BAB 5 IMPLEMENTASI

Bab ini menjelaskan tentang batasan implementasi dan hasil implementasi sistem yang dibuat untuk penelitian ini seperti *preprocessing text*, pembobotan *term* atau kata, dan klasifikasi KNN.

5.1 Batasan Implementasi

Batasan implementasi merupakan batasan-batasan yang harus dipenuhi pada saat mengimplementasi sistem berdasarkan perancangan yang telah dibuat pada bab sebelumnya. Tujuan dari batasan implementasi adalah agar sistem yang akan dibuat menjadi lebih terarah dan dapat fokus pada tujuan utama sistem. Batasan implementasi sistem untuk penelitian ini adalah sebagai berikut.

1. Sistem ini diimplementasikan menggunakan bahasa pemrograman Python.
2. Metode yang digunakan untuk klasifikasi kepribadian adalah *K-Nearest Neighbor*.
3. Data-data yang digunakan berasal dari *tweet* para karyawan di suatu perusahaan. Data tersebut disimpan ke dalam *database* MySQL.
4. Hasil keluaran sistem ini berupa empat macam kepribadian yaitu Artisan, Guardian, Idealist, dan Rasional.

5.2 Implementasi Sistem

Pada subbab implementasi sistem akan diuraikan berbagai tahapan klasifikasi kepribadian untuk mengetahui karakter calon karyawan dengan algoritma KNN. Berbagai tahap dalam implementasi sistem mengikuti berbagai tahapan proses yang telah dirancang pada bab sebelumnya dan menampilkan potongan kode program untuk setiap prosesnya.

Tahapan sistem pada penelitian ini mencakup *preprocessing text* yang mengolah data berupa dokumen teks menjadi fitur-fitur kata, pembobotan *term* atau kata yang akan memberi bobot pada setiap kata hasil *preprocessing text*, dan klasifikasi KNN yang akan menentukan kelas kepribadian dari setiap data uji. Fungsi yang ada pada sistem dapat dilihat pada Tabel 5.1.

Tabel 5.1 Daftar Fungsi Sistem

No.	Proses	Nama Fungsi	Keterangan
1	*Preprocessing Text*	*tolowercase()*	Fungsi yang digunakan untuk mengubah seluruh karakter huruf didalam data menjadi karakter huruf kecil.
		tokenization()	Fungsi yang digunakan untuk memecah kalimat atau dokumen di dalam data menjadi satuan kata. Sebelum itu, akan dilakukan penghapusan tanda baca dan angka

			pada data.
		stoplist()	Fungsi yang digunakan untuk memanggil daftar *stopword* atau *stoplist* dari *file* csv yang berguna untuk proses *filtering*.
		filtering()	Fungsi yang digunakan untuk menghapus kata-kata yang tidak bermakna penting atau *stopword* berdasarkan *stoplist*.
		stemming()	Fungsi yang digunakan untuk menghapus imbuhan pada setiap kata pada data.
2	Pembobotan *Term*	*termsunique()*	Fungsi yang digunakan untuk membuat *array* yang berisi kata-kata unik dari hasil *preprocessing text*.
		calculatewtf()	Fungsi yang digunakan untuk menghitung jumlah fitur kata yang sama dalam satu dokumen dan pembobotan logaritma dari jumlah fitur tersebut.
		calculateidf()	Fungsi yang digunakan untuk menghitung jumlah dokumen yang mengandung suatu fitur kata dan nilai kebalikan dari jumlah dokumen tersebut.
		calculatetfidf()	Fungsi yang digunakan untuk menghitung hasil perkalian dari TF dan IDF dari suatu fiturkata.
4	KNN	*dotproduct()*	Fungsi yang digunakan menghitung nilai *dot product*.
		panjangdokumen()	Fungsi yang digunakan untuk menghitung panjang dokumen.
		cosinesimilarity()	Fungsi yang digunakan untuk menghitung *cosine similarity* yang merupakan perkalian dari *dot product* dan panjang dokumen.
		seleksinilaik)	Fungsi yang digunakan untuk mengurutkan nilai hasil cosine similarity dari yang terbesar hingga yang terkecil dan menyeleksinya sebanyak nilai k.
		classificationknn()	Fungsi yang digunakan untuk menentukan kelas akhir dari hasil prediksi sistem pada setiap data uji.
5	Pengujian	*akurasi()*	Fungsi yang digunakan untuk menghitung tingkat akurasi sistem.

5.3 *Prapocessing Text*

Preprocessing text merupakan tahapan paling awal dalam sistem ini yang berguna untuk mendapat fitur-fitur kata yang membantu proses klasifikasi dari data-data berupa dokumen teks. Tahapan preprocessing text terdiri dari *to lowercase*, *tokenization*, *filtering*, dan *stemming*.

5.3.1 Implementasi *To Lowercase*

To lowercase merupakan tahap paling awal dalam *preprocessing text*. Pada tahap ini, seluruh karakter huruf dalam data akan menjadi karakter huruf kecil. Implementasi *to lowercase* ditunjukkan pada Kode Program 5.1.

```
1  def tolowercase(self,teks):
2      teksbaru = teks.lower()
3      return teksbaru
```

Kode Program 5.1 Implementasi *To Lowercase*

Uraian potongan Kode Program 5.1 tentang implementasi *to lowercase* ditunjukkan sebagai berikut:

1. Pada baris 2 menunjukkan proses pengubahan setiap karakter pada data menjadi karakter huruf kecil.

5.3.2 Implementasi *Tokenization*

Tokenization merupakan tahap setelah *to lowercase* dalam *preprocessing text*. Tahap ini bertujuan untuk menghapus tanda baca dan angka dalam data serta memecah kalimat menjadi satuan kata. Implementasi *tokenization* dapat dilihat pada Kode Program 5.2.

```
1  def tokenization(self, text):
2      text  ="".join([a for a in text if a not in string.punctuation])
3      text  ="".join([a for a in text if a not in string.digits])
4      terms = text.split()
5      return  terms
```

Kode Program 5.2 Implementasi *Tokenization*

Uraian potongan Kode Program 5.2 tentang implementasi tokenization ditunjukkan sebagai berikut:

1. Pada baris 2 menunjukkan proses penghapus tanda baca dalam data.
2. Pada baris 3 menunjukkan proses penghapusan angka dalam data.
3. Pada baris 4 menunjukkan proses pemecahan kalimat menjadi satuan kata.

5.3.3 Implementasi *Filtering*

Filtering merupakan tahap selanjutnya setelah *tokenization* dalam *processing text*. Tahap ini bertujuan untuk menghapus kata-kata yang tidak memiliki makna penting dalam proses klasifikasi. Implementasi *filtering* ditunjukkan pada Kode Program 5.3.

```
1  def stoplist(self):
2      with open('stoplist.csv', 'r') as filestoplist:
3          datastoplist = csv.reader(filestoplist)
```

```
4           data = []
5           for a in datastoplist:
6               data += a
7           return data
8   def filtering(self, term):
10      stopword = self.stoplist()
11      if term not in stopword:
12          return term
```

Kode Program 5.3 Implementasi *Filtering*

Uraian potongan Kode Program 5.3 tentang implementasi *filtering* ditunjukkan sebagai berikut.

1. Pada baris 2 – 6, sistem akan memanggil *file* stoplist.csv yang berisi daftar *stopword* atau kata-kata yang tidak memiliki makna penting. Kemudian daftar tersebut dimasukkan ke dalam *array*. Cuplikan *file* stoplist.csv ditunjukkan pada Gambar 5.1.

Gambar 5.1 Cuplikan *File* stoplist.csv

2. Pada baris 10 menunjukkan proses pemanggilan *array* yang berisi daftar *stopword*.

3. Pada baris 11 – 12 menunjukkan proses pengecekan apakah kata-kata dari data tidak terdapat pada *array stoplist* atau daftar *stopword*. Apabila iya, maka kata-kata tersebut dapat dilanjutkan pada proses selanjutnya.

5.3.4 Implementasi *Stemming*

Stemming merupakan tahap terakhir dalam *preprocessing text*. Tahap ini bertujuan untuk menghapus imbuhan kata pada setiap kata dalam seluruh data. Implementasi *stemming* ditunjukkan pada Kode Program 5.4.

```
1  def stemming(self, term):
2      stemmer = self.factory.create_stemmer()
3      term = stemmer.stem(term)
4      return term
```

Kode Program 5.4 Implementasi *Stemming*

Uraian potongan Kode Program 5.4 tentang implementasi *stemming* ditunjukkan sebagai berikut.

1. Pada baris 2 menunjukkan proses pembuatan *stemmer* menggunakan *library* Sastrawi.

2. Pada baris 3 menunjukkan proses penghapusan imbuhan pada kata-kata dengan *stemmer* yang telah dibuat.

5.4 Pembobotan *Term* atau Kata

Pembobotan *term* atau kata merupakan tahapan selanjutnya setelah *preprocessing text* dalam sistem pada penelitian ini. Tujuan dari tahap ini adalah untuk memberi bobot pada setiap kata hasil dari preprocessing text berdasarkan frekuensi kemunculannya di satu dokumen, banyak dokumen, atau gabungan dari keduanya. Tahapan implementasi pembobotan term atau kata terdiri dari implementasi pembentukan kata unik, perhitungan W_{tf}, IDF, dan TF-IDF.

5.4.1 Implementasi Pembentukan Kata Unik

Pembentukan kata unik merupakan tahapan pertama dalam implementasi pembobotan *term* atau kata. Proses ini bertujuan untuk mengambil kata-kata yang unik hasil *preprocessing text* dari seluruh dokumen. Kata-kata unik ini akan berguna untuk perhitungan pembobotan term atau kata. Proses pembentukan kata unik dapat dilihat pada Kode Program 5.5.

```
1  def termsunique(self,allterms):
2      unik = []
3      for terms in allterms:
4          for term in terms:
5              if term not in unik:
6                  unik.append(term)
7      return unik
```

Kode Program 5.5 Implementasi Pembentukan Kata Unik

Uraian potongan Kode Program 5.5 tentang implementasi pembentukan kata unik ditunjukkan sebagai berikut.

1. Pada baris 3 – 6 menunjukkan proses pengambilan kata-kata hasil preprocessing text dan dimasukkan ke dalam *array* baru dengan aturan apabila kata tersebut sudah pernah masuk ke dalam *array* tersebut tidak bisa masuk lagi ke dalam array tersebut. Sehingga akan terbentuk *array* yang hanya berisi kata-kata unik.

5.4.2 Implementasi Perhitungan W_{tf}

Perhitungan W_{tf} merupakan implementasi perhitungan pertama dalam pembobotan *term* atau kata. Tujuan dari proses ini yaitu untuk menghitung frekuensi kemunculan suatu kata dalam satu dokumen dan bobot logaritmanya. Implementasi perhitungan Wtf dapat dilihat pada Kode Program 5.6.

```
1  def calculatewtf(self, allterms, termsunique):
2      wtfdoc = []
3      for i in allterms:
4          wtf = {}
5          for a in termsunique:
6              vtf = i.count(a)
7              if vtf > 0:
8                  vwtf = 1 + math.log10(vtf)
9              else:
10                 vwtf = 0
11             wtf.update({'%s' % (a): vwtf})
12         wtfdoc.append(wtf)
13     return wtfdoc
```

Kode Program 5.6 Implementasi Perhitungan W_{tf}

Uraian potongan Kode Program 5.6 tentang implementasi perhitungan W_{tf} ditunjukkan sebagai berikut.

1. Pada baris 3 – 12 menunjukkan proses perhitungan W_{tf}. Pada baris itu terdapat perulangan seluruh dokumen dan seluruh kata unik. Pada baris 6 menunjukkan proses perhitungan jumlah kata yang sama dalam satu dokumen. Kata tersebut berasal dari kumpulan kata uni yang telah dibentuk sebelumnya. Sedangkan pada baris 8 – 10 menunjukkan proses pembobotan logaritmanya apabila jumlah frekuensi kemunculan kata lebih dari 0, namun apabila tidak maka pembobotan logaritmanya juga 0. Kemudian hasil pembobotan W_{tf} akan disimpan dalam baris ke 11 dan 12.

5.4.3 Implementasi perhitungan IDF

Perhitungan IDF merupakan implemetasi perhitungan selanjutnya setelah perhitungan W_{tf} dalam pembobotan *term* atau kata. Proses ini bertujuan untuk menghitung frekuensi kemunculan kata pada banyak dokumen dan mencari nilai kebalikan atau *inverse*-nya. Implementasi perhitungan IDF dapat dilihat pada Kode Program 5.7.

```
1  def calculateidf(self, alltf, termsunique):
2      idf = {}
3      for a in termsunique:
4          vdf = sum(1 for b in alltf if a in b and b.get(a) > 0)
5          vidf = math.log10(len(alltf) / float(vdf))
6          idf.update({a: vidf})
7      return idf
```

Kode Program 5.7 Implementasi Perhitungan IDF

Uraian potongan Kode Program 5.7 tentang implementasi perhitungan IDF ditunjukkan sebagai berikut.

1. Pada baris 3 – 6 menunjukkan proses perhitungan IDF dengan melakukan perulangan sebanyak kata unik yang telah terbentuk di awal. Pada baris 4 menunjukkan proses perhitungan jumlah dokumen yang mengandung kata yang berasal dari kata unik. Sedangkan baris 5 menunjukkan proses perhitungan nilai *inverse* dari jumlah dokumen yang telah dihitung diatasnya lalu disimpan pada baris 6.

5.4.4 Implementasi perhitungan TF-IDF

Perhitungan TF-IDF merupakan tahapan perhitungan terakhir dalam pembobotan *term* atau kata. Proses ini bertujuan untuk mendapat bobot kata terakhir berupa gabungan nilai W_{tf} yang merupakan bobot logaritma dari frekuensi kemunculan kata dalam satu dokumen dan nilai IDF yang merupakan nilai *inverse* dari frekuensi kemunculan kata di banyak dokumen. Implementasi perhitungan TF-IDF dapat dilihat pada Kode Program 5.8.

```
1  def calculatetfidf(self,allterms):
2      termsunique = self.termsunique(allterms)
3      allwtf = self.calculatewtf(allterms,termsunique)
4      idf = self.calculateidf(allwtf,termsunique)
```

```
5       tfidfdoc  = []
6        for tf in allwtf:
7            tfidf = {}
8            for a in idf:
9                vtfidf = idf.get(a) * tf.get(a)
10               tfidf.update({'%s' % (a): vtfidf})
11           tfidfdoc.append(tfidf)
12      return tfidfdoc
```

Kode Program 5.8 Implementasi Perhitungan TF-IDF

Uraian potongan Kode Program 5.8 tentang implementasi perhitungan TF-IDF ditunjukkan sebagai berikut.

1. Pada baris 2 menunjukkan pemanggilan fungsi untuk membentuk kata unik dari seluruh data.
2. Pada baris 3 menunjukkan pemanggilan fungsi untuk menghitung W_{tf} pada seluruh data.
3. Pada baris 4 menunjukkan pemanggilan fungsi untuk menghitung IDF pada seluruh data.
4. Pada baris 6 – 11 menunjukkan proses perulangan sebanyak jumlah kata unik yang telah dihitung bobot W_{tf} dan jumlah dokumen. Pada baris 9 menunjukkan proses perhitungan TF-IDF berupa perkalian antara nilai W_{tf} dan nilai IDF kemudian hasilnya akan disimpan dalam *array* pada baris 10 dan 11.

5.5 Klasifikasi *K-Nearest Neighbor*

Klasifikasi K-Nearest Neighbor merupakan tahapan paling akhir dalam sistem klasifikasi kepribadian ini. Proses ini bertujuan untuk mendapatkan kelas hasil prediksi untuk setiap data uji berdasarkan tetangga atau kelas data latih yang terdekat. Implementasi klasifikasi KNN terdiri dari perhitungan dot product, panjang dokumen, cosine similarity, seleksi nilai k, dan penentuan klasifikasi akhir KNN.

5.5.1 Implementasi Hitung *Dot Product*

Hitung *dot product* merupakan tahapan perhitungan paling awal dalam implementasi klasifikasi KNN. Pada tahap ini dilakukan proses perkalian nilai TF-IDF antara data latih dan data uji kemudian dijumlahkan per dokumen latih. Implementasi perhitungan *dot product* dapat dilihat pada Kode Program 5.9.

```
1   def dotproduct(self,fiturlatih,fituruji):
2       total = []
3       for a in range(0, len(fituruji)):
4           uji = fituruji[a].values()
5           jml = []
6           for b in range(0, len(fiturlatih)):
7               latih = fiturlatih[b].values()
8               jumlah = 0
9               for c in range(0, len(fiturlatih[0])):
10                  hasil = uji[c] * latih[c]
11                  jumlah += hasil
12              jml.append(jumlah)
13          total.append(jml)
```

```
14      return total
```

Kode Program 5.9 Implementasi Hitung *Dot Product*

Uraian potongan Kode Program 5.9 tentang implementasi perhitungan *dot product* ditunjukkan sebagai berikut.

1. Pada baris 3 – 13 menunjukkan perulangan sebanyak jumlah data uji, jumlah data latih, dan jumlah fitur kata dari seluruh data. Pada baris 8 – 11 menunjukkan perhitungan dot product berupa perkalian nilai TF-IDF dari fitur kata data uji dan nilai TF-IDF dari fitur kata data latih kemudian dijumlah berdasarkan per dokumen atau data kemudian disimpan ke *array* pada baris 12 dan 13.

5.5.2 Implementasi Hitung Panjang Dokumen

Hitung panjang dokumen merupakan tahapan implementasi perhitungan selanjutnya setelah perhitungan dot product dalam implementasi klasifikasi KNN. Proses ini bertujuan untuk mendapatkan nilai panjang dari setiap dokumen baik pada data uji maupun data latih. Implementasi hitung panjang dokumen dapat dilihat pada Kode Program 5.10.

```
1   def panjangdokumen(self,tfidf):
2       pd = []
3       for a in range(0, len(tfidf)):
4           fitur = tfidf[a].values()
5           total = 0
6           for b in range(0, len(fitur)):
7               kuadrat = (fitur[b])**2
8               total += kuadrat
9           akar = math.sqrt(total)
10          pd.append(akar)
11      return pd
```

Kode Program 5.10 Implementasi Hitung Panjang Dokumen

Uraian potongan Kode Program 5.10 tentang implementasi perhitungan panjang dokumen ditunjukkan sebagai berikut.

1. Pada baris 3 - 10 menunjukkan perulangan sebanyak jumlah dokumen atau data dan jumlah fitur kata dari data tersebut. Pada baris 5 – 9 menunjukkan perhitungan panjang dokumen dengan menguadratkan nilai TF-IDF dari setiap fitur kata kemudian dijumlah per dokumen atau data dan diakar. Kemudian hasil dari perhitungan tersebut akan disimpan ke *array* pada baris 10.

5.5.3 Implementasi Hitung *Cosine Similarity*

Hitung *cosine similarity* merupakan tahapan implementasi selanjutnya setelah perhitungan panjang dokumen dalam implementasi klasifikasi KNN. Proses ini bertujuan untuk mendapatkan nilai tingkat kemiripan antar dokumen atau data berdasarkan nilai *dot product* dan nilai panjang dokumen. Implementasi hitung *cosine similarity* dapat dilihat pada Kode Program 5.11.

```
1   def cosinesimilarity(self, fiturlatih, fituruji):
2       arrcossim = []
3       dotproduct = self.dotproduct(fiturlatih,fituruji)
```

```
4       pdoklatih = self.panjangdokumen(fiturlatih)
5       pdokuji = self.panjangdokumen(fituruji)
6       for x in range(0, len(dotproduct)):
7           row = []
8           for y in range(0, len(dotproduct[x])):
9               if pdoklatih[y] == 0 or pdokuji[x]==0:
10                  cossim = 0
11              else:
12                  cossim = dotproduct[x][y]/(pdoklatih[y]*pdokuji[x])
13              row.append(cossim)
14          arrcossim.append(row)
15      return arrcossim
```

Kode Program 5.11 Implementasi Hitung *Cosine Similarity*

Uraian potongan Kode Program 5.11 tentang implementasi perhitungan *cosine similarity* ditunjukkan sebagai berikut.

1. Pada baris 3 menunjukkan pemanggilan fungsi untuk menghitung nilai *dot product* dari seluruh data.
2. Pada baris 4 menunjukkan pemanggilan fungsi untuk menghitung panjang dokumen untuk data latih.
3. Pada baris 5 menunjukkan pemanggilan fungsi untuk menghitung panjang dokumen untuk data uji.
4. Pada baris 6 – 14 menunjukkan perulangan sebanyak jumlah data uji dan data latih. Pada baris 9 menunjukkan kondisi apabila nilai panjang dokumen dari data latih atau data uji sama dengan 0, maka nilai cosine similarity sama dengan 0. Namun apabila panjang dokumen dari dua data tersebut tidak sama dengan 0, maka akan dilakukan proses perhitungan *cosine similarity* dengan persamaan nilai *dot product* dibagi dengan hasil perkalian dari panjang dokumen dari data latih dan panjang dokumen dari data uji. Hasil dari perhitungan tersebut disimpan ke dalam *array* pada baris 13 dan 14.

5.5.4 Implementasi Seleksi Nilai K

Seleksi nilai k merupakan tahapan selanjutnya setelah seluruh tahapan perhitungan dilakukan di dalam implementasi klasifikasi KNN. Proses ini bertujuan untuk memperoleh data latih dengan nilai *cosine similarity* tertinggi sebanyak jumlah nilai k untuk setiap data uji. Implementasi seleksi nilai k dapat dilihat pada Kode Program 5.12.

```
1   def seleksinilaik(self, arrcossim, kelas):
2       nilaik = 3
3       arrselcossim = []
4       arrselkelas = []
5       for x in range(0, len(arrcossim)):
6           sortrow = sorted(arrcossim[x],reverse=True)
7           selcossim = sortrow[:(nilaik)]
8           arrselcossim.append(selcossim)
9           rowselkelas = []
10          for i in range(0, nilaik):
11              selid = arrcossim[x].index(selcossim[i])
12              selkelas = kelas[selid]
13              rowselkelas.append(selkelas)
14          arrselkelas.append(rowselkelas)
15      return arrselcossim, arrselkelas
```

Kode Program 5.12 Implementasi Seleksi Nilai K

Uraian potongan Kode Program 5.12 tentang implementasi seleksi nilai k ditunjukkan sebagai berikut.

1. Pada baris 2 menunjukkan deklarasi dan inisialisasi nilai k sama dengan 3
2. Pada baris 5 – 14 menunjukkan perulangan sebanyak jumlah data uji dan jumlah nilai k.
3. Pada baris 6 menunjukkan proses pengurutan nilai *cosine similarity* dimulai dari yang terbesar hingga yang terkecil.
4. Pada baris 7 menunjukkan pengambilan nilai *cosine similarity* sebanyak jumlah nilai k.
5. Pada baris 11 menunjukkan pengambilan id dari data latih yang memiliki nilai *cosine similarity* tertinggi.
6. Pada baris 12 menunjukkan pengambilan kelas kepribadian yang melekat pada *id* data latih yang telah ditentukan diatasnya.
7. Pada baris 13 menunjukkan penyimpanan kelas-kelas yang telah diambil sebanyak nilai k, kemudian disimpan per data uji pada baris 14.

5.5.5 Implementasi Penentuan Klasifikasi Akhir KNN

Penentuan klasifikasi akhir KNN merupakan tahapan paling akhir dalam implementasi klasifikasi KNN. Prose ini bertujuan untuk menentukan kelas akhir untuk setiap data uji berdasarkan kelas dari data latih yang memiliki nilai cosine similarity tertinggi. Implementasi penentuan klasifikasi akhir KNN dapat dilihatpada Kode Program 5,13.

```
1  def classificationknn(self, arrselkelas, arrselcossim):
2      kelas = ['Artisan','Guardian','Idealist','Rasional']
3      hasilkelas = []
4      for x in range(0, len(arrselkelas)):
5          print (x+1)
6          hasilcossim = []
7          for z in range(0, len(kelas)):
8              totcossim = 0
9              for y in range(0, len(arrselkelas[x])):
10                 if arrselkelas[x][y] == kelas[z]:
11                     totcossim += arrselcossim[x][y]
12             print kelas[z]," = ",totcossim
13             hasilcossim.append(totcossim)
14         maxcossim = max(hasilcossim)
15         id = hasilcossim.index(maxcossim)
16         hasilkelas.append(kelas[id])
17     return hasilkelas
```

Kode Program 5.13 Implementasi Penentuan Klasifikasi Akhir KNN

Uraian potongan Kode Program 5.13 tentang implementasi penentuan klasifikasi akhir KNN ditunjukkan sebagai berikut.

1. Pada baris 2 menunjukkan deklarasi dan inisialisasi *array* kelas kepribadian seperti Artisan, Guardian, Idealist, dan dan Rasional.

2. Pada baris 4 – 16 menunjukkan perulangan sebanyak jumlah data uji, kelas, dan nilai k.
3. Pada baris 5 menampilkan urutan dari data uji.
4. Pada baris 8 – 11 menunjukkan proses penjumlahan nilai *cosine similarity* berdasarkan kelas mereka yang sama.
5. Pada baris 12 menmpilkan hasil dari total *cosine similarity* per kelas.
6. Pada baris 13 menunjukkan proses menyimapan hasil dari total nilai *cosine similarity* berdasarkan kelas ke dalam *array*.
7. Pada baris 14 menunjukkan pencarian nilai total *cosine similarity* yang paling maksimal diantara lainnya.
8. Pada baris 15 menunjukkan pengambilan id dari dari total *cosine similarity* yang paling maksimal dari *array* kelas.
9. Pada baris 16 menunjukkan proses penyimpanan hasil kelas prediksi berdasarkan id total *cosine similarity* yang paling maksimal ke dalam *array*.

5.6 Pengujian

Pengujian merupakan tahapan paling akhir dalam sistem pada penelitian ini. Tujuan dari tahapn ini yaitu untuk mengukur tingkat keakuran suatu sistem dalam melakukan klasifikasi pada suatu masalah tertentu. Implementasi pengujian ini terdiri dari implementasi perhitungan akurasi.

5.6.1 Implementasi Hitung Akurasi

Hitung akurasi merupakan tahapan dari implementasi pengujian. Tujuan dari proses ini yaitu menghitung jumlah data uji yang mendapat hasil klasifikasi dengan benar atau salah serta menghitung tingkat keberhasilan sistem dalam melakukan klasifikasi. Implementasi perhitungan akurasi dapat dilihat pada Kode Program 5.14.

```
1  def akurasi(self,resultsystem,resultdb):
2      true = false = 0.0
3      for x in range(0, len(resultdb)):
4          if resultdb[x] == resultsystem[x]:
5              true += 1
6          else:
7              false += 1
8      result = float(true)/float(true+false)
9      return result
```

Kode Program 5.14 Implementasi Hitung Akurasi

Uraian potongan Kode Program 5.13 tentang implementasi hitung akurasi ditunjukkan sebagai berikut.

1. Pada baris 2 menunjukkan deklarasi dan inisiliasi variabel benar dan salah dengan berisi nilai 0.
2. Pada baris 3 – 7 menunjukkan proses perhitungan jumlah data uji yang menghasilkan klasifikasi dengan benar dan salah.

3. Pada baris 8 menunjukkan perhitungan akurasi dengan membandingkan jumlah data uji yang melakukan klasifikasi dengan hasil kelas yang benar atau tepat dan seluruh jumlah data uji.