BAB 5 IMPLEMENTASI SISTEM

Bab ini berisi pembahasan hasil implementasi dari perancangan yang sudah dibuat sebelumnya. Pembahasan yang dilakukan meliputi spesifikasi sistem yang digunakan, implementasi program dan antarmuka.

5.1 Spesifikasi Sistem

Pembahasan spesifikasi sistem meliputi perangkat keras dan lunak yang akan digunakan dalam pembuatan aplikasi klasifikasi dokumen tanaman obat dengan metode *Improved k-Nearest Neighbor.*

5.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang akan digunakan dalam penelitian ini terdapat pada Tabel 5.1.

Tabel 0.1 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
Prosesor	Intel(R) Celeron (R) CPU N3050 @1.60GHz
Memori RAM	2GB
Harddisk	500GB

5.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang akan digunakan dalam penelitian ini terdapat pada Tabel 5.2.

Tabel 0.2 Spesifikasi Perangkat Lunak

Nama	Spesifikasi
Sistem operasi	Microsoft Windows 10, 64-bit
Bahasa pemrograman	Java
Aplikasi Pemrograman	Netbeans IDE 8.1 Java Development Kit 1.8.0_101

5.2 Implementasi Program

Implementasi program dilakukan berdasarkan perancangan yang telah dibuat pada bab sebelumnya. Implementasi program pada penelitian ini secara umum mencakup tiga proses yaitu *preprocessing,* pembobotan, dan klasifikasi.

5.2.1 Preprocessing

Implementasi *preprocessing* mencakup empat tahap yaitu *tokenizing, case folding, filtering,* dan *stemming.* Data yang akan diproses berupa file dalam format .txt. Proses *input* data terdapat pada Kode Implementasi 5.1.

```
1  private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
2   JFileChooser J = new JFileChooser();
3   J.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
4     if (J.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
5      filesLatih = J.getSelectedFile();
6      DirektoriDokLat = filesLatih.getAbsolutePath();
7      //menampilkan path dari file
8      DatLatPath.setText(DirektoriDokLat);
9      fileLatih = filesLatih.listFiles();
10     DatLatText = new String[fileLatih.length];
11     for (int i = 0; i < fileLatih.length; i++) {
12     BacaFile bf = new
13     BacaFile(fileLatih[i].getAbsolutePath());
14     DatLatText[i] = bf.getText();
15     //data dimasukkan dalam Array DatLatText
16      }
17     }
18    }
```

Kode Implementasi 0.1 Proses *Input* Data

5.2.1.1 Tokenizing

Tahap pertama dalam *preprocessing* adalah *tokenizing.* Di tahap ini dokumen dipecah menjadi *token* atau *term.* Kode Implementasi 5.2 menunjukkan proses *tokenizing.*

```
1  public ArrayList<String> Tokenizing(String a){
2   String []token = a.split(" ");
3   ArrayList<String> token1 = new ArrayList<String>();
4   token1.addAll(Arrays.asList(token));
5   return token1;
6  }
```

Kode Implementasi 0.2 Proses *Tokenizing*

5.2.1.2 Case Folding

Tahap kedua adalah *case folding,* yaitu proses penghapusan karakter selain huruf dan membuat huruf kapital menjadi kecil seperti yang ditunjukkan pada Kode Implementasi 5.3.

```
1  public String CaseFolding(String a){
2   String text = a.toLowerCase().replaceAll("[^A-Za-z]", " ").trim();
3    text = text.replaceAll("\\s+", " ");
4    return text;
5  }
```

Kode Implementasi 0.3 Proses *Case Folding*

5.2.1.3 Filtering

Proses selanjutnya adalah *filtering* yaitu penghapusan *term* yang kurang penting atau disebut dengan *stopwords.* Daftar *stopwords* diperoleh dari jurnal Fadillah Z. Tala. Kode Implementasi 5.4 menunjukkan proses *filtering.*

```
1  public String  Filtering (String a){
2   ArrayList<String> stopword;
3   String HasilFilter;
4   stopword = new ArrayList<String>();
5   HasilFilter = " ";
6   try{
7    BufferedReader stops = new BufferedReader (new
8    FileReader("filtering.txt"));
9    Scanner check = new Scanner(stops);
```

```
10    String kata;
11    while (check.hasNext())
12    {
13      kata = check.next().toLowerCase().trim();
14      stopword.add(kata);//proses baca file stopword
15    }
16    }catch (Exception e) {
17       System.out.println("File tidak ditemukan");
18    }
19     if(!isStopword( a,stopword)){
20     HasilFilter=a;//proses pengecekan apakah term adalah stopword
21    }
22     return HasilFilter;
23    }
```

Kode Implementasi 0.4 Proses *Filtering*

5.2.1.4 Stemming

Setelah melalui proses *filtering,* selanjutnya dilakukan *stemming* yaitu pengubahan kata menjadi bentuk kata dasar dengan menggunakan algoritma *Porter Stemmer* untuk Bahasa Indonesia oleh Fadillah seperti yang ditunjukkan pada Kode Implementasi 5.5.

```
1   public String Stemming(String s){
2    String stem=s;
3    if (s.length() >= 3) {
4    stem = hapusawalan1(hapusposesive(hapuspartikel(s)));
5    //implementasi stemming porter-fadillah
6     if (stem.equalsIgnoreCase(hapusposesive(hapuspartikel(s)))) {
7      stem = hapusakhiran(hapusawalan2(stem));
8     } else {
9      if (!hapusakhiran(stem).equalsIgnoreCase(stem)) {
10      stem = hapusawalan2(hapusakhiran(stem));
11      }
12     }
13    }
14    return stem;
15  }
```

Kode Implementasi 0.5 Proses *Stemming*

5.2.2 Pembobotan

Setelah melalui *preprocessing,* selanjutnya dilakukan pembobotan dengan menggunakan metode TFIDF. Proses ini meliputi tahap mendapatkan *term* unik, menghitung *term frequency, document frequency, inverse document frequency,* dan menghitung bobot akhir dengan metode TFIDF.

5.2.2.1 Mendapatkan *Term* Unik

Tahap ini dilakukan untuk memperoleh *term* unik dari data latih dan data uji seperti yang ditunjukkan pada Kode Implementasi 5.6.

```
1   public ArrayList<String> BentukKamus(ArrayList<String> a){
2    ArrayList<String> Kamus = new ArrayList<String>();
3    for (int i=0; i<a.size();i++){
4     if (!Kamus.contains(a.get(i))){
5     Kamus.add(a.get(i));  }
6    }return Kamus;
7    }
```

Kode Implementasi 0.6 Proses Mendapatkan *Term* Unik

5.2.2.2 Menghitung *Term Frequency*

Proses ini dilakukan untuk menghitung jumlah frekuensi kemunculan *term* unik pada setiap dokumen latih dan uji. Proses menghitung *term frequency* terdapat pada Kode Implementasi 5.7.

```
1  public ArrayList<Integer> hitungTF(String kata,ArrayList<Datalatih>
2  a) { //hitung dalam dok x ada berapa kata
3   ArrayList<Integer> TF = new ArrayList<Integer>();
4    for (int i = 0; i < a.size(); i++) {
5      int jumdat = 0;
6      for (int j = 0; j < a.get(i).term.size(); j++) {
7       if (kata.equals(a.get(i).term.get(j))) {
8        jumdat++;
9        }
10      }
11      TF.add(jumdat);
12      }
13   return TF;
14 }
```

Kode Implementasi 0.7 Proses Menghitung *Term Frequency*

5.2.2.3 Menghitung *Document Frequency*

Proses selanjutnya adalah menghitung jumlah frekuensi dokumen yang memiliki setidaknya satu *term* unik seperti yang ditunjukkan pada Kode Implementasi 5.8.

```
1  public int hitungDF(ArrayList<Integer> a, ArrayList<String> b) {
2   //hitung dokumen frekuensi
3   int DF = 1;
4   if (a.size() == 1) {
5     DF = 1;
6     } else {
7     for (int i = 0; i < a.size() - 1; i++) { //syarat DF bertambah
8     //bila minimal ada 1 term unik pada dokumen
9      if (a.get(i) != a.get(i + 1) && b.get(i).equals(b.get(i + 1))) {
10     DF++;//bila nomor berbeda dan kategori sama,
11      }else if (a.get(i) == a.get(i + 1) && !b.get(i).equals(b.get(i +
12      1))) {
13      DF++;//bila nomor sama tetapi kategori berbeda
14      }else if (a.get(i) != a.get(i + 1) && !b.get(i).equals(b.get(i +
15      1))) {
16      DF++;// nomor dan kategori berbeda
17     }
18    }
19   }
20   return DF;
21 }
```

Kode Implementasi 0.8 Proses Menghitung *Document Frequency*

5.2.2.4 Menghitung *Inverse Document Frequency*

Setelah diperoleh nilai *document frequency,* proses selanjutnya adalah penghitungan *inverse document frequency.* Proses menghitung *invers document frequency* ditunjukkan pada Kode Implementasi 5.9.

```
1  public double hitungIDF(int jumDok, int DF) {// hitung IDF
2   double IDF;
3   IDF = Math.log10(jumDok / DF);
4   return IDF;
5  }
```

Kode Implementasi 0.9 Proses Menghitung *Inverse Document Frequency*

5.2.2.5 Menghitung TFIDF

Proses pembobotan akhir adalah penghitungan TFIDF yaitu perkalian antara *term frequency* dan *document frequency* seperti ditunjukkan pada Kode Implementasi 5.10.

```
1  public ArrayList<Double> hitungTFIDF(ArrayList<Integer> tf, double
2  idf) { //hitung tfidf
3   ArrayList<Double> tfidf = new ArrayList<Double>();
4   for (int i = 0; i < tf.size(); i++) {
5     tfidf.add(i, tf.get(i) * idf); //perkalian tf dan idf
6   }
7   return tfidf;
8  }
```

Kode Implementasi 0.10 Proses Menghitung TFIDF

5.2.3 Menghitung Cosine Similarity

Cosine Similarity dihitung untuk memperoleh nilai kedekatan antara dokumen uji terhadap dokumen latih. Rumus menghitung *cosine similarity* terdapat pada Persamaan (2.3). Kode Implementasi 5.11 menunjukkan proses menghitung nilai *cosine similarity.*

```
1  Public double hitungCosine(ArrayList<TFIDFlat> latih,
2  ArrayList<TFIDFlat> uji, int datuji, int datlatih) {
3  //method hitung pembilang cosine (total(tfidf lat * uji))
4    ArrayList<Double> cosine = new ArrayList<Double>();
5    double pembilang = 0;
6    for (int i = 0; i < uji.size(); i++) {
7    cosine.add(latih.get(i).tfidf.get(datlatih)
8    *uji.get(i).tfidf.get(datuji));
9    //tfidf data latih dikali tfidf data uji
10   }
11   for (int i = 0; i < cosine.size(); i++) {
12   pembilang = pembilang + cosine.get(i);
13   //hasil perkalian dijumlahkan
14   }
15   return pembilang;
16   }
17
18 //menghitung pembilang dari rumus cosine
19      totcos.add(knn.hitungCosine(matriktfidf, matriktfidfuji, i, j));
20 //perkalian tfidf tiap soal uji dengan seluruh soal latih kemudian
21 //semua nilai ditotal
22      }
23      cos.cosine = totcos;
24      totaltflatxtfuji.add(cos);
25      }
26
27 //menghitung penyebut dari rumus cosine
28      totaltfidflatih = knn.hitungTotalTFIDF(jumlahDokumen,
29      matriktfidf);//hitung total tfidf tiap dokumen latih
30      totaltfidfuji = knn.hitungTotalTFIDF(jumlahDokumenUji,
```

```
31      matriktfidfuji);//hitung total tfidf tiap dokumen uji
32  //menghitung akar kuadrat total tfidf data latih sebagai penyebut
33      double akarkuadrat = Math.sqrt((totaltfidflatih.get(i) *
34      totaltfidflatih.get(i)));
35      tottfidflat2.add(akarkuadrat);
36
37  //menghitung akar kuadrat total tfidf data uji sebagai penyebut
38      double akarkuadrat = Math.sqrt((totaltfidfuji.get(i) *
39      totaltfidfuji.get(i)));
40      tottfidfuji2.add(akarkuadrat);
41
42  //menghitung nilai cosine
43      cos.cosine2[j] = (double) totaltflatxtfuji.get(i).cosine.get(j) /
44      (double) (tottfidflat2.get(j) * tottfidfuji2.get(i));
45      cossinesim.add(cos);
```

Kode Implementasi 0.11 Proses Menghitung *Cosine Similarity*

5.2.4 Mengurutkan Nilai *Cosine Similarity*

Setelah nilai *cosine similarity* diperoleh, langkah selanjutnya adalah mengurutkan nilai tersebut dengan menggunakan metode *bubble sort*. Pada proses ini kategori dari dokumen latih juga ikut diurutkan berdasarkan nilai *cosine similarity* yang terbesar hingga terkecil seperti ditunjukkan pada Kode Implementasi 5.12.

```
1   for (int i = 0; i < cossinesim.get(k).cosine2.length; i++) {
2   for (int j = i + 1; j < cossinesim.get(k).cosine2.length; j++) {
3    if (cossinesim.get(k).cosine2[i] < cossinesim.get(k).cosine2[j]) {
4     double temp;
5     String tem;
6       temp = cossinesim.get(k).cosine2[i];
7       tem = cossinesim.get(k).kategori2[i];
8       cossinesim.get(k).cosine2[i] = cossinesim.get(k).cosine2[j];
9       cossinesim.get(k).kategori2[i] = cossinesim.get(k).kategori2[j];
10      cossinesim.get(k).cosine2[j] = temp;
11      cossinesim.get(k).kategori2[j] = tem;
12   }
13  }
14  }
```

Kode Implementasi 0.12 Proses Mengurutkan Nilai *Cosine Similarity*

5.2.5 Klasifikasi Improved k-Nearest Neighbor

Proses implementasi terkahir adalah penerapan metode klasifikasi *Improved k-Nearest Neighbor.* Proses ini meliputi dua tahap yaitu penghitungan nilai *k* baru dari *k* awal yang ditentukan dan penghitungan peluang dokumen uji terhadap kategori tertentu.

5.2.5.1 Menentukan Nilai *k* Baru

Nilai *k* adalah banyaknya jumlah tetangga atau nilai *cosine similarity* yang sudah diurutkan dan akan digunakan pada proses penghitungan peluang. Nilai *k* yang digunakan dalam metode *Improved k-Nearest Neighbor* dihitung berdasarkan Persamaan (2.4). Proses menentukan nilai *k* baru ditunjukkan pada Kode Implementasi 5.13.

```
1  public int kbaruapiaceae(int k, int max, ArrayList<Datalatih>
2  termlatih) {
3   int ttlapiaceae = 0;
4   int kbaruapiaceae;
5   int kakhirapiaceae;
6    for (int i = 0; i < termlatih.size(); i++) {
7     if (termlatih.get(i).kategori.get(0).equals("Apiaceae")) {
8       ttlapiaceae++;
9     }
10   }
11 //rumus menentukan k baru = k inputan kali jumlah data kategori yang
12 //akan dihitung bagi jumlah data terbanyak dari seluruh kategori
13   kbaruapiaceae = (k * ttlapiaceae) / max;
14     if (kbaruapiaceae < 1) {
15     kakhirapiaceae = 1; //bila k < 1 maka dianggap 1
16     } else {
17     kakhirapiaceae = kbaruapiaceae;
18     }
19     return kakhirapiaceae;
20    }
```

Kode Implementasi 0.13 Proses Menentukan Nilai *k* baru

Kode Implementasi 5.13 adalah penghitungan nilai *k* baru untuk kategori *Apiaceae.* Kode implementasi tersebut juga diterapkan pada 12 kategori tumbuhan obat lainnya.

5.2.5.2 Menghitung Peluang

Tahap ini dilakukan untuk mengetahui peluang data uji termasuk ke dalam kategori tertentu. Rumus penghitungan peluang terdapat pada Persamaan (2.5). Proses menghitung peluang ditunjukkan pada Kode Implementasi 5.14.

```
1  public double hitungPeluangIKNN(int kbaru, int noDataUji,
2  ArrayList<cossine> cosin, String kategoriygdicari) {
3   double peluang = 0;
4   double pembilang = 0;
5   double penyebut = 0;
6
7  //menghitung pembilang dari rumus peluang
8     for (int i = 0; i < kbaru; i++) {
9  //jika kategori data uji yang dicari sama dengan cosine similaritynya
10 //maka cosine similarity kali 1
11    if (cosin.get(noDataUji).kategori2[i].equals(kategoriygdicari)) {
12    pembilang = pembilang + (cosin.get(noDataUji).cosine2[i] * 1);
13 //jika tidak sama cosine similarity kali 0
14    }else if
15    (!cosin.get(noDataUji).kategori2[i].equals(kategoriygdicari)) {
16    pembilang = pembilang + 0;
17    }
18 }
19
20 //menghitung penyebut dari rumus peluang
21    for (int i = 0; i < kbaru; i++) {
22    penyebut = penyebut + cosin.get(noDataUji).cosine2[i];
23
24 //menghitung peluang, pembilang bagi penyebut
25    peluang = pembilang / penyebut;
26    return peluang;
27    }
```

Kode Implementasi 5.14 Proses Menghitung Peluang

5.2.5.3 Menentukan Kategori Data Uji

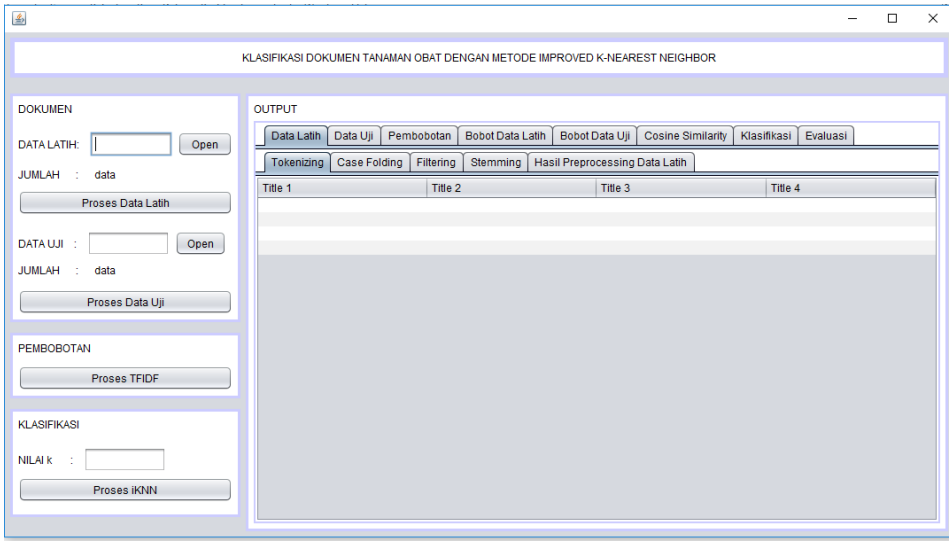
Tahap penentuan kategori data uji ditentukan dengan mengurutkan nilai peluang menggunakan metode *bubble sort* dan mengambil nilai peluang terbesar seperti yang ditunjukkan pada Kode Implementasi 5.15.

```
for (int j = 0; j < peluang.length; j++) {
    for (int k = j + 1; k < peluang.length; k++) {
//bubble sort
    if (peluang[j] < peluang[k]) {
     double temp;
     String tem;

     temp = peluang[j];
     tem = kategori[j];

     peluang[j] = peluang[k];
     kategori[j] = kategori[k];

     peluang[k] = temp;
     kategori[k] = tem;
    }
    }
}
     imknn.noData = i + 1;
     imknn.peluang = peluang;
     imknn.kategori = kategori;
     impKNN.add(imknn);
//kategori data uji
     impKNN.get(i).kategori[0]
```

Kode Implementasi 5.15 Proses Menentukan Kategori Data Uji

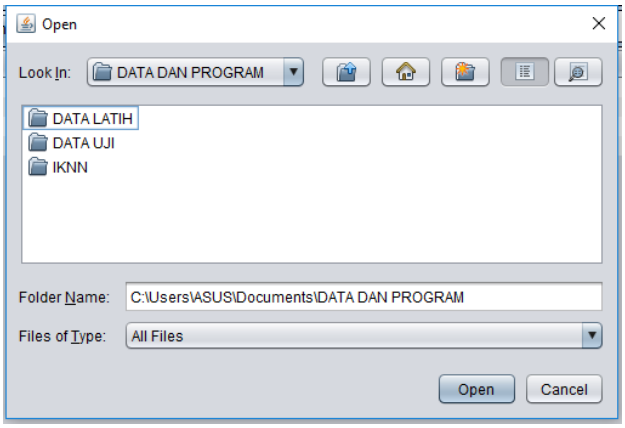
5.3 Implementasi *User Interface*

Tahap ini menunjukkan implementasi dari *user interface* atau antarmuka sistem klasifikasi dokumen tanaman obat menggunakan metode *Improved k-Nearest Neighbor.* Proses diawali dengan melakukan *input* data berupa dokumen latih, uji, dan nilai *k.* Memasukkan data dokumen latih dan uji dilakukan dengan menekan tombol *open.*



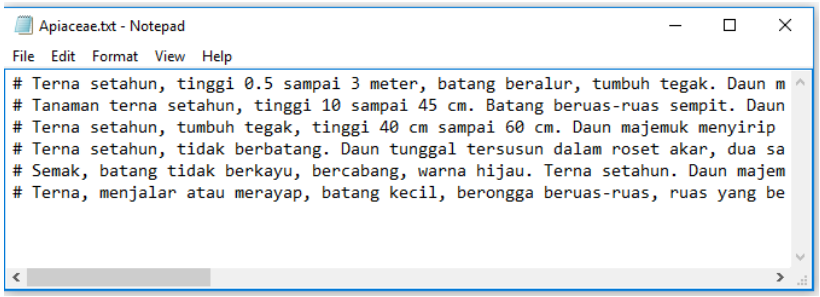
Gambar 0.1 Implementasi Antarmuka Halaman Awal

Gambar 5.2 menunjukkan implementasi antarmuka untuk memilih *file* data latih dan data uji dengan menekan tombol *open.*



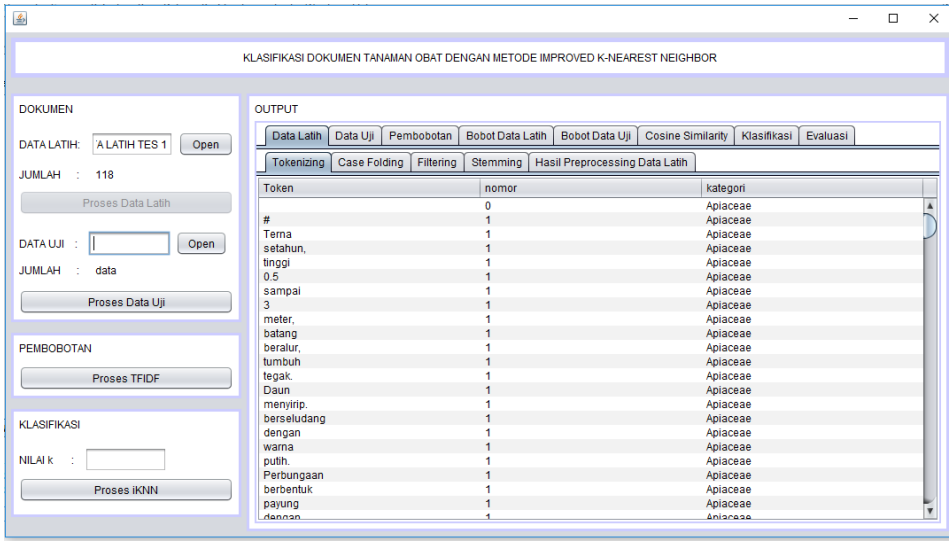
Gambar 0.2 Implementasi Antarmuka *Open* Data

Gambar 5.3 menunjukkan *file* data latih dan data uji yang dibaca oleh sistem yaitu dalam format *file .txt.*



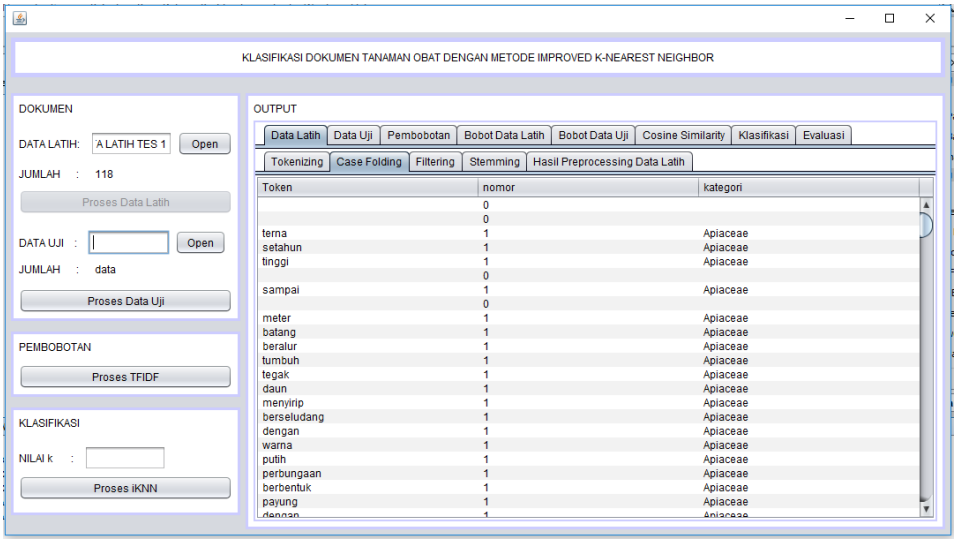
Gambar 0.3 *File* Data Latih dan Uji

Tahap selanjutnya adalah pemrosesan data latih dan uji. *Preprocessing* data latih diawali dengan *tokenizing* yaitu pemecahan kalimat dalam dokumen menjadi *token* seperti pada Gambar 5.4.



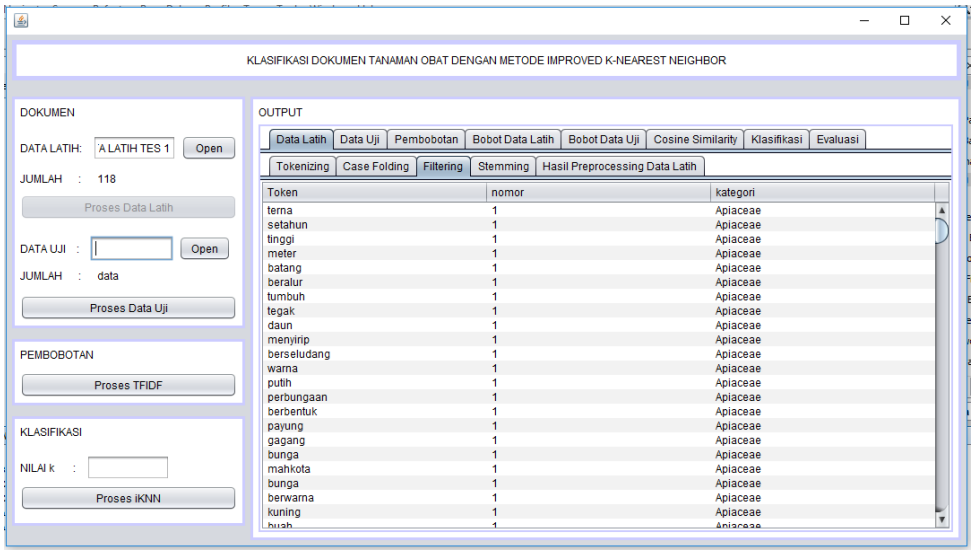
Gambar 0.4 Implementasi Antarmuka Proses *Tokenizing*

Selanjutnya adalah tampilan proses *case folding* yaitu mengubah huruf kapital menjadi huruf kecil dan menghilangkan tanda baca atau *delimiter.* Implementasi antarmuka proses *case folding* ditunjukkan pada Gambar 5.5.



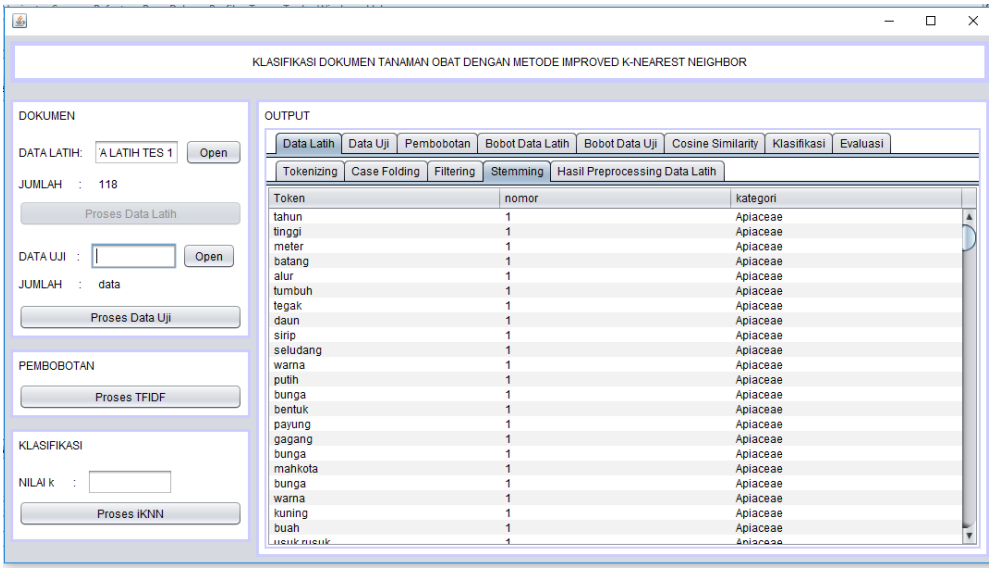
Gambar 0.5 Implementasi Antarmuka Proses *Case Folding*

Setelah *case folding,* proses selanjutnya adalah *filtering* yaitu menghapus *term* yang termasuk dalam *stopwords.* Implementasi antarmuka proses *filtering* ditunjukkan pada Gambar 5.6.



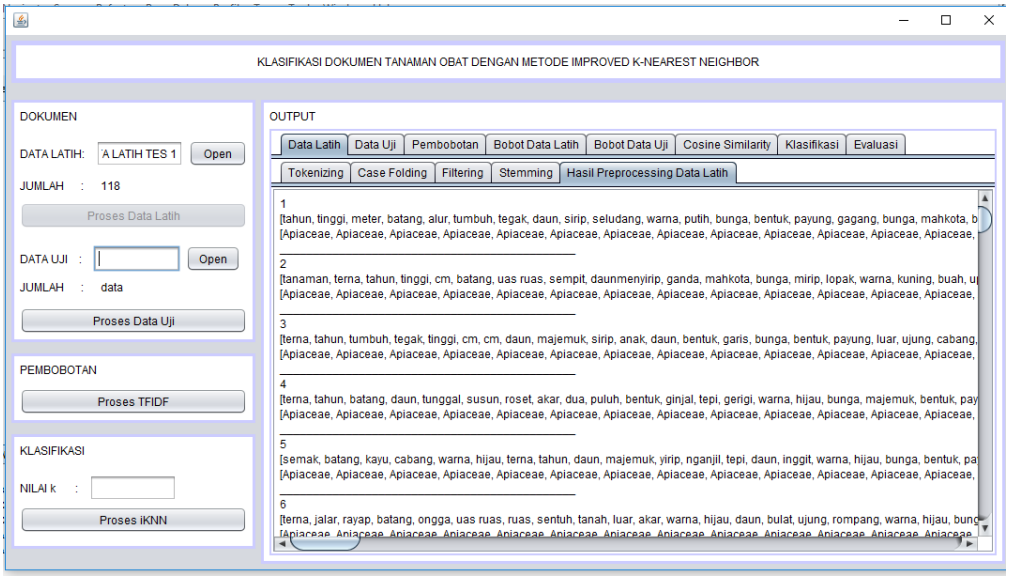
Gambar 0.6 Implementasi Antarmuka Proses *Filtering*

Berikutnya adalah implementasi antarmuka proses *stemming* yang berisi kata dasar dari *term* hasil pemrosesan *filtering* seperti ditunjukkan pada Gambar 5.7.



Gambar 0.7 Implementasi Antarmuka Proses *Stemming*

Data yang sudah melalui *preprocessing* selanjutnya ditampilkan pada tab Data Latih. Implementasi antarmuka hasil *preprocessing* data latih ditunjukkan pada Gambar 5.8.

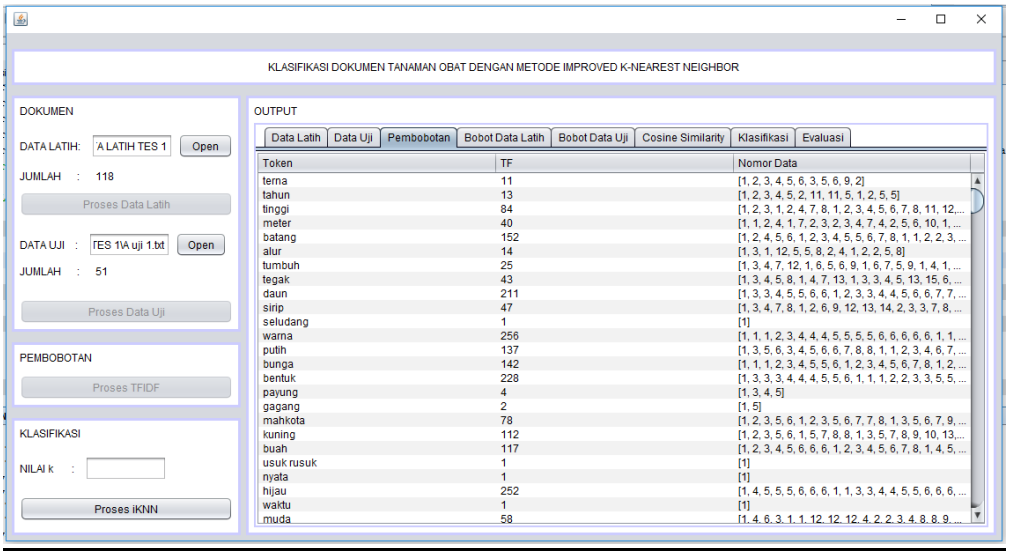


Gambar 0.8 Implementasi Antarmuka Hasil *Preprocessing* Data Latih

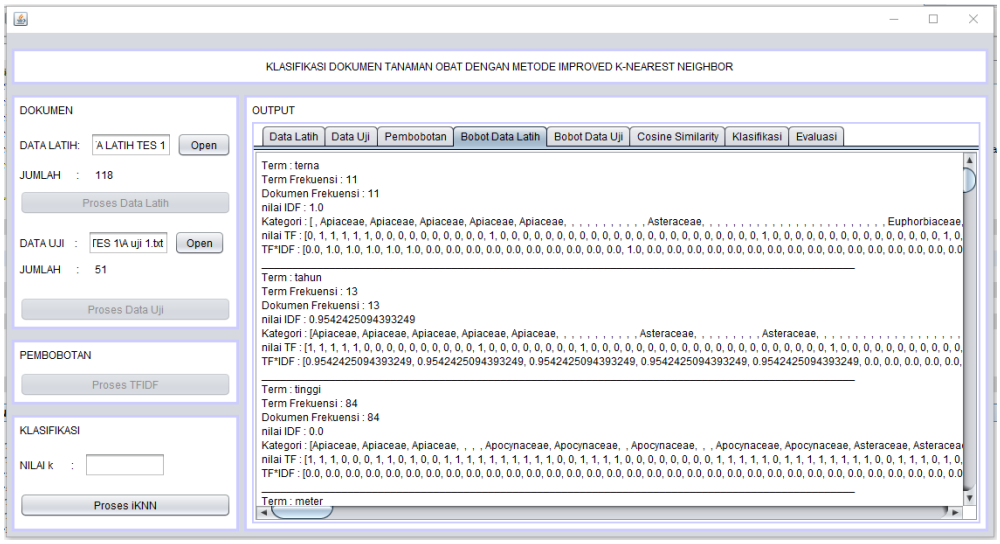
Tampilan proses *preprocessing* yang mencakup tahap *tokenizing, case folding, filtering, stemming,* dan hasil *preprocessing* juga berlaku sama untuk pemrosesan data uji seperti ditunjukkan pada Gambar 5.9.

Gambar 0.9 Implementasi Antarmuka Hasil *Preprocessing* Data Uji

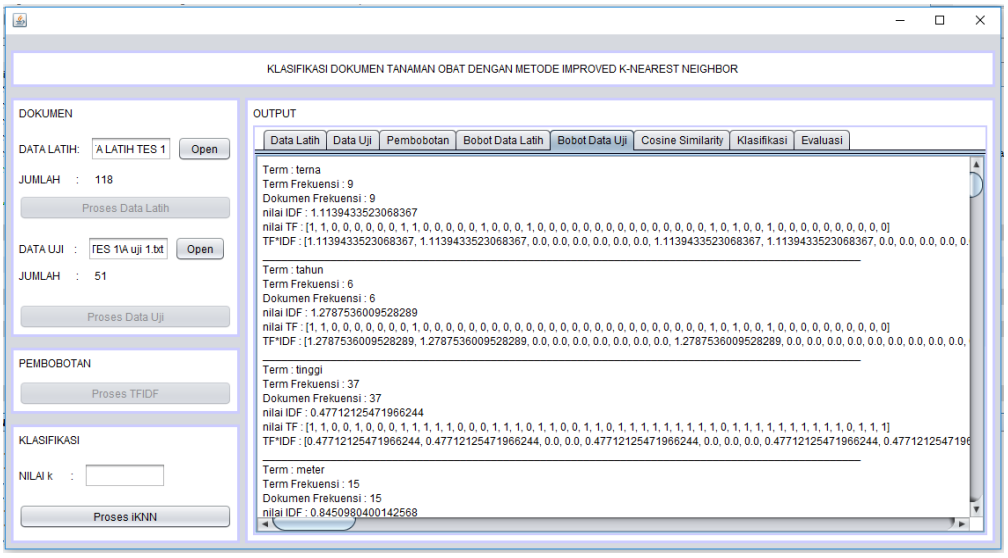
Setelah melalui *preprocessing,* tahap selanjutnya adalah pembobotan data latih dan data uji. Hasil dari proses ini adalah vektor bobot setiap *term* pada dokumen. Implementasi antarmuka proses pembobotan ditunjukkan pada Gambar 5.10, Gambar 5.11, dan Gambar 5.12.



Gambar 0.10 Implementasi Antarmuka *Term Frequency* Data Latih

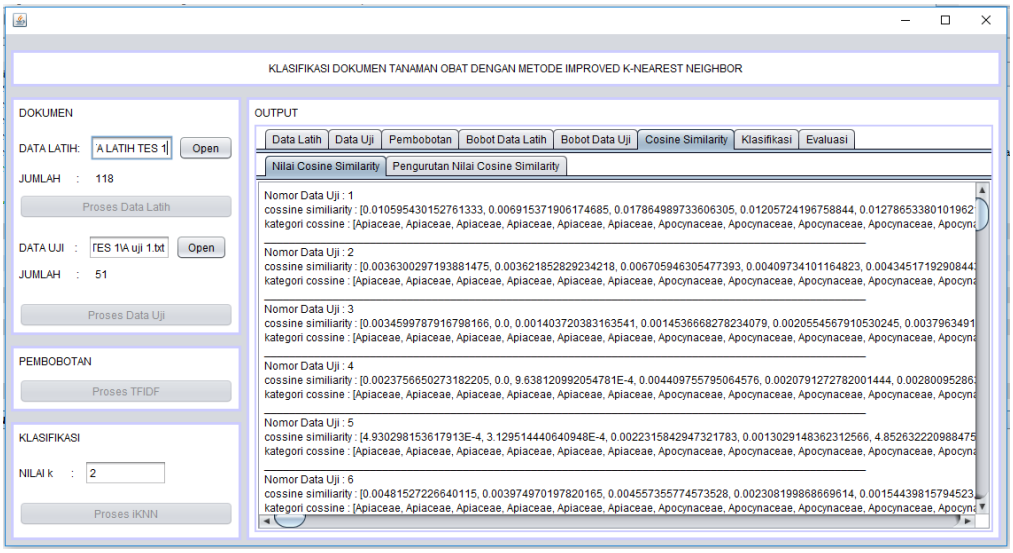


Gambar 0.11 Implementasi Antarmuka Hasil Pembobotan Data Latih



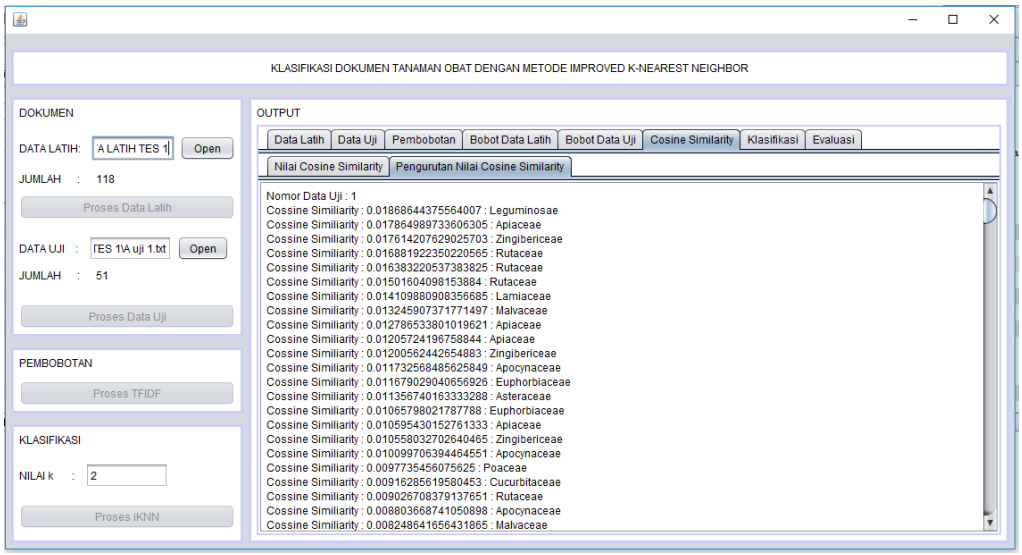
Gambar 0.12 Implementasi Antarmuka Hasil Pembobotan Data Uji

Tahap selanjutnya adalah proses klasifikasi yang terdiri dari *input* nilai *k,* penghitungan dan pengurutan nilai *cosine similarity* serta klasifikasi dokumen uji menggunakan metode *Improved k-Nearest Neighbor.* Implementasi antarmuka penghitungan nilai *cosine similarity* ditunjukkan pada Gambar 5.13.



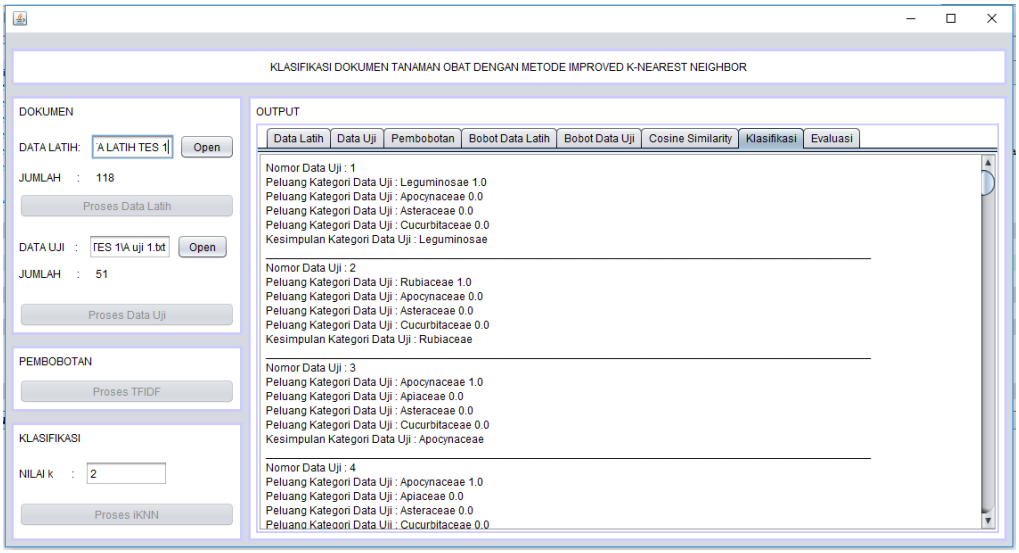
Gambar 0.13 Implementasi Antarmuka Hasil Penghitungan *Cosine Similarity*

Setelah dilakukan proses penghitungan nilai *cosine similarity,* selanjutnya dilakukan proses pengurutan nilai *cosine similarity* diikuti dengan kategorinya seperti ditunjukkan pada Gambar 5.14.



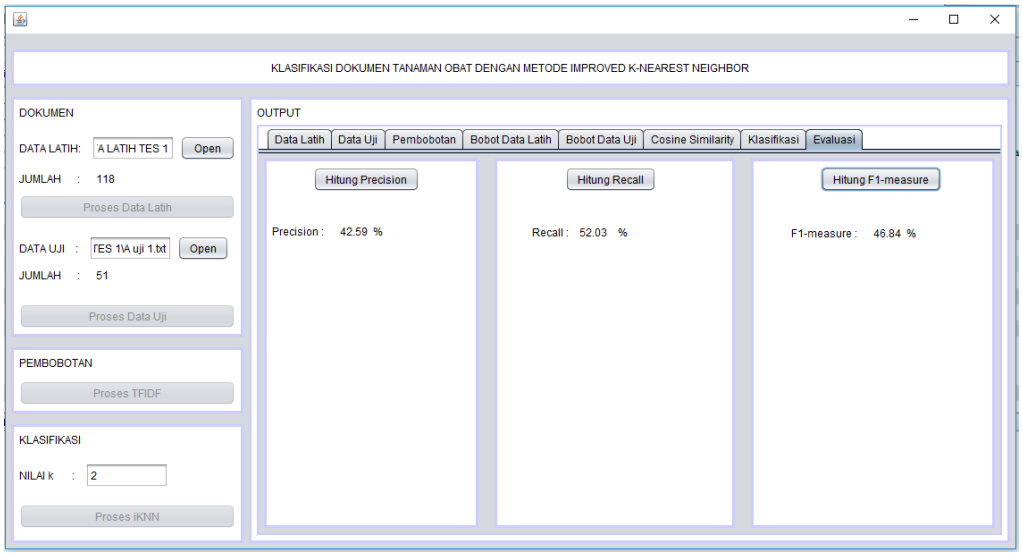
Gambar 0.14 Implementasi Antarmuka Hasil Pengurutan *Cosine Similarity*

Tahap klasifikasi dengan metode *Improved k-Nearest Neighbor* dilakukan setelah menghitung dan mengurutkan nilai *cosine similarity.* Pada tahap ini diperoleh kesimpulan dari kategori data uji. Implementasi antarmuka proses klasifikasi ditunjukkan pada Gambar 5.15.



Gambar 0.15 Implementasi Antarmuka Hasil Klasifikasi

Tahap akhir adalah proses evaluasi dengan menghitung akurasi berdasarkan nilai *precision, recall,* dan *f1-measure.* Hasil evaluasi ditunjukkan pada Gambar 5.16.



Gambar 0.16 Implementasi Antarmuka Hasil Evaluasi