BAB 5 IMPLEMENTASI

5.1 Spesifikasi Sistem

Dalam melakukan implementasi digunakan perangkat keras dan perangkat lunak dengan spesifikasi tertentu sehingga sistem dapat berjalan.

5.1.1 Spesifikasi Perangkat Keras

Tabel 5.1 menunjukkan komponen perangkat keras serta bagaimana spesifikasinya.

Tabel 5.1 Spesifikasi Perangkat Keras

Nama Komponen	**Spesifikasi**
Processor	*CPU AMD Quad-Core Processor* (1,50 GHz)
Memory (RAM)	*Memory 6144MB RAM*
HDD	1 TB
Display	*Display AMD Radeon™ R2 Graphics*

5.1.2 Spesifikasi Perangkat Lunak

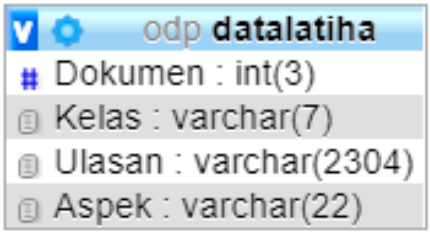
Tabel 5.2 menunjukkan komponen perangkat lunak serta bagaimana spesifikasinya.

Tabel 5.2 Spesifikasi Perangkat Lunak

Nama Komponen	**Spesifikasi**
Bahasa Pemrograman	Python 2.7
Editor	Spyder, Notepad++
DBMS	SQLite

5.2 Implementasi Data

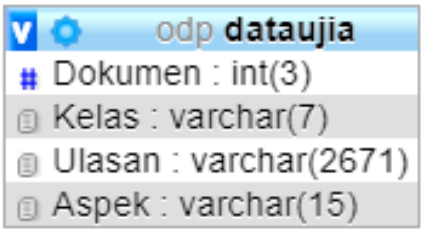
Berdasarkan perancangan sebelumnya, Gambar 5.1 menunjukkan implementasi basis data untuk data latih, Gambar 5.2 menunjukkan implementasi basis data untuk data uji, dan Gambar 5.3 menunjukkan perancangan basis data untuk hasil *POS-Tagging*.



Gambar 5.1 Implementasi Data Latih

Penjelasan singkat tentang atribut untuk data latih adalah sebagai berikut:

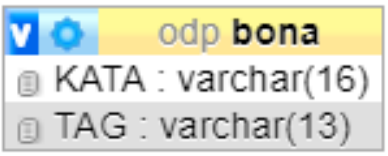
1. Dokumen, yaitu nama dokumen yang dalam kasus ini adalah nomor dokumen.
2. Kelas, yaitu kelas ulasan tersebut termasuk ke dalam kelas positif atau negatif berdasarkan pakar.
3. Ulasan, yaitu teks dokumen itu sendiri.
4. Aspek, dari hasil klasterisasi setiap dokumen akan mendapatkan aspeknya.



Gambar 5.2 Implementasi Data Uji

Penjelasan singkat tentang atribut untuk data uji adalah sebagai berikut:

1. Dokumen, yaitu nama dokumen yang dalam kasus ini adalah nomor dokumen.
2. Kelas, yaitu kelas ulasan tersebut termasuk ke dalam kelas positif atau negatif berdasarkan pakar.
3. Ulasan, yaitu teks dokumen itu sendiri.
4. Aspek, dari hasil klasterisasi setiap dokumen akan mendapatkan aspeknya.



Gambar 5.3 Implementasi Data Kelas Kata

Penjelasan singkat tentang atribut untuk data latih adalah sebagai berikut:

1. Kata, yaitu daftar kata dari keseluruhan data latih.
2. *Tag*, yaitu kelas kata yang diambil dari Kateglo.

5.3 Implementasi Sistem

5.3.1 Proses untuk menangani basis data

Untuk menangani basis data dibuat sebuah kelas khusus dengan fungsi-fungsi sesuai keperluan data yang ingin diolah.

```
class DBHelper:
    def __init__(self, dbname="dataskripsi.db",latih='dataLatihA',uji='dataUjiA',bon='BONA'):
        self.dbname = dbname
        self.latih = latih
        self.uji = uji
        self.bon= bon
        self.conn = sqlite3.connect(dbname)

    def get_nama_dokumen(self):
        stmt = "SELECT Dokumen FROM "+self.latih
        return [x[0] for x in self.conn.execute(stmt)]

    def get_nama_dokumen_uji(self):
        stmt = "SELECT Dokumen FROM "+self.uji
        return [x[0] for x in self.conn.execute(stmt)]

    def get_data_latih(self):
        stmt = "SELECT ulasan FROM "+self.latih
        return [x[0] for x in self.conn.execute(stmt)]

    def get_kelas_latih(self):
        stmt = "SELECT kelas FROM "+self.latih
        return [x[0] for x in self.conn.execute(stmt)]

    def get_data_uji(self):
        stmt = "SELECT ulasan FROM "+self.uji
        return [x[0] for x in self.conn.execute(stmt)]

    def get_kelas_uji(self):
        stmt = "SELECT kelas FROM "+self.uji
        return [x[0] for x in self.conn.execute(stmt)]

    def get_ulasan_latih(self, kelas):
        stmt = "select ulasan from {} WHERE Kelas = (?)".format(self.latih)
        args = (kelas, )
        return [x[0] for x in self.conn.execute(stmt, args)]

    def get_kata_nomina(self):
        stmt = "SELECT KATA FROM {} WHERE TAG = 'Nomina'".format(self.bon)
        return [x[0] for x in self.conn.execute(stmt)]

    def insert_tag(self, data):

        c = self.conn.cursor()
        drop_table = 'DROP TABLE IF EXISTS {}'.format(self.bon)
        c.execute(drop_table)
        create_table = 'CREATE TABLE IF NOT EXISTS {} ( `KATA` TEXT, `TAG` TEXT)'.format(self.bon)
        c.execute(create_table)
        tagidx = 'CREATE INDEX IF NOT EXISTS tagIndex ON {} (KATA ASC)'.format(self.bon)
        c.execute(tagidx)

        for row in data:
```

```
58              c.execute('INSERT INTO {} VALUES (?,?)'.format(self.bon),
59 row )
60          self.conn.commit()
61 
62      def set_aspek(self,aspek,dok):
63          c = self.conn.cursor()
64          c.execute('UPDATE {} SET Aspek = "{}" WHERE Dokumen =
65 "{}"'.format(self.latih,aspek,dok))
66          self.conn.commit()
67      def set_aspek_uji(self,aspek,dok):
68          c = self.conn.cursor()
69          c.execute('UPDATE {} SET Aspek = "{}" WHERE Dokumen =
70 "{}"'.format(self.uji,aspek,dok))
71          self.conn.commit()
```

Kode Program 5.1 Penanganan Basis Data

Penjelasan Kode Program 5.1 berdasarkan fungsi di dalamnya adalah sebagai berikut:

1. __init__ adalah untuk inisialisasi awal variable-bariabel yang akan dibutuhkan selanjutnya seperti nama basis data kemudian membangun koneksi ke basis data tersebut.
2. get_nama_dokumen adalah untuk mengambil seluruh nama dokumen atau dalam kasus ini merupakan nomor dokumen yang menjadi identitasnya pada data latih.
3. get_nama_dokumen_uji adalah untuk mengambil seluruh nama dokumen atau dalam kasus ini merupakan nomor dokumen yang menjadi identitasnya pada data uji.
4. get_data_latih adalah untuk mengambil seluruh ulasan pada data latih.
5. get_kelas_latih adalah untuk mengambil seluruh kelas untuk setiap ulasan pada data latih.
6. get_data_uji adalah untuk mengambil seluruh ulasan pada data uji.
7. get_kelas_uji adalah untuk mengambil seluruh kelas untuk setiap ulasan pada data uji.
8. get_ulasan_latih adalah untuk mengambil ulasan pada data latih yang memiliki kelas sesuai parameter (positif atau negatif).
9. get_kata_nomina adalah untuk mengambil kata yang memiliki kelas kata nomina.
10. insert_tag adalah untuk membuat data berupa kata serta kelas katanya.
11. set_aspek adalah untuk menambahkan data aspek kepada ulasan data latih.
12. set_aspek adalah untuk menambahkan data aspek kepada ulasan data uji.

5.3.2 Proses *Preprocessing* Data

Untuk melakukan klasterisasi diperlukan *Bag of Nouns.* Untuk itu pada proses ini akan dilakukan *preprocessing* hingga *pos-tagging* sehingga nantinya dapat disaring mana yang termasuk kata benda atau nomina.

```
1   def pos_tagging():
2       stopword = open('id.stopwords.02.01.2016.txt','r')
3       isistopword = stopword.read()
4       stopword.close()
5       sw = re.split('\n', isistopword)
6
7       list_data = db.get_data_latih()
8       list_cf = []
9       for each in list_data:
10          list_cf.append(each.lower())
11
12      list_f = []
13      for x in list_cf:
14          teks = x
15          for y in sw:
16              teks = re.sub(r'\b{}\b'.format(y), ' ', teks)
17          list_f.append(teks)
18
19      list_f2 = []
20      for x in list_f:
21          list_f2.append(re.sub(r'[^A-Za-z]+', ' ', x))
22
23      list_t = []
24      for each in list_f2:
25          list_t.append(re.split(' ', each))
26
27      list_word = []
28      for x in list_t:
29          for y in x:
30              if y and len(y) > 2:
31                  list_word.append(y)
32
33      set_word = set(list_word)
34      list_word = list(set_word)
35      tag_word = []
36      non_tag = []
37      for each in list_word:
38          try:
39              tag = cek_tag(each)
40              tag_word.append([each, tag])
41          except Exception as e:
42              print(e)
43              print("\n----\n")
44              print each
45              non_tag.append([each])
46          time.sleep(2)
47      db.insert_tag(tag_word)
48  def cek_tag(kata):
49      response                                                    =
50  requests.get('http://kateglo.com/api.php?format=json&phrase='+kata)
51      content = response.content.decode("utf8")
52      js = json.loads(content)
53      return js['kateglo']['lex_class_name']
```

Kode Program 5.2 Proses *Preprocessing*

Penjelasan Kode Program 5.2 adalah sebagai berikut:

1. Baris 2-5 adalah pembacaan dan penyimpanan daftar kata pada *stopwords*

ke dalam variable local.

2. Baris 9-10 adalah *case folding,* membuat semua huruf menjadi huruf kecil
3. Baris 13-17 adalah penghilangan kata yang termasuk *stopwords*
4. Baris 20-21 adalah untuk menghilangkan karakter yang bukan alfabet
5. Baris 24-25 adalah untuk tokenisasi dengan cara membagi suatu *string* berdasarkan karakter spasi.
6. Baris 28-31 adalah untuk mengumpulkan seluruh kata kedalam satu *list* dengan syarat kata tersebut memiliki Panjang karakter lebih dari dua karakter.
7. Baris 33-34 adalah untuk melakukan penghilangan kata atau token yang duplikat.
8. Baris 37-47 adalah untuk melakukan proses penentuan kelas kata untuk setiap kata dengan memanggil fungsi cek_tag() kemudian memasukkannya ke dalam basis data.
9. Baris 48-53 adalah untuk melakukan pemeriksaan terhadap suatu kata melalui API yang disediakan oleh Kateglo kemudian diambil data kelas katanya.

5.3.3 Proses Klasterisasi dengan K-Modes

5.3.3.1 Proses Klasterisasi Data Latih

Dimulai dengan pembentukan dimensi daftar dokumen dan daftar kata nomina kemdian dilakukan klasterisasi hingga diberikan sekumpulan kata yang merepresentasikan setiap klaster.

```
1
2   for x in list_bon_real:
3       c = [x for dok in list_data if re.search(r'\b{}\b'.format(x),
4   dok.lower())]
5       if len(c) > 2:
6           list_bon.append(x)
7
8   n_klaster = 6
9   vek_dok = []
10  jumlah_nomina = []
11  for x in range(len(list_data)):
12      vek_kata = []
13      for y in list_bon:
14          if re.search(r'\b{}\b'.format(y), list_data[x].lower()):
15              vek_kata.append(1)
16          else:
17              vek_kata.append(0)
18      vek_dok.append([vek_kata, str(list_nama_dokumen[x])])
19      jumlah_nomina.append(sum(vek_kata))
20
21  term_freq = {}
22  for x in range(len(list_data)):
23      term_doc = []
24      for y in list_bon:
25          term_occ = len(re.findall(r'\b{}\b'.format(y), list_data[x]
26  .lower()))
```

```
        term_doc.append(term_occ)
    term_freq[str(list_nama_dokumen[x])] = term_doc

centroid = []
centroid.append(vek_dok[2][0])
centroid.append(vek_dok[6][0])
centroid.append(vek_dok[9][0])
centroid.append(vek_dok[104][0])
centroid.append(vek_dok[51][0])
centroid.append(vek_dok[65][0])

cek_k = []

for iterasi in range(10):
    centroid_value =[]
    for each in vek_dok:
        sum_x = [None] * n_klaster
        for x in range(len(sum_x)):
            sum_x[x] = []
        temp_centroid = []
        hd = [None] * n_klaster
        for indx in range(len(each[0])):
            for indy in range(n_klaster):

                if   each[0][indx]   ==   centroid[indy][indx]   or
centroid[indy][indx] == 2:
                    sum_x[indy].append(0)
                else:
                    sum_x[indy].append(1)
        for indz in range(len(hd)):
            hd[indz] = sum(sum_x[indz])
        centroid_value.append(hd)

    result_cluster = []
    for each in centroid_value:
        closest = min(each)
        result_cluster.append(each.index(closest)+1)

    k = [None] * n_klaster
    for indx in range(len(k)):
        k[indx] = []
    for x in range(len(result_cluster)):
        for y in range(n_klaster):
            if result_cluster[x] == y+1:
                k[y].append(vek_dok[x])

    for pjn in range(len(k)):
        print  'Anggota  klaster  {}  sebanyak  {}.'.format(pjn+1,
len(k[pjn]))
    for kx in range(n_klaster):
        temp_new_centroid = []
        for x in range(len(k[kx][0][0])):
            mode_temp = []
            for y in k[kx]:
                mode_temp.append(y[0][x])
            mode_data = Counter(mode_temp)
            if len(mode_data.most_common()) > 1:
                if         mode_data.most_common()[0][1]         !=
mode_data.most_common()[1][1]:
                    mode_val = mode_data.most_common()[0][0]
                    temp_new_centroid.append(mode_val)
                elif        mode_data.most_common()[0][1]        ==
mode_data.most_common()[1][1]:
                    temp_new_centroid.append(2)
            else:
```

```
92                  mode_val = mode_data.most_common()[0][0]
93                  temp_new_centroid.append(mode_val)
94          centroid[kx] = temp_new_centroid
95
96      if cek_k == k:
97          f = open('centroid.txt', 'w')
98          for each in centroid:
99              if each == centroid[len(centroid)-1]:
100                 f.write('{}'.format(each))
101             else:
102                 f.write('{},'.format(each))
103         f.close()
104         break
105     elif cek_k != k:
106         cek_k = k
107         continue
108
109 term_freq_sum = []
110 for i in range(n_klaster):
111     term_freq_temp = []
112     for x in k[i]:
113         term_freq_temp.append([x[1], term_freq[x[1]]])
114
115     term_freq_sum.append(term_freq_temp)
116
117 s1 = []
118 for i in range(n_klaster):
119     s2 = []
120     for x in range(len(term_freq_sum[i][0][1])):
121         sum_temp = []
122         for y in term_freq_sum[i]:
123             sum_temp.append(y[1][x])
124         s2.append(sum(sum_temp))
125     s1.append(s2)
126
127 list_aspek = []
128 for x in range(len(s1)):
129     temp_list = {}
130     for y in range(len(s1[x])):
131         temp_list[list_bon[y]] = s1[x][y]
132     list_aspek.append(sorted(temp_list,   key=temp_list.__getitem__,
133 reverse=True))
134
135 term_is = []
136 f = open('aspek.txt','w')
137 for i in range(n_klaster):
138     w1 = list_aspek[i][0]
139     w2 = list_aspek[i][1]
140     w3 = list_aspek[i][2]
141     term_is.append('{}-{}-{}'.format(w1,w2,w3))
142     if i == n_klaster-1:
143         f.write('{}'.format(term_is[i]))
144     else:
145         f.write('{}\n'.format(term_is[i]))
146 f.close()
147 for i in range(n_klaster):
148     for each in k[i]:
149         dok = each[1]
150         aspek = term_is[i]
151         db.set_aspek(aspek,dok)
```

Kode Program 5.3 Proses Klasterisasi Data Latih

Penjelasan Kode Program 5.3 adalah sebagai berikut:

1. Baris 2-6 adalah untuk membentuk daftar kata nomina yang kemunculannya lebih dari dua kali untuk keseluruhan dokumen.
2. Baris 11-18 adalah untuk membentuk vektor dokumen dengan nilai 1 jika suatu kata terdapat dalam dokumen tersebut dan bernilai 0 jika tidak terdapat.
3. Baris 22-28 adalah untuk membentuk vector dokumen dengan nilai kemunculan suatu kata pada dokumen tersebut. Hal ini akan digunakan nantinya saat klaster sudah terbentuk.
4. Baris 30-36 adalah untuk penentuan *centroid* awal.
5. Baris 51-55 adalah untuk pemeriksaan jika nilai suatu atribut pada dokumen dengan *centroid* sama maka akan diberikan nilai 0 dan jika berbeda akan diberikan nilai 1.
6. Baris 56-58 adalah untuk mendapatkan jarak antara dokumen dengan *centroid* dengan cara menjumlah berapa atribut yang tidak cocok kemudian semua jarak untuk semua *centroid* dihimpun.
7. Baris 61-63 adalah untuk penentuan klater suatu dokumen dengan melihat kepada jarak terdekat atau nilai jarak terkecil.
8. Baris 65-71 membuat sebuah daftar klaster dengan anggotanya dokumen yang mana saja.
9. Baris 76-94 adalah untuk menentukan *centroid* baru dengan cara melihat modus untuk suatu atribut berdasarkan dokumen-dokumen yang ada dalam satu klaster.
10. Baris 96-100 adalah untuk pemeriksaan apakah hasil klaster pada iterasi tertentu sama dengan hasil klaster sebelumnya, jika iya maka iterasi dihentikan, jika tidak iterasi dilajutkan hingga batas maksimal iterasi.
11. Baris 109-115 adalah untuk pembentukan vektor dokumen dengan *term frequency* berdasarkan klasternya.
12. Baris 117-125 adalah untuk melihat total kemunculan setiap kata untuk setiap klaster.
13. Baris 127-133 adalah melakukan pengurutan jumlah kemunculan kata pada setiap klaster.
14. Baris 136-141 adalah untuk mengetahui tiga kata dengan kemunculan paling banyak pada setiap klaster.
15. Baris 147-151 adalah untuk memasukkan aspek pada setiap dokumen ke dalam basis data.

5.3.3.2 Proses Pengujian Klasterisasi

Proses ini melakukan pengujian seberapa tepat suatu dokumen dimasukkan ke dalam klasternya dengan menggunakan *Silhoutte Coefficient*.

```
1   def silhoutte_coefficient(n_klaster, k):
2       avg_k = [None] * n_klaster
3       for each in range(len(k)):
4           avg_k[each] = [None] * len(k[each])
5           for each2 in range(len(k[each])):
6               avg_k[each][each2] = [None] * n_klaster
7       for i in range(n_klaster):
8           for x in range(len(k[i])):
9               for ii in range(n_klaster):
10                  sum_k = []
11                  for y in range(len(k[ii])):
12                      sum_d = []
13                      for z in range(len(k[i][x][0])):
14                          if i == ii and x == y:
15                              pass
16                          else:
17                              if k[i][x][0][z] == k[ii][y][0][z]:
18                                  sum_d.append(0)
19                              else:
20                                  sum_d.append(1)
21                      sum_k.append(sum(sum_d))
22                  avg_k[i][x][ii] = sum(sum_k) / len(sum_k)
23      f = open('silhoutte coefficient.csv', 'w')
24      sfc = [None] * n_klaster
25      for i in range(len(avg_k)):
26          sfc[i] = []
27          for ii in range(len(avg_k[i])):
28              a = avg_k[i][ii].pop(i)
29              b = min(avg_k[i][ii])
30              sf = (b-a)/max(a,b)
31              sfc[i].append(float(sf))
32              f.write('{},{},{}\n'.format(i, ii, sf))
33      f.close()
34      sfc_c = []
35      for i in range(len(sfc)):
36          avg = sum(sfc[i]) / len(sfc[i])
37          sfc_c.append(avg)
38
39      sfc_all = []
40      for x in sfc:
41          for y in x:
42              sfc_all.append(y)
43      avg_sfc = sum(sfc_all) / len(sfc_all)
44      return sfc_c, avg_sfc
```

Kode Program 5.4 Proses Pengujian Klasterisasi

Penjelasan Kode Program 5.4 adalah sebagai berikut:

1. Baris 14 adalah untuk melakukan pemeriksaan bahwa dokumen yang sedang diuji tidak menghitung jarak dengan dokumen itu sendiri.
2. Baris 17-21 adalah proses penghitungan jarak antar dokumen baik dalam satu klaster maupun dengan anggota klaster yang lain.
3. Baris 22 adalah nilai rata-rata dari jarak suatu dokumen dengan dokumen lain pada suatu klaster.

4. Baris 25-32 adalah mengambil nilai untuk variabel a yaitu rata-rata jarak dari suatu dokumen dengan anggota lain pada klaster dokumen tersebut dan untuk variabel b yaitu nilai minimum dari rata-rata jarak suatu dokumen dengan anggota klaster lain. Dengan kedua variabel ini maka dapat dihitung nilai *Silhoutte Coefficient* untuk setiap dokumen.
5. Baris 35-37 adalah untuk mengetahui nilai *Silhoutte Coefficient* per klaster.
6. Baris 39-43 adalah untuk mengetahui nilai rata-rata *Silhoutte Coefficient* secara keseluruhan.

5.3.3.3 Proses Klasterisasi Data Uji

Proses untuk mengetauhi aspek suatu data uji dengan menempatkan data tersebut terlebih dahulu ke dalam klaster yang sesuai.

```
1  def get_aspek (inp, tipe, nama_dok = 0):
2      f =  open('centroid.txt', 'r')
3      cntrd = f.read()
4      f.close()
5      centroid = cntrd.split('-')
6      f = open('aspek.txt', 'r')
7      aspk = f.read()
8      f.close()
9      aspek = aspk.split('\n')
10
11     list_bon_real = db.get_kata_nomina()
12     list_bon = []
13     list_data = db.get_data_latih()
14
15     for x in list_bon_real:
16         c = [x for dok in list_data if re.search(r'\b{}\b'.format(x),
17 dok.lower())]
18         if len(c) > 2:
19             list_bon.append(x)
20
21     vek_kata = []
22     for y in list_bon:
23         if re.search(r'\b{}\b'.format(y), inp.lower()):
24             vek_kata.append(1)
25         else:
26             vek_kata.append(0)
27     vek_dok = [vek_kata, str(nama_dok)]
28
29     n_klaster = len(centroid)
30
31
32
33     sum_x = [None] * n_klaster
34     for x in range(len(sum_x)):
35         sum_x[x] = []
36
37     hd = [None] * n_klaster
38     for indx in range(len(vek_dok[0])):
39         for indy in range(n_klaster):
40             if   vek_dok[0][indx]   ==   centroid[indy][indx]   or
41 centroid[indy][indx] == 2:
42                 sum_x[indy].append(0)
43             else:
44                 sum_x[indy].append(1)
45     for indz in range(len(hd)):
46         hd[indz] = sum(sum_x[indz])
```

```
47      centroid_value = hd
48      closest = min(centroid_value)
49      result_cluster = centroid_value.index(closest)+1
50      return aspek[result_cluster]
```

Kode Program 5.5 Proses Klasterisasi Data Uji

Penjelasan Kode Program 5.5 adalah sebagai berikut:

1. Baris 2-9 adalah untuk mengambil nilai *centroid* dan aspek yang telah menjadi representasi dari suatu klaster.
2. Baris 15-27 adalah pembentukan vektor dokumen berdasarkan kemunculan kata pada dokumen tersebut.
3. Baris 33-47 adalah penghitungan jarak dokumen dengan setiap klaster.
4. Baris 48-49 adalah penentuan dokumen tersebut masuk ke klaster mana.
5. Baris 50 adalah untuk mengetahui aspek yang relevan terhadap dokumen tersebut.

5.3.4 Proses Hitung Skor Ulasan

Untuk melakukan ekstraksi fitur berupa skor ulasan, sebelumnya telah dilakukan proses preprocessing hingga didapat sekumpulan dokumen dengan kata-kata unik yang ada di dalamnya.

```
1   w = []
2   for x in range(len(list_t)):
3       term_doc = []
4       for y in list_t[x]:
5           if y:
6               term_occ      =      len(re.findall(r'\b{}\b'.format(y),
7   list_f2[x]))
8               term_doc.append(term_occ)
9       w.append(term_doc)
10
11  s_term_pos = []
12  s_term_neg = []
13  for x in range(len(list_t)):
14      term_doc_pos = []
15      term_doc_neg = []
16      for y in list_t[x]:
17          if y:
18              try:
19
20                  term_pos = len([y for dok in list_dok_positif if
21  re.search(r'\b{}\b'.format(y), dok.lower())])
22                  term_neg = len([y for dok in list_dok_negatif if
23  re.search(r'\b{}\b'.format(y), dok.lower())])
24                  if term_pos == 0 and term_neg == 0:
25                      print '{},{}'.format(y,x)
26
27                  term_doc_pos.append(term_pos/(term_pos+term_neg))
28                  term_doc_neg.append(term_neg/(term_pos+term_neg))
29              except Exception as e:
30                  print e
31      s_term_pos.append(term_doc_pos)
32      s_term_neg.append(term_doc_neg)
33
34  s_doc = []
35  f = open('score_representations.csv', 'w')
```

```
36  for x in range(len(w)):
37      s_doc_pos = 0
38      s_doc_neg = 0
39      for y in range(len(w[x])):
40          s_doc_pos += w[x][y] * s_term_pos[x][y]
41          s_doc_neg += w[x][y] * s_term_neg[x][y]
42      s_doc.append([s_doc_pos, s_doc_neg, list_kelas[x]])
43      if x == len(w)-1:
44          f.write('{},{},{}'.format(s_doc_pos,              s_doc_neg,
45  list_kelas[x]))
46      else:
47          f.write('{},{},{}\n'.format(s_doc_pos,            s_doc_neg,
48  list_kelas[x]))
49  f.close()
```

Kode Program 5.6 Proses Hitung Skor Ulasan

Penjelasan Kode Program 5.6 adalah sebagai berikut:

1. Baris 1-9 adalah untuk menghitung bobot kata pada setiap dokumen dengan melihat kepada jumlah kemunculannya.
2. Baris 20-25 adalah untuk mengetahui suatu kata terdapat di dalam berapa dokumen positif dan berapa dokumen negatif.
3. Baris 27-32 adalah pembentukan vektor skor kata positif dan negatif.
4. Baris 36-48 adalah penghitungan skor dokumen untuk positif dan negatif.

5.3.5 Proses Klasifikasi dengan LVQ

5.3.5.1 Proses Pelatihan dengan LVQ

Dari hasil merepresentaikan dokumen dalam bentuk skor kemudian dilakukan pelatihan dengan metode LVQ 2 hingga diambil bobot terakhir sebagai bobot ideal untuk klasifikasi.

```
1           alpha = 0.5
2           epsilon = 0.5
3           decalpha= 0.5
4           s_r_doc = open('score_representations.csv','r')
5           s_r = s_r_doc.read()
6           s_r_doc.close()
7
8           s = s_r.split('\n')
9           for x in range(len(s)):
10              s[x] = s[x].split(',')
11
12          for each in s:
13              if len(each) > 1:
14                  if each[2] == 'positif':
15                      each[2] = 1
16                  elif each[2] == 'negatif':
17                      each[2] = 2
18              else:
19                  s.remove(each)
20          s_doc = [[float(y) for y in s[x]] for x in range(len(s))]
21          w0_pos = next((x for x in s_doc if x[2] == 1))
22          w0_neg = next((x for x in s_doc if x[2] == 2))
23          s_doc.remove(w0_pos)
24          s_doc.remove(w0_neg)
```

```
25
26          for epoch in range(maxepoch):
27              comp = []
28              d = []
29              match = 0
30              notmatch = 0
31              for indx in range(len(s_doc)):
32
33                  d1     =     math.sqrt(((float(s_doc[indx][0])     -
34  float(w0_pos[0]))**2)+((float(s_doc[indx][1])                        -
35  float(w0_pos[1]))**2))
36                  d2     =     math.sqrt(((float(s_doc[indx][0])     -
37  float(w0_neg[0]))**2)+((float(s_doc[indx][1])                        -
38  float(w0_neg[1]))**2))
39
40                  min_d = min(d1,d2)
41                  if min_d == d1:
42                      c = [1, d1]
43                      r = [2, d2]
44                  elif min_d == d2:
45                      c = [2, d2]
46                      r = [1, d1]
47                  d.append([indx,c[0],r[0]])
48                  if d[indx][1] == s_doc[indx][2]:
49                      match+=1
50                  elif d[indx][1] != s_doc[indx][2]:
51                      notmatch+=1
52
53                  comp.append([s_doc[indx][2],c[0]])
54
55                  if c[1]/r[1] > (1 - epsilon) and r[1]/c[1] < (1 +
56  epsilon):
57                      window = 1
58                  else:
59                      window = 0
60
61                  if window == 1 and r[0] == s_doc[indx][2]:
62                      if c[0] == 1:
63                          w0_pos[0]    =    w0_pos[0]    -    alpha    *
64  (s_doc[indx][0] - w0_pos[0])
65                      elif r[0] == 1:
66                          w0_pos[0]    =    w0_pos[0]    +    alpha    *
67  (s_doc[indx][0] - w0_pos[0])
68                      else:
69                          w0_pos[0] = w0_pos[0]
70                  else:
71                      if c[0] == 1:
72                          if c[0] == s_doc[indx][2]:
73                              w0_pos[0]   =   w0_pos[0]   +   alpha   *
74  (s_doc[indx][0] - w0_pos[0])
75                          else:
76                              w0_pos[0]   =   w0_pos[0]   -   alpha   *
77  (s_doc[indx][0] - w0_pos[0])
78                      else:
79                          w0_pos[0] = w0_pos[0]
80
81
82                  if window == 1 and r[0] == s_doc[indx][2]:
83                      if c[0] == 1:
84                          w0_pos[1]    =    w0_pos[1]    -    alpha    *
85  (s_doc[indx][1] - w0_pos[1])
86                      elif r[0] == 1:
87                          w0_pos[1]    =    w0_pos[1]    +    alpha    *
88  (s_doc[indx][1] - w0_pos[1])
89                      else:
```

```
90                          w0_pos[1] = w0_pos[1]
91                  else:
92                      if c[0] == 1:
93                          if c[0] == s_doc[indx][2]:
94                              w0_pos[1]   =   w0_pos[1]   +   alpha   *
95  (s_doc[indx][1] - w0_pos[1])
96                          else:
97                              w0_pos[1]   =   w0_pos[1]   -   alpha   *
98  (s_doc[indx][1] - w0_pos[1])
99                      else:
100                         w0_pos[1] = w0_pos[1]
101
102
103                 if window == 1 and r[0] == s_doc[indx][2]:
104                     if c[0] == 2:
105                         w0_neg[0]    =    w0_neg[0]    -    alpha    *
106 (s_doc[indx][0] - w0_neg[0])
107                     elif r[0] == 2:
108                         w0_neg[0]    =    w0_neg[0]    +    alpha    *
109 (s_doc[indx][0] - w0_neg[0])
110                     else:
111                         w0_neg[0] = w0_neg[0]
112                 else:
113                     if c[0] == 2:
114                         if c[0] == s_doc[indx][2]:
115                             w0_neg[0]   =   w0_neg[0]   +   alpha   *
116 (s_doc[indx][0] - w0_neg[0])
117                         else:
118                             w0_neg[0]   =   w0_neg[0]   -   alpha   *
119 (s_doc[indx][0] - w0_neg[0])
120                     else:
121                         w0_neg[0] = w0_neg[0]
122
123
124                 if window == 1 and r[0] == s_doc[indx][2]:
125                     if c[0] == 2:
126                         w0_neg[1]    =    w0_neg[1]    -    alpha    *
127 (s_doc[indx][1] - w0_neg[1])
128                     elif r[0] == 2:
129                         w0_neg[1]    =    w0_neg[1]    +    alpha    *
130 (s_doc[indx][1] - w0_neg[1])
131                     else:
132                         w0_neg[1] = w0_neg[1]
133                 else:
134                     if c[0] == 2:
135                         if c[0] == s_doc[indx][2]:
136                             w0_neg[1]   =   w0_neg[1]   +   alpha   *
137 (s_doc[indx][1] - w0_neg[1])
138                         else:
139                             w0_neg[1]   =   w0_neg[1]   -   alpha   *
140 (s_doc[indx][1] - w0_neg[1])
141                     else:
142                         w0_neg[1] = w0_neg[1]
143             alpha = alpha * decalpha
144
145
```

Kode Program 5.7 Proses Pelatihan dengan LVQ

Penjelasan Kode Program 5.7 adalah sebagai berikut:

1. Baris 21-22 adalah untuk inisialisasi bobot awal.

2. Baris 34-39 adalah penghitungan jarak antara bobot dengan masukan.

3. Baris 41-48 adalah penentuan kelas untuk masukan. Bernilai 1 berarti positif dan 2 berarti negatif.
4. Baris 56-60 adalah untuk mengetahui apakah masukan tersebut berada dalam *window* atau tidak.
5. Baris 62-102 adalah proses untuk melakukan pembaharuan terhadap bobot positif.
6. Baris 105-144 adalah proses untuk melakukan pembaharuan terhadap bobot negatif.

5.3.5.2 Proses Pengujian Klasifikasi

Untuk persiapan melakukan pengujian, telah dilakukan *filtering* untuk karakter selain alfabet untuk kemudian dilakukan pengujian.

```
1           d = []
2           comp = []
3           match = 0
4           notmatch =  0
5           for each in list_data_uji:
6               c, r = klasifikasi(each)
7               indx = list_data_uji.index(each)
8               t = list_kelas_uji[list_data_uji.index(each)]
9               if t == 'positif':
10                  t = 1
11              elif t == 'negatif':
12                  t = 2
13              d.append([indx,c,r])
14
15              if d[indx][1] == t:
16                  match+=1
17              elif d[indx][1] != t:
18                  notmatch+=1
19              comp.append([t,c])
20
21          pp = 0
22          pn = 0
23          np = 0
24          nn = 0
25          for each in comp:
26              if each[0] == 1:
27                  if each[1] == 1:
28                      pp += 1
29                  elif each[1] == 2:
30                      pn += 1
31              if each[0] == 2:
32                  if each[1] == 1:
33                      np += 1
34                  elif each[1] == 2:
35                      nn += 1
36
37          precision_pos = pp / (pp + np)
38          recall_pos = pp / (pp + pn)
39          f1_pos = 2 * precision_pos * recall_pos / (precision_pos +
40  recall_pos)
41          precision_neg = nn / (nn + np)
42          recall_neg = nn / (nn + pn)
43          f1_neg = 2 * precision_neg * recall_neg / (precision_neg +
44  recall_neg)
```

Kode Program 5.8 Proses Pengujian Klasifikasi

Penjelasan Kode Program 5.8 adalah sebagai berikut:

1. Baris 6 adalah untuk mengetahui hasil klasifikasi suatu masukan termasuk kelas mana.
2. Baris 15-18 adalah untuk melakukan pemeriksaan berapa hasil klasifikasi yang sesuai dengan *ground truth* dan berapa yang tidak sesuai.
3. Baris 25-35 adalah untuk pembentukan *confussion matrix* untuk positif da negatif.
4. Baris 37-44 adalah untuk penghitungan *precision, recall,* dan *f1-score.*

5.3.5.3 Proses Klasifikasi

Proses ini adalah mengklasifikasikan suatu ulasan ke dalam kelas positif atau negatif.

```
1   def get_sentimen (inp):
2               inp = inp.lower()
3               for y in sw:
4                   inp = re.sub(r'\b{}\b'.format(y), ' ', inp)
5               inp = re.sub('[^A-Za-z]+', ' ', inp)
6               inp2 = inp.split()
7               inp2 = list(set(inp2))
8               term_doc = []
9               for word in inp2:
10                  if word:
11                      term_occ                                        =
12  len(re.findall(r'\b{}\b'.format(word), inp))
13                      term_doc.append(term_occ)
14              w = term_doc
15              term_doc_pos = []
16              term_doc_neg = []
17              for word in inp2:
18                  if word:
19                      try:
20                          term_pos    =    len([word    for    dok    in
21  list_dok_positif if re.search(r'\b{}\b'.format(word), dok.lower())])
22                          term_neg    =    len([word    for    dok    in
23  list_dok_negatif if re.search(r'\b{}\b'.format(word), dok.lower())])
24                          if term_pos == 0 and term_neg == 0:
25                              term_doc_pos.append(0)
26                              term_doc_neg.append(0)
27                          else:
28
29  term_doc_pos.append(term_pos/(term_pos+term_neg))
30
31  term_doc_neg.append(term_neg/(term_pos+term_neg))
32                      except Exception as e:
33                          print e
34              s_term_pos = term_doc_pos
35              s_term_neg = term_doc_neg
36              s_doc_pos = 0
37              s_doc_neg = 0
38              for y in range(len(w)):
39                  s_doc_pos += w[y] * s_term_pos[y]
40                  s_doc_neg += w[y] * s_term_neg[y]
41              s_doc= [s_doc_pos, s_doc_neg]
42
43              s_doc = [float(y) for y in s_doc]
44              d1        =        math.sqrt(((float(s_doc[0])        -
45  float(w0_pos[0]))**2)+((float(s_doc[1]) - float(w0_pos[1]))**2))
```

```
46              d2          =          math.sqrt(((float(s_doc[0])          -
47  float(w0_neg[0]))**2)+((float(s_doc[1]) - float(w0_neg[1]))**2))
48
49              min_d = min(d1,d2)
50              if min_d == d1:
51                  c = 1
52                  r = 2
53              elif min_d == d2:
54                  c = 2
55                  r = 1
56              return  c, r
```

Kode Program 5.9 Proses Klasifikasi

Penjelasan Kode Program 5.9 adalah sebagai berikut:

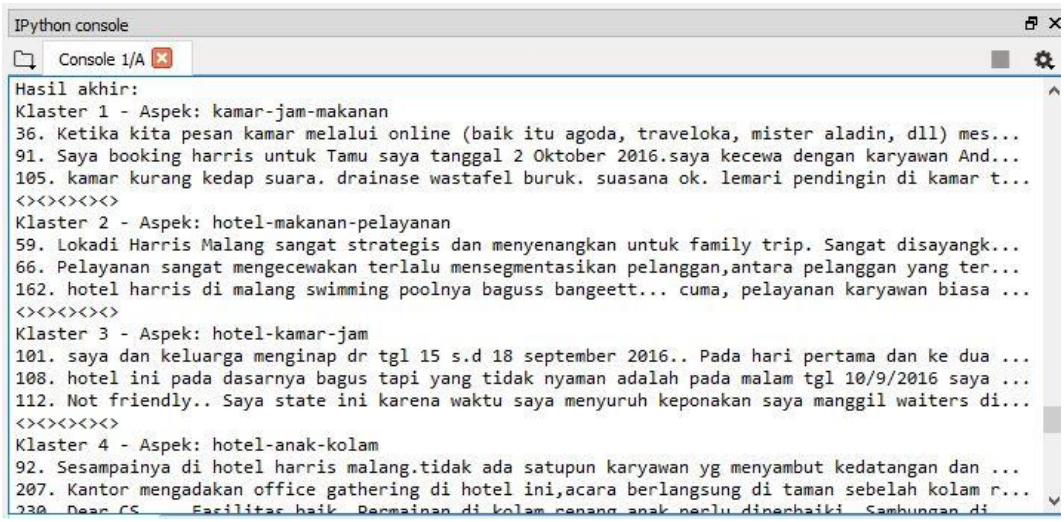
1. Baris 2-7 adalah untuk *pre-processing input*.

2. Baris 9-41 adalah untuk ekstraksi fitur *score representation*.

3. Baris 44-56 adalah untuk mendapatkan hasil klasifikasi sesuai metode LVQ

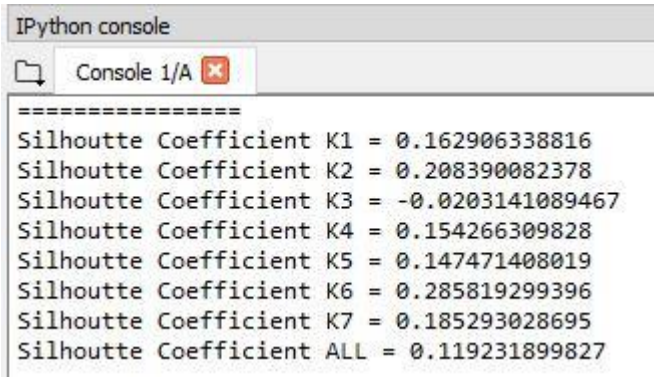
5.4 Implementasi Antarmuka

Gambar 5.4 menunjukkan hasil klasterisasi dengan menampilkan empat dokumen pada setiap klaster.



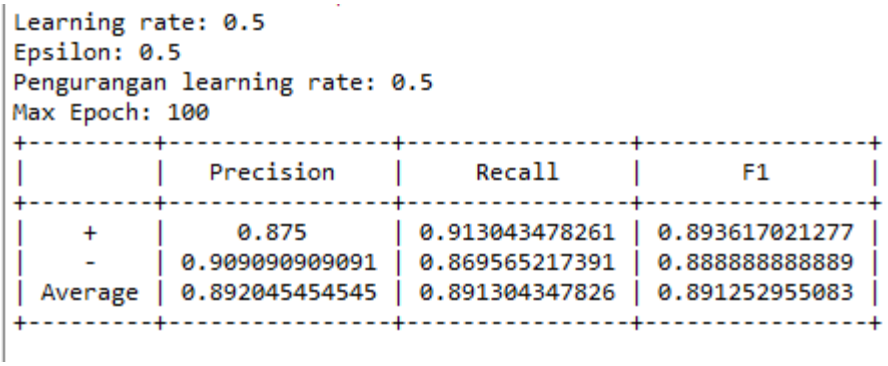
Gambar 5.4 Antarmuka Hasil Klasterisasi

Gambar 5.5 menunjukkan hasil pengujian klasterisasi dengan menampilkan nilai rata-rata *Silhouette Coefficient* pada setiap klaster kemudian rata-rata secara keseluruhan.



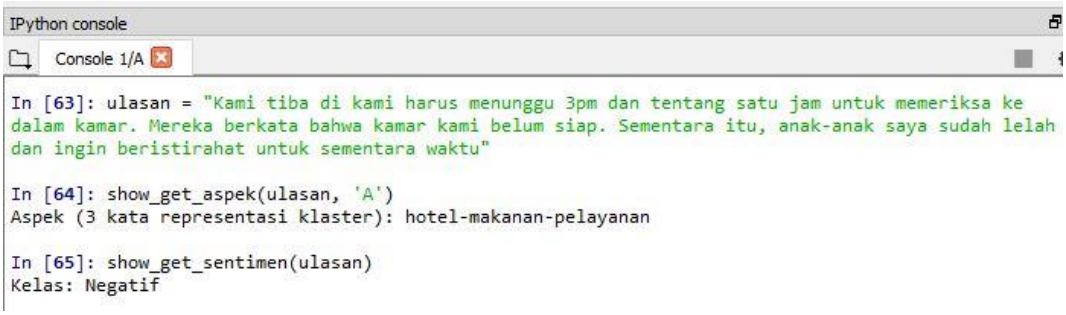
Gambar 5.5 Antarmuka Hasil Pengujian Klasterisasi

Gambar 5.6 menunjukkan hasil pengujian klasifikasi dengan menampilkan nilai *precisioin, recall,* dan *f1-score* untuk setiap kelas positif dan negatif kemudian rata-ratanya.



Gambar 5.6 Antarmuka Hasil Pengujian Klasifikasi

Gambar 5.7 menunjukkan hasil pengujian pada satu data untuk aspek dan kelas sentimennya.



Gambar 5.7 Antarmuka Uji Data