# BAB IV
# IMPLEMENTASI DAN PENGUJIAN

## 4.1 Pengumpulan Data

Proses pengumpulan dataset yang digunakan didalam penelitian ini adalah melalui media sosial yaitu Twitter. Tahapan ini melibatkan *library* Python bernama *tweepy*. *Tweepy* adalah salah satu *library* Python yang berfungsi untuk melakukan proses *crawling* Twitter dengan cara mengakses Twitter API. Twitter API adalah antarmuka pemrograman aplikasi yang berfungsi sebagai akses programatik ke data Twitter kepada perusahaan, pengembangan, dan pengguna. Sebelum menggunakan *tweepy*, perlu dilakukan proses instalasi *library* Python dengan mengetikan perintah pada *command prompt*. Dalam hal ini, peneliti melakukan instalasi *library* menggunakan perintah *pip* sebagaimana ditunjukan oleh Gambar 4.1.

```
pip install tweepy
```

Gambar 4.1 Perintah installansi *library tweepy*

Pada proses pengumpulan data, peneliti membuat program *crawling* kecil agar memudahkan dalam pengambilan data *tweet*. Program kecil ini membutuhkan *consumer token* dan *consumer secret* sebagai variabel untuk bisa terhubung dengan API Twitter. *Consumer token* dan *consumer secret* didapatkan dengan cara mendaftarkan aplikasi klien sistem pada halaman *Twitter Developers* di situs *developer.twitter.com*. Proses inisialisasi *consumer token* dan *consumer secret* diimplementasikan ke dalam konstruktor yang terdapat pada Gambar 4.2.

```
def __init__(self, consumer_key=None, consumer_secret=None, access_token=None,
access_secret=None):
	if consumer_key != None:
		self.consumer_key = consumer_key
		self.consumer_secret = consumer_secret
		self.access_token = access_token
		self.access_secret = access_secret
	else:
		self.consumer_key = 'DmHd7uZy5Gr4g7trOOKqphuhW'
	self.consumer_secret='Xfjg012XcljlbmoPOFW4Rxw7hwI77ol06DAHCztaf2nssdjxVZ'
		self.access_token='1951831687-aqX3J0MTGegdFQDdcRXiNUS8cN5dtWNWg9qOIAA'
		self.access_secret = 't2ObeFAUqVTZBS5eFN27aYK6e9K1VVHX2OIgCv4FHoCRJ'
	p.set_options(p.OPT.URL, p.OPT.MENTION, p.OPT.HASHTAG, p.OPT.RESERVED,
p.OPT.EMOJI, p.OPT.SMILEY, p.OPT.NUMBER)
	self.resultDump = []
```

Gambar 4.2 Inisialisasi *consumer token* dan *consumer secret*

Selama melakukan *crawling*, *tweet* yang diambil adalah *tweet* yang berdasarkan *timeline* yang terdapat di akun Twitter @Lambe_Turah. Data *tweet* yang diambil pada akun @Lambe_Turah ini, terakhir kali diambil pada tanggal 24 April 2019. Data tweet yang dikumpulkan akan dianalisis untuk mendapatkan kata-kata yang sering digunakan dalam *tweet* tersebut. Hasil analisis adalah kumpulan kata standar yang digunakan sebagai kamus pada algoritma *Levenshtein Distance.* Selain itu juga daftar kamus diperoleh dari data yang peneliti sudah kumpulkan sebelumnya. Daftar kata ini disimpan dalam sebuah *file* dengan nama *list_crawling.*

Selanjutnya, pengumpulan data *tweet* diimplementasikan melalui fungsi yang diberi nama *crawlingToSentences*. Fungsi *crawlingToSentences* ini bertugas sebagai proses pengambilan *tweet* dengan memasukan parameter teks yang dicari. Sebelum itu fungsi *crawlingToSentences* membutuhkan proses otentikasi dengan API Twitter yang diimplementasikan pada fungsi *authCon.* Otentikasi ini berfungsi sebagai penjembataan saat pengguna melakukan permintaan pengambilan data pada Twitter. Implementasi fungsi *authCon* dapat dilihat juga pada Gambar 4.3.

```
def authCon(self):
	auth = tweepy.OAuthHandler(self.consumer_key, self.consumer_secret)
	auth.set_access_token(self.access_token, self.access_secret)
	api = tweepy.API(auth, wait_on_rate_limit=True)
	return api

def crawlingToSentences(self, text, n ,clean=False):
	data = self.authCon()
	result = []
	for status in tweepy.Cursor(data.search, q=text,tweet_mode= "extended",
lang="in").items(n):
		result.append(status.full_text.lower())
	result_2 = []
	if clean==True:
		for text in result:
			result_2.append(self.preprocess(text))
		self.resultDump = result_2
		return result_2
	else:
		self.resultDump = result
		return result
```

Gambar 4.3 Kode fungsi *authcon* dan fungsi *crawlingtosentences*

Contoh hasil dari implementasi tahap *crawling* dapat dilihat pada Gambar 4.4. Hasil dari implementasi ini disimpan di sebuah *file* dengan format *txt* dan diberi nama *hasil_crawling.* Dari hasil *crawling* dapat dilihat bahwa teks mengandung beberapa data yaitu teks *retweet*, *username*, isi *tweet* dan beberapa *noise* pada isi *tweet* tersebut. Hasil ini masih membutuhkan

tahap *preprocessing* agar mendapatkan teks *tweet* yang diinginkan sebelum ke tahap selanjutnya. *Tweet* yang dikumpulkan dari hasill *crawling* dapat dilihat pada Gambar 4.4.

```
rt @jackvardan: bagus nih banyak yang sorot tanah prabowo. lagi pula hgu dan negara bisa ambil kapan aja. sikat semua, setuju kalau
rt @bima_____: walhi tantang jokowi buka kartu, beber orang-orang di tim suksesnya yang kuasai lahan negara
https://t.co/9nqnzfo8hc
rt @panca66: detik juga sudah berani menulis kebohongan jokowi #jokowibohonglagi https://t.co/2avz1t4vcq
rt @manuel_agil: pak @jokowi serius dalam masalah menyatukan setiap kota, daerah, dan provinsi. .
dan pak #jokowimemberikanbukti dari setia…
orang di sekitar jokowi banyak yang kuasai tanah negara, yang mana pak riza? https://t.co/wz8yrkmu0q
rt @kafiradikal: ...ya ampun parah banget level kebodohan fans jokowi ini ya tuhan yesus kristus...!!
diksi "impor air" aja dengan bejibu…
rt @ekowboy: debat pertama. menyerang gerindra calonkan caleg mantan napi koruptor
debat kedua. menyerang kepemilikan lahan prabowo yang d…
@asepmisbahudin5 @in_doni_sia @ardani72683507 @jokowi @prabowo wkwk
pak jokowi telah membangun jalan tol, pelabuhan dan juga airport yang sudah dikerjakan dan akan diteruskan lebih banyak lagi
#jokowimenangprabowonyerah
rt @tanyoana: @sudirmansaid @cumarachel @jokowi bicara soal tanah hgu yang di kelola perusahaan @prabowo
tapi dia lupa cukong-cukong di se…
```

Gambar 4.4 Teks hasil implementasi *crawling*

## 4.2 Implementasi *Preprocessing*

*Preprocessing* diimplementasikan dengan membuat beberapa fungsi yang mewakili setiap proses yang dilakukan dalam *preprocessing* antara lain fungsi *cleanrt* untuk menghapus *retweet* pada status Twitter, *remove_punctuation* untuk menghapus tanda baca yang tidak diperlukan lagi, dan *preprocess* yang mewakili beberapa pembersihan *noise* lainnya. Gambar 4.5 menunjukan implementasi *preprocessing* untuk data *tweet* hasil *crawling* yang didapat.

```
def cleanRT(self, s):
	return re.sub(r'^rt[\s]+', '', s, flags=re.MULTILINE)
def remove_punctuation(self, s):
	for t in list(punctuation):
		s = s.replace(t,'')
	return self.cleanRT(s)
def preprocess(self, text):
	text = p.clean(text)
	text = self.remove_punctuation(text)
	text = " ".join(text.split())
	text = text.replace(' - ', '-')
	text = re.sub(r'[^\x00-\x7F]+',' ', text)
	text = self.replaceWithMeaning(self.slangword, self.abbreviation,
text.split())
	return text
```

Gambar 4.5 Kode program beberapa fungsi *preprocessing*

a. *Case Folding*

Hasil implementasi *case folding* dapat dilihat pada Tabel 4.1. Pada tabel sebelah kiri adalah teks sebelum dilakukan *case folding* dan tabel sebelah kanan adalah teks yang sudah

dilakukan *case folding*. Pada tabel dapat dilihat contoh sebagian data *tweet* yang diambil untuk dilakukan *case folding*. *Case folding* disini mengubah semua karakter huruf menjadi kecil.

Tabel 4.1 Hasil *preprocessing* tahap *case folding*

| ***Tweet*** | ***Tweet* setelah dilakukan case folding** |
|---|---|
| @antaresoh Noh dua biar afdol https://t.co/2yaHSsDWWV | @antaresoh noh dua biar afdol https://t.co/2yahssdwwv |
| Kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal.... | kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal.... |
| # dewi afdol.... menemani anda di Palapa Goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin... https://t.co/5QlchaaiUn | # dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin... https://t.co/5qlchaaiun |
| puasa udh semakin deket, ada yang punya acara ghibah closingan ga? | puasa udh semakin deket, ada yang punya acara ghibah closingan ga? |
| biar afdol aja ntr tobatnya<br>Kangmas mbakyu | biar afdol aja ntr tobatnya<br>kangmas mbakyu |
| tanggal muda itu seneng opo malah bingung alesane opo jal.... | tanggal muda itu seneng opo malah bingung alesane opo jal.... |
| # dewi afdol.... menemani anda di Palapa Goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin hepi... https://t.co/wya7gJlSmd | # dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin hepi... https://t.co/wya7gjlsmd |
| @YG_TREASURE13 kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim | @yg_treasure13 kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim |
| makanya kita nikah aja kak biar afdol...<br>@mxconfess Yukk | makanya kita nikah aja kak biar afdol...<br>@mxconfess yukk |
| @RPEBASE Yukk | @rpebase yukk |

b. Menghapus URL

Hasil implementasi Menghapus URL dapat dilihat pada Tabel 4.2. Pada tabel sebelah kiri adalah teks sebelum dilakukan penghapusan URL dan tabel sebelah kanan adalah teks yang sudah dibersihkan URL nya. URL yang dihapus dapat dilihat pada tabel seperti "https://t.co/2yahssdwwv", "https://t.co/5qlchaaiun", dan "https://t.co/wya7gjlsmd".

Tabel 4.2 Hasil *preprocessing* tahap menghapus URL

| ***Tweet*** | ***Tweet* setelah dibersihkan URL** |
|---|---|
| @antaresoh noh dua biar afdol https://t.co/2yahssdwwv<br>kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal.... | @antaresoh noh dua biar afdol<br>kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal.... |

| | |
|---|---|
| # dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin... https://t.co/5qlchaaiun<br>puasa udh semakin deket, ada yang punya acara ghibah closingan ga?<br>biar afdol aja ntr tobatnya<br>kangmas mbakyu<br>tanggal muda itu seneng opo malah bingung alesane opo jal....<br># dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin hepi... https://t.co/wya7gjlsmd<br>@yg_treasure13 kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim<br>makanya kita nikah aja kak biar afdol...<br>@mxconfess yukk<br>@rpebase yukk | # dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin...<br>puasa udh semakin deket, ada yang punya acara ghibah closingan ga?<br>biar afdol aja ntr tobatnya<br>kangmas mbakyu<br>tanggal muda itu seneng opo malah bingung alesane opo jal....<br># dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin hepi...<br>@yg_treasure13 kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim<br>makanya kita nikah aja kak biar afdol...<br>@mxconfess yukk<br>@rpebase yukk |

c. Menghapus *Emoticon*, *Symbol* dan Tanda Baca

Hasil *crawling* data *tweet* yang diperoleh juga perlu dilakukan pembersihan pada *emoticon*, *symbol*, dan tanda baca pada teksnya. Karena sudah menjadi ciri khas teks *tweet* bila terdapat banyak *noise* tersebut. Hasil implementasi dari menghapus *emoticon*, *symbol* dan tanda baca dapat dilihat pada Tabel 4.3. Tabel sebelah kiri menunjukan teks yang belum dibersihkan dan tabel sebelah kanan adalah hasil dari pembersihan tersebut.

Tabel 4.3 Hasil *preprocessing* tahap menghapus *emoticon*, *symbol*, dan tanda baca

| ***Tweet*** | ***Tweet* setelah dihapus *emoticon*, *symbol*, dan tanda baca** |
|---|---|
| @antaresoh noh dua biar afdol<br>kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal....<br># dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin...<br>puasa udh semakin deket, ada yang punya acara ghibah closingan ga?<br>biar afdol aja ntr tobatnya<br>kangmas mbakyu<br>tanggal muda itu seneng opo malah bingung alesane opo jal....<br># dewi afdol.... menemani anda di palapa goyang sepanjang hari di frekuwensi 90.5, full lagu dangdut yang bikin hepi... | noh dua biar afdol<br>kangmas mbakyu<br>paling asik piknik itu naik motor opo bis yaa alesane opo jal<br>dewi afdol menemani anda di palapa goyang sepanjang hari di frekuwensi full lagu dangdut yang bikin<br>puasa udh semakin deket ada yang punya acara ghibah closingan ga<br>biar afdol aja ntr tobatnya<br>kangmas mbakyu<br>tanggal muda itu seneng opo malah bingung alesane opo jal<br>dewi afdol menemani anda di palapa goyang sepanjang hari di frekuwensi full lagu dangdut yang bikin hepi |

| | |
|---|---|
| @yg_treasure13 kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim<br>makanya kita nikah aja kak biar afdol... | kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim<br>makanya kita nikah aja kak biar afdol |

d. Menghapus karakter berulang (*char repetition*)

Data *tweet* hasil *crawling* banyak ditemukan kata yang memiliki karakter berulang berurutan. Kata yang digunakan dalam penulisan ini biasanya untuk menekankan kata yang ditulis pada status *tweet*. Penggunaan karakter berulang berurutan hanya dibatasi maksimal dua karakter, sehingga apabila lebih dari dua karakter selebihnya akan dihapus. Contoh kata dengan karakter berulang yang sudah ditemukan dapat dilihat pada Tabel 4.4.

Tabel 4.4 Contoh kata dengan karakter berulang

| Kata | Hasil perbaikan |
|---|---|
| maaaaaap | maap |
| suaranyaaaa | suaranyaa |
| yaaaaaaa | yaa |
| nddddutttttt | nddutt |
| wuaddddduhhhhh | wuadduhh |
| asyeeeeqqq | asyeeqq |
| pulaaangggg | pulaangg |
| bingungggggg | bingungg |
| duhhhh | duhh |
| naaaaahhhh | naahh |
| mariiiii | marii |
| jadiiiii | jadii |

## 4.3 Implementasi Deteksi *Out of Vocabulary*

### 4.3.1 Implementasi Pengecekan *Slangword*

*Tweet* yang memiliki *noise* dan sudah berhasil dibersihkan, maka selanjutya akan melalui tahap pengecekan *slangword* dengan kamus yang sudah dikumpulkan pada *file* berformat *txt* dengan nama *slang_word_list.txt*. Apabila saat pengecekan kata tidak ditemukan di kamus, maka akan diabaikan dan dilanjutkan pada tahap pengecekan kata singkatan. Implementasi pengecekan *slangword* dapat dilihat pada Gambar 4.6.

```
def replaceWithMeaning(self, words):
	new_words = []
	for i in words:
		if i in self.slangword.keys():
			new_words.append(self.slangword[i])
		else:
			new_words.append(i)
		return " ".join(new_words)
```

Gambar 4.6 Kode program pengecekan *slangword*

Adapun contoh *slangword* yang ditemukan pada data *tweet* dan perbaikannya dapat dilihat pada Tabel 4.5.

Tabel 4.5 Contoh kata *slangword* yang didapat pada data *tweet*

| ***Slangword*** | **Perbaikan *slangword*** |
|---|---|
| Bijimane | bagaimana |
| Chuyunk | sayang |
| Gile | gila |
| Ashyap | ah siap |
| Ciyus | serius |
| Mevvah | mewah |

### 4.3.2 Implementasi Pengecekan Singkatan

Tahap selanjutnya pada deteksi OOV adalah tahap pengecekan singkatan kata di *tweet*. Pengecekan singkatan memiliki implementasi yang sama pada pengecekan *slangword*, yaitu mencocokan di kamus singkatan, apabila kata terdapat di kamus maka akan diganti dengan kata yang sesungguhnya, jika kata terdapat pada kamus maka tidak akan diabaikan. Implementasi pengecekan singkatan dan pengecekan *slangword* berbeda hanya pada *file* yang dipisah saja. *File* singkatan yang sudah dikumpulkan disimpan pada *file txt* dan diberi nama *singkatan_list.txt*. Implementasi pengecekan singkatan dapat dilihat pada Gambar 4.7.

```
def replaceAbbreviation(self, words):
	new_words = []
	for i in words:
		if i in self.abbreviation.keys():
			new_words.append(self.abbreviation[i])
		else:
			new_words.append(i)
	return " ".join(new_words)
```

Gambar 4.7 Kode program implementasi pengecekan singkatan

Adapun contoh singkatan yang ditemukan pada data *tweet* dan perbaikannya dapat dilihat pada Tabel 4.6.

Tabel 4.6 Contoh kata singkatan yang didapat pada data *tweet*

| Singkatan | Perbaikan singkatan |
|---|---|
| Nnt | nanti |
| Ntr | sebentar |
| Sy | saya |
| Sbntr | sebentar |
| Drpd | daripada |
| Dsb | dan sebagainya |

### 4.3.3 Implementasi Pengecekan Kata Berulang

Tahap terakhir pendeteksi OOV adalah mengecekan kata berulang. Tahap pengecekan kata berulang diimplementasikan pada Gambar 4.8. Pada tahap ini dilakukan pengecekan pada kata berulang yang ditulis dengan singkat.

```
def replace2(teks):
	replace = []
	for i in teks.split():
		if i[-1:] == "2" and len(i)>3:
			replace.append(i[:-1]+"-"+i[:-1])
		else:
			replace.append(i)
	return " ".join(replace)
```

Gambar 4.8 Kode program implementasi pengecekan kata berulang

Pengecekan dilakukan dengan memeriksa setiap kata di data *tweet.* Kata yang akan diperbaiki akan diperiksa karakter pada akhir kata, kata berulang yang disingkat diakhiri dengan karakter “2” pada akhiran kata. Setelah terdekteksi karakter “2” pada akhir kata, selanjutnya karakter “2” akan dihapus dan diganti dengan kata berulang kata tersebut dibarengi dengan karakter “-”.

Adapun kata yang ditemukan dalam *tweet* terutama dalam penanganan kata berulang dapat diperlihatkan pada Tabel 4.7. Contoh pada Tabel 4.7 adalah kata berulang yang peneliti temukan dalam data *tweet* akun @Lambe_Turah.

Tabel 4.7 Perbaikan kata berulang yang ditemukan pada data *tweet*

| Kata | Perbaikan kata berulang |
|---|---|
| baik2 | baik-baik |
| orang2 | orang-rang |
| mereka2 | mereka-mereka |
| tiba2 | tiba-tiba |
| aneh2 | aneh-aneh |

## 4.4 Implementasi Normalisasi Algoritma *Levenshtein Distance*

Implementasi algoritma *Levenshtein Distance* dapat dilihat pada Gambar 4.9. Gambar menunjukan *source code* implementasi algoritma *Levenstein Distance* untuk menghitung jarak antara dua *string* yang berbeda. Ada tiga operasi yang dilakukan pada *source code* yaitu penyisipan, penghapusan, dan permutasi. Algoritma *Leveshtein Distance* bekerja untuk menghitung jumlah minimum mentransformasikan suatu *string* menjadi *string* lain. Tahap mentransformasikan dapat merupakan penggantian, penghapusan, dan penyisipan pada kata.

```
def levenshteinDistance(self, s1, s2):
    if len(s1) > len(s2):
        s1, s2 = s2, s1
    distances = range(len(s1) + 1)
    for i2, c2 in enumerate(s2):
        distances_ = [i2+1]
        for i1, c1 in enumerate(s1):
            if c1 == c2:
                distances_.append(distances[i1])
            else:
                distances_.append(1 + min((distances[i1], distances[i1 + 1],
distances_[-1])))
        distances = distances_
    return distances[-1]
```

Gambar 4.9 Kode program implementasi *Levenshtein Distance*

Setiap kata di kamus akan dihitung dengan *string* masukan untuk mendapatkan *edit distance*, proses tersebut diimplementasikana pada Gambar 4.10. Proses pada tahap ini mengimplementasikan dengan menghitung *edit distance* dari setiap kamus dan memasukannya pada kandidat *distance* dengan *threshold* sebesar 3.

```
def find_closest_word(self, s, dictionary):
    distances = []
    for key in dictionary:
        distance = self.levenshteinDistance(s, key)
        #distance = qwerty_LD(s, key,1,1)
        if distance < 3:
            distances.append(key)
    return distances
```

Gambar 4.10 Kode program ambang batas *Levenshtein Distance*

Gambar 4.11 adalah implementasi untuk pemeriksaan setiap kata yang akan diperbaiki dengan kamus yang dimiliki. Kata yang akan diperbaiki akan dicocokan pada kamus. Apabila, kata terdapat pada kamus maka kata akan dikembalikan. Jika, kata tidak terdapat kamus maka kata akan diperbaiki dengan memanggil fungsi *Levenshtein Distance.* Selain perbaikan fungsi pada Gambar 4.11 juga dibuat pengurutan untuk menampilkan rekomendasi kata dari *edit distance* terkecil.

```
def test_LD(self, word):
            dic = self.openFile()
            result = []
            data1 = []
            data2 = []
            data3 = []
            if word in dic:
                  return [word]
            else:
                  a = self.find_closest_word(word, dic)
            for i in a:
                  if i[1]==1: data1.append(i[0])
                  elif i[1]==2:data2.append(i[0])
                  else:  data3.append(i[0])
            if len(data1) > 0:
                  #print([word, data1])
                  data1.sort(key=len, reverse = True)
                  return [word, data1]
            elif len(data2) > 0:
                  #print([word, data2])
                  data2.sort(key=len, reverse = True)
                  return [word, data2]
            else:
                  #print([word, data3])
                  data3.sort(key=len, reverse = True)
                  return [word, data3]
```

Gambar 4.11 Kode program pencocokan kata pada kamus dan perbaikan kata

Tabel 4.8 menunjukan hasil implementasi dari kode program yang sudah dijalankan. Hasil ini adalah hasil yang sudah melalui tahap *preprocessing* yang sudah dijelaskan sebelumnya dan tahap deteksi OOV, sehingga kata-kata yang diperbaiki oleh algoritma ini adalah yang benar-benar kata salah ejaan.

Tabel 4.8 Hasil perbaikan sampel *tweet* menggunakan *Levenshtein Distance*

| Teks | Teks setelah dinormalisasi |
|---|---|
| noh dua biar afdol | aneh dua biar afdal |
| kangmas mbakyu | kangmas mbakyu |
| paling asik piknik itu naik motor opo bis yaa | paling asyik piknik itu naik motor bopok bis ya |
| alesane opo jal | alasan bopok jalan |
| dewi afdol menemani anda di palapa goyang sepanjang hari di frekuwensi full lagu dangdut yang bikin | dewi afdal menemani kamu di palapa goyang sepanjang hari di frekuensi fulus lagu dangdut yang bikin |

| | |
|---|---|
| puasa udh semakin deket ada yang punya acara ghibah closingan ga<br>biar afdol aja ntr tobatnya<br>kangmas mbakyu<br>tanggal muda itu seneng opo malah bingung alesane opo jal<br>dewi afdol menemani anda di palapa goyang sepanjang hari di frekuwensi full lagu dangdut yang bikin hepi<br>kak kl pagi pagi aku dikasi konten trs aku kepikiran kakak2 mulu nnti puasa aku gk afdol krn mikirin yg belum muhrim<br>makanya kita nikah aja kak biar afdol<br>yukk<br>yukk | puasa sudah semakin dekat ada yang punya acara khitbah closingan tidak<br>biar afdal saja sebentar tobatnya<br>kangmas mbakyu<br>tanggal muda itu senang bopok bahkan bingung<br>alasan bopok jalan<br>dewi afdal menemani kamu di palapa goyang sepanjang hari di frekuensi fulus lagu dangdut yang bikin nyepi<br>kak kalau pagi pagi saya dikasi konten terus saya terpikirkan kakang melulu nanti puasa saya tidak afdal karena pikiran yang belum muhrim<br>karenanya kita nikah saja kak biar afdal<br>kayak kayak |

## 4.5 Pengujian Dan Evaluasi

### 4.5.1 Pengujian Pada Algoritma *Levenshtein Distance*

Pengujian yang dilakukan mengambil sampel data *crawling* sebanyak 90 *tweet* dan ditemukan sebanyak 106 kata yang terdeteksi dapat diperbaiki oleh *Levenshtein Distance*. Berikut adalah tabel hasil kata yang berhasil dikumpulkan beserta hasil dari normalisasi dengan menggunakan algoritma *Levenshtein Distance*, kata yang berhasil dinormalisasi dapat dilihat pada Tabel 4.9. Pada tabel ini dapat ditemukan sebanyak 71 kata yang dapat diperbaiki secara benar. Kata kata pada tabel tersebut adalah kata hasil dari perbaikan *Levenshtein Distance* yang berhasil diperbaiki sesuai kata yang benar atau memprediksi dengan target yang benar.

Tabel 4.9 Hasil pengujian *Levenshtein Distance* dengan target yang benar

| No. | Kata | Prediksi | No. | Kata | Prediksi |
|---|---|---|---|---|---|
| 1 | Waahhhh | wah | 37 | neh | nih |
| 2 | maaah | mah | 38 | Duhhh | duh |
| 3 | TSAHHHH | sah | 39 | Maaah | mah |
| 4 | Selamaaaat | selamat | 40 | Abiezzz | abis |
| 5 | Idol | idola | 41 | Abies | abis |
| 6 | Duhhh | duh | 42 | Duhhh | duh |
| 7 | dechhhh | deh | 43 | Tamvaaaan | tampan |
| 8 | Trusss | terus | 44 | cantiknyaaaa | cantiknya |
| 9 | Embak | mbak | 45 | Ehhhh | eh |
| 10 | Syurhat | curhat | 46 | Eeeh | eh |
| 11 | Yaaah | yah | 47 | Eeeeh | eh |
| 12 | Akhirnyaaa | akhirnya | 48 | Ngidamnyaaa | mengidam |
| 13 | Duhh | duh | 49 | sich | sih |

| | | | | | |
|---|---|---|---|---|---|
| 14 | siapaaa | siapa | 50 | bu | ibu |
| 15 | putusssss | putus | 51 | Bisikin | bisikan |
| 16 | kesha | kesah | 52 | Yaaah | yah |
| 17 | Wuihhhhh | wah | 53 | Asyeeeek | asyik |
| 18 | besket | basket | 54 | Ehh | eh |
| 19 | Yaaaah | yah | 55 | Akhirnyaaaa | akhirnya |
| 20 | sayembaranyaaaa | sayembara | 56 | eMbak | mbak |
| 21 | diumumin | diumumkan | 57 | emesh | gemes |
| 22 | Duhhh | duh | 58 | nonton | tonton |
| 23 | esmosi | smosi | 59 | Efect | efek |
| 24 | sich | sih | 60 | nich | bih |
| 25 | Kumpulll | kumpul | 61 | Yaaah | yah |
| 26 | Kumpulll | kumpul | 62 | Asyeeeek | asyik |
| 27 | Asyeeeek | asyik | 63 | Ehh | eh |
| 28 | Asyeeeeek | asyik | 64 | keserupan | kesurupan |
| 29 | nich | nih | 65 | yaaaah | yah |
| 30 | Duuuhhh | duh | 66 | pilemnya | filmnya |
| 31 | dikirimin | dikirimkan | 67 | pengeeeen | pingin |
| 32 | pidio | video | 68 | Sharapan | sarapan |
| 33 | saksisan | saksikan | 69 | Ahhh | ah |
| 34 | Selamaaaaat | selamat | 70 | Abiesss | abis |
| 35 | TSAHHHH | sah | 71 | nonton | tonton |
| 36 | hureee | hore | | | |
| 37 | penganten | pengantin | | | |

Selain itu dari 106 data uji yang sudah diprediksi, perbaikan kata dengan algoritma ini juga mengalami kegagalan pada beberapa kata. Terdapat 35 kata yang tidak berhasil diperbaiki pada tahap ini. Adapun kata yang perbaikan tidak sesuai target ditunjukan pada Tabel 4.10.

Tabel 4.10 Hasil pengujian *Levenshtein Distance* yang tidak sesuai target

| No. | Kata | No. | Kata |
|---|---|---|---|
| 1 | yeaaaah | 19 | dungs |
| 2 | yaaaaks | 20 | dungs |
| 3 | woless | 21 | dungs |
| 4 | syuting | 22 | doya |
| 5 | sosialita | 23 | disindang |
| 6 | sayembara | 24 | dedek |
| 7 | rektor | 25 | cuzz |
| 8 | pieeee | 26 | cus |
| 9 | ngidam | 27 | ciyussss |
| 10 | minceuu | 28 | ciyeee |

| | | | |
|---|---|---|---|
| 11 | mie | 29 | cans |
| 12 | malmingan | 30 | atutt |
| 13 | lho | 31 | atut |
| 14 | kanjeng | 32 | aqooh |
| 15 | ipar | 33 | ahad |
| 16 | ihhhh | 34 | ngintilin |
| 17 | gengsss | 35 | cuzz |
| 18 | gapapalah | | |

Persentase jumlah kata benar diperoleh dengan cara mengujikan 106 kata yang tidak terdapat pada kamus. Jumlah kata benar tersebut merupakan hasil pendekatan dari metode *Levenshtein Distance*. Berdasarkan tabel yang diperoleh beserta hasil yang didapatkan dengan algoritma ini, nilai akurasi yang diperoleh adalah 66.98% dengan jumlah kata yang benar adalah 71 Kata dari 106 kata yang diuji. Akurasi diperoleh berdasarkan perhitungan berikut:

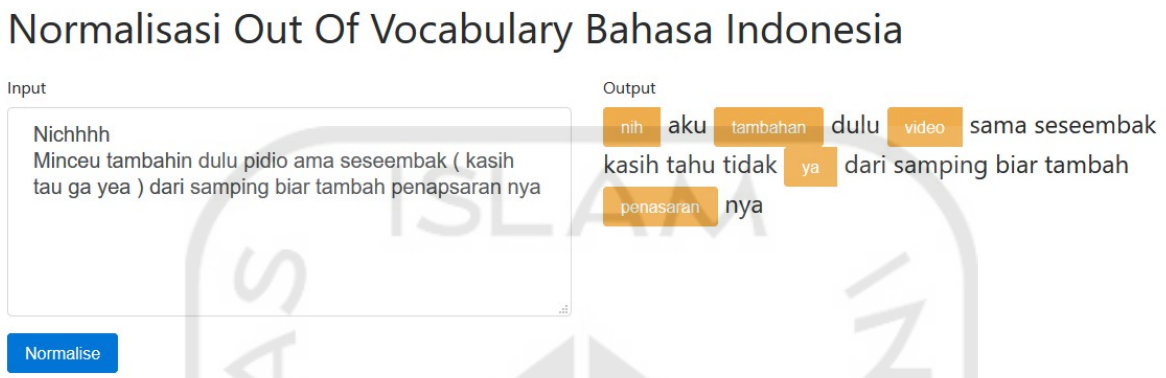
$$Persentase = \frac{71}{106} \times 100\% = 66.98\%$$

Setelah melihat hasil dari tabel, ditemukan banyak kata yang tidak sering atau lazim digunakan dalam bahasa sehari-hari. Hal ini menyebabkan perbaikan kata yang ditampilkan ada yang tidak sesuai dengan seharusnya. Contoh pada Tabel 4.10 adalah kata "yeaaaah" yang seharusnya memiliki perbaikan kata adalah "yes" atau "ya". Namun, prediksi perbaikan kata yang disarankan pada sistem adalah "kearah". Kata "kearah" merupakan kata bahasa Indonesia yang artinya "menuju ke suatu arah" pada situs kamus besar bahasa Indonesia (KBBI). Dengan demikian, makna kata "kearah" sangat berbeda dengan makna yang seharusnya yaitu "yes". Selain itu kata "kearah" kurang sering digunakan di bahasa sehari-hari sehingga akan sering direkomendasikan pada kata selanjutnya. Oleh karena itu perlu ada pemilahan kata pada kamus yang berisi kata-kata yang sering digunakan oleh masyarakat atau perlu ada penilaian probabilitas kata untuk digunakan.

### 4.5.2 Pengujian Normalisasi Pada Antarmuka Sistem

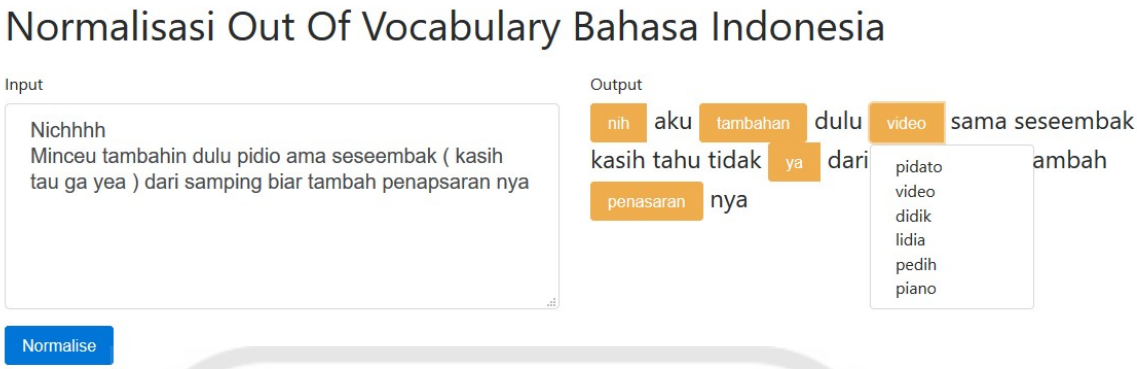
Normalisasi pada antarmuka sistem dilakukan oleh *user* dengan menginputkan teks yang akan dinormalisasi pada halaman utama sistem. Halaman utama sistem ini dibuat dari *framework* Django. *Framework* Django dipilih karena mendukung pengembangan dalam bahasa pemrograman Python. Adapun pengujian antarmuka akan mengambil contoh salah satu

*tweet* yang dipilih dari hasil *crawling* dan dan memasukannya tanpa *preprocessing.* sampel yang diambil untuk evaluasi hanya salah satu *tweet* dari yang sudah dikumpulkan. Karena *preprocessing* akan dilakukan di dalam sistem.




Gambar 4.12 Uji coba pertama sampel teks pada tampilan sistem

Gambar 4.12 menunjukan hasil percobaan sistem menggunakan teks yang didapatkan dari hasil *crawling*. Hasil menunjukan bahwa sistem dapat memperbaiki kata lumayan baik. Sampel yang dinormalisasi terdapat *slangword* seperti kata "tau" yang diperbaiki dengan kamus yang sudah dikumpulkan dan kata yang salah ejaan seperti kata "nichhhh" yang diperbaiki menjadi "nih" dan kata "pidio" yang diperbaiki menjadi "video". Selain itu, kata yang diperbaiki menggunakan *Levenshtein Distance* dapat menampilkan rekomendasi perbaikan kata sesuai kebutuhan pengguna. Untuk memilih rekomendasi kata hanya dengan menekan tombol pada kata yang kurang pengguna kurang tepat di antarmuka. Pemilihan rekomendasi kata dapat dilihat pada Gambar 4.13. Namun, apabila kata tidak terdapat pada kamus maka kata akan dianggap OOV namun tidak bisa memberikan rekomendasi kata tersebut.




Gambar 4.13 Rekomendasi kata hasil *Levenshtein Distance*