BAB IV
IMPLEMENTASI

Pada bab implementasi ini akan dibahas penerapan sistem dari proses perancangan yang telah di paparkan pada bab sebelumnya. Bab implementasi ini terdiri dari lingkungan implementasi, implementasi program, dan implementasi antar muka.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang sesuai dibutuhkan agar proses-proses dapat diimplementasikan dan berjalan tepat. Sistem dibangun pada lingkungan implementasi perangkat keras dan perangkat lunak. Lingkungan implementasi perangkat keras dan perangkat lunak yang digunakan adalah sebagai berikut :

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pembuatan sistem memiliki beberapa spesifikasi sebagai berikut :

1. Processor Intel® Core™ i7-3610QM CPU @2.3GHZ
2. VGA NVIDIA Geforce GT630M 2GB
3. Memory 4096 MB RAM
4. Harddisk 750 GB
5. Monitor 14”

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan untuk implementasi sistem ini adalah sebagai berikut:

1. Sistem Operasi Windows 7 Professional 64-bit
2. Java version 6 (build 1.6.0_16)
3. Netbeans IDE 6.7.1
4. Wordnet 2.1
5. Text Editor Notepad
6. Enterprise Architect 8.0.860
7. Microsoft Office Visio 2007

4.2 Implementasi Program

4.2.1 Kelas dan Fungsi

Pada sub bab ini membahas mengenai kelas-kelas yang terdapat pada sistem beserta fungsi (*method*) pada setiap kelas beserta kegunaanya. Kelas-kelas tersebut terdiri dari kelas preprocessing, PorterAlgo, postingList, KNN, evaluasi, addStopword, dan kelas gui.

1. Kelas preprocessing

   Kelas preprocessing berisi fungsi-fungsi untuk melakukan proses preprocessing pada dokumen latih dan dokumen uji. Kelas preprocessing ini terdiri dari fungsi-fungsi yang dapat dilihat pada tabel 4.1

Tabel 4.1 Fungsi-Fungsi pada Kelas Preprocesing

No	Fungsi (*Method*)	Keterangan
1	caseFolding	Fungsi untuk melakukan proses *Case Folding* pada dokumen
2	tampilDok	Fungsi untuk menampilkan isi dari suatu dokumen
3	preprocessingDokLatih	Fungsi untuk melakukan proses preprocessing setiap term pada dokumen latih. Pada fungsi ini proses yang dilakukan adalah *case folding* , *tokenizing*, *filtering*, *stemming*, *dictionary construction*, *ontology extraction*, dan *feature weighting*
4	cariSynset	Fungsi untuk mencari *synonym set* (*synset*) dari setiap *term* dengan bantuan database *WordNet*.
5	simpanNonStem	Fungsi untuk menyimpan *term* yang tidak melalui proses *stemming*
6	simpanStem	Fungsi untuk menyimpan *term* yang sudah melalui proses *stemming*
7	dictionaryConstruction	Fungsi untuk membentuk kamus atau *inverted index* untuk setiap *term*

8	tambahTermFrequency	Fungsi untuk menambahkan *Term Frequency* pada *term* yang saling bersinonim
9	tambahPostingList	Fungsi untuk menambahkan *Posting List* pada *term* yang saling bersinonim
10	joinTerm	Fungsi untuk menggabungkan *term* yang saling bersinonim
11	compareDF	Fungsi untuk membandingkan *Document Frequency* (DF) dari *term* yang saling bersinonim
12	tfTerm	Fungsi untuk menampilkan *Term Frequency* dari *term* yang saling bersinonim
13	hapusSynset	Fungsi untuk menghapus *term* yang sudah digabungkan dari kamus atau *inverted index*
14	featureSelection	Fungsi untuk menghapus *term* yang memiliki *Document Frequency* (DF) dibawah nilai *Threshold* yang telah ditentukan
15	preprocessingTesting	Fungsi untuk melakukan proses *preprocessing* pada dokumen uji. Pada fungsi ini proses yang dilakukan adalah *case folding, tokenizing, filtering, stemming*, dan *feature weighting.*
16	hitungQuery	Fungsi untuk menghitung *Term Frequency* dari setiap *term* pada dokumen uji
17	bobotLatih	Fungsi untuk menghitung bobot dari setiap *term* pada setiap dokumen latih
18	bobotUji	Fungsi untuk menghitung bobot dari setiap *term* pada setiap dokumen uji
19	tampilTerm	Fungsi untuk menyimpan *term-term* pada *inverted index* ke dalam arrayList untuk kemudian ditampilkan pada *interface* sistem
20	tampil	Fungsi untuk menampilkan *Term Frequency* dari setmua *term*

21	cosineSimilarity	Fungsi untuk menghitung cosine similarity dokumen uji dengan setiap dokumen latih
22	synsetNoun	Fungsi untuk mencari synset setiap term pada dokumen latih dan dokumen uji

2. Kelas PorterAlgo

Kelas PorterAlgo berisi algoritma Porter-Stemmer dimana *term-term* pada dokumen latih dan dokumen uji akan diubah menjadi bentuk kata dasarnya. Fungsi-fungsi pada kelas PorterAlgo dapat dilihat pada tabel 4.2

Tabel 4.2 Fungsi-Fungsi pada Kelas PorterAlgo

No	Fungsi (*Method*)	Keterangan
1	hasSuffix	Fungsi untuk mengecek apakah suatu *term* memiliki akhiran atau tidak
2	vowel	Fungsi untuk mengetahui apakah *term* berisi huruf vokal
3	measure	Fungsi untuk mengetahui *measure* dari suatu *term*
4	containsVowel	Fungsi untuk mengetahui apakah *term* berisi huruf vokal atau tidak. Pada fungsi containsVowel terdapat fungsi vowel di dalamnya
5	cvc	Fungsi untuk mengetahui apakah *term* memiliki bentuk "cvc" didalamnya
6	step1	Fungsi untuk menjalankan algoritma Porter-Stemmer pada Step 1 beserta aturan penghapusan aturan pada step 1
7	step 2	Fungsi untuk menjalankan algoritma Porter-Stemmer pada Step 2 beserta aturan penghapusan aturan pada step 2
8	step3	Fungsi untuk menjalankan algoritma Porter-Stemmer pada Step 3 beserta aturan penghapusan aturan pada step 3

9	step4	Fungsi untuk menjalankan algoritma Porter-Stemmer pada Step 4 beserta aturan penghapusan aturan pada step 4
10	step5	Fungsi untuk menjalankan algoritma Porter-Stemmer pada Step 5 beserta aturan penghapusan aturan pada step 5
11	stripSuffixes	Fungsi utama pada kelas PorterAlgo dimana pada fungsi ini fungsi step1 sampai step5 dijalankan secara sekuensial

3. Kelas postingList

   Kelas postingList ini merupakan struktur data untuk membentuk *inverted index*. Jadi nantinya isi dari *inverted index* adalah objek yang menginstan kelas postingList. Kelas postingList terdiri dari tiga konstruktor dengan tiga parameter yang berbeda. Konstruktor dari kelas postingList dapat dilihat pada tabel 4.3.

Tabel 4.3 Konstruktor dari Kelas PostingList

No	Konstruktor dan Parameter	Keterangan
1	postingList()	Konstruktor tanpa parameter untuk inisialisasi awal postingList pada *inverted index*
2	postingList(String berita, String kat, String idD, int i, int tf, Boolean baru)	Konstruktor untuk memasukkan *term* baru yang belum ada pada *inverted index*
3	postingList(String berita, String kat, String idD, int tf, Boolean baru)	Konstruktor untuk *term* yang sudah ada pada *inverted index*. Konstruktor ini digunakan untuk menambahkan frekuensi *term.*

4. Kelas addStopword

   Kelas addStopword merupakan kelas untuk menyimpan daftar kata yang merupakan *stopword*. Fungsi-fungsi pada kelas addStopword dapat dilihat pada tabel 4.4

Tabel 4.4 Konstruktor dan Fungsi dari Kelas AddStopword

No	Fungsi (*Method*)	Keterangan
1	addStopword (konstruktor)	Konstruktor pada kelas addStopword yang berguna untuk membaca daftar stopword dari file .txt dan menyimpannya pada arrayList
2	isStopword	Fungsi untuk menentukan apakah suatu *term* merupakan *stopword* atau bukan

5. Kelas KNN

   Kelas KNN merupakan kelas yang berisi proses penentuan kategori pada dokumen uji. Fungsi-fungsi pada kelas KNN dapat dilihat pada tabel 4.5

Tabel 4.5 Fungsi-Fungsi dari Kelas KNN

No	Fungsi (*Method*)	Keterangan
1	KNN	Fungsi untuk memberikan skor kategori pada dokumen uji. Kategori dengan nilai skor paling tinggi akan dijadikan kategori dari dokumen uji
2	klasifikasiDokumen	Fungsi untuk menentukan skor tertinggi dari kategori yang telah dihitung pada *method* KNN sebelumnya
3	Sorting	Fungsi untuk mengurutkan nilai *cosine similarity* yang telah diambil sejumlah “k” . Proses *sorting* adalah menggunakan algoritma *Bubble Sort*
4	sortSimilarity	Fungsi untuk menentukan kategori pada dokumen latih yang telah diambil sejumlah “k”

6. Kelas evaluasi

   Kelas evaluasi berisi proses untuk menghitung hasil evaluasi yang dihasilkan oleh sistem. Evaluasi sistem diukur dengan nilai *recall*, *precision*, dan *F1measure*. Fungsi-fungsi dari kelas evaluasi dapat dilihat pada tabel 4.6

Tabel 4.6 Fungsi-Fungsi pada Kelas Evaluasi

No	Fungsi (*Method*)	Keterangan
1	hitungEvaluasi	Fungsi untuk menghitung *recall*, *precision*, dan *F1measure* dari setiap kategori
2	averageEvaluation	Fungsi untuk menghitung rata-rata dari *recal*, *precision* dan *F1measure* dari setiap skenario pengujian

4.22 Tahapan Pemrosesan

Proses pertama pada sistem klasifikasi berita berbahasa inggris menggunakan algoritma *K-Nearest Neighbor* (KNN) berbasis ontologi adalah proses *preprocessing* dokumen latih. *Preprocessing* dokumen latih dilakukan pada *Method* preprocessingDokLatih yang terdapat pada kelas *preprocessing*. *Source code method preprocessing* dokumen latih dapat dilihat pada gambar 4.1.

```
//preprocessing setiap dokumen latih
        for(int i=0; i<dokLatih.length; i++)
      {
          idDokumen = "latih"+(i+1);
          idd = i+1;
            isiDokumen = caseFolding(dokLatih[i][0]);
            token = new StringTokenizer(isiDokumen);
            int pos = 1;
            while(token.hasMoreTokens())
            {
            term = token.nextToken();
//jika term tidak merupakan stopword maka dilakukan proses
selanjutnya
            if(!stop.isStopword(term))
            {
//term yang memiliki panjang kurang dari 3 karakter tidak diproses
              if(term.length()>2)
               {
                //mengecek apakah term memliki Part Of Speech
```

```
if(wordnet.isAdjective(term)||wordnet.isAdverb(term)||wordnet.is
Noun(term)||wordnet.isVerb(term))
                        {
                        sebelumStem = term;
                        //proses stemming
                        term = stemming.stripSuffixes(term);
                        sesudahStem = term;
                        simpanStem(sebelumStem, sesudahStem);
                         //proses pembentukan kamus (inverted
index)
                        dictionaryConstruction(term,
dokLatih[i][1], dokLatih[i][2], pos, idDokumen,idd);
                        pos++;
                        }
                        else
//jika term tidak memiliki Part Of Speech maka term langsung
disimpan
                         {
                            simpanNonStem(term);
dictionaryConstruction(term,dokLatih[i][1],dokLatih[i][2],pos,id
Dokumen,idd);
                            pos++;
                        }
                    }
                }
            }
        }
```

Gambar 4.1 *Source Code Preprocessing* Dokumen Latih

Pada *method* preprocessingDokLatih ini terdiri dari proses *case folding* yang dijalankan pada *method* caseFolding pada kelas preprocessing. *Source code* dari proses *case folding* dapat dilihat pada gambar 4.2. Setelah melalui proses *case folding*, langkah selanjutnya adalah proses *tokenizing* dengan menggunakan fungsi *StringTokenizer* dari *Java*. Setelah isi dari setiap dokumen latih dipecah menjadi kumpulan *token*, langkah selanjutnya adalah proses *filtering*. Proses *filtering* ini terdapat dapat dilihat pada *source code* 4.3. *Term* yang bukan *stopword* akan melalui proses selanjutnya yaitu proses *stemming*. Namun sebelum *term* tersebut melalui proses *stemming*, *term* tersebut dicek terlebih dahulu *term* tersebut memiliki POS (*Part of Speech*) dengan menggunakan *method* isNoun(), isAdjective(), isVerb(), dan isAdverb() yang telah disediakan oleh *RiTa.WordNet*. Jika *term* tidak

memiliki POS, maka *term* langsung disimpan di dalam kamus. *Term* yang telah melalui proses *stemming* kemudian disimpan di dalam kamus.

```
    public String caseFolding(String teks)
    {
      String isiTeks = "";
      try{
         BufferedReader read = new BufferedReader(new
FileReader(teks));
            Scanner input = new Scanner(read);
            //membaca setiap kata pada dokumen
            while(input.hasNext()){
            isiTeks = isiTeks+'
'+input.next().toLowerCase().replaceAll("[^A-Za-z\']", "
").trim();
            }
            input.close();
       } catch (Exception e){
            System.out.println("File tidak ditemukan");

       }
      return isiTeks;
    }
```

Gambar 4.2 *Source Code* Proses *Case Folding*

```
public boolean isStopword(String kata)
    {
      return(stopword.contains(kata));
    }
```

Gambar 4.3 *Source Code* Proses *Filtering*

Setelah melalui proses *stemming*, langkah selanjutnya adalah proses *Dictionary Construction* atau pembentukan kamus. Setiap *term* akan disimpan di dalam *inverted index* yang dibangun menggunakan *hashmap* dimana *key* dari *hashmap* tersebut adalah *string term* dengan value adalah objek pada kelas PostingList. Isi dari kelas PostingList dapat dilihat pada gambar 4.4 serta proses *dictionary construction* dapat dilihat pada gambar 4.5

```
public postingList(String berita, String kat, String idD, int i,
int tf, Boolean baru)
    {
      ArrayList<Integer> posisi = new ArrayList<Integer>();
      namaDokumen = new ArrayList<String>();
      kategori = new ArrayList<String>();
      idDokumen = new ArrayList<String>();
     id = new ArrayList<Integer>();
```

```
        frekuensiTerm = new ArrayList<ArrayList<Integer>>();
      namaDokumen.add(berita);
        kategori.add(kat);
        idDokumen.add(idD);
        id.add(i);
        posisi.add(tf);
        frekuensiTerm.add(posisi);
    }
```

Gambar 4.4 *Source Code* Posting List

```
public void dictionaryConstruction(String term, String dokLatih,
String kategori, int tf, String id,int i)
    {
      if(invertedIndex.containsKey(term)){ //jika term sudah ada
pada kamus
            postList = invertedIndex.get(term);
            //jika nama dok belum ada pada kamus
            if(postList.namaDokumen.indexOf(dokLatih) < 0){
                postList.namaDokumen.add(dokLatih);
                postList.kategori.add(kategori);
                postList.idDokumen.add(id);
                postList.id.add(i);
                postList.frekuensiTerm.add(new
ArrayList<Integer>());
            }

postList.frekuensiTerm.get(postList.namaDokumen.indexOf(dokLatih
)).add(tf);
            invertedIndex.put(term, postList);
      } else { //jika term belum ada
            invertedIndex.put(term,   new   postingList(dokLatih,
kategori, id, i, tf, true));
      }
    }
```

Gambar 4.5 *Source Code Dictionary Construction*

Setelah semua *term* disimpan ke dalam *inverted index*, langkah selanjutnya adalah proses *Feature Selection*. Proses ini dijalankan pada method featureSelection dengan parameter nilai *document threshold*. *Source code* untuk menghitung *Document Frequency* dapat dilihat pada gambar 4.6.

Term yang memiliki *Document Frequency* kurang dari nilai *Document Threshold* akan dihapus. *Source code* proses *feature selection* dapat dilihat pada gambar 4.7.

```
df = new int[invertedIndex.size()];
      idf = new double [invertedIndex.size()];
      double tempIDF;
        //menghitung nilai IDF setiap term dan menyimpannya di
inverted index
      for     (Map.Entry<String,     postingList>     entry     :
invertedIndex.entrySet())
      {
            postList = invertedIndex.get(entry.getKey());
            df[i] = entry.getValue().idDokumen.size();
            tempIDF = jumDokLatih/df[i];
            idf[i] = Math.log(tempIDF)/Math.log(2);
            postList.idf = idf[i];
            invertedIndex.put(entry.getKey(), postList);
            i++;
      }
```

Gambar 4.6 *Source Code* Menghitung DF dan IDF

```
    public void featureSelection(int threshold)
    {
      //menyimpan term yang memiliki DF < threshold
      for(Map.Entry<String, postingList> entry :
invertedIndex.entrySet())
      {
            if(entry.getValue().namaDokumen.size()< threshold)
            {

DFtermThreshold.add(entry.getValue().frekuensiTerm.size());
                  termThreshold.add(entry.getKey());
            }
      }

        //menghapus term tersebut dari inverted index
      for(int j=0; j<termThreshold.size(); j++)
      {
            invertedIndex.remove(termThreshold.get(j));
      }
    }
```

Gambar 4.7 *Source Code Feature Selection*

Proses selanjutnya setelah *feature selection* adalah *ontology extraction*. Proses ini mencari apakah *term-term* pada *inverted index* saling bersinonim atau tidak. Jika *term-term* tersebut saling bersinonim maka akan digabung frekuensi term dari *term-term* tersebut. Sebelum digabungkan, *term-term* tersebut dibandingkan nilai DF (*Document Frequency*) nya.

Term yang memiliki DF lebih kecil akan digabungkan dengan *term* yang memiliki DF lebih besar. Proses *ontology extraction* dilakukan pada *method* cariSynset dan *source code* nya dapat dilihat pada gambar 4.8. Proses penggabungan *term* dapat dilihat pada *method* joinTerm dan *source code* nya dapat dilihat pada gambar 4.9. Proses membandingkan DF dilakukan pada *method* compareDF dan *source code* nya dapat dilihat pada gambar 4.10

```
  void cariSynset(ArrayList<String> tempTerm)
    {

        for (int i=0;i<tempTerm.size();i++)
            {
               String []hasilNoun=synsetNoun(tempTerm.get(i));
               if(hasilNoun!=null)
               {
                     for(String str :hasilNoun)
                     {
                         synset.add(str);
                         if(tempTerm.contains(str)
&&!synset.contains(tempTerm.get(i)))
                           {
                               hasilSinonim.add("Term
\t\t"+tempTerm.get(i)+"\t\t bersinonim dengan term \t\t"+str);
                               term1.add(tfTerm(tempTerm.get(i)));
                               term1.add(tfTerm(str));
                               joinTerm(tempTerm.get(i),str);
                           }
                     }

                }
        }
```

Gambar 4.8 *Source Code Ontology Extraction*

```
public boolean compareDF (String term,String synset)
    {
        boolean hasil=true;
        int DFterm=0,DFsynset=0;
        for      (Map.Entry<String,postingList>      entry      :
invertedIndex.entrySet())
        {
           if (entry.getKey().equals(term))
           {
               DFterm=entry.getValue().namaDokumen.size();
           }
```

```
        }
        for     (Map.Entry<String,postingList>     entry     :
invertedIndex.entrySet())
        {
           if (entry.getKey().equals(synset))
           {
               DFsynset=entry.getValue().namaDokumen.size();
           }
        }
        if (DFterm > DFsynset)
        {
            hasil=true;
        }
        else if (DFterm < DFsynset)
        {
            hasil=false;
        }
        else if (DFterm==DFsynset)
        {
            hasil=true;
        }
        return hasil;
    }
```

Gambar 4.9 *Source Code* compareDF

```
public void joinTerm(String term, String synset)
{
/*membandingkan nilai DF dari kedua term.term yang memiliki DF
lebih kecil akan digabungkan dengan
*term yang memiliki DF lebih besar. jika kedua term memiliki
nilai DF yang sama,pada program ini
sudah ditetapkan kalau term kedua yang akan digabung dengan term
pertama.
*/
String kata1="",kata2="";
if (compareDF(term,synset)) //mengecek document frequency dari
term
{
kata1=term;
kata2=synset;
}
else if (!compareDF(term,synset))
{
kata1=synset;
kata2=term;
}
String namaDokumen="";
```

```
for (Map.Entry<String,postingList> entry :
invertedIndex.entrySet()) //proses mencari term pertama pada
inverted index
{
if (entry.getKey().equals(kata1))
{
for (Map.Entry<String,postingList>entry1 :
invertedIndex.entrySet()) //proses mencari term kedua pada
inverted index
if(entry1.getKey().equals(kata2))
{

for (int i=0;i<entry1.getValue().idDokumen.size();i++)
{
//jika id dokumen pada term kedua sudah ada pada dokumen kedua
if
(entry.getValue().idDokumen.contains(entry1.getValue().idDokumen
.get(i)))
{
int counter=0;
while (counter<entry.getValue().idDokumen.size())
{
if
(entry1.getValue().idDokumen.get(i).equals(entry.getValue().idDo
kumen.get(counter)))
{
namaDokumen=entry.getValue().namaDokumen.get(counter);
//
System.out.println(entry1.getValue().frekuensiTerm.get(i).size()
);
int count=0;
while (count<entry1.getValue().frekuensiTerm.get(i).size())
{
tambahTermFrequency(entry.getKey(),count+1,namaDokumen);
count++;
}
}
counter++;
}
term2.add(tfTerm(entry.getKey()));
removeSynset.add(entry1.getKey());
}
else //jika id dokumen belum ada maka akan dibuat posting list
baru
{

namaDokumen=entry1.getValue().namaDokumen.get(i);
String kategori=entry1.getValue().kategori.get(i);
String idDokumen=entry1.getValue().idDokumen.get(i);
int id=entry1.getValue().id.get(i);
```

```
tambahPostingList(entry.getKey(),namaDokumen,kategori,idDokumen,
id);

int count=0;
while(count<entry1.getValue().frekuensiTerm.get(i).size())
{
tambahTermFrequency(entry.getKey(),count+1,namaDokumen);
count++;
                                    }
term2.add(tfTerm(entry.getKey()));
removeSynset.add(entry1.getKey());
                                    }
                                }
                            }
                        }
                }
        }
}
```

Gambar 4.10 *Source Code* Penggabungan *Term*

Proses *preprocessing* dok latih telah selesai dilakukan. Proses *preprocessing* ini juga dilakukan pada dokumen uji. *Preprocessing* dokumen uji juga melalui proses *case folding*, *tokennizing*, *filtering*, dan *stemming*. *Term-term* yang melalui proses *stemming* adalah *term* yang memiliki POS (*Part Of Speech*). *Preprocessing* dokumen uji dilakukan di dalam *method* preprocessingTesting dan *source code*nya dapat dilihat pada gambar 4.11.

```
public void preprocessingTesting(String dokUji, String namaDok,
String kategori, String id)
    {
      String term;
      isiDokumen =caseFolding(dokUji);
      token = new StringTokenizer(isiDokumen);
      int pos = 1;

       while(token.hasMoreTokens())
      {
            term = token.nextToken();
            if(!stop.isStopword(term))
            {
                if(term.length()>2)
                {

if(wordnet.isAdjective(term)||wordnet.isAdverb(term)||wordnet.is
Noun(term)||wordnet.isVerb(term))
                    {
                     sebelumStem = term;
```

```
                    term = stemming.stripSuffixes(term);
                    sesudahStem = term;
                    simpanStem(sebelumStem, sesudahStem);
                    //menyimpan frekuensi term uji pada inverted
index
                    hitungQuery(term,  namaDok,  kategori,  pos,
id);
                    pos++;
                    }
                    else
                    {
                        simpanNonStem(term);

hitungQuery(term,namaDok,kategori,pos,id);
                        pos++;
                    }

                }
            }
        }
    }
```

Gambar 4.11 *Source Code Preprocessing* Dokumen Uji

Setelah *preprocessing* kedua jenis dokumen dilakukan, langkah selanjutnya adalah menentukan bobot dari setiap *term* pada dokumen latih dan dokumen uji. Pertama adalah menentukan bobot pada dokumen uji. Bobot pada *term* dihitung dengan menggunakan metode TFIDF yaitu mengalikan *Term Frequency* setiap *term* pada setiap *posting list* dengan nilai IDF (*inverse document frequency*) dari *term* tersebut. *Source code* dari proses pembobotan dokumen latih dapat dilihat pada gambar 4.12.

```
public double[][] bobotLatih()
    {
        bobotDokLatih = new
double[invertedIndex.size()][(int)jumDokLatih];
        int x=0;

        for(Map.Entry<String, postingList> entry :
invertedIndex.entrySet())
      {
            for(int i =0; i <
entry.getValue().namaDokumen.size(); i++)
            {
                //TFIDF = tf * idf
```

```
                tfidf =
entry.getValue().frekuensiTerm.get(i).size()*entry.getValue().id
f;
                //simpan pada array
                bobotDokLatih[x][entry.getValue().id.get(i)-1] =
tfidf;
            }
            x++;
        }
        tampilTerm();

        return bobotDokLatih;
    }
```

Gambar 4.12 *Source Code* Pembobotan Dokumen Latih

Nilai dari TFIDF *term-term* pada dokumen latih akan disimpan pada *array* dua dimensi. Selanjutnya adalah *term-term* pada dokumen uji jug dihitung bobotnya. Pada dokumen uji tidak terdapat proses *dictionary construction* tetapi bobot dari dokumen uji ditentukan berdasarkan kemuculan *term* tersebut pada *inverted index* dokumen latih.

Langkah pertama untuk menentukan bobot dari dokumen uji adalah menghitung *Term Frequency* dari dokumen uji. *Source code* dari perhitungan *Term Frequency* dari dokumen uji dapat dilihat pada gambar 4.13. Setelah dihitung *term frequency* dari dokumen uji maka langkah selanjutnya adalah menghitung TFIDF dari dokumen uji. IDF yang digunakan pada penghitungan bobot dokumen uji adalah IDF pada dokumen latih. Proses penghitungan TFIDF dokumen uji dapat dilihat pada gambar 4.14

```
public void hitungQuery(String term, String dokUji, String
kategori, int tf, String id)
    {
      if(invertedIndex.containsKey(term)) //jika term sudah ada
pada kamus
       {
            postList = invertedIndex.get(term);
            if(postList.namaDokumen.indexOf(dokUji)<0)     //jika
dokumen sudah ada
            {
            postList.namaDokumen.add(dokUji);
            postList.kategori.add(kategori);
                postList.idDokumen.add(id);
```

```
            postList.frekuensiTerm.add(new
ArrayList<Integer>());
            }
            //jika dokumen belum ada

postList.frekuensiTerm.get(postList.namaDokumen.indexOf(dokUji))
.add(tf);
            invertedIndex.put(term, postList);
      }
        else //jika term tidak ada maka mengecek synset dari dok
latih
        {
            String [] hasilNoun=synsetNoun(term);
            if (hasilNoun!=null)
            {
                for (String str : hasilNoun)
                {
                    if (tempTerm.contains(str))
                    {
                        postList=invertedIndex.get(str);
                        if
(postList.namaDokumen.indexOf(dokUji)<0) //jika dokumen sudah ada
                        {
                            postList.namaDokumen.add(dokUji);
                            postList.kategori.add(kategori);
                            postList.frekuensiTerm.add(new
ArrayList<Integer>());
                        }
                        //jika dokumen belum ada

postList.frekuensiTerm.get(postList.namaDokumen.indexOf(dokUji))
.add(tf);
                        invertedIndex.put(str, postList);
                    }
                }
            }
        }
    }
```

Gambar 4.13 *Source Code* Hitung Frekuensi *Term* Dokumen Uji

```
  public double[] bobotUji(String dokUji)
    {
      bobotDokUji = new double[invertedIndex.size()];
      int j=0;
      for(Map.Entry<String, postingList> entry :
invertedIndex.entrySet())
      {
```

```
            for (int i=0; i <
entry.getValue().namaDokumen.size(); i++)
            {

      if(entry.getValue().namaDokumen.get(i).equals(dokUji))
                  {
                        tfidf =
entry.getValue().frekuensiTerm.get(i).size()*entry.getValue().id
f;
                        bobotDokUji[j] = tfidf;
            } else{
                        bobotDokUji[j]=0;
            }
            }
            j++;
      }
      return bobotDokUji;
    }
```

Gambar 4.14 *Source Code* Hitung Bobot Dokumen Uji

Bobot dari kedua jenis dokumen telah ditentukan dan langkah selanjutnya adalah menghitung *cosine similarity* antara dokumen uji dengan setiap dokumen latih. *Source code* dari proses *cosine similarity* ini dapat dilihat pada gambar 4.15

```
public double[] CossineSimilarity(double[][] bobotLatih,
double[] bobotUji)
    {
      panjangBUji=0;
        double dot = 0;
        double length = 0;
        int jumDok = bobotLatih[0].length;

        perkalian = new double[jumDok];
      panjangBLatih = new double[jumDok];
        double[] CosSim = new double[jumDok];

        //menghitung norm dokumen uji
        for(int i=0; i<bobotUji.length; i++)
      {
            panjangBUji = panjangBUji+Math.pow(bobotUji[i], 2);
        }

        //menghitung norm dokumen latih dan dot product bobot
dokumen latih dan dokumen uji
        for(int i=0; i<bobotLatih.length; i++)
        {
            for(int j=0; j<bobotLatih[i].length; j++)
            {
```

```
                length = Math.pow(bobotLatih[i][j], 2);
                panjangBLatih[j] += length;
                dot = bobotLatih[i][j]*bobotUji[i];
                perkalian[j] += dot;
            }
        }

        //cossim = dot product / akar (panjang dok latih *
panjang dok uji)
        for(int d=0; d<jumDok; d++)
        {
                CosSim[d] =
perkalian[d]/(Math.sqrt(panjangBLatih[d]*panjangBUji));
      }
        return CosSim;
    }
```

Gambar 4.15 *Source Code* Proses *Cosine Similarity*

Langkah selanjutnya adalah menentukan kategori dari dokumen uji dengan menggunakan metode KNN. Proses nya adalah hasil *cosine similarity* dari dokumen uji dengan dokumen latih diurutkan dengan algoritma *bubble sort*. *Source code* dari pengurutan hasil *cosine similarity* dapat dilihat pada *source code* 4.16. Setelah diurutkan, langkah selanjutnya adalah mengambil nilai *cosine similarity* sebanyak "k" teratas dengan jumlah "k" telah ditentukan sebelumnya. Setelah diambil sejumlah "k" maka langkah selanjutnya adalah menghitung skor dari setiap kategori pada *method* KNN. Kategori dengan nilai skor tertinggi akan dijadikan kategori pada dokumen uji. *Source code* untuk *method* KNN dan penentuan kategori dokumen uji dapat dilihat pada gambar 4.17 dan 4.18

```
public String[][] Sorting(String[][] sim)
    {
      String temp[];
      int NDok = sim.length;
      for(int i=1; i<NDok; i++)
      {
            for(int j=0; j<NDok-1; j++)
            {

      if(Double.parseDouble(sim[i][2])>Double.parseDouble(sim[j]
[2]))
            {
                  temp = sim[j];
                  sim[j] = sim[i];
```

```
                    sim[i] = temp;
                }
                }
        }
        return sim;
    }
```

Gambar 4.16 *Source Code* Algoritma *Bubble Sort*

```
public double [] KNN(int k, String[][] kategori, String[][]
simUrut)
    {

      double[] skor = new double[kategori.length];

        for(int i=0; i<kategori.length; i++)
      {
            double temp =0;
            for(int l=0; l<k; l++)
            {
            if(simUrut[l][1].equals(kategori[i][0]))
                    temp                                    =
temp+Double.parseDouble(simUrut[l][2])*1;
            else
                    temp                                    =
temp+Double.parseDouble(simUrut[l][2])*0;
            }
            skor[i] = temp;
      }
      return skor;
    }
```

Gambar 4.17 *Source Code* Proses KNN

```
public String klasifikasiDokumen(double[] skor, String[][]
kategori)
    {
        String hasilKlasifikasi;
      double max = skor[0];
      int index = 0;

        for(int i=0; i<skor.length; i++)
      {
            if(max < skor[i])
            {
            max = skor[i];
            index = i;
            }
```

```
        }
        hasilKlasifikasi = kategori[index][0];

        return hasilKlasifikasi;
    }
```

Gambar 4.18 *Source Code* Penentuan Kategori Dokumen Uji

Setelah menentukan kategori dari dokume uji, maka dapat ditentukan kinerja sistem dengan menghitung *precision,recall,*dan *F1measure*. Gambar untuk menghitung *precision,recall,* dan *F1measure* dapat dilihat pada gambar 4.19. Selain itu, terdapat juga fungsi untuk menghitung rata-rata dari *precision, recall,* dan *F1measure* untuk menetukan keseluruhan dari kinerja sistem setelah melakukan beberapa kali skenario percobaan. Gambar dari perhitungan rata-rata *precision,recall,* dan *F1measure* dapat dilihat pada gambar 4.20.

```
public double[][] hitungEvaluasi(String[][] dokUji, String[]
hasilKlasifikasi, String[][] kategori)
    {
      evaluasi = new double[kategori.length][3];
      for(int i=0; i<kategori.length; i++)
      {
            truePositive=0;falsePositive=0; falseNegative=0;

            for(int j=0; j<dokUji.length; j++)
            {
                  if((hasilKlasifikasi[j] == null ?
kategori[i][0] == null :
hasilKlasifikasi[j].equals(kategori[i][0]))&&(dokUji[j][2] ==
null ? kategori[i][0] == null :
dokUji[j][2].equals(kategori[i][0])) )
                        truePositive++;
            else if((hasilKlasifikasi[j] == null ?
kategori[i][0] == null :
hasilKlasifikasi[j].equals(kategori[i][0])) && (dokUji[j][2] ==
null ? kategori[i][0] != null :
!dokUji[j][2].equals(kategori[i][0])))
                        falsePositive++;
            else if((hasilKlasifikasi[j] == null ?
kategori[i][0] != null :
!hasilKlasifikasi[j].equals(kategori[i][0])) && (dokUji[j][2] ==
null ? kategori[i][0] == null :
dokUji[j][2].equals(kategori[i][0])))
                        falseNegative++;
            }
```

```
 if (truePositive==0)//jika tidak ada nilai dari true positive
maka precision recall dan f1 measure kategori tersebut diset 0
            {
                evaluasi [i][0]=0;//precision di set 0
                evaluasi [i][1]=0;//recall di set 0
                evaluasi [i][2]=0;//f1 measure di set 0
            }
            else
            {
                evaluasi[i][0] =
truePositive/(truePositive+falsePositive); //precision
                evaluasi[i][1] =
truePositive/(truePositive+falseNegative); //recall
                //F1 measure
                evaluasi[i][2] =
(2*evaluasi[i][0]*evaluasi[i][1])/(evaluasi[i][0]+evaluasi[i][1]
);
            }     }

      return evaluasi;
    }
```

Gambar 4.19 *Source Code* Menghitung Evaluasi Hasil Klasifikasi Sistem

```
public double[] averageEvaluation(double[][] evaluasi)
      {
            double avgPrecision=0, avgRecall=0, avgF1=0;
            average = new double[evaluasi.length];
            for(int i=0; i<evaluasi.length; i++)
            {
                  avgPrecision = avgPrecision+evaluasi[i][0];
                  avgRecall = avgRecall+evaluasi[i][1];
                  avgF1 = avgF1+evaluasi[i][2];
            }

            average[0] = avgPrecision/evaluasi.length;
            average[1] = avgRecall/evaluasi.length;
            average[2] = avgF1/evaluasi.length;

            return average;
      }
```

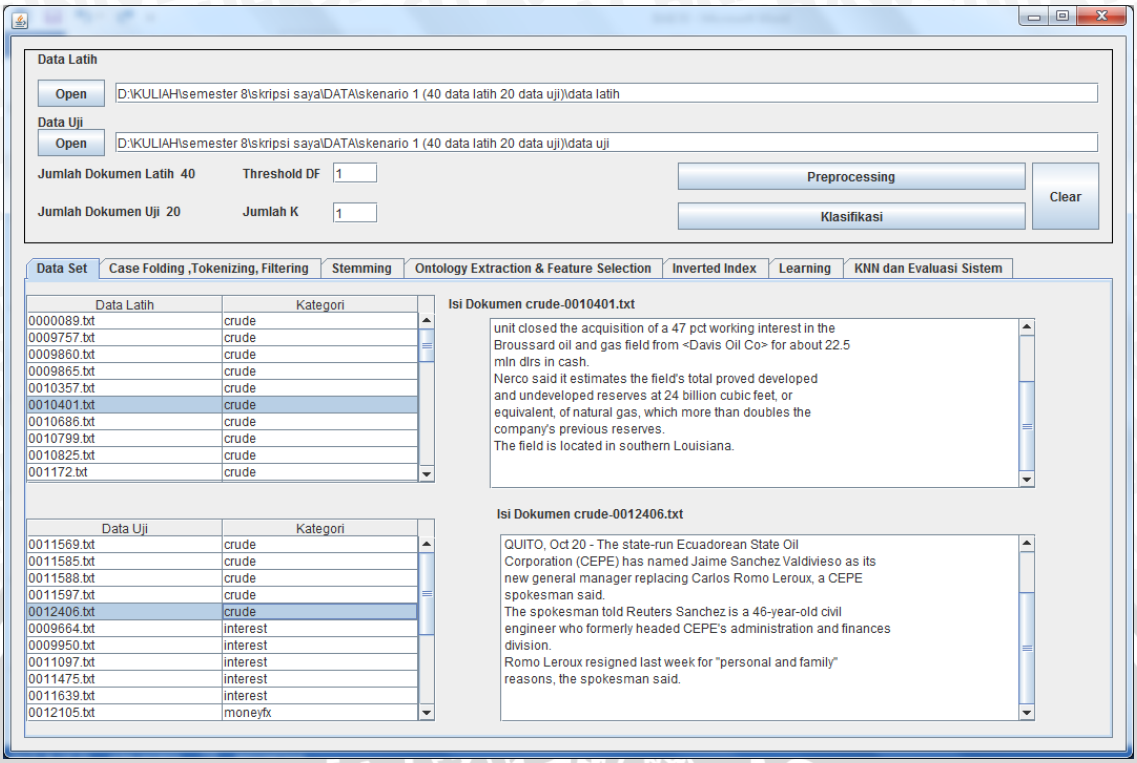
Gambar 4.20 *Source Code* Menghitung Rata-Rata Evaluasi Hasil Klasifikasi Sistem

4.3 Implementasi Antar Muka

Pada sub bab ini akan dijelaskan implementasi dari antar muka yang telah dirancang pada bab sebelumnya. Antar muka sistem klasifikasi berita berbahasa inggris menggunakan algoritma KNN (*K-Nearest Neighbor*) berbasis ontologi ini terdiri dari satu *form* utama dengan beberapa *field input*, beberapa tombol serta beberapa *tab*. *Field input* adalah jumlah "k" dan *Document Frequency Threshold*. Tombol utama terdiri dari dua tombol utama yaitu tombol *preprocessing* dan tombol klasifikasi. Tombol lainnya adalah tombol *clear* untuk menghapus sluruh isi *field* pada semua tab serta dua tombol untuk membuka direktori dokumen latih dan dokumen uji.

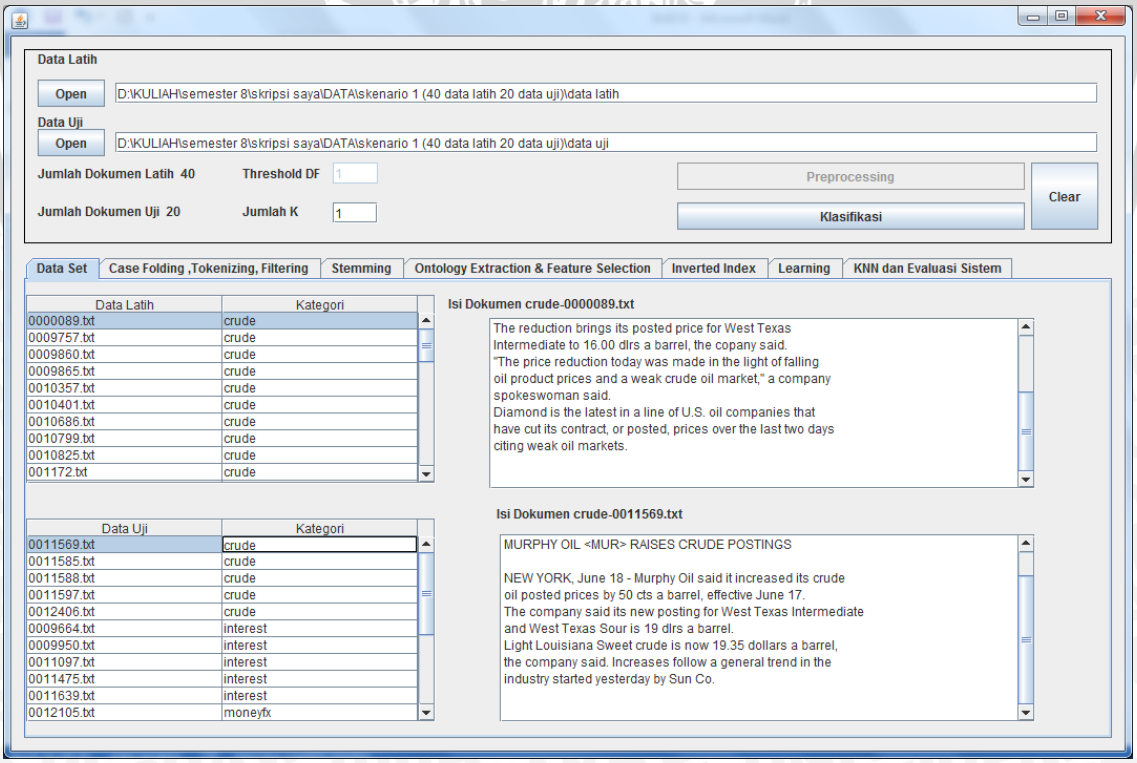
Field jumlah "k" dan *field Document Frequency Threshold* harus diisi. Selanjutnya adalah mengklik kedua tombol open untuk membuka direktori dokumen latih dan dokumen uji. Untuk dokumen latih hanya dapat memilih direktori tidak dapat memilih satu *file* dokumen sedangkan untuk dokumen uji pemilihan *single file* dapat dilakukan. Daftar dokumen latih dan dokumen uji yang terdiri dari judul dokumen, kategori dokumen beserta isi dokumen ditampilkan pada tab Data Set. Ketika judul dokumen di-klik maka *text area* disebelah tabel daftar dokumen baik dokumen latih maupun dokumen uji akan menampilkan isi berita yang di-klik sebelumnya. Jumlah dokumen latih dan jumlah dokumen uji akan ditampilkan pada *text field* disebelah *field Threshold DF* dan jumlah "k".Untuk tab data set dapat dilihat pada gambar 4.21.

Setelah mengisi *field* Threshold DF dan *field* K maka langkah selanjutnya adalah klik tombol Preprocessing. Ketika proses *preprocessing* telah selesai dijalankan maka *field* threshold DF dan tombol preprocessing akan di-set menjadi *false* untuk mencegah user men-klik tombol preprocessing dua kali. Gambar 4.22 menunjukkan antar muka setelah proses *preprocessing* selesai dijalankan.



Gambar 4.21 Tab Data Set Setelah Dipilih Dokumen Latih dan Dokumen Uji

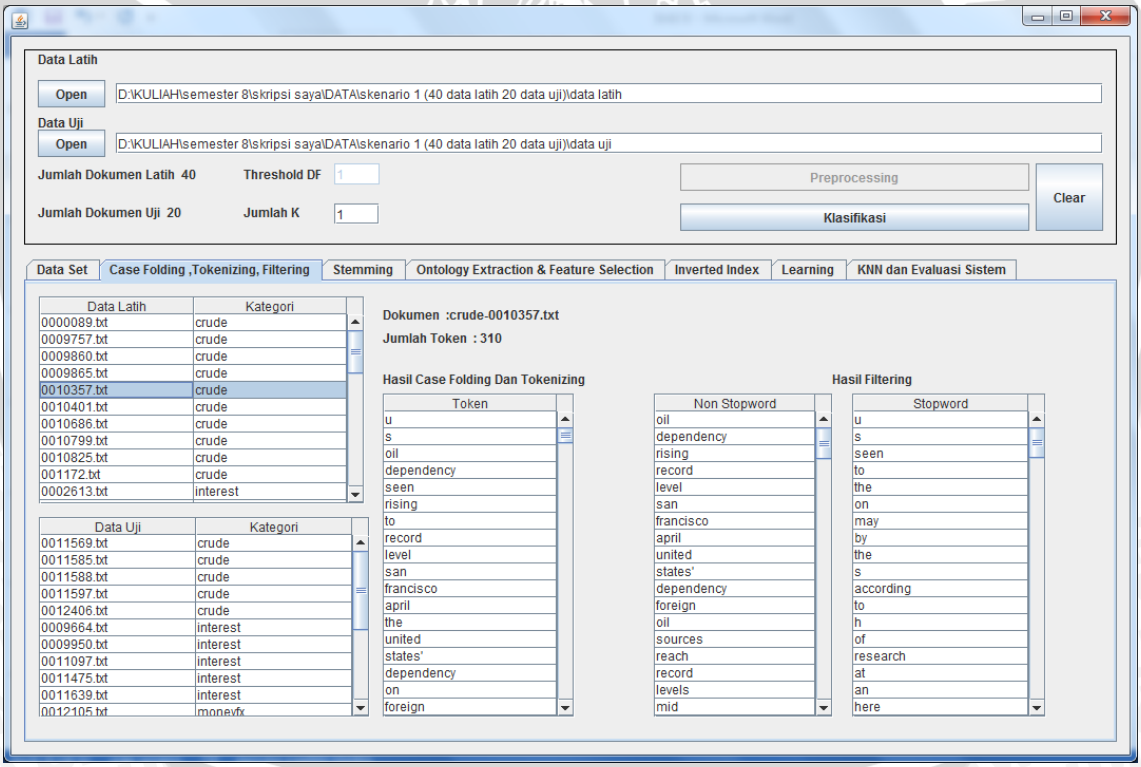
Sumber : [Implementasi]



Gambar 4.22 Tampilan Antar Muka Ketika *Preprocessing* Selesai Dijalankan

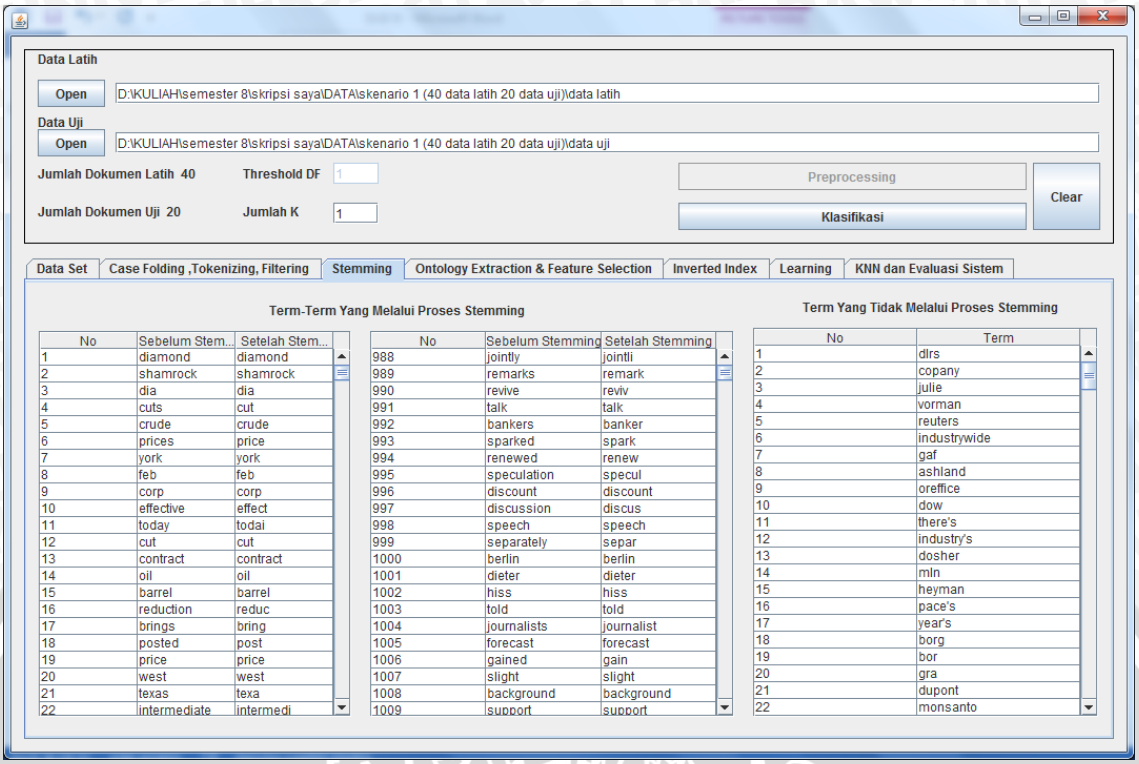
Sumber : [Implementasi]

Setelah *preprocessing* selesai dijalankan maka hasil dari setiap proses *preprocessing* akan ditampilkan pada tab *Case Folding*, *Stemming*, dan *Ontology Extraction*. Urutan tab-tab pada antar muka sistem ini menunjukkan urutan-urutan proses pada sistem.Pada tab *Case Folding, Tokenizining, Filtering* akan ditampikan hasil dari proses *Case Folding, Tokenizing, Filtering*. Daftar Dokumen akan ditampilkan pada tabel Data Latih dan Data Uji. Ketika judul dokumen di-klik maka akan muncul jumlah Token pada dokumen tersebut serta hasil proses *case folding* dan *tokenizing* akan ditampilkan pada tabel Hasil Case Folding dan Tokenizing. *Term-term* yang bukan merupakan *stopword* akan ditampilkan pada tabel Non Stopword sementara *term-term* yang merupakan *stopword* akan ditampilkan pada tabel Stopword. Tampilan antar muka pada tab *Case Folding, Tokenizing,Filtering* dapat dilihat pada gambar 4.23.



Gambar 4.23 Tampilan Antar Muka Tab *Case Folding, Tokenizing, Filtering*

Sumber : [Implementasi]



No	Sebelum Stem...	Setelah Stem...
1	diamond	diamond
2	shamrock	shamrock
3	dia	dia
4	cuts	cut
5	crude	crude
6	prices	price
7	york	york
8	feb	feb
9	corp	corp
10	effective	effect
11	today	todai
12	cut	cut
13	contract	contract
14	oil	oil
15	barrel	barrel
16	reduction	reduc
17	brings	bring
18	posted	post
19	price	price
20	west	west
21	texas	texa
22	intermediate	intermedi

No	Sebelum Stemming	Setelah Stemming
988	jointly	jointli
989	remarks	remark
990	revive	reviv
991	talk	talk
992	bankers	banker
993	sparked	spark
994	renewed	renew
995	speculation	specul
996	discount	discount
997	discussion	discus
998	speech	speech
999	separately	separ
1000	berlin	berlin
1001	dieter	dieter
1002	hiss	hiss
1003	told	told
1004	journalists	journalist
1005	forecast	forecast
1006	gained	gain
1007	slight	slight
1008	background	background
1009	support	support

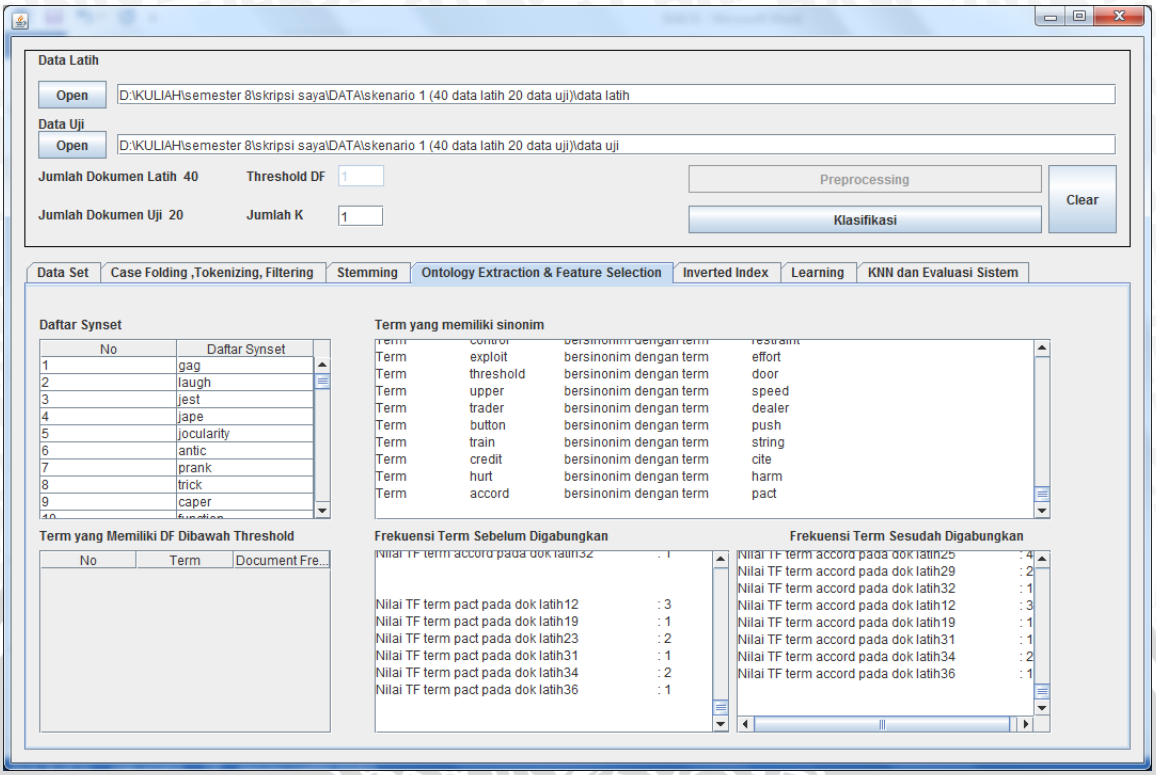
No	Term
1	dlrs
2	copany
3	julie
4	vorman
5	reuters
6	industrywide
7	gaf
8	ashland
9	oreffice
10	dow
11	there's
12	industry's
13	dosher
14	mln
15	heyman
16	pace's
17	year's
18	borg
19	bor
20	gra
21	dupont
22	monsanto

Gambar 4.24 Tampilan Antar Muka Tab *Stemming*

Sumber : [Implementasi]

Tab selanjutnya adalah tab *stemming*. Pada tab ini akan ditampilkan *term-term* yang tidak melalui proses *stemming* dan yang melalui proses *stemming*. *Term-term* yang melalui proses *stemming* akan ditampilkan bentuk *term* sebelum melalui proses *stemming* dan hasil setelah melalui proses *stemming*. Tampilan antar mukan tab *stemming* dapat dilihat pada gambar 4.24.

Tab selanjutnya adalah tab *ontology extraction*. *Synset* setiap *term* pada dokumen latih akan ditampilkan pada tabel daftar *synset* sementara *term-term* yang saling bersinonim di dalam *inverted index* akan ditampilkan pada *text area term* yang memiliki sinonim. *Term-term* yang memiliki *document frequency* di bawah *threshold* akan ditampilkan di bawah tabel daftar *synset.* Selanjutnya *text field* frekuensi *term* sebelum digabungkan akan menampilkan frekuensi *term* setiap *term* yang saling bersinonim sebelum digabungkan sementar *text area* frekuensi *term* sesudah digabungkan akan menampilkan frekuensi *term* setelah *term-term* tersebut digabungkan. Tampilan antar muka tab *ontology extraction* dapat dilihat pada gambar 4.25.

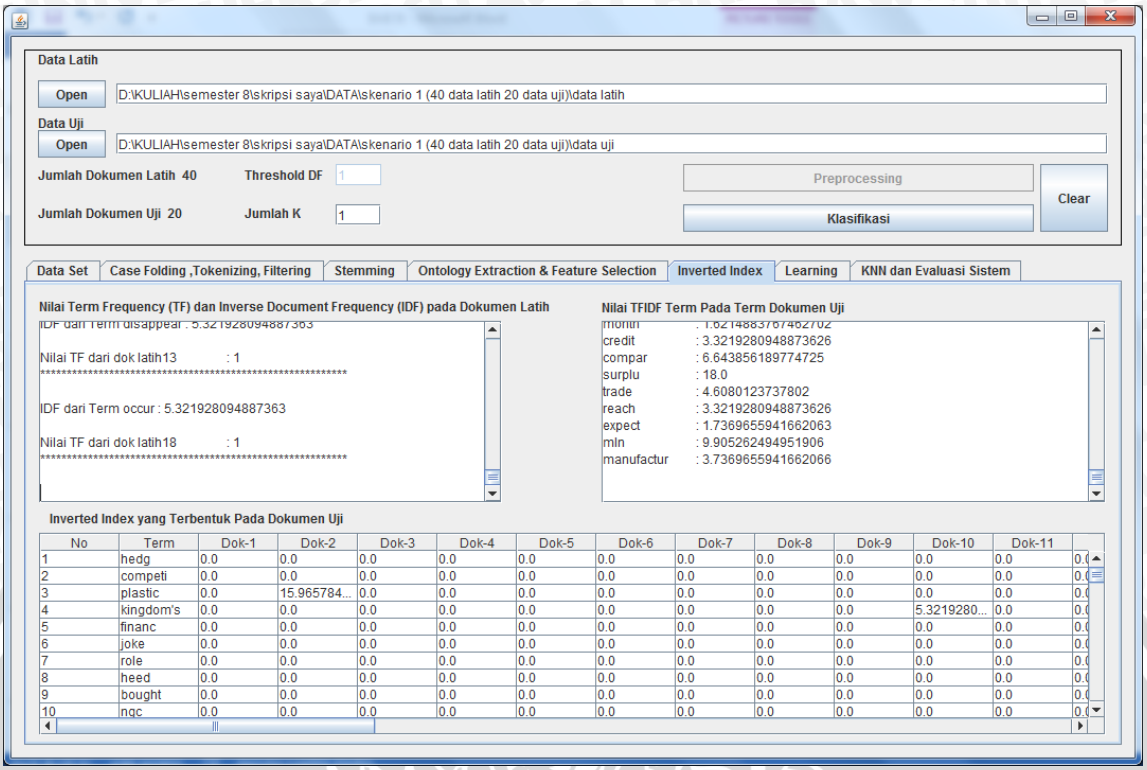


Gambar 4.25 Tampilan Antar Muka Tab *Ontology Extraction*

Sumber : [Implementasi]

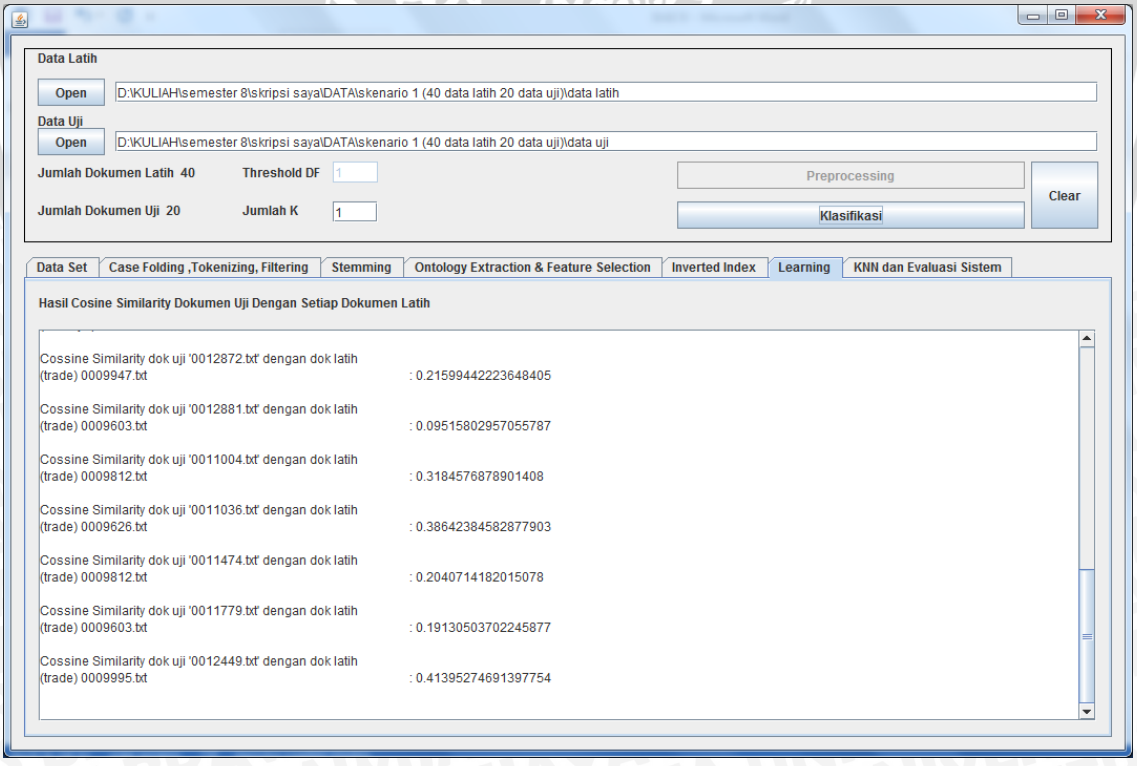
Setelah proses *ontology extraction*, maka proses selanjutnya adalah menentukan bobot dari dokumen latih dan dokumen uji yang akan ditampilkan pada tab *Inverted Index*. Bobot dari dokumen latih akan ditampilkan pada tabel *inverted index* yang terbentuk sedangkan bobot dari dokumen uji akan ditampilkan pada *text area* TFIDF *term* pada *term* dokumen uji. *Term-term* pada dokumen uji tersebut juga sebelumnya telah melalui proses *ontology extraction*. Tampilan antar muka tab *inverted index* dapat dilihat pada gambar 4.26.

Proses *preproecssing* telah selesai dijalankan dan langkah selanjutnya adalah proses *cosine similarity*. Ketika tombol klasifikasi di-klik maka pada tab learning akan tampil hasil *cosine similarity* dokumen uji dengan setiap dokumen latih. Nilai *cosine similarity* yang ditampilkan adalah nilai yang sudah diurutkan dari yang paling besar serta yang ditampiilkan hanya sejumlah “k” yang telah diinputkan sebelumnya. Tampilan antar muka tab *cosine similarity* dapat dilihat pada gambar 4.27.



Gambar 4.26 Tampilan Antar Muka Tab *Inverted Index*

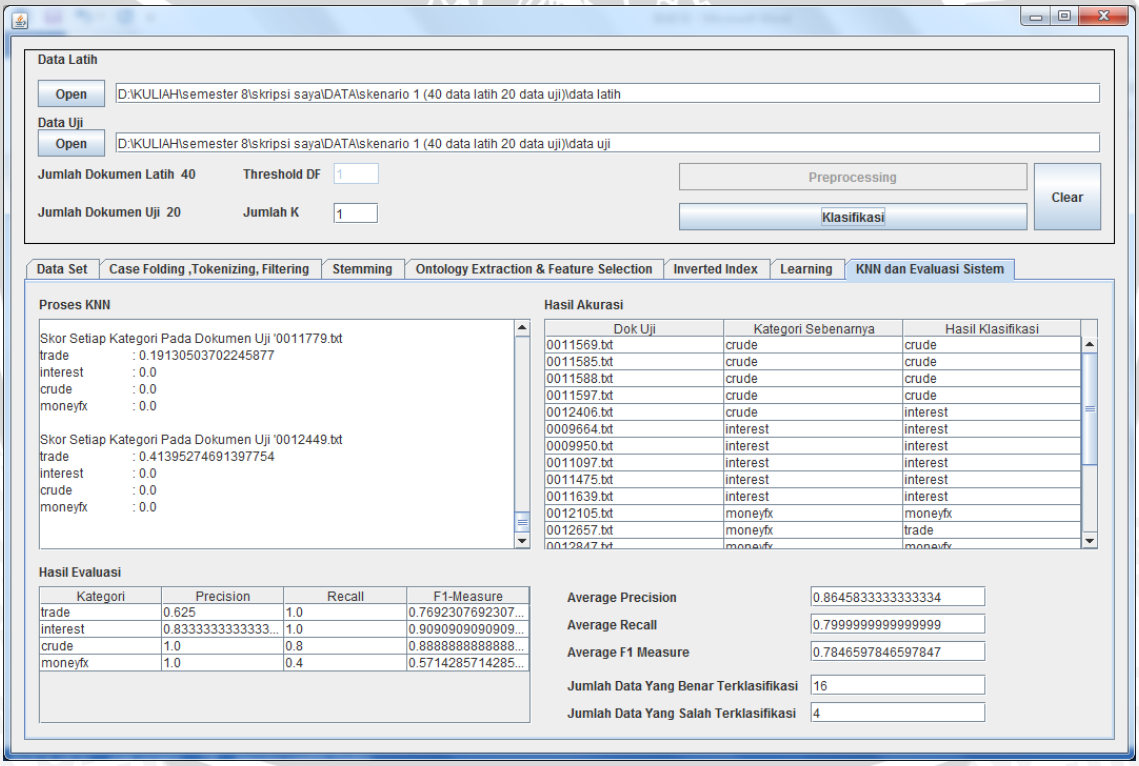
Sumber : [Implementasi]



Gambar 4.27 Tampilan Antar Muka Tab *Cosine Similarity*

Sumber : [Implementasi]

Setelah proses *cosine similarity*, langkah selanjutnya adalah menentukan kategori dari dokumen uji dengan menggunakan metode KNN yang akan ditampilkan pada tab KNN dan evaluasi sistem. Pemberian skor tiap kategori akan ditampilkan pada *text area* proses KNN. Selanjutnya kategori dari setiap dokumen uji hasil perhitungan sistem akan ditampilkan pada tabel hasil akurasi pada kolom hasil klasifikasi sementara kolom kategori sebenarnya menampilkan kategori dokumen sebenarnya. Evaluasi sistem akan ditampilkan pada tabel hasil evaluasi. *Precision, recall*, dan *F1 measure* dari setiap kategori akan ditampilkan pada tabel tersebut pada setiap skenario uji coba sementara rata-rata dari *Precision, recall*, dan *F1 measure* yang berguna untuk mengukur evaluasi sistem secara keseluruhan akan ditampilkan pada *text field* disebelah tabel hasil evaluasi. Tampilan antar muka dari tab KNN dan evaluasi dapat dilihat pada gambar 4.28.



Gambar 4.28 Tampilan Antar Muka Tab KNN dan Evaluasi

Sumber : [Implementasi]