BAB 5 IMPLEMENTASI

Berdasarkan perancangan pada Bab 4, maka dibuat implementasi dengan menggunakan bahasa pemrograman Python.

5.1 Preprocessing

Implementasi *preprocessing* dilakukan dengan melakukan beberapa *import* library yang ada di python

- Import *counter* untuk menghitung jumlah kata yang digunakan pada proses setelah preprocessing
- Import csv untuk melakukan pembacaan file csv
- Import re (regex) dilakukan untuk proses tokenisasi dimana regex bisa dengan efektif melakukan *case folding* dan *cleaning* pada tokenisasi
- Import Sastrawi untuk proses *stemming* secara efektif dari Sastrawi

Preprocessing sendiri terdiri dari tiga sub-proses yaitu tokenisasi, *filtering,* dan *stemming*. Setiap subproses dibuat *method* sendiri supaya lebih mudah diimplementasikan, untuk tokenisasi dilakukan penggantian karakter dan angka dengan spasi, dilanjutkan dengan *case folding,* lalu pemisahan dokumen menjadi perkata (*split*). Filtering dilakukan dengan membuka file *stopword* yang berisi kata-kata yang harus dihilangkan lalu membuat list yang berisi dokumen yang sudah dihilangkan dihilangkan *stopword*-nya. Untuk *stemming*, proses ini dilakukan dengan pemanggilan library Sastrawi, sehingga prosesnya cukup dengan memanggil library tersebut untuk dilakukan *stemming* pada setiap katanya.

Setelah method tersebut selesai dibuat, untuk mengaplikasikannya pada data dilakukan pembacaan file csv dan pemanggilan *method* tokenisasi, *filtering,* dan *stemming* untuk melakukan proses setiap baris. Setiap baris pada data csv dianggap sebagai satu data atau dokumen, kolom pertama berisi ulasan dan kolom kedua berisi kelas sentimennya. Implementasi *preprocessing* bisa dilihat pada Kode Program 5.1.

```
1   from collections import Counter
2   import csv
3   import re
4   from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
5   import operator
6
7   # buka file latih
8   with open("datanyakecil.csv", 'rb') as file:
9       reviews = list(csv.reader(file))
10
11  with open('stopword_list_tala.txt','r') as file:
12      file = map(lambda s: s.strip(), file)
13      listsw = list(file)
14
15  def tokenisasi(data):
16      data = re.sub(r'\W+|[0-9]+',' ',data)
17      data = data.lower()
18      data = data.split()
```

```
    return data

def filtering(data, stopwords):
    return [w for w in data if w not in stopwords]

def stemming (data):
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    return [stemmer.stem(w) for w in data]

dtBaru=[]
for row in reviews:
    sentimen = row[1]
    ulasan = row[0]
    prosestoken = tokenisasi(ulasan)
    prosesfilter = filtering (prosestoken,listsw)
    prosesstemming = stemming (prosesfilter)
    stringstemming=' '.join(prosesstemming)
    dtBaru.append((stringstemming, sentimen))
```

Kode Program 5.1 Implementasi preprocessing

5.2 Seleksi Fitur *Query Expansion Ranking*

Implementasi seleksi fitur *Query Expansion Ranking* dilakukan dengan membuat *method* untuk melakukan pengecekan keberadaan fitur terpilih pada tiap dokumen positif dan negatif bernama pfqf(dokumen,listbaru) *method* ini mengembalikan nilai df untuk data positif dan negative dan tampung nilainya pada sebuah *list*.

Setelah itu nilai pf, qf, dan *Query Expansion Ranking* dicari sesuai persamaan 2.6, 2.7, dan 2.8. Kemudian urutkan fitur dan nilai *Query Expansion Ranking* sesuai nilai *Query Expansion Ranking* dari terkecil hingga terbesarl lalu potong (pilih) jumlah fiturnya sesuai rasio yang diinginkan. Implementasi *Query Expansion Ranking* ditunjukkan pada Kode Program 5.2.

```
pf=[]
qf=[]
psfs=[]

pfSort=[]
qfSort=[]

def pfqf(dokumen,listbaru):
    a=0;
    for kata in listUnik:
        #print kata
        if kata in dokumen:
            a=a+1
        else:
            a=0
        listbaru.append((kata,a))

for row in isiPos:
    pf1= pfqf(row,pf)

for row in isiNeg:
    qf1= pfqf(row,qf)

def sortHuruf(listabcd,listbaru):
    count=Counter()
    for k,v in listabcd:
        count[k]+=v
    listbaru=                              sorted(count.items(),
key=operator.itemgetter(0))
    return listbaru

pff=sortHuruf(pf,pfSort)
qff=sortHuruf(qf,qfSort)

jmlDataPos=len(isiPos)
jmlDataNeg=len(isiNeg)

datapfqf=[]
dataqer=[]
dataqerurut=[]
listqerbaru=[]
i=0

for kata,pf in pff:
    qf=qff[i][1]
    i=i+1
    pfhasil= (pf+0.5)/(jmlDataPos+1.0)
    qfhasil= (qf+0.5)/(jmlDataNeg+0.5)
    atas= pfhasil+qfhasil
    bawah= pfhasil-qfhasil
    qerhasil= (abs(atas)/abs(bawah))
    datapfqf.append((kata,pfhasil,qfhasil,qerhasil))
    dataqer.append((kata,qerhasil))
```

Kode Program 5.2 Implementasi seleksi fitur *Query Expansion Ranking*

5.3 Klasifikasi Sentimen dengan Naïve Bayes

Implementasi Naive Bayes dilakukan dengan melakukan *preprocessing* pada data uji. Walaupun Naive Bayes hanya diterapkan pada data uji, namun data uji tetap dilakukan *preprocessing* supaya data bisa diolah dengan lebih mudah. Setelah *preprocessing* selesai maka dilakukan pencocokan setiap *term* atau kata

pada data uji dengan *feature selection* menggunakan algoritme *wordlist* yang cara kerjanya berkebalikan dengan *filtering,* algoritme *wordlist* mengembalikan kata yang ada pada daftar kata *wordlist* (*feature selection*).

Supaya implementasi Naive Bayes ini lebih mudah, maka dibuat sebuah *method* bernama naivebayes. *Method* ini bertujuan untuk menghitung nilai Naive Bayes sebuah data atau dokumen. Karena algoritme ini bekerja dengan cara mengalikan peluang setiap kata pada masing-masing kelasnya maka dibuat sebuah variabel bernama prediksi yang bernilai 1 yang tidak akan mempengaruhi hasil perkalian jika dikalikan dengan variabel prediksi.

Setelah nilai dari Naive Bayes tiap kelas didapat, selanjutnya adalah membuat *method* untuk menentukan bahwa dokumen tersebut masuk kedalam kelas sentimen positif atau negatif. Cara menentukannya adalah jika hasil Naive Bayes negatif lebih besar dari Naive Bayes positif maka dokumen tersebut bersentimen negatif, namun jika tidak maka dokumen tersebut bernilai positif. Implementasi klasifikasi sentimen dengan Naive Bayes dapat dilihat melalui Kode Program 5.3.

```
1   def Wordlist(dokumen,wordlist):
2       return [w for w in dokumen if w in wordlist]
3
4   def naiveBayes(dokumenTest, listKataKelas, hitungKataKelas,
5   peluangKelas):
6       prediksi = 1.000
7       isiDok=dokumenTest.split()
8       pencocokan = Wordlist(isiDok,seleksiFitur)
9       for kata in pencocokan:
10          prediksi*=
11  (float(listKataKelas.count(kata))+1)/(float(hitungKataKelas
12  )+jmlUnik)
13      return prediksi*peluangKelas
14
15  with open("datanyakeciltest.csv", 'rb') as file:
16      #test = file.read()
17      testfile = list(csv.reader(file))
18
19  dtBaruTest=[]
20  for row in testfile:
21      sentimen = row[1]
22      ulasan = row[0]
23      prosestoken = tokenisasi(ulasan)
24      stringtoken= ' '.join(prosestoken)
25      #listtoken = stringtoken.split()
26      prosesfilter = filtering (prosestoken,listsw)
27      stringfilter=' '.join(prosesfilter)
28      prosesstemming = stemming (prosesfilter)
29      stringstemming=' '.join(prosesstemming)
30      dtBaruTest.append((stringstemming, sentimen))
31  dokTokenTest  =csv.writer(open('datanyakeciltestprepos.csv',
32  'wb'), delimiter=',')
33  dokTokenTest.writerows(dtBaruTest)
34
35  with open("datanyakeciltestprepos.csv", 'rb') as file:
36      test = list(csv.reader(file))
37
38  def masukKelas(dokumen):
39      print "===POSITIF==="
40
```

```
41      prediksiPos=      naiveBayes(dokumen,      listKataPos,
42  jmlKataPos, peluangPos)
43      print "===NEGATIF==="
44      prediksiNeg=      naiveBayes(dokumen,      listKataNeg,
45  jmlKataNeg, peluangNeg)
46      if prediksiPos>prediksiNeg:
47          a=prediksiPos
48          b="Positif"
49          c=1
50      else:
51          a=prediksiNeg
52          b="Negatif"
53          c=-1
54      #print dokumen,a,b
        return a,b,c
```

Kode Program 5.3 Klasifikasi sentimen dengan Naive Bayes