BAB 1 IMPLEMENTASI

1.1 Implementasi Kode Program

1.1.1 Proses Rule Template

```
def rule_tmplt(open_DUji):
    doc_testing = open(open_DUji)
    doc_testing = doc_testing.read()
    #rule tipe
    string = re.sub(r'([aA-zZ]{1,2}[0-9]{1,4}[aA-zZ+]{0,4})', r'\1<TIPE>', doc_testing)
    string = re.sub(r'(\b[0-9]{1,1}(?![gG])[aA-zZ]{1,1}\b)', r'\1<TIPE>', string)
    #rule harga
    string = re.sub(r'([rR][pP]|)(([1-9](|[.])([0-9]{1,2}|[0-9]{1,3}).([0-9]{1,3}).[0-
9]{1,3}(|,[0]{1,2})))', r'\1\2<HARGA>', string)
    string =
re.sub(r'(\d+)(.|,)(\d+)([jJ][uta]+[\w|an]|[rR][atus]+[\w|an]|[rR][ibu]+[\w|an])',
r'\1\2\3\4<HARGA>', string)
    string = re.sub(r'(\d+)(\s)([jJ][uta]+[\w|an]|[rR][atus]+[\w|an]|[rR][ibu]+[\w|an])',
r'\1<HARGA> \3<HARGA>', string)
    string =
re.sub(r'(?<![.|,])(\d+)([jJ][uta]+[\w|an]|[rR][atus]+[\w|an]|[rR][ibu]+[\w|an])',
r'\1\2<HARGA>', string)
    #rule n_spek
    string = re.sub(r'(\d+(|[ ])+([gG][bB]|[mM][pP]|[iI][n][a-
z]*|[fF][pP][sS]|[mM][aA][hH]|[gG]|[lL][tT][eE]|[gG]))', r'\1<N_SPEK>', string)
    string = re.sub(r'(\?+$)', r'\1<N_TAG>', string)
    #rule pattern
    #string = re.sub("\w+(?![<\w+>])", r_pattern, string)
    string = re.sub("(?!\d)\w+(?![<\w+>])", r_pattern, string)
    hsl_rule= open('hsl_ner.txt', 'w')
    hsl_rule.write("\tHASIL RULE TEMPLATE\n")
    hsl_rule.write (string)
    hsl_rule.write("\n\n")
    hsl_rule.close()

def r_pattern(m):
    v = m.group(0)
    MEREK = {"merek", "mereknya", "samsung", "apple",
"nokia","sony","motorola","xiaomi","oppo",
"oneplus","meizu","blackberry","vivo","htc","asus","zte"}
    tipe = {"tipe", "tipenya", "ipad","ThinkPad", "iphone", "galaxy", "tab" ,"plus", "pro",
"note", "lite", "mini", "redmi" ,"mi", "xperia","elite","max","prime"}
    harga = {"harga","harganya"}
    spek = {"spesifikasi", "fitur", "jaringan", "sinyal", "chipset", "cpu", "gpu" ,"memori",
"internal", "kamera",
"ram","os","bluetooth","usb","sensor","warna","berat","layar","baterai"}
    spek2 = []
    for i in spek:
        if i not in spek2:
```

```
33          spek2.append(i+'nya')
    n_spek = {"fingerprint", "accelerometer", "proximity", "kompas", "gyro",
34  "barometer", "gps"}
35
36      if v in MEREK:
37          return v+"<MEREK>"
38      elif v in tipe:
39          return v+"<TIPE>"
40      elif v in harga:
41          return v+"<HARGA>"
42      elif v in spek:
43          return v+"<SPEK>"
44      elif v in spek2:
45          return v+"<SPEK>"
46      elif v in n_spek:
47          return v+"<N_SPEK>"
48      else:
49          return v+"<N_TAG>"
```

Kode Program 1.1 Rule Template

Penjelasan Kode Program 5.1 adalah sebagai berikut:

1. Baris 1-3 merupakan deklarasi *fuction* untuk membaca data dokumen uji.
2. Baris 4-14 merupakan *fuction regular expression* subs dimana mengganti *string* sesuai *rule template* yang sudah dibuat.
3. Baris 17 merupakan pemanggilan *fuction r_pattern(m)* dimana menyeleksi *string.*
4. Baris 18-22 merupakan proses membuka dokumen dan menulis dokumen sampai menutup dokumen.
5. Baris 24-49 merupakan *fuction* untuk menyeleksi *string* apakah *string* tersebut termasuk MEREK, TIPE, HARGA, SPEK, N_SPEK jika tidak maka *string* tersebut termasuk N_TAG.

1.1.2 Proses Pre-processing Data Latih

```
1  mnl = open('mnl.txt', 'a')
2  doc_train = open("data_train.txt")
3   text_string = doc_train.read()
4   doc_train.close()
5    #tokenisasi
6    text_string = text_string.split()
```

Kode Program 1.2 Pre-processing dokumen

Penjelasan Kode Program 5.2 adalah sebagai berikut:

1. Baris 1-2 merupakan proses membuka dokumen hasil perhitungan dan membuka dokumen data latih.
2. Baris 3 merupakan proses membaca dokumen latih dan menyimpan ke sebuah *variables*.

3. Baris 4 proses menutup proses membuka dan membaca dokumen latih.
4. Baris 6 merupakan proses tokenisasi yang mana proses ini dilakukan pemecahan kalimat menjadi kata tunggal.

1.1.3 Proses Frekuensi

1.1.3.1 Proses Frekuensi Word

```
1   #data train unik
2   list_kata = []
3   for i in kata:
4       if i not in list_kata:
5           list_kata.append(i)
6
7   #menghitung frekuensi list kata
8   frek_kata = {}
9   for i in range(len(list_kata)):
10      frek_kata[i] = 0
11
12  for i in range(len(list_kata)):
13      for j in range(len(token)):
14          if (list_kata[i] == token[j]):
15              frek_kata[i] += 1
16
17  mnl.write("\tFREKUENSI WORD\n")
18  for i in range(len(list_kata)):
19      mnl.write('{:<20}\t{}\n'.format(list_kata[i], frek_kata[i]))
```

Kode Program 1.3 Proses Frekuensi Word

Penjelasan Kode Program 5.3 adalah sebagai berikut:

1. Baris 2-5 merupakan proses perulangan sebanyak kata dimana jika kata tidak ada di list_kata maka melakukan *fuction append(i)* yaitu penambahan data pada *list* list_kata.
2. Baris 8-10 meruapakan proses pemberian nilai awal pada frekuensi *word* sebanyak data list_kata.
3. Baris 12-15 merupakan proses perhitungan kemunculan frekuensi *word.*
4. Baris 17-19 merupakan proses menyimpan hasil perhitungan ke *txt*.

1.1.3.2 Proses Frekuensi Entitas

```
1   #kumpulan entitas pada data train
2   string = str(token)
3   data_entitas = re.findall(r'<\w+>', string)
4
5   #Menghitung frekuensi entitas
6   frek_entitas = {}
7   for i in range(len(entitas)):
8       frek_entitas[i] = 0
```

```
9
10    for i in range(len(entitas)):
11        for j in range(len(data_entitas)):
12            if (entitas[i] == data_entitas[j]):
13                frek_entitas[i] += 1
14
15    mnl.write("\n\tFREKUENSI ENTITAS\n")
16    for i in range(len(entitas)):
17        mnl.write('{:<10}\t\t{}\n'.format(entitas[i], frek_entitas[i]))
```

Kode Program 1.4 Proses Frekuensi Entitas

Penjelasan Kode Program 5.4 adalah sebagai berikut:

1. Baris 2-3 merupakan menyimpan *string* dan mencari *string* yang memiliki *entitas*.
2. Baris 6-8 merupakan proses pemberian nilai awal pada frekuensi *entitas.*
3. Baris 10-13 merupakan proses perhitungan kemunculan frekuensi *entitas*.
4. Baris 15-17 merupakan proses menyimpan hasil perhitungan ke *txt*.

1.1.3.3 Proses Frekuensi Transisi

```
1     #Memberikan nilai awal pada jumlah sumtrans
2     frek_trans = {}
3     frek_trans2 = {}
4     for i in range (len(entitas)*len(entitas)):
5         frek_trans[i] = 0
6         frek_trans2[i] = 0
7
8     #Menyimpan status transisi
9     trans = []
10    transke = []
11    before = data_entitas[0];
12    for i in range(1, len(data_entitas)):
13        count = 0
14        while count < len(entitas):
15            counts = 0
16            while counts < len(entitas):
17                if (before == entitas[count]) and (data_entitas[i] == entitas[counts]):
18                    trans.append(before + data_entitas[i])
19                    transke.append(before +'\t->\t'+ data_entitas[i])
20                    break
21                counts += 1
22            count += 1
23        before = data_entitas[i]
24
25    #Menyimpan list transisi
26    list_trans = []
27    for i in trans:
```

```
28          if i not in list_trans:
29              list_trans.append(i)
30
31      # Frek Trans List
32      for i in range(len(list_trans)):
33          for j in range(len(trans)):
34              if (list_trans[i] == trans[j]):
35                  frek_trans[i] += 1
36
37      #tampil frekuensi transisi
38      trans_listke = []
39      for i in transke:
40          if i not in trans_listke:
41              trans_listke.append(i)
42      trans_listke.sort(key=str)
43      for i in range(len(trans_listke)):
44          for j in range(len(transke)):
45              if (trans_listke[i] == transke[j]):
46                  frek_trans2[i] += 1
47      mnl.write('\n\tFREKUENSI TRANSISI\n' )
48      for i in range(len(trans_listke)):
49          mnl.write('{}\t{}\n'.format(trans_listke[i], frek_trans2[i]))
```

Kode Program 1.5 Proses Frekuensi Transisi

Penjelasan Kode Program 5.5 adalah sebagai berikut:

1. Baris 2-6 merupakan proses pemberian nilai 0 pada frekuensi transisi.
2. Baris 9-23 merupakan proses menyimpan transisi yang terjadi pada dokumen latih.
3. Baris 25-35 merupakan proses menghitung kemunculan frekuensi transisi.
4. Baris 37-46 merupakan proses yang sama seperti proses baris 25-35, proses ini terdapat tambahan sorting berdasarkan transisi.
5. Baris 47-49 merupakan proses menyimpan hasil perhitungan frekuensi transisi yang sudah disorting ke *txt*.

1.1.4 Proses Perhitungan Transition Probability

```
1       #Menghitung Transition Probabilty
2       matrix_trans = []
3       temp1, temp2 = 0, 0
4
5       for i in range(len(entitas)+1):
6           matrix_trans.append([])
7           for j in range(len(entitas)+1):
8               if (i == 0) and (j == 0):
9                   matrix_trans[i].append('Transition')
10              elif (i == 0) and (j > 0):
11                  matrix_trans[i].append(entitas[temp1])
```

```
12              temp1 += 1
13          elif (i > 0) and (j == 0):
14              matrix_trans[i].append(entitas[temp2])
15              temp2 += 1
16          elif (i > 0) and (j > 0):
17              count = 0
18              while count < len(list_trans):
19                  if (entitas[i-1]+entitas[j-1] == list_trans[count]):
                        matrix_trans[i].append(round(((float(frek_trans[count])) /
20  (float(frek_entitas[i-1]))), 6))
21                      break
22                  count += 1
23              else:
24                  matrix_trans[i].append(0)
25
26      mnl.write('\n\tTRANSITION PROBABILITY\n')
27      for i in matrix_trans:
28          mnl.write("%s\n"% i)
```

Kode Program 1.6 Proses Perhitungan *Transition Probability*

Penjelasan Kode Program 5.6 adalah sebagai berikut:

1. Baris 2-3 merupakan inisialisasi *variables list* untuk hasil dan *temp* untuk mempermudah perulangan.
2. Baris 5-15 merupakan proses pembuatan *matrix* sesuai dengan entitas dan perhitungan.
3. Baris 17-24 merupakan proses perhitungan *transition probability* berdasarkan frekuensi transisi dan frekuensi entitas yang didapatkan pada proses sebelumnya.
4. Baris 26-28 merupakan proses menyimpan hasil perhitungan *transition probability* ke *txt*.

1.1.5 Proses Perhitungan Emission Probability

```
1   #Menghitung Emission Probability
2   matrix_emis = []
3   temp3, temp4 = 0, 0
4
5   for i in range(len(token_list)+1):
6       matrix_emis.append([])
7       for j in range(len(entitas)+1):
8           if (i == 0) and (j == 0):
9               matrix_emis[i].append('Emission')
10          elif (i == 0) and (j > 0):
11              matrix_emis[i].append(entitas[temp3])
12              temp3 += 1
13          elif (i > 0) and (j == 0):
14              matrix_emis[i].append(token_list[temp4])
```

```
15              temp4 += 1
16          elif (i > 0) and (j > 0):
17              count = 0
18              while count < len(list_kata):
19                  if (token_list[i-1] + entitas[j-1] == list_kata[count]):
                        matrix_emis[i].append(round(((float(frek_kata[count]))/
20  (float(frek_entitas[j-1]))), 6))
21                      break
22                  count += 1
23              else:
24                  matrix_emis[i].append(0)
25
26      mnl.write('\n\tEMISSION PROBABILITY\n')
27      for i in matrix_emis:
28          mnl.write("%s\n"%i)
```

Kode Program 1.7 Proses Perhitungan *Emission Probability*

Penjelasan Kode Program 5.7 adalah sebagai berikut:

1. Baris 2-3 merupakan inisialisasi *variables list* untuk hasil dan *temp* untuk mempermudah perulangan.
2. Baris 5-15 merupakan proses pembuatan *matrix* sesuai dengan entitas dan kalimat pada data latih serta perhitungan.
3. Baris 17-24 merupakan proses perhitungan *emission probability* dilakukan dengan membagi frekuensi kemunculan kata dengan entitas tertentu dengan banyaknya kemunculan entitas tersebut.
4. Baris 26-28 merupakan proses menyimpan hasil perhitungan *emission probability* ke *txt*.

1.1.6 Proses Pre-processing Data uji

```
1   #Data Uji
2       document_testing = open(open_DUji)
3       text_testing = document_testing.read()
4       document_testing.close()
5       #tokenisasi
6       data_testing = text_testing.split()
```

Kode Program 1.8 Proses Pre-processing Data Uji

Penjelasan Kode Program 5.8 adalah sebagai berikut:

1. Baris 1-2 merupakan proses membuka dokumen data uji.
2. Baris 3 merupakan proses membaca dokumen uji dan menyimpan ke sebuah *variables*.
3. Baris 4 proses menutup proses membuka dan membaca dokumen uji.
4. Baris 6 merupakan proses tokenisasi yang mana proses ini dilakukan pemecahan kalimat menjadi kata tunggal.

1.1.7 Proses Algoritme Viterbi

```
#Menghitung Viterbi
    temp5, temp6 = 0, 0
    go = 0
    matrix_viterbi = []
    trans_prob = []
    for i in range(len(data_testing)+1):
        matrix_viterbi.append([])
        for j in range(len(entitas)+1):
            if (i == 0) and (j == 0):
                matrix_viterbi[go].append('Viterbi')
            elif (i == 0) and (j > 0):
                matrix_viterbi[go].append(entitas[temp5])
                temp5 += 1
            elif (i > 0) and (j == 0):
                matrix_viterbi[go].append(data_testing[temp6])
                temp6 += 1
            elif (i == 1) and (j > 0):
                count = 0
                while count < len(matrix_emis):
                    entitas_before = matrix_trans[1][0]
                    if (data_testing[i-1] == matrix_emis[count][0]):
                        matrix_viterbi[go].append(round(float(matrix_emis[count][j]) *
float(matrix_trans[i][j]), 6))
                        temp = 1
                        max_tag = 0
                        while temp < len(matrix_viterbi[i]):
                            if (matrix_viterbi[go][temp] > max_tag):
                                max_tag = matrix_viterbi[go][temp]
                                entitas_before = matrix_viterbi[0][temp]
                            temp += 1
                        for a in range(1, len(matrix_trans)):
                            for b in range(1, len(matrix_trans[0])):
                                if (matrix_trans[a][0] == entitas_before):
                                    trans_prob.append(matrix_trans[a][b])
                        break
                    count += 1
                else:
                    matrix_viterbi[go].append(0)
                    temp = 1
                    max_tag = 0
                    while temp < len(matrix_viterbi[i]):
                        if (matrix_viterbi[go][temp] > max_tag):
                            max_tag = matrix_viterbi[i][temp]
                            entitas_before = matrix_viterbi[0][temp]
                        temp += 1
```

```
45                for a in range(1, len(matrix_trans)):
46                   for b in range(1, len(matrix_trans[0])):
47                      if (matrix_trans[a][0] == entitas_before):
48                         trans_prob.append(matrix_trans[a][b])
49             elif (i > 1) and (j > 0):
50                counts = 0
51                while counts < len(matrix_emis):
52                   if(data_testing[i-1] == matrix_emis[counts][0]):
                        matrix_viterbi[go].append(round(float(matrix_emis[counts][j]) *
53   float(trans_prob[j-1]), 6))
54                   temp = 1
55                   max_tag = 0
56                   while temp < len (matrix_viterbi[i]):
57                      if (matrix_viterbi[go][temp] > max_tag):
58                         max_tag = matrix_viterbi[go][temp]
59                         entitas_before = matrix_viterbi[0][temp]
60                      temp += 1
61                   if (j == len(entitas)-1):
62                      trans_prob = []
63                      for a in range(1, len(matrix_trans)):
64                         for b in range(1, len(matrix_trans[0])):
65                            if (matrix_trans[a][0] == entitas_before):
66                               trans_prob.append(matrix_trans[a][b])
67                   break
68                counts += 1
69
70                else:
71                   matrix_viterbi[go].append(0)
72                   temp = 1
73                   max_tag = 0
74                   while temp < len (matrix_viterbi[i]):
75                      if (matrix_viterbi[go][j] > max_tag):
76                         max_tag = matrix_viterbi[go][j]
77                         entitas_before = matrix_viterbi[0][j]
78                      temp += 1
79                   if (j == len(entitas)-1):
80                      trans_prob = []
81                      for a in range(1, len(matrix_trans)):
82                         for b in range(1, len(matrix_trans[0])):
83                            if (matrix_trans[a][0] == entitas_before):
84                               trans_prob.append(matrix_trans[a][b])
85          go +=1
86       mnl.write('\n\tALGORITMA VITERBI\n')
87       for i in matrix_viterbi:
88          mnl.write("%s\n"%i)
89
```

```
 90      mnl.close()
 91
 92      #mencari jalur terbaik (NER)
 93      ner = {}
 94      for i in range(1, len(matrix_viterbi)):
 95          j, max = 1, 0
 96          while j < len(matrix_viterbi[0]):
 97              if (matrix_viterbi[i][j] > max):
 98                  max = matrix_viterbi[i][j]
 99                  ner[i] = matrix_viterbi[i][0] + matrix_viterbi[0][j]
100              elif(max == matrix_viterbi[i][j]) :
101                  ner[i] = matrix_viterbi[i][0]
102              j += 1
103
104  hsl_ner= open('hsl_ner.txt', 'a')
105      hsl_ner.write('\tHASIL HMM\n')
106      for i in ner:
107          hsl_ner.write (str(ner[i]))
108          hsl_ner.write (str(" "))
109      hsl_ner.close()
```

Kode Program 1.9 Proses Algoritme Viterbi

Penjelasan Kode Program 5.9 adalah sebagai berikut:

1. Baris 2-5 merupakan inisialisasi *variables* dan *list* untuk mempermudah perulangan dan perhitungan.
2. Baris 6-17 merupakan proses pembuatan *matrix* sesuai dengan entitas dan kalimat pada data uji serta perhitungan.
3. Baris 18-85 merupakan proses perhitungan *viterbi*, dimana perhitungan ini didapatkan dari hasil perkalian nilai *transition probability* dengan *emission probability*. Hasil perkalian tersebut guna untuk mencari jalur terbaik. Setelah perhitungan pada kalimat pertama kemudian melakukan kembali perkalian *transition probability* dengan *emission probability* pada kalimat berikutnya. Nilai *transition probability* yang digunakan pada perhitungan kalimat kedua dan berikutnya merupakan nilai maksimum terhadap entitas sebelumnya.
4. Baris 86-88 merupakan proses menyimpan hasil perhitungan *viterbi* ke *txt*.
5. Baris 90-100 merupakan proses akhir dalam *viterbi* dimana memberikan entitas pada data uji tersebut. Proses ini dilakukan dengan mencari entitas yang memiliki nilai maksimum untuk setiap kalimat pada dokumen data uji yang telah dilakukan pada proses sebelumnya. Jika nilai maksimum yang didapatkan pada kalimat tersebut bernilai 0, maka kalimat tersebut tidak memiliki entitas.
6. Baris 102-107 merupakan proses menyimpan hasil perhitungan data uji dengan viterbi ke *txt* yang sudah memiliki entitas.

1.1.8 Proses Hybird NER

```
1  def Hybird():
2      #membaca file hasi NER
```

```
f = open('hsl_ner.txt')
lines=f.readlines()
for i, line in enumerate(lines):
    rule = lines[1]
    rule = rule.rstrip('\n')
    hmm = lines[4]
    hmm = hmm.rstrip('\n')
    hmmSmooth = lines[7]
    hmmSmooth = hmmSmooth.rstrip('\n')

#dictionary rule template
rule_items = [s for s in rule.split(' ') if s]
d_rule = {}
for rule_item in rule_items:
    if len(rule_item.split('<')) < 2: continue
    key,value = rule_item.split('<')
    d_rule[key] = '<'+value

#dictionary hmm
hmm_items = [s for s in hmm.split(' ') if s]
d_hmm = {}
for hmm_item in hmm_items:
    if len(hmm_item.split('<')) < 2: continue
    key,value = hmm_item.split('<')
    d_hmm[key] = '<'+value

#dictionary hmm+add smoothing
hmmSmooth_items = [s for s in hmmSmooth.split(' ') if s]
d_hmmSmooth = {}
for hmmSmooth_item in hmmSmooth_items:
    if len(hmmSmooth_item.split('<')) < 2: continue
    key,value = hmmSmooth_item.split('<')
    d_hmmSmooth[key] = '<'+value

rule_hmm = d_rule.copy()
rule_hmmsmooth= d_rule.copy()

#Hybird hasil rule dengan hmm
for key, value in d_ rule_hmm.items():
    if value == '<N_TAG>':
        rule_hmm [key] = d_hmm.get(key)
for key, value in rule_hmm.items():
    if value is None:
        value = '<N_TAG>'
        rule_hmm [key] = value
Rule_Hmm = rule_hmm
```

```
49
50
51      #Hybird hasil rule dengan hmm+add smooth
52      for key, value in rule_hmmsmooth.items():
53          if value == '<N_TAG>':
54              rule_hmmsmooth [key] = d_hmmSmooth.get(key)
55      Rule_HmmSmooth = rule_hmmsmooth
56
57      #simpan hasil
58      hsl_ner= open('hsl_ner.txt', 'a')
59      hsl_ner.write('\n\n\tHASIL Rule + HMM\n')
60      a = ' '.join("{!s}{!s}".format(key,val) for (key,val) in Rule_Hmm.items())
61      hsl_ner.write(a)
62      hsl_ner.write('\n\n\tHASIL Rule + HMM dan Additve Smoothing\n')
63      b = ' '.join("{!s}{!s}".format(key,val) for (key,val) in Rule_HmmSmooth.items())
64      hsl_ner.write(b)
65      hsl_ner.close()
```

Kode Program 1.10 Proses Hasil NER

Penjelasan Kode Program 5.10 adalah sebagai berikut:

1. Baris 1-38 setelah mendapatkan hasil proses *Rule template dan Hidden Markov Model* (HMM) langkah ini menyimpan hasil pengenalan entitas ke dalam bentuk dictionary.
2. Baris 41-55 merupakan proses menukar nilai entitas N_TAG pada hasil pengenalan *Rule Template* terhadapat hasil pengenalan HMM maupun HMM dengan penambahan *Additive Smoothing.*
3. Baris 51-65 merupakan proses menyimpan hasil pengenalan *Rule Template* dengan HMM, dan hasil pengenalan *rule template* dengan HMM ditambah *additive smoothing*.

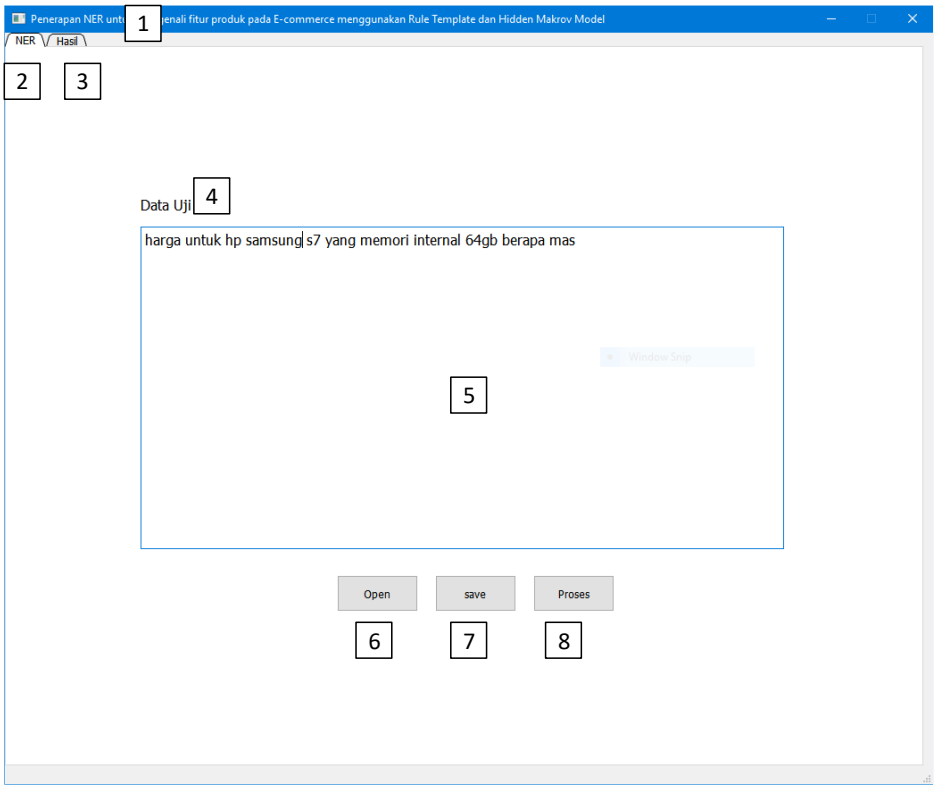
1.2 Implementasi Antarmuka

Pada antarmuka terdiri dari 2 panel antarmuka yaitu antarmuka untuk masukan data dan antarmuka untuk menampilkan hasil. Antarmuka yang diimplementasikan pada penelitian ini ditujukan untuk memudahkan *user* dalam memilih, mengubah dokumen uji. Terdapat tombol yang dapat digunakan *user* dalam memilih dokumen uji dan tombol untuk menyimpan jika terdapat perubahan pada data uji yang dipilih.

Berikut ini merupakan penjelasan panel pada Gambar 5.1 antarmuka masukan data:

1. Judul sistem Penerapan *Named Entity Recognition* untuk mengenali fitur produk pada *E-commerce* menggunakan *Rule Template* dan *Hidden Markov Model*.
2. *Button* untuk beralih ke antarmuka masukan data.
3. *Button* untuk beralih ke antarmuka hasil.
4. *Text label* data uji.
5. *Text field* untuk menampilkan mengubah data uji.
6. *Button* untuk memilih *file* data uji tipe .txt.

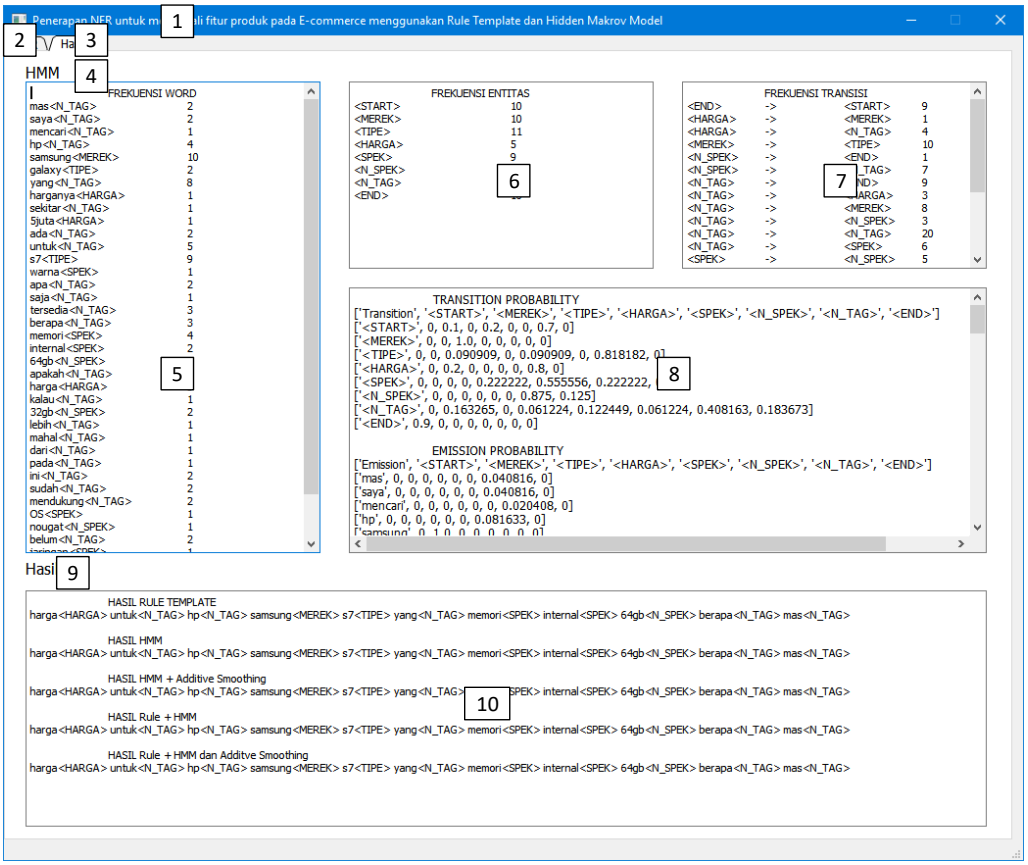
7. *Button* untuk menyimpan file data uji yang telah diubah.
8. *Button* untuk menjalankan program dalam penerapan NER pada *e-commerce* menggunakan *Rule template* dan *Hidden Markov Model.*



Gambar 1.1 Implementasi Antarmuka Masukan Data

Berikut ini merupakan tampilan antarmuka hasil dimana menampilkan hasil perhitungan *Hidden Markov Model* dan menampilkan hasil dari metode *Hidden Markov model dan Rule Template.* Berikut penjelasan panel pada Gambar 5.2 antarmuka hasil:

1. Judul sistem Penerapan *Named Entity Recognition* untuk mengenali fitur produk pada *E-commerce* menggunakan *Rule Template* dan *Hidden Markov Model*.
2. *Button* untuk beralih ke antarmuka masukan data.
3. *Button* untuk beralih ke antarmuka hasil.
4. *Text label* HMM.
5. *Text field* untuk menampilkan frekuensi word data latih.
6. *Text field* untuk menampilkan frekuensi entitas data latih.
7. *Text field* untuk menampilkan frekuensi transisi data latih.
8. *Text field* untuk menampilkan perhitungan HMM dan Perhitungan HMM dengan penambahan *Additive Smoothing* yang terdiri dari probabilitas transisi, probabilitas emisi dan perhitungan data uji menggunakan algoritme Viterbi.
9. *Text label* Hasil.
10. *Text field* untuk menampilkan hasil pengenalan *Named Entity Recognition*.



Gambar 1.2 Implementasi Antarmuka Hasil