
2               b = list1.get(i).toString();
3     StringTokenizer st1 = new StringTokenizer(b, ",[] ");
4               while (st1.hasMoreTokens()) {
5                   word = st1.nextToken();
6                   list.add((word));
7                   temp.add(Integer.parseInt(word));
8               }
9               Collections.sort(temp);
10              for (int k = 0; k < list.size() - 1; k++) {
11                  if (list2.contains(list.get(k))) {
12                      right = 1;
13                  } else {
14                      right = 0;
15                  }
16                  benar *= right;
17                  right = 1;
18              }
19              FCAR[i] = hitung.hitungConf(list) * benar;
20              if (list.contains("15")) {
21                  kelas1 += FCAR[i];
22              } else if (list.contains("16")) {
```

```
23                      kelas2 += FCAR[i];
24                  }
25                  totFCAR += FCAR[i];
26                  right = 1;
27                  benar = 1;
28                  list.clear();
29              }
30              hasil1 = kelas1 / totFCAR;
31              hasil2 = kelas2 / totFCAR;
32              if (hasil1 >= hasil2) {
33                  kelas = "Lulus";
34                  hasilAkhir = 15;
35              } else {
36                  kelas = "Tidak Lulus";
37                  hasilAkhir = 16;
38              }
39              if (hasilAkhir == Trans.get(j).getLulus()) {
40                  totAkurasi += 1;
41              }
42              Object[] one = {j + 1, hasil1, hasil2, kelas};
43              hslKelas.addRow(one);
44              hasil1 = 0;
45              hasil2 = 0;
46              list2.clear();
47          }
48          rataAkurasi = totAkurasi / Trans.size() * 100;
49          hasilAkur.setText("Akurasi = " + rataAkurasi + "%");
50      }
51  }
```

***Source Code* 4.15. Proses Klasifikasi Data Uji**

4.3. Implementasi Antarmuka

Implementasi antarmuka sistem seperti yang telah dijelaskan dalam rancangan antarmuka pada subbab 3.4, terdiri dari bagian *input*, bagian *output*, tombol proses dan tombol *refresh*. Bagian input terdiri dari tiga data *input*, yaitu data *input* jumlah *record*, data *input* minimum *support* dan data *input* minimum *confidence*. Bagian output terdiri dari 2 menu utama. Menu pertama adalah tab

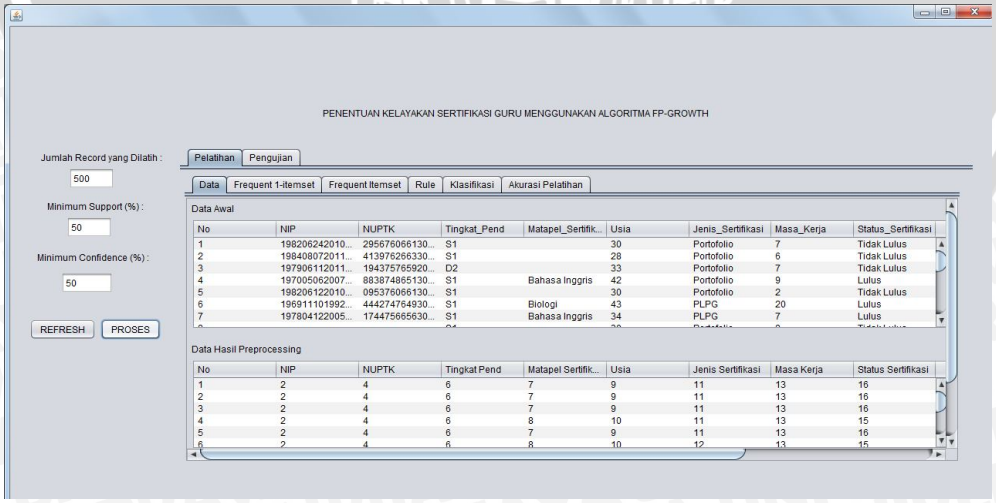
pelatihan yang memiliki enam submenu yaitu submenu Data, submenu *Frequent 1-itemset*, submenu *Frequent Itemset*, submenu *Rule*, submenu Klasifikasi dan submenu Akurasi Pelatihan. Menu kedua yaitu tab pengujian dimana tab ini berisi 3 submenu yaitu submenu Data, submenu Klasifikasi dan submenu Pengujian Data Uji.

Bagian *input* yang pertama adalah *input* jumlah *record* dari data yang ingin dijadikan data latih, i*nput* yang kedua adalah *field input* nilai minimum *support* dan *input* yang ketiga adalah *field input* nilai minimum *confidence*. Pertama-tama user menginputkan jumlah *record* yang diinginkan, kemudian menginputkan *minimum support dan minimum confidence* dengan skala 1-100. Setelah data inputan terisi dengan benar, maka ketika tombol 'proses' ditekan maka sistem akan menjalankan proses dan data *output* pada submenu-submenu yang ada akan ditampilkan pada masing – masing tabel.

4.3.1. Antarmuka Menu Pelatihan

4.3.1.1. Antarmuka Data Awal dan Data Hasil Preprocessing

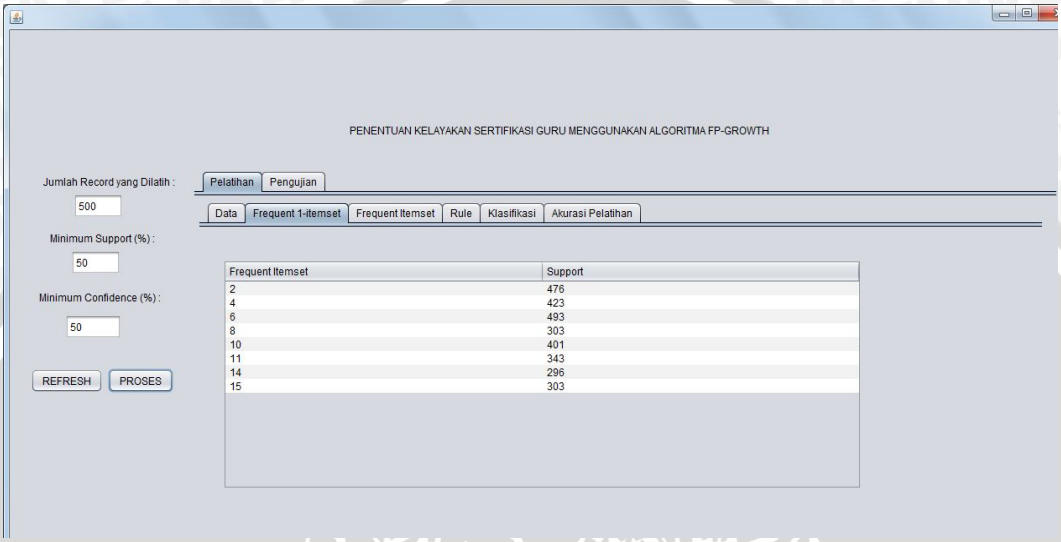
Data awal dan data hasil *preprocessing* terdiri dari Nomor, NIP, NUPTK, tingkat pendidikan, mata pelajaran sertifikasi, usia, jenis sertifikasi, masa kerja, dan status sertifikasi. Pada tab ini, *parameter-parameter* akan dinormalisasi menjadi angka-angka agar dapat diproses pada tahap selanjutnya. Implementasi antarmuka data awal dan data hasil sertifikasi dapat dilihat pada gambar 4.1.



Gambar 4.1. Antarmuka Data Awal dan Data Hasil *Preprocessing*

4.3.1.2. Antarmuka Submenu *Frequent 1-Itemset*

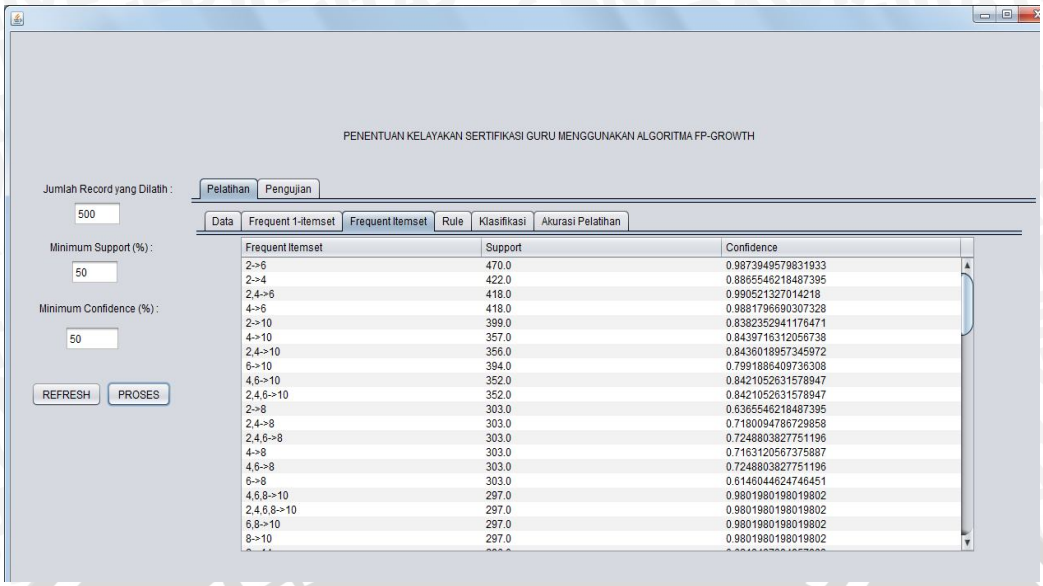
Tab submenu *frequent 1-itemset* ini terdiri dari kolom *frequent itemset* dan *support*. Pada tab ini akan ditampilkan *parameter-parameter* yang nilainya memenuhi nilai *minimum support* yang telah ditentukan, yang tidak memnuhi akan ditolak dan tidak diproses pada proses selanjutnya.



Gambar 4.2. Antarmuka Submenu *Frequent 1-Itemset*

4.3.1.3. Antarmuka Submenu *Frequent Itemset*

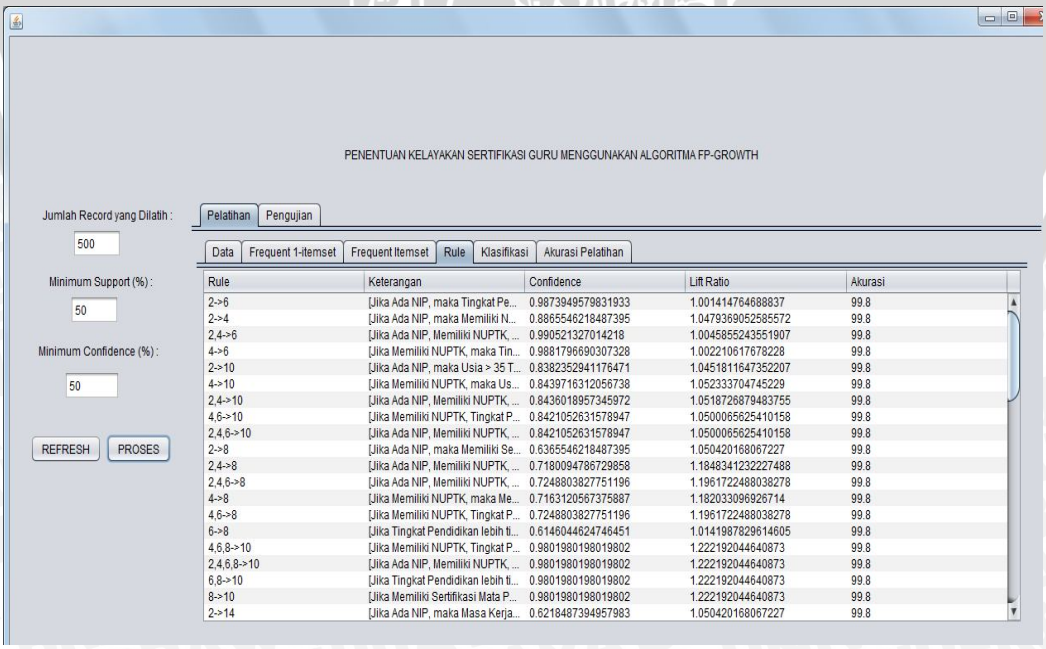
Tab submenu *frequent itemset* terdiri dari kolom *frequent itemset* yang telah dibentuk menjadi *conditional patterb base*, kolom *support* dan *confidence*. Pada tab ini, nilai *support* setiap lintasan *prefix* tiap *suffix* yang memiliki nilai *support* lebih besar atau sama dengan nilai minimum *support* akan ditampilkan sebagai *frequent itemset* beserta nilai *support* dan *confidence*nya.



Gambar 4.3. Antarmuka Submenu *Frequent Itemset*

4.3.1.4. Antarmuka Submenu *Rule*

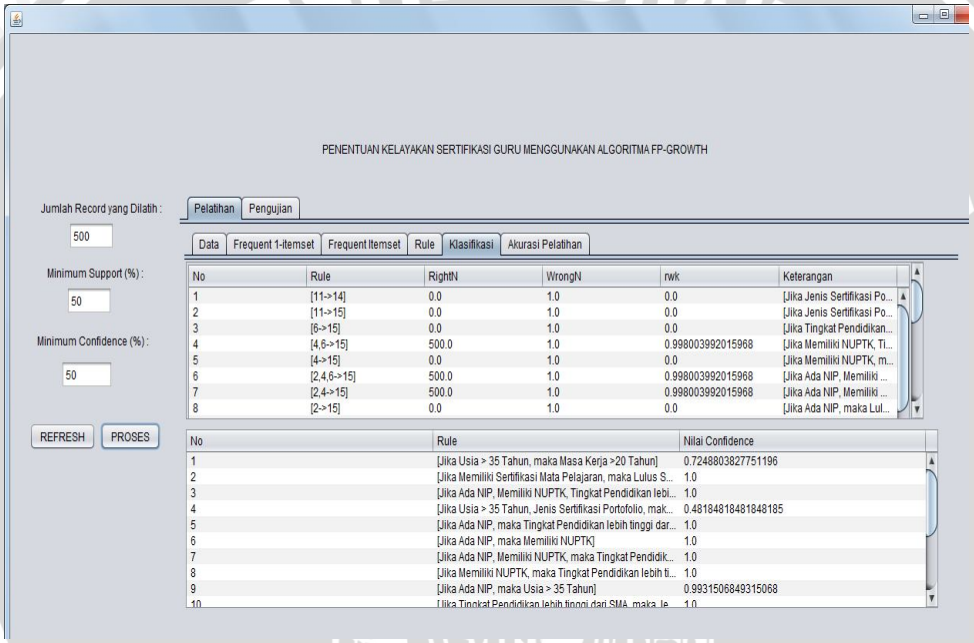
Pada tab submenu *rule* ini ditampilkan *rule* yang telah lolos seleksi, yaitu *frequent itemset* yang memiliki nilai *confidence* memenuhi nilai *minimum confidence* yang telah ditentukan. Pada submenu ini diberikan kolom keterangan yang berisi kode *rule* yang telah diubah ke dalam bentuk kalimat. Selain itu nilai *confidence, lift ratio,* dan akurasi tiap *rule* juga ditampilkan.



Gambar 4.4. Antarmuka Submenu *Rule*

4.3.1.5. Antarmuka Submenu Klasifikasi

Pada tab submenu klasifikasi ini ditampilkan *rule-rule* yang terbentuk untuk proses klasifikasi. Pada submenu ini diberikan dua *panel*, dimana *panel* pertama berisi hasil perhitungan nilai *rightN, wrongN* dan r.wk. Pada panel ini terdapat kolom *rule* yang menampilkan *rule* yang lolos seleksi serta terdapat kolom keterangan yang berisi kode *rule* yang telah diubah ke dalam bentuk kalimat. *Rule* yang memiliki nilai r.wk terkecil akan dihapus dan tidak dimasukkan pada *panel* kedua, dimana pada *panel* tersebut terdapat kolom *rule* yang menampilkan *rule* untuk proses klasifikasi beserta nilai *confidence*-nya.

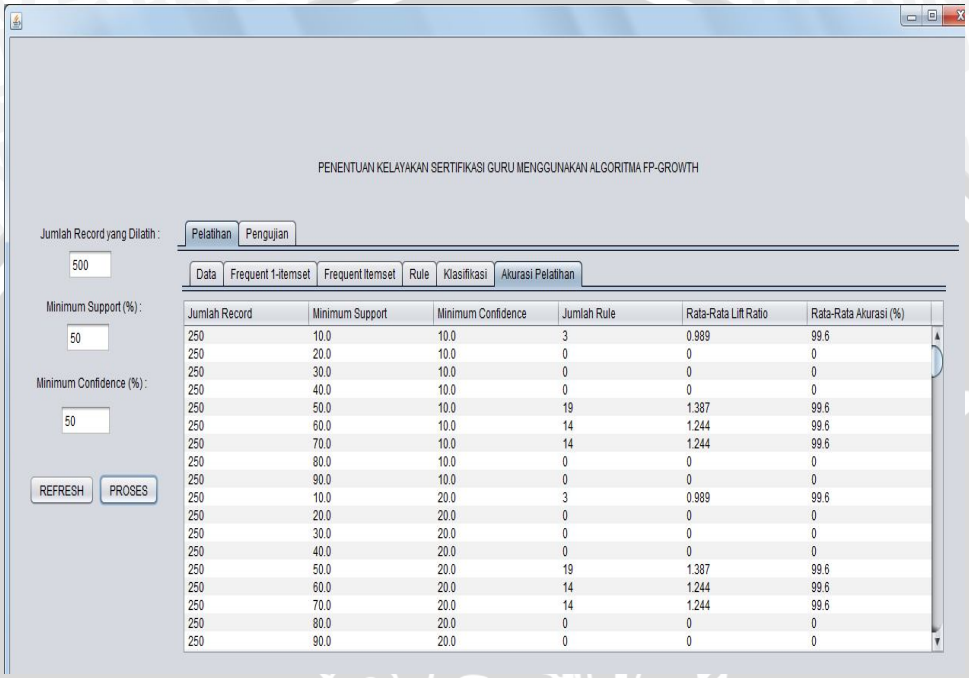


Gambar 4.5. Antarmuka Submenu Klasifikasi

4.3.1.6. Antarmuka Submenu Akurasi Pelatihan

Submenu Akurasi Pelatihan ini difungsikan untuk menampilkan hasil pengujian dari data latih yang telah diproses sebelumnya dan mempermudah proses analisis pengujian. Pada tab ini terdapat kolom jumlah *record* untuk menampilkan variasi jumlah data yang dilatih, kolom *minimum support* untuk menampilkan variasi *minimum support* yang dilatih, kolom *minimum confidence* untuk menampilkan variasi *minimum confidence* yang dilatih, kolom jumlah *rule* untuk menampilkan hasil jumlah *rule* yang didapat dengan variasi jumlah *record*,

minimum support dan *minimum confidence* yang dilatih, kolom rata-rata *lift ratio* untuk menampilkan hasil rata-rata *lift ratio* yang didapat dengan variasi jumlah *record*, *minimum support* dan *minimum confidence* yang dilatih, serta kolom rata-rata akurasi untuk menampilkan hasil rata-rata akurasi *rule* yang didapat dengan variasi jumlah *record, minimum support* dan *minimum confidence* yang dilatih.

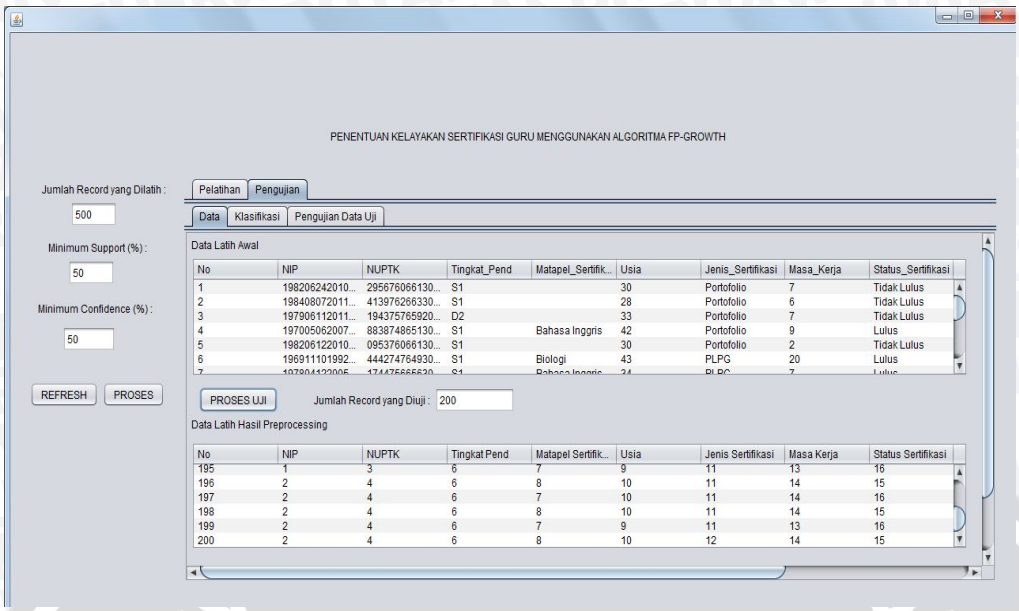


Gambar 4.6. Antarmuka Submenu Akurasi Pelatihan

4.3.2. Antarmuka Menu Pengujian

4.3.2.1. Antarmuka Submenu Data

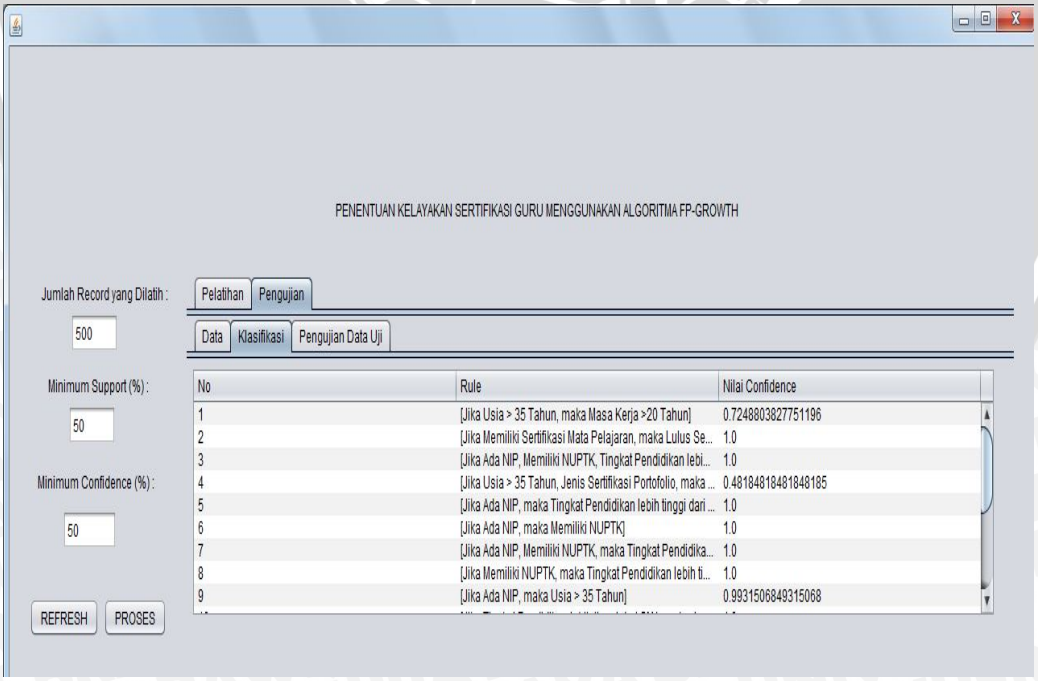
Submenu ini merupakan tampilan data awal dan data hasil *preprocessing* yang sama dengan data awal pada menu pelatihan. Pada tab ini, terdapat *field input* jumlah *record* yang akan diuji.



Gambar 4.7. Antarmuka Submenu Data

4.3.2.2. Antarmuka Submenu Klasifikasi

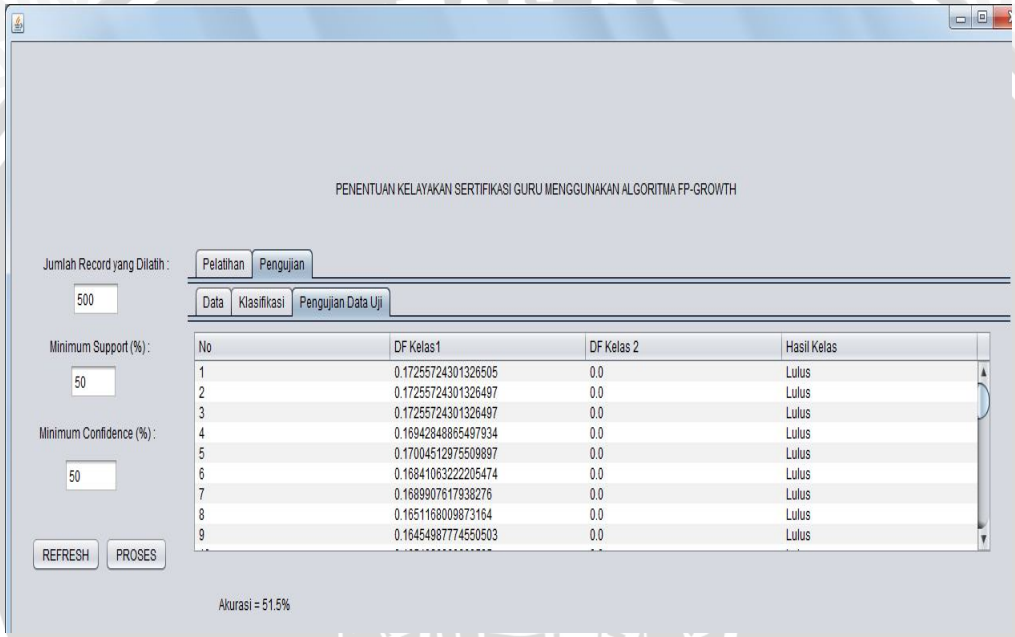
Submenu ini merupakan tampilan dari *rule-rule* yang akan digunakan untuk proses klasifikasi data baru atau data uji. *Rule-rule* tersebut didapatkan dari *rule* yang telah dilatih pada proses pelatihan sebelumnya. Pada tab ini, terdapat kolom *rule* yang menampilkan *rule* yang telah berbentuk kalimat beserta *confidence*-nya.



Gambar 4.8. Antarmuka Submenu Klasifikasi

4.3.2.3. Antarmuka Submenu Pengujian Data Uji

Submenu ini menampilkan kecenderungan kelas untuk data baru atau data uji. Pada tab ini, terdapat kolom DF kelas 1 yang menampilkan nilai DF bagi data yang nilai DF-nya lebih besar terhadap kelas C1, kolom DF kelas 2 yang menampilkan nilai DF bagi data yang nilai DF-nya lebih besar terhadap kelas C2 dan kolom hasil kelas yang menampilkan hasil klasifikasi data yang diuji. Selain itu ditampilkan juga akurasi dari data uji tersebut, dimana akurasi tersebut diukur dari kecocokan data uji yang diproses dengan data aktualnya.



Gambar 4.9. Antarmuka Submenu Pengujian Data Uji