BAB 5

IMPLEMENTASI

Bab ini menjelaskan implementasi yang dibangun berdasarkan metodologi dan perancangan yang telah dijelaskan pada bab sebelumnya.

5.1 Batasan Implementasi

Batasan implementasi merupakan sub bab yang menjelaskan batasan proses yang dapat dilakukan oleh sistem berdasarkan perancangan yang telah diuraikan dari bab sebelumnya. Berikut merupakan batasan implementasi sistem pada penelitian ini sebagai berikut.

1. Penerapan Klasifikasi *Tweets* Pada Berita *Twitter* Menggunakan Metode *K-Nearest Neighbor* Dan *Query Expansion* Berbasis *Distributional Semantic* dirancang dan dibangun menggunakan aplikasi *desktop* bahasa Java.
2. Metode yang digunakan untuk penyelesaian masalah adalah K-Nearest Neighbor dan *Query Expansion*.
3. Data uji dan data latih yang digunakan merupakan berasal dari *tweets* akun kompas dan detik.
4. Hasil keluaran aplikasi berupa klasifikasi kategori otomotif, teknologi, makanan, olahraga, travel, hiburan, kesehatan, dan ekonomi.

5.2 Implementasi Algoritme

Pada sub-bab Implementasi algoritme ini akan menjelaskan klasifikasi *tweets* yang terdiri dari beberapa algoritme utama yaitu, *text preprocessing, query expansion* dan klasifikasi teks. Berikut merupakan tahapan-tahapan tersebut.

5.2.1 Implementasi Algoritme *Preprocessing Text*

Algoritme *text preprocessing* merupakan tahapan paling awal yang terjadi pada sistem sebelum melangkah ke tahap berikutnya seperti *query expansion* dan klasifikasi. Tahapan tersebut terdapat 4 proses yang terdiri sebagai berikut.

1. *Preprocessing Case Folding*

   Proses pada tahap ini adalah mengubah berbagai huruf yang berbentuk *Uppercase* atau huruf besar menjadi huruf kecil. Proses tersebut dapat dilihat pada Kode Program 5.1 beserta dengan penjelasannya.

   Baris 1-3 : proses untuk menghilangkan tanda baca dan angka

   Baris 4 : proses untuk mengubah kata menjadi *lowercase*

```
1  for (int i = 0; i < words.length; i++) {
2   words[i] = words[i].replaceAll("\\W", "");//hilangkan simbol
3  words[i] = words[i].replaceAll("[0-9]", "");
4  words[i] = words[i].toLowerCase(); //mengubah lowercase
```

Kode Program 5.1 Implementasi *Case Folding*

2. *Preprocessing Tokenizing*

Proses ini melakukan pemecahan kalimat menjadi kata per kata dan menghilangkan beberapa simbol maupun karakter yang dianggap sistem sebagai tanda baca. Proses tersebut dapat dilihat pada Kode Program 5.2 beserta penjelasannya.

Baris 1 : Memecah kalimat menjadi kata per kata

```
1  String[] words = id_dok.split("\\s+"); //mecah
```

Kode Program 5.2 Implementasi *Tokenizing*

3. *Preprocessing Filtering*

Tahap ini memproses pengambilan kata-kata yang dari hasil *tokenizing* yang dianggap penting. Namun cara tersebut menggunakan algoritme *stoplist* (membuang kata yang tidak perlu digunakan atau menghapus kata tidak penting) dan menyimpan kata-kata yang telah dianggap penting (*wordlist*) (Insan, 2013). Proses tersebut dapat dilihat pada Kode Program 5.3 beserta penjelasannya.

Baris 1-19 : Digunakan untuk memanggil file *stoplist* berisi teks sebagai acuan untuk menghilangkan kata tidak penting.

Baris 20-28: Kode program untuk perulangan jika terdapat kata yang tidak penting pada *term* unik akan dihilangkan dan kata selain itu akan disimpan untuk proses berikutnya.

```
1  String[] stoplist;
2   BufferedReader br2 = new BufferedReader(new
3  FileReader("C:/Users/alfiranita/Documents/NetBeansProjects/
4  skripsinih/src/Model/stoplist.txt"));
5      try {StringBuilder sb2 = new StringBuilder();
6  String line = br2.readLine();
7      while (line != null) {
8              sb2.append(line);
9              sb2.append(System.lineSeparator());
10             line = br2.readLine();
11             dataStoplist.add(line);
12       }
13 stoplist = sb2.toString().split("[\n]");
14
15    } finally {
16
17    br.close();
18        }
19 boolean kt = this.stopList(words[i]);
```

```
20  if (!kt) {
21        int ketemu = 0;
22        for (int j = 0; j < termUnik.size(); j++) {
23          if (words[i].matches(termUnik.get(j))) {
24           ketemu++;
25           }
26     }
27        if (ketemu == 0 && !words[i].matches("")) {
28           termUnik.add(words[i]); } }
29
30                              num++; }
```

Kode Program 5.3 Implementasi *Filtering*

4. Preprocessing Stemming

Pada tahap ini dilakukan merubah berbagai kata-kata yang terdapat pada dokumen menjadi kata-kata yang berupa akarnya atau kata aslinya *(base word*) dengan aturan tertentu dari sistem. Proses tersebut dapat dilihat pada Kode Program 5.4 beserta penjelasannya.

Baris 1-5 : membaca file yang berisi teks *base word* sebagai kata asli yang tidak dapat di hilangkan setiap bagian kata maupun imbuhannya.

Baris 6-21: Kode program untuk memulai dan memanggil fungsi *stemming,* dengan mengecek kata atau men-*scan* setiap baris kalimat.

```
1   dictionary = new HashSet<String>();
2   InputStream in =
3   Lemmatizer.class.getResourceAsStream("/root-words.txt");
4   BufferedReader br = new BufferedReader(new
5   InputStreamReader(in));
6   try {
7      StringBuilder sb = new StringBuilder();
8      String line = br.readLine();
9      while (line != null) {
10             sb.append(line);
11             sb.append(System.lineSeparator());
12             dictionary.add(line);
13             line = br.readLine();
14        }
15  this.kamus = sb.toString().split("\\r?\\n");
16  this.preprocessing.setKamus(kamus);
17  this.preprocessing.setDictionary(dictionary);
18   } finally {
19             br.close();
20             }
21  Lemmatizer lemmatizer = new DefaultLemmatizer(dictionary);
```

Kode Program 5.4 Implementasi *Stemming*

5.2.2 *Query Expansion*

Proses mengekspansi kata baru atau *query expansion* memiliki beberapa tahapan yang dilakukan. Pertama, yaitu memproses data yang berbentuk kamus berita untuk dihitung kedekatan kata dengan setiap kata pada data uji dengan menghasilkan nilai terpendek. Kedua, memproses penggabungan kata yang telah terpilih oleh sistem memiliki jarak terdekat yang kemudian disisipkan di antara kata yang terdekat dalam satu kalimat data uji tersebut.

1. *Distributional Semantic*

   Algoritme yang digunakan untuk mencari kedekatan kata pada dokumen kamus berita adalah dengan teknik perhitungan *Euclidean Distance*. Teknik tersebut memproses antar kata yang dihitung kedekatannya. Proses perhitungan kedekatan kata tersebut dapat dilihat pada kode program 5.5 beserta dengan penjelasannya.

   Baris 1-25 : Digunakan untuk menemukan kata pada data uji yang ada pada dokumen berita untuk dihutung kedekatannya.

   Baris 27-54 : Kode program untuk menghitung kedekatan kata pada data uji dengan dokumen berita, lalu dihitung kedekatan kata tersebut dengan setiap dokumen berita yang sudah menjadi *term* unik tersebut menggunakan rumus *Euclidean distance.*

```
1  private void hitungKedekatan() {
2   DefaultTableModel tabelgProc = new DefaultTableModel();
3   dataujiqepre.setModel(tabelgProc);
4
5   tabelgProc.addColumn("Kategori");
6
7  System.out.println("====================================");
8     ArrayList<String> katakanlah = new ArrayList<String>();
9    for (int i = 0; i < dokumen_uji.size(); i++) {
10       System.out.println(dokumen_uji.get(i));
11       String[] parts = dokumen_uji.get(i).split(" ");
12       String kata_baru = "";
13
14      for (int j = 0; j < parts.length; j++) {
15
16     preprocessing.setKata(parts[j]);
17     preprocessing.process();
18     String p = preprocessing.getHasil();
19     ArrayList<Integer> ketemu = new ArrayList<Integer>();
20            boolean status_ketemu = false;
21            boolean stop = this.stopList(p);
22
23            if (!stop) {
24                     kata_baru = kata_baru + " " + p;
25            }
26
27            int index_ketemu = 0;
28            for (int k = 0; k < termUnik.size(); k++) {
29                 if (p.matches(termUnik.get(k))) {
```

```
30                      status_ketemu = true;
31                      index_ketemu = k;
32
33                for (int l = 0; l < termUnik.size(); l++)
34 {     ketemu.add(frekuensi.get(k).get(l));
35                         }
36                     }
37                 }
38            if (status_ketemu) {
39               ArrayList<Double> subTotals = new
40 ArrayList<Double>();
41               for (int k = 0; k < termUnik.size(); k++) {
42                if (k == index_ketemu) {
43                            subTotals.add(999999999999.0);
44 }               else {
45                 double total = 0;
46                for (int l = 0; l < termUnik.size(); l++)
47  {
48 double pengurangan=ketemu.get(l) -
49 frekuensi.get(k).get(l);
50 double pangkat = Math.pow(pengurangan, 2);
51               total = total + pangkat;
52                            }
53               double akar = Math.sqrt(total);
54                            subTotals.add(akar);
55                         }
56                      }
57
```

Kode Program 5.5 Implementasi *Distributional Semantic*

2. Proses Ekspansi Kata

Proses pada algoritme ini menggunakan *threshold* yang selanjutnya menentukan kata yang terpilih berdasarkan *threshold* tersebut. Kata yang memiliki kedekatan dengan arti kata lain memiliki nilai di bawah *threshold* yang dianggap pantas untuk diekspan ke dalam dokumen uji. Proses tersebut dapat dlihat pada Kode Program 5.6 beserta dengan penjelasannya.

Baris 1-16 : Digunakan untuk memberikan batasan perolehan nilai tingkat kedekatan yang sudah dihitung, dan diambil untuk ditambahkan sebagai kata baru untuk diekspansi ke data uji.

Baris 27-38 : Kode program untuk menambahkan kata yang diperoleh dari perhitungan sebelumnya sebagai kata baru yang akan diekspansi ke dalam data uji dan ditampilkan ke tabel hasil.

```
1  ArrayList<Integer> idx = new ArrayList<Integer>();
2  String[] kata;
3
4
5  double treshold = 1.5;
6  int x = 0;
7  for (int k = 0; k < subTotals.size(); k++) {
8        if (subTotals.get(k) < treshold){
```

```
 9
10                  idx.add(k);
11                              }
12                          }
13
14      for (int k = 0; k < idx.size(); k++) {
15
16      kata_baru = kata_baru + " "+ termUnik.get(idx.get(k));
17      System.out.println("k = "+termUnik.get(idx.get(k)));
18
19                          }
20
21
22                      }
23
24
25                  }
26
27      ArrayList<String> data_term = new ArrayList<String>();
28      String[] pecah_kata = kata_baru.split(" ");
29      for (int j = 0; j < pecah_kata.length; j++) {
30                      if (!pecah_kata[j].matches("")) {
31                          data_term.add(pecah_kata[j]);
32                      }
33
34                  }
35                  this.data_term_pre.add(data_term);
36                  System.out.println(kata_baru);
37                  String[] tampil = {kata_baru};
38                  tabelgProc.addRow(tampil);
39              }
40          }
```

Kode Program 5.6 Implementasi Ekspansi kata

5.2.3 Klasifikasi Teks

Setelah melewati proses *preprocessing* teks dan *query expansion*, berikutnya adalah proses klasifikasi teks. Klasifikasi teks memiliki beberapa tahapan, yang pertama adalah menentukan setiap *term* yang digunakan dari setiap dokumen yang ada, menghitung pembobotan, dan perhitungan jarak antar dokumen untuk menentukan kategori terdekat.

1. Menentukan *Term*

   Proses ini menentukan semua *term* dari setiap dokumen yang akan digunakan klasifikasi. Algoritme tersebut dapat dilihat pada kode program 5.7.

   Baris 1-17 : Kode program untuk memetakan *term* data uji yang sudah diekspansi dengan data latih

   Baris 18-33 : Kode program untuk mencari frekuensi *term* yang dimiliki setiap dokumen.

```
1  public void knn() {
2  System.out.println(data_term_pre.size());
3  System.out.println("ooooooooooooooooooooooooooooooooo");
4  System.out.println(data_term_pre.toString());
5  List<String> per_kata = new ArrayList<String>();
6     for (int i = 0; i < this.data_term_pre.size(); i++) {
7
8      for (int j = 0; j < data_term_pre.get(i).size(); j++)
9  {
10              per_kata.add(data_term_pre.get(i).get(j));
11              }
12
13          }
14          List<Tf_Idf> tf_Idfs = new ArrayList<Tf_Idf>();
15
16      for (int i = 0; i < per_kata.size(); i++) {
17        List<Integer> per_kata2 = new ArrayList<Integer>();
18            for (int j = 0; j < data_term_pre.size(); j++) {
19
20             int ketemu = 0;
21            for (int k = 0; k < data_term_pre.get(j).size();
22 k++) {
23            if
24 (per_kata.get(i).matches(data_term_pre.get(j).get(k))) {
25                          ketemu = 1;
26                      }
27                  }
28                  per_kata2.add(ketemu);
29              }
30              Tf_Idf tf_Idf = new Tf_Idf(per_kata.get(i));
31              tf_Idfs.add(tf_Idf);
32
33              this.frekuensi_data_latih.add(per_kata2);
34
35          }
```

Kode Program 5.7 Implementasi Penentuan *Term* Setiap Dokumen

2. Pembobotan

Pada tahap ini dilakukan pembobotan setiap kata pada antar dokumen dengan menggunakan teknik *TF-Idf*. Algoritme tersebut dapat dilihat pada kode program 5.8.

Baris 1-14 : Digunakan untuk menghasilkan jumlah *df* yang terhitung

Baris 15-23 : Kode program untuk menghitung pembobotan dengan menggunakan *Tf-Idf*

```
1  for (int i = 0; i < frekuensi_data_latih.size(); i++) {
2              //System.out.println("bobot : " +
3  frekuensi_data_latih.get(i).toString());
4              int df = 0;
5              for (int j = 0; j <
6  frekuensi_data_latih.get(i).size(); j++) {
7                  if (frekuensi_data_latih.get(i).get(j) == 1) {
```

```
8                       df++;
9                   }
10              }
11               System.out.println("jml tf = "+
12  frekuensi_data_latih.get(i).size());
13              System.out.println("df = "+ df);
14              double log = 0.0;
15              if (df > 0) {
16                  log =
171 Math.log(frekuensi_data_latih.get(i).size() / df);
8               }
19              tf_Idfs.get(i).setDf(df);
20
21  tf_Idfs.get(i).setBobot(frekuensi_data_latih.get(i));
22              tf_Idfs.get(i).setLog(log);
23
24
25
```

Kode Program 5.8 Implementasi Pembobotan TF-Idf

3. Permetaan Bobot dan Menampilan hasil *cosine*

Pemetaan bobot yang dilakukan berguna untuk menempatkan nilai hasil *tf-idf* atau bobot awal ke dalam setiap *term* yang memiliki nilai hasil *tf-idf* tersebut, yang selanjutnya akan diproses pembobotan akhir dengan normalisasi. Proses tersebut dapat dilihat pada Kode Program 5.9 beserta penjelasannya.

Baris 1-44 : Digunakan untuk memetakan setiap nilai *term* yang berhasil dihitung dengan menyesuaikan setiap kolom dokumen yand dimiliki

Baris 22-43: Kode program untuk menampilkan hasil perhitungan *cosine* yang dilakukan pada tahap klasifikasi akhir.

```
1   List<Double> pemetaan_bobot = new ArrayList<Double>();
2               for (int j = 0; j <
3   frekuensi_data_latih.get(i).size(); j++) {
4                   if (frekuensi_data_latih.get(i).get(j) == 1) {
5                       pemetaan_bobot.add(log);
6                   } else {
7                       pemetaan_bobot.add(0.0);
8                   }
9               }
10              tf_Idfs.get(i).setPemetaan_bobot(pemetaan_bobot);
11          }
12
13          tf_Idfs = this.normalisasi(tf_Idfs);
14          for (int i = 0; i < tf_Idfs.size(); i++) {
15              System.out.println(tf_Idfs.get(i).getTerm() + " = "
16  + tf_Idfs.get(i).getBobot().toString() + ", DF= " +
17  tf_Idfs.get(i).getDf() + ", log= " + tf_Idfs.get(i).getLog() + "
18  || pembobotan = " +
19  tf_Idfs.get(i).getPemetaan_bobot().toString() + ", ==
20  normalisasi = " + tf_Idfs.get(i).getNormalisasi().toString());
21          }
22          List<List<Double>> cosine = this.hitungCosine(tf_Idfs);
23          System.out.println("Cosine : ");
24          DefaultTableModel tabelgProc = new DefaultTableModel();
```

```
25          jTable7.setModel(tabelgProc);
26
27          tabelgProc.addColumn("Dokumen Uji");
28          tabelgProc.addColumn("Kategori");
29          for (int i = 0; i < cosine.size(); i++) {
30              System.out.println(cosine.get(i).size());
31              System.out.println(cosine.get(i).toString());
32              int index_kategori = 0;
33              double t_jumlah_cosine = 0;
34              for (int j = 0; j < cosine.get(i).size(); j++) {
35                  if (t_jumlah_cosine < cosine.get(i).get(j)) {
36                      t_jumlah_cosine = cosine.get(i).get(j);
37                      index_kategori = j;
38                  }
39              }
40              System.out.println(index_kategori);
41              String[] tampil = {"" + dokumen_uji.get(i),
42  kategori_latih.get(index_kategori)};
43              tabelgProc.addRow(tampil);
44      }
45  }
```

Kode Program 5.9 Implementasi Pemetaan Pembobotan Awal

4. Normalisasi

Normalisasi yang dimaksud merupakan penghitungan ketetapan nilai bobot di setiap *term* pada masing-masing dokumen yang akan digunakan untuk penghitungan jarak setelah normalisasi tersebut selesai. Perhitungan tersebut dapat dilihat pada kode program 5.10 beserta penjelasannya.

Baris 1-8 : Digunakan untuk menghasilkan perhitungan normalisasi dari setiap *term* klasifikasi yang dimiliki dari perolehan perhitungan hasil *TF-idf*.

```
1   private List<Tf_Idf> normalisasi(List<Tf_Idf> tf_Idfs) {
2           System.out.println("Jml kakaak");
3           System.out.println(tf_Idfs.size());
4           for (int i = 0; i < tf_Idfs.size(); i++) {
5               List<Double> normalisasi = new ArrayList<Double>();
6               for (int j = 0; j < data_term_pre.size(); j++) {
7                   double total_pangkat = 0.0;
8                   for (int k = 0; k < tf_Idfs.size(); k++) {
9                       double value =
10  tf_Idfs.get(k).getPemetaan_bobot().get(j);
11                      double pk = value * value;
12                      total_pangkat = total_pangkat + pk;
13                  }
14                  double pembilang = Math.sqrt(total_pangkat);
15                  double nilai_normalisasi =
16  tf_Idfs.get(i).getPemetaan_bobot().get(j) / pembilang;
17                  normalisasi.add(nilai_normalisasi);
18              }
19              tf_Idfs.get(i).setNormalisasi(normalisasi);
20
21          }
22          return tf_Idfs;
```

Kode Program 5.10 Implementasi Normaliasasi Ketetapan Bobot

5. Perhitungan *Cosine Similarity*

Perhitungan tingkat kemiripan yang dilakukan menggunakan teknik *cosine similarity* untuk menghasilkan nilai tertinggi sebagai tingkat kemiripan untuk dijadikan sebagai kategori yang terdekat. Proses tersebut dapat dilihat pada kode program 5.11 beserta penjelasannya.

Baris 1-21 : Kode program yang digunakan untuk menghitung *cosine similarity* dari hasil perhitungan normalisasi sebelumnya.

Baris 22-27 : Digunakan untuk menampilkan hasil tingkat kemiripan perhitungan *k-nn* antara data uji ekspansi dengan data latih

```
1  private List<List<Double>> hitungCosine(List<Tf_Idf> tf_Idfs) {
2          int batas_doukemen_latih = data_term_pre.size() -
3  this.dokumen_uji.size();
4          System.out.println(batas_doukemen_latih + " = " +
5  data_term_pre.size() + " - " + this.dokumen_uji.size());
6          List<List<Double>> cosine = new
7  ArrayList<List<Double>>();
8          for (int i = batas_doukemen_latih; i <
9  data_term_pre.size(); i++) {
10             System.out.println("perulangan ke :" + i);
11             List<Double> t_cos = new ArrayList<Double>();
12             for (int j = 0; j < batas_doukemen_latih; j++) {
13                 double jumlah_cosin = 0;
14
15 cosin = new double[400];
16 for (int k = 0; k < tf_Idfs.size(); k++) {
17  double t_data_uji = tf_Idfs.get(k).getNormalisasi().get(i);
18  double t_data_latih = tf_Idfs.get(k).getNormalisasi().get(j);
19  double hasil_kali = t_data_uji * t_data_latih;
20            jumlah_cosin = jumlah_cosin + hasil_kali;
21
22       }
23 t_cos.add(jumlah_cosin);
24  cosin[j] = jumlah_cosin;
25  if (cosin[j] != 0.0) {
26     System.out.println("Cosin D"+j+" dan Dbaru = " + cosin[j]);
27                 }
            }
            cosine.add(t_cos);
  }}
```

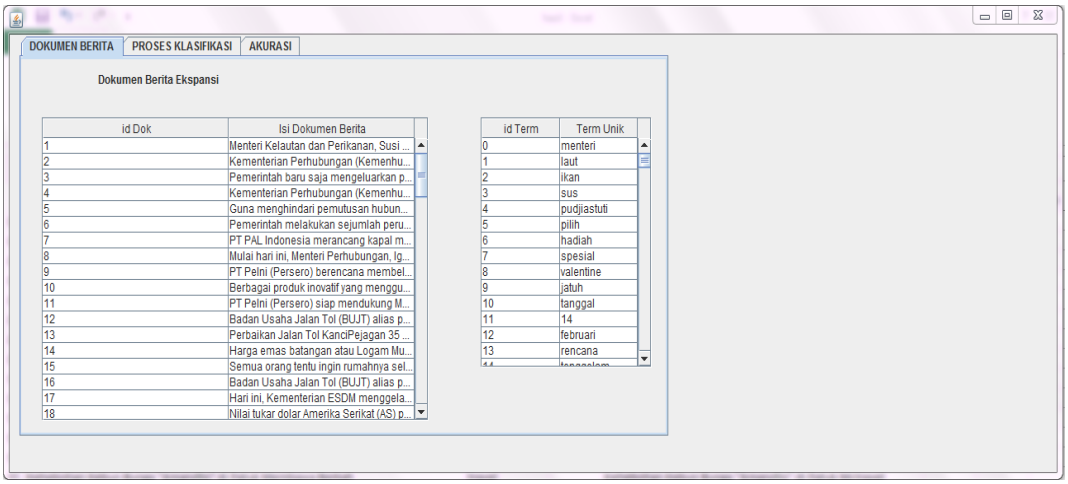
Kode Program 5.11 Implementasi Perhitungan *Cosine Similarity*

5.3 Implementasi Antarmuka

Implementasi antarmuka yang dimaksud adalah tampilan sistem yang digunakan untuk layar interaksi antar pengguna dengan sistem. Implementasi yang ditunjukkan adalah tampilan dokumen berita, tampilan tahap klasifikasi dan antarmuka perhitungan akurasi.

5.3.1 Implementasi Antarmuka Dokumen Berita

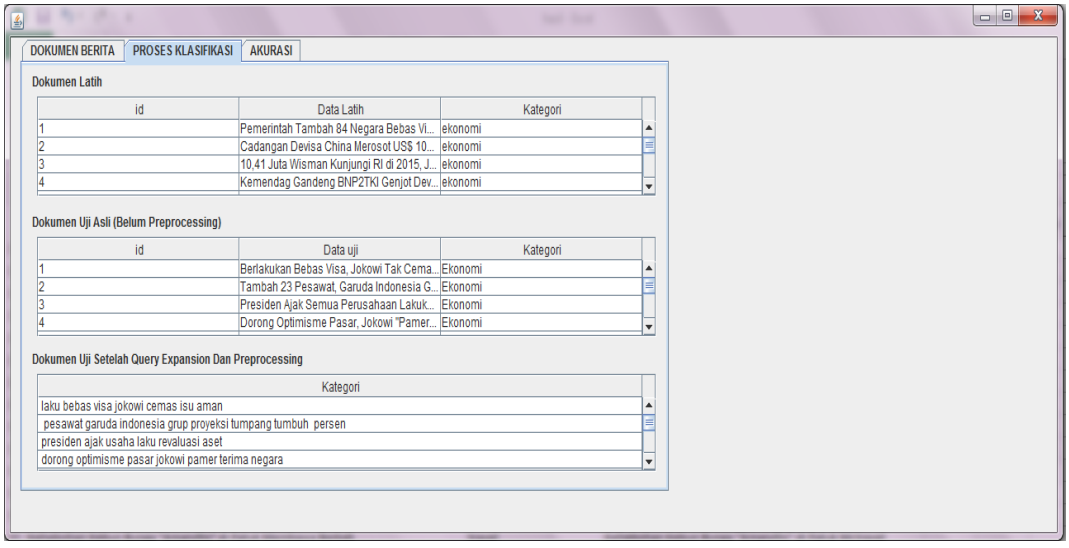
Halaman dokumen berita ini digunakan untuk menentukan kamus kata yang akan digunakan sebagai kedekatan kata. Halaman ini juga menampilkan *term-term* unik yang ada dari dokumen berita. Namun proses perhitungan jarak antar kata tidak diperlihatkan dalam halaman ini. Implementasi halaman tersebut dapat ditunjukkan pada Gambar 5.1.



Gambar 5.1 Tampilan Dokumen berita Ekspansi Kata

5.3.2 Implementasi Antarmuka Klasifikasi Teks

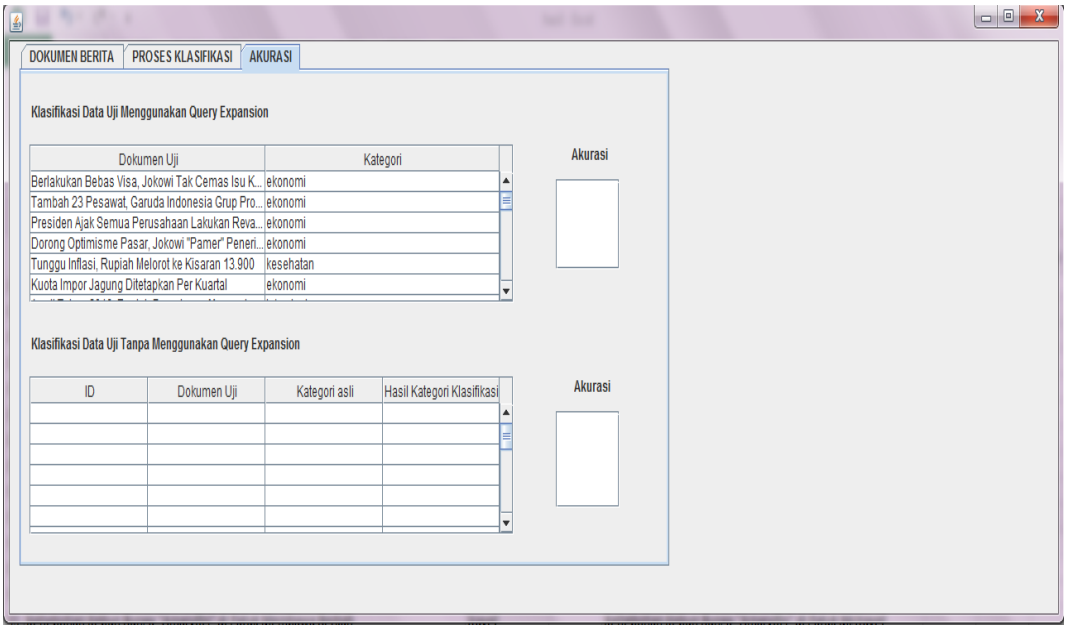
Antarmuka yang ditampilkan pada halaman berikut berupakan proses klasifikasi teks yang memperlihatkan tabel dokumen yang sudah terproses oleh *preprocessing* dan ekspansi kata. Implementasi tersebut dapat dilihat pada Gambar 5.2.



Gambar 5.2 Tampilan Halaman Klasifikasi Teks

5.3.3 Implementasi Antarmuka Hasil Klasifikasi

Tampilan pada halaman ini adalah hasil proses klasifikasi yang telah diproses. Pada halaman ini juga akan menampilkan kategori yang diproses oleh sistem dan juga kategori asli dari setiap data uji. Hasil yang ditampilkan terbagi menjadi dua yaitu klasifikasi menggunakan ekspansi kata dengan klasifikasi tanpa menggunakan ekspansi kata. Pada halaman tersebut dapat dilihat pada Gambar 5.3.



Gambar 5.3 Tampilan Halaman Hasil Klasifikasi