BAB VI. PENUTUP

6.1. Kesimpulan

Pada penelitian analisis sentimen dengan metode *text mining* terhadap restoran di masa sebelum dan selama masa *covid-19* di D.I. Yogyakarta memiliki kesimpulan yaitu pelanggan sudah puas terhadap layanan yang diberikan restoran meskipun pada masa pandemi *covid-19* sekalipun. Kepuasan ini dilihat melalui kata dari ulasan pelanggan yang merepresentasikan kepuasan menurut variabel kepuasan dengan menggunakan Skala Likert. Selain itu, aspek produk maupun layanan merupakan aspek utama dalam menentukan kepuasan dan ketidakpuasan pelanggan. Beberapa aspek pendukung lainnya yang turut mempengaruhi yaitu aspek keramahan dan keterampilan pegawai, fasilitas pendukung, kestragisan lokasi, dan arsitektur bangunan restoran. Di sisi lain terdapat aspek yang membuat pelanggan menjadi tidak puas yaitu prosedur pemesanan, harga yang ditawarkan, dan dari segi arsitektur bangunan restoran.

Padandangan dari pelanggan adalah ekspetasi dan penghargaan yang tinggi atas aspek produk maupun layanan. Oleh karena itu, pelanggan akan berfokus pada aspek produk dan layanan dari restoran tersebut. Aspek inilah yang menjadi salah satu *core* dari empat *core* utama bisnis restoran yang terkuat serta sangat berpengaruh sehingga perlu lebih diperhatikan pihak restoran. Penelitian ini dapat dimanfaatkan oleh pihak restoran dalam memahami aspek-aspek kepuasan dan ketidakpuasan pelanggan, sehingga pihak restoran pun dapat memperbaiki, sekaligus meningkatkan kualitas layanan agar tepat dan sesuai dengan ekpetasi serta kebutuhan pelanggan.

6.2. Saran

Dari penelitian yang sudah dilakukan, maka beberapa saran yang ada untuk penelitian berikutnya khususnya terkait *text mining* dalam analisis

sentimen dengan metode *sentistrength* adalah sebagai berikut :

1. Penelitian yang menyangkut analisis sentimen terhadap data ulasan teks bisa melaui berbagai sumber *platform* termasuk melalui media sosial.
2. Pada tahap *pre-processing*, bagian normalisasi dapat lebih diperbanyak dalam kata untuk singkatan, kesalahan ketik, dan lain-lain. Semakin banyak normalisasi kata maka dapat mempermudah dalam mengolah data.
3. Daftar kata yang ada pada kamus atau leksikan berbahasa Indonesia dapat disisipkan kata baru lagi sehingga dapat menghasilkan klasifikasi sentimen yang lebih baik.

LAMPIRAN

1. *Source Code* Pengumpulan Data (*Data Scraping*)

```
import requests
import pandas
from bs4 import BeautifulSoup
import sys

url = 'https://www.tripadvisor.co.id/Restaurant_Review-
g14782503-d7371162-Reviews-Bosskid_Watu_Samudra-
Yogyakarta_Yogyakarta_Region_Java.html'
# front = url[:52]
# back = url[51:]

req = requests.get(url)
soup = BeautifulSoup(req.text, 'html.parser')
dataFrame = pandas.DataFrame()

all_date = []
all_title = []
all_content = []

# Get Title
getTitle = soup.find('h1', {'class': '_3a1XQ88S'})
csvName = getTitle.text

for e in soup.find_all('span', {'class': 'ratingDate'}):
    all_date.append(e.text)

for e in soup.find_all('span', {'class': 'noQuotes'}):
    all_title.append(e.text)

# OLD
# for e in soup.find_all('div', {'class': 'prw_rup
prw_reviews_text_summary_hsx'}):
#     review = soup.find_all('p', {'class':
'partial_entry'})

#     all_content.append(review.text)

# NEW
content = soup.find_all('div', {'class': 'ui_column is-9'})

for review in content:
    data = review.find('p', {'class': 'partial_entry'})

    all_content.append(data.text)
```

```
# print(len(all_date))
# print(all_date)
# print(len(all_title))
# print(all_title)
# print(len(all_content))
# print(all_content)

createTable = {
    'Date': all_date,
    'Title': all_title,
    'Review': all_content
}

dataFrame = pandas.DataFrame(createTable, columns=['Date',
'Title', 'Review'])

dataFrame.index += 1
dataFrame.to_csv(r'./' + csvName + '.csv')

print('\nComplete')
```

2. *Source Code Pre-Processing* Data

```
# Load
library("tm")
library("SnowballC")
library("wordcloud")
library("RColorBrewer")
library("stringr")

setwd("D:/CSV")
docs<-readLines("Restoran-Jogja-Sebelum-Masa-Pandemi.csv")

# Load the data as a corpus
docs <- Corpus(VectorSource(docs))

#Cleaning the text
# Convert the text to lower case
docs <- tm_map(docs, content_transformer(tolower))

#Remove punctuation
docs <- tm_map(docs, toSpace, "[[:punct:]]")

# Remove URL
removeURL <- function(x) gsub("http[[:alnum:]]*", " ", x)
docs <- tm_map(docs, removeURL)

#Replacing "/", "@" and "|" with space:
toSpace <- content_transformer(function (x , pattern )
gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")
docs <- tm_map(docs, toSpace, "\\|")

#Remove numbers
```

```
docs <- tm_map(docs, toSpace, "[[:digit:]]")

# add stopwords & remove from corpus
myStopwords = readLines("stopwords_new.csv")
docs <- tm_map(docs, removeWords, myStopwords)

#Replace words
docs <- tm_map(docs, gsub, pattern="sgt",
replacement="sangat")

# Eliminate extra white spaces
docs <- tm_map(docs, stripWhitespace)

# Save Result
dataframe < data.frame(text = unlist(sapply(docs, `[`)),
stringsAsFactors = F)
write.csv(dataframe, "D:/CSV/Restoran-Jogja-Sebelum-Masa-
Pandemi-PreProcessing.csv")
```

3. *Source Code* Metode *Sentistrength*

```
import re
from collections import OrderedDict
import numpy as np
import pandas as pd
import csv


class sentistrength:

    def __init__(self, config=dict()):
        self.negasi = [line.replace('\n', '') for line in
open("negatingword.txt").read().splitlines()]
        self.tanya = [line.replace('\n', '') for line in
open("questionword.txt").read().splitlines()]
        # create sentiment words dictionary
        self.sentiwords_txt = [line.replace('\n',
'').split(":") for line in

open("sentiwords_id.txt").read().splitlines()]
        self.sentiwords_dict = OrderedDict()
        for term in self.sentiwords_txt:
            self.sentiwords_dict[term[0]] = int(term[1])
        # create emoticon dictionary
        self.emoticon_txt = [line.replace('\n', '').split("
| ") for line in

open("emoticon_id.txt").read().splitlines()]
        self.emoticon_dict = OrderedDict()
        for term in self.emoticon_txt:
            self.emoticon_dict[term[0]] = int(term[1])
        # create idioms dictionary
        self.idioms_txt = [line.replace('\n', '').split(":")
for line in open("idioms_id.txt").read().splitlines()]
        self.idioms_dict = OrderedDict()
```

```
        for term in self.idioms_txt:
            self.idioms_dict[term[0]] = int(term[1])
        # create boosterwords dictionary
        self.boosterwords_txt = [line.replace('\n',
'').split(":") for line in

open("boosterwords_id.txt").read().splitlines()]
        self.boosterwords_dict = OrderedDict()
        for term in self.boosterwords_txt:
            self.boosterwords_dict[term[0]] = int(term[1])
        self.negation_conf = config["negation"]
        self.booster_conf = config["booster"]
        self.ungkapan_conf = config["ungkapan"]
        self.consecutive_conf = config["consecutive"]
        self.repeated_conf = config["repeated"]
        self.emoticon_conf = config["emoticon"]
        self.question_conf = config["question"]
        self.exclamation_conf = config["exclamation"]
        self.punctuation_conf = config["punctuation"]
        self.mean_conf = False
        self.dfObj = pd.DataFrame(columns=['Reviews',
'Result', 'Max_Pos', 'Max_Neg', 'Kelas'])

    def senti(self, term):
        try:
            return self.sentiwords_dict[term]
        except:
            return 0

    def emosikon(self, term):
        try:
            return self.emoticon_dict[term]
        except:
            return 0

    def ungkapan(self, term):
        try:
            return self.idioms_dict[term]
        except:
            return 0

    def booster(self, term):
        try:
            return self.boosterwords_dict[term]
        except:
            return 0

    def cek_negationword(self, prev_term, prev_term2):
        # jika kata sebelumnya (index-1) adalah kata negasi,
negasikan nilai -+nya
        if prev_term in self.negasi or prev_term2 + " " +
prev_term in self.negasi:
            # print prev_term
            self.score = -abs(self.score) if self.score > 0
else abs(self.score)
```

```
    def cek_boosterword(self, term):
        booster_score = self.booster(term)
        if booster_score != 0 and self.score > 0: self.score
+= booster_score
        if booster_score != 0 and self.score < 0: self.score
-= booster_score

    def cek_consecutive_term(self, prev_term):
        if self.prev_score > 0 and self.score >= 3:
self.score += 1
        if self.prev_score < 0 and self.score <= -3:
self.score -= 1

    def cek_ungkapan(self, bigram, trigram, i):
        bigram = ' '.join(bigram)
        trigram = ' '.join(trigram)
        ungkapan_score = self.ungkapan(bigram)
        if ungkapan_score == 0:
            ungkapan_score = self.ungkapan(trigram)
        if ungkapan_score != 0:
            self.score = ungkapan_score
            self.prev_score = 0
            self.pre_max_pos[i - 1] = 1
            self.pre_max_neg[i - 1] = -1
            self.max_pos = self.pre_max_pos[i - 2]  # if
len(self.pre_max_pos)>1 else 1
            self.max_neg = self.pre_max_neg[i - 2]  # if
len(self.pre_max_neg)>1 else -1
            self.sentence_score[i - 1] = re.sub(r'\[\d\]',
'', self.sentence_score[i - 1])

    def cek_repeated_punctuation(self, next_term):
        if re.search(r'!{2,}', next_term) and self.score >=
3: self.score += 1
        if re.search(r'!{2,}', next_term) and self.score <=
-3: self.score -= 1

    def remove_extra_repeated_char(self, term):
        return re.sub(r'([A-Za-z])\1{2,}', r'\1', term)

    def plural_to_singular(self, term):
        return re.sub(r'([A-Za-z]+)\-\1', r'\1', term)

    def classify(self):
        result = "neutral"
        try:
            if self.mean_conf:
                mean_p = np.mean(self.mean_pos)
                mean_n = np.mean(self.mean_neg)
                print(mean_p, mean_n)
                if mean_p > mean_n:
                    result = "positive"
                elif mean_p < mean_n and not self.is_tanya:
                    result = "negative"
                elif mean_p < mean_n and self.is_tanya:
                    result = "neutral"
```

```
            else:
                if abs(self.sentences_max_pos) >
abs(self.sentences_max_neg):
                    result = "positive"
                elif abs(self.sentences_max_pos) <
abs(self.sentences_max_neg):
                    result = "negative"
                elif abs(self.sentences_max_pos) ==
abs(self.sentences_max_neg):
                    result = "neutral"
        except:
            print("error ", self.sentences_max_pos,
self.sentences_max_neg)
        return result

    def cek_neutral_term(self, terms, i):
        if terms[i - 1] in self.neutral_term or terms[i + 1]
in self.neutral_term: self.score = 1

    def main(self, sentence):
        self.neutral_term = ['jika', 'kalau']
        sentences = sentence.split('.')
        self.sentences_max_neg = -1
        self.sentences_max_pos = 1
        self.sentences_score = []
        self.sentences_text = []
        for sentence in sentences:
            self.max_neg = -1
            self.max_pos = 1
            self.mean_neg = [1]
            self.mean_pos = [1]
            self.sentence_score = []
            terms = sentence.split()
            # terms = re.split(r'[\s,.]',sentence)
            terms_length = len(terms)
            self.is_tanya = False
            self.sentence_text = ''
            # print self.max_pos, self.max_neg
            # SEMUA KALIMAT YANG MEMILIKI TANDA SERU
MEMILIKI +ve minimal 2
            if self.exclamation_conf and re.search('!',
sentence): self.max_pos = 2
            self.prev_score = 0
            self.pre_max_pos = []
            self.pre_max_neg = []
            for i, term in enumerate(terms):
                # repeated_term = ''
                is_extra_char = False
                plural = ''
                self.score = 0
                # if re.search(r'[A-Za-z\-.]+',term):
                # print term
                if re.search(r'([A-Za-z])\1{3,}', term):
                    is_extra_char = True
                    # repeated_term =term
                term = self.remove_extra_repeated_char(term)
```

```
                if re.search(r'([A-Za-z]+)\-\1', term):
                    plural = term
                    term = self.plural_to_singular(term)
                # GET SENTI SCORE#
                self.score = self.senti(term)
                # print "senti score",term, self.score

                # NEGATION HANDLER#
                if self.negation_conf and self.score != 0
and i > 0: self.cek_negationword(terms[i - 1], terms[i - 2])
                # print  "negation score",term, self.score

                # BOOSTERWORD HANDLER#
                if self.booster_conf and self.score != 0 and
i > 0 and i <= (terms_length - 1): self.cek_boosterword(
                    terms[i - 1])
                if self.booster_conf and self.score != 0 and
i >= 0 and i < (terms_length - 1): self.cek_boosterword(
                    terms[i + 1])
                # print  "booster score",term, self.score

                # IDIOM/UNGKAPAN HANDLER#
                if self.ungkapan_conf and i > 0 and i <=
(terms_length - 1): self.cek_ungkapan([terms[i - 1], term],

[terms[i - 2],

terms[i - 1], term], i)
                # if self.ungkapan_conf and i>=0 and
i<(terms_length-1):self.cek_ungkapan([term,terms[i+1]])
                # print  "idiom score",term, self.score

                # CONSECUTIVE SENTIMENT WORD#
                if self.consecutive_conf and i > 0 and i <=
(
                        terms_length - 1) and self.score !=
0: self.cek_consecutive_term(terms[i - 1])
                # print  "consecutive score",term,
self.score

                # +1 SENTI SCORE IF REPEATED CHAR ON
POSITIVE/NEGATIVE +2 IF NEUTRAL TERM
                if self.repeated_conf and is_extra_char ==
True and self.score > 0: self.score += 1
                if self.repeated_conf and is_extra_char ==
True and self.score < 0: self.score -= 1
                if self.repeated_conf and is_extra_char ==
True and self.score == 0: self.score = 2
                # print  "repeat char score", term,
self.score
                if self.punctuation_conf and i >= 0 and i <
(terms_length - 1): self.cek_repeated_punctuation(
                    terms[i + 1])
                # CEK APAKAH TERDAPAT KATA TANYA
                if self.question_conf and (term in
self.tanya or re.search(r'\?', term)): self.is_tanya = True
```

```
                # CEK neutral term
                if self.score != 0 and i > 1 and i <
(terms_length - 2): self.cek_neutral_term(terms, i)
                # if self.score!=0 and i>0 and
i<(terms_length-4): self.cek_neutral_term(terms,i)
                if self.emoticon_conf and self.score == 0:
self.score = self.emosikon(term)

                self.prev_score = self.score
                if self.mean_conf and self.score > 0:
self.mean_pos.append(self.score)
                if self.mean_conf and self.score < 0:
self.mean_neg.append(abs(self.score))
                # GET MAX SCORE +ve/-ve
                self.max_pos = self.score if self.score >
self.max_pos else self.max_pos
                self.max_neg = self.score if self.score <
self.max_neg else self.max_neg
                # insert score info current term
                self.pre_max_pos.append(self.max_pos)
                self.pre_max_neg.append(self.max_neg)
                # print self.pre_max_pos, self.pre_max_neg
                if plural != '': term = plural
                self.sentence_text += ' {}'.format(term)
                if self.score != 0: term = "{}
[{}]".format(term, self.score)
                self.sentence_score.append(term)

            self.sentences_text.append(self.sentence_text)
            self.sentences_score.append("
".join(self.sentence_score))
            if self.is_tanya:
                self.max_neg = -1
            self.sentences_max_pos = self.max_pos if
self.max_pos > self.sentences_max_pos else
self.sentences_max_pos
            self.sentences_max_neg = self.max_neg if
self.max_neg < self.sentences_max_neg else
self.sentences_max_neg
            # print self.sentences_max_pos,
self.sentences_max_neg
        sentence_result = self.classify()
        self.dfObj = self.dfObj.append(
            {'Reviews': self.sentences_text, 'Result':
self.sentences_score, 'Max_Pos': self.sentences_max_pos,
             'Max_Neg': self.sentences_max_neg, 'Kelas':
sentence_result}, ignore_index=True)
        print(self.dfObj);
        self.dfObj.to_csv('D:/SENTISTRENGTH/Restoran-Jogja-
Masa-Pandemi-PreProcessing.csv', header=True, index=True);
        return {"classified_text": ".
".join(self.sentences_score), "tweet_text": ".
".join(self.sentences_text),
                "sentence_score": self.sentences_score,
"max_positive": self.sentences_max_pos,
```

```
                "max_negative": self.sentences_max_neg,
"kelas": sentence_result}
        # return {"classified_text":".
".join(self.sentences_score)}
        # print self.sentences_text
        # return {"classified_text":".
".join(self.sentences_score),"tweet_text":".
".join(self.sentences_text),"sentence_score":self.sentences_
score,"max_positive":self.sentences_max_pos,"max_negative":s
elf.sentences_max_neg,"kelas":sentence_result}

    # return {"RESULT : ": self.sentences_max_pos +
self.sentences_max_neg,
    #                      "max_positif":
self.sentences_max_pos,
    #                      "neg_positif":
self.sentences_max_neg,
    #                      "kelas": sentence_result}
    # return {"classified_text":".
".join(self.sentences_score)}
    # return{sentece_result}


config = dict()
config["negation"] = True
config["booster"] = True
config["ungkapan"] = True
config["consecutive"] = True
config["repeated"] = True
config["emoticon"] = True
config["question"] = True
config["exclamation"] = True
config["punctuation"] = True
senti = sentistrength(config)
# print(senti.main("agnezmo pintar dan cantik sekali tetapi
lintah darat :)"))

# print(senti.main("Tempatnya strategis, dekat dengan mall.
Kamarnya bersih. Saya suka menginap di royal ambarukmo.
Resepsionisnya namanya Wigati, sangat welcome dan membantu
kami, memberi info destinasi wisata maupun kuliner"))
# file_new = open("D:/reviews.csv","r");
list2 = [line.strip() for line in
        open("D:/SENTISTRENGTH/Restoran-Jogja-Masa-Pandemi-
PreProcessing.csv", "r", errors='ignore')];
for p in list2: print(senti.main(p))
#                          file_new.write(senti.main(p));

# file_new.write(list2);
# with open("D:/reviews.csv","w") as writeFile:
#                           writer = csv.writer(writeFile)
#                           writer.writerow(list2)

# writeFile.close();
# file_new.close();
```

4. *Source Code* Metode *N-Gram*

```
setwd("D:/BahanNgram")
docs <- readLines("Restoran-Jogja-Masa-Pandemi-
PreProcessing-SS-NegativePenuh.csv")
docs <- Corpus(VectorSource(docs))

dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing = TRUE)
d <- data.frame(word = names(v),freq=v)
head(d, 15)

# Save Result
Dataframe <- data.frame(text = unlist(sapply(v, `[`)),
stringsAsFactors = F)
write.csv(dataframe, "D:/HasilNgram/ Restoran-Jogja-Masa-
Pandemi-PreProcessing-SS-NegativePenuh(NEW).csv")
```

5. *Source Code* Identifikasi Empat *Core* Bisnis Restoran

```
import re
from collections import OrderedDict
import numpy as np
import pandas as pd
import csv


class sentistrength:

    def __init__(self, config=dict()):
        self.negasi = [line.replace('\n', '') for line in
open("negatingword.txt").read().splitlines()]
        self.tanya = [line.replace('\n', '') for line in
open("questionword.txt").read().splitlines()]
        # create sentiment words dictionary
        self.sentiwords_txt = [line.replace('\n',
'').split(":") for line in

open("kamus_main_business.txt").read().splitlines()]
        self.sentiwords_dict = OrderedDict()
        for term in self.sentiwords_txt:
            self.sentiwords_dict[term[0]] = int(term[1])
        # create emoticon dictionary
        self.emoticon_txt = [line.replace('\n', '').split("
| ") for line in

open("emoticon_id.txt").read().splitlines()]
        self.emoticon_dict = OrderedDict()
        for term in self.emoticon_txt:
            self.emoticon_dict[term[0]] = int(term[1])
        # create idioms dictionary
        self.idioms_txt = [line.replace('\n', '').split(":")
for line in open("idioms_id.txt").read().splitlines()]
        self.idioms_dict = OrderedDict()
```

```
        for term in self.idioms_txt:
            self.idioms_dict[term[0]] = int(term[1])
        # create boosterwords dictionary
        self.boosterwords_txt = [line.replace('\n',
'').split(":") for line in

open("boosterwords_id.txt").read().splitlines()]
        self.boosterwords_dict = OrderedDict()
        for term in self.boosterwords_txt:
            self.boosterwords_dict[term[0]] = int(term[1])
        self.negation_conf = config["negation"]
        self.booster_conf = config["booster"]
        self.ungkapan_conf = config["ungkapan"]
        self.consecutive_conf = config["consecutive"]
        self.repeated_conf = config["repeated"]
        self.emoticon_conf = config["emoticon"]
        self.question_conf = config["question"]
        self.exclamation_conf = config["exclamation"]
        self.punctuation_conf = config["punctuation"]
        self.mean_conf = False
        self.dfObj = pd.DataFrame(columns=['Reviews',
'Result', 'Max_Pos', 'Max_Neg', 'Kelas'])

    def senti(self, term):
        try:
            return self.sentiwords_dict[term]
        except:
            return 0

    def emosikon(self, term):
        try:
            return self.emoticon_dict[term]
        except:
            return 0

    def ungkapan(self, term):
        try:
            return self.idioms_dict[term]
        except:
            return 0

    def booster(self, term):
        try:
            return self.boosterwords_dict[term]
        except:
            return 0

    def cek_negationword(self, prev_term, prev_term2):
        # jika kata sebelumnya (index-1) adalah kata negasi,
negasikan nilai -+nya
        if prev_term in self.negasi or prev_term2 + " " +
prev_term in self.negasi:
            # print prev_term
            self.score = -abs(self.score) if self.score > 0
else abs(self.score)
```

```
    def cek_boosterword(self, term):
        booster_score = self.booster(term)
        if booster_score != 0 and self.score > 0: self.score
+= booster_score
        if booster_score != 0 and self.score < 0: self.score
-= booster_score

    def cek_consecutive_term(self, prev_term):
        if self.prev_score > 0 and self.score >= 3:
self.score += 1
        if self.prev_score < 0 and self.score <= -3:
self.score -= 1

    def cek_ungkapan(self, bigram, trigram, i):
        bigram = ' '.join(bigram)
        trigram = ' '.join(trigram)
        ungkapan_score = self.ungkapan(bigram)
        if ungkapan_score == 0:
            ungkapan_score = self.ungkapan(trigram)
        if ungkapan_score != 0:
            self.score = ungkapan_score
            self.prev_score = 0
            self.pre_max_pos[i - 1] = 1
            self.pre_max_neg[i - 1] = -1
            self.max_pos = self.pre_max_pos[i - 2]  # if
len(self.pre_max_pos)>1 else 1
            self.max_neg = self.pre_max_neg[i - 2]  # if
len(self.pre_max_neg)>1 else -1
            self.sentence_score[i - 1] = re.sub(r'\[\d\]',
'', self.sentence_score[i - 1])

    def cek_repeated_punctuation(self, next_term):
        if re.search(r'!{2,}', next_term) and self.score >=
3: self.score += 1
        if re.search(r'!{2,}', next_term) and self.score <=
-3: self.score -= 1

    def remove_extra_repeated_char(self, term):
        return re.sub(r'([A-Za-z])\1{2,}', r'\1', term)

    def plural_to_singular(self, term):
        return re.sub(r'([A-Za-z]+)\-\1', r'\1', term)

    def classify(self):
        result = "neutral"
        try:
            if self.mean_conf:
                mean_p = np.mean(self.mean_pos)
                mean_n = np.mean(self.mean_neg)
                print(mean_p, mean_n)
                if mean_p > mean_n:
                    result = "positive"
                elif mean_p < mean_n and not self.is_tanya:
                    result = "negative"
                elif mean_p < mean_n and self.is_tanya:
                    result = "neutral"
```

```
            else:
                if abs(self.sentences_max_pos) >
abs(self.sentences_max_neg):
                    result = "positive"
                elif abs(self.sentences_max_pos) <
abs(self.sentences_max_neg):
                    result = "negative"
                elif abs(self.sentences_max_pos) ==
abs(self.sentences_max_neg):
                    result = "neutral"
        except:
            print("error ", self.sentences_max_pos,
self.sentences_max_neg)
        return result

    def cek_neutral_term(self, terms, i):
        if terms[i - 1] in self.neutral_term or terms[i + 1]
in self.neutral_term: self.score = 1

    def main(self, sentence):
        self.neutral_term = ['jika', 'kalau']
        sentences = sentence.split('.')
        self.sentences_max_neg = -1
        self.sentences_max_pos = 1
        self.sentences_score = []
        self.sentences_text = []
        for sentence in sentences:
            self.max_neg = -1
            self.max_pos = 1
            self.mean_neg = [1]
            self.mean_pos = [1]
            self.sentence_score = []
            terms = sentence.split()
            # terms = re.split(r'[\s,.]',sentence)
            terms_length = len(terms)
            self.is_tanya = False
            self.sentence_text = ''
            # print self.max_pos, self.max_neg
            # SEMUA KALIMAT YANG MEMILIKI TANDA SERU
MEMILIKI +ve minimal 2
            if self.exclamation_conf and re.search('!',
sentence): self.max_pos = 2
            self.prev_score = 0
            self.pre_max_pos = []
            self.pre_max_neg = []
            for i, term in enumerate(terms):
                # repeated_term = ''
                is_extra_char = False
                plural = ''
                self.score = 0
                # if re.search(r'[A-Za-z\-.]+',term):
                # print term
                if re.search(r'([A-Za-z])\1{3,}', term):
                    is_extra_char = True
                    # repeated_term =term
                term = self.remove_extra_repeated_char(term)
```

```
                if re.search(r'([A-Za-z]+)\-\1', term):
                    plural = term
                    term = self.plural_to_singular(term)
                # GET SENTI SCORE#
                self.score = self.senti(term)
                # print "senti score",term, self.score

                # NEGATION HANDLER#
                if self.negation_conf and self.score != 0
and i > 0: self.cek_negationword(terms[i - 1], terms[i - 2])
                # print  "negation score",term, self.score

                # BOOSTERWORD HANDLER#
                if self.booster_conf and self.score != 0 and
i > 0 and i <= (terms_length - 1): self.cek_boosterword(
                    terms[i - 1])
                if self.booster_conf and self.score != 0 and
i >= 0 and i < (terms_length - 1): self.cek_boosterword(
                    terms[i + 1])
                # print  "booster score",term, self.score

                # IDIOM/UNGKAPAN HANDLER#
                if self.ungkapan_conf and i > 0 and i <=
(terms_length - 1): self.cek_ungkapan([terms[i - 1], term],

[terms[i - 2],

terms[i - 1], term], i)
                # if self.ungkapan_conf and i>=0 and
i<(terms_length-1):self.cek_ungkapan([term,terms[i+1]])
                # print  "idiom score",term, self.score

                # CONSECUTIVE SENTIMENT WORD#
                if self.consecutive_conf and i > 0 and i <=
(
                        terms_length - 1) and self.score !=
0: self.cek_consecutive_term(terms[i - 1])
                # print  "consecutive score",term,
self.score

                # +1 SENTI SCORE IF REPEATED CHAR ON
POSITIVE/NEGATIVE +2 IF NEUTRAL TERM
                if self.repeated_conf and is_extra_char ==
True and self.score > 0: self.score += 1
                if self.repeated_conf and is_extra_char ==
True and self.score < 0: self.score -= 1
                if self.repeated_conf and is_extra_char ==
True and self.score == 0: self.score = 2
                # print  "repeat char score", term,
self.score
                if self.punctuation_conf and i >= 0 and i <
(terms_length - 1): self.cek_repeated_punctuation(
                    terms[i + 1])
                # CEK APAKAH TERDAPAT KATA TANYA
                if self.question_conf and (term in
self.tanya or re.search(r'\?', term)): self.is_tanya = True
```

```
                # CEK neutral term
                if self.score != 0 and i > 1 and i <
(terms_length - 2): self.cek_neutral_term(terms, i)
                # if self.score!=0 and i>0 and
i<(terms_length-4): self.cek_neutral_term(terms,i)
                if self.emoticon_conf and self.score == 0:
self.score = self.emosikon(term)

                self.prev_score = self.score
                if self.mean_conf and self.score > 0:
self.mean_pos.append(self.score)
                if self.mean_conf and self.score < 0:
self.mean_neg.append(abs(self.score))
                # GET MAX SCORE +ve/-ve
                self.max_pos = self.score if self.score >
self.max_pos else self.max_pos
                self.max_neg = self.score if self.score <
self.max_neg else self.max_neg
                # insert score info current term
                self.pre_max_pos.append(self.max_pos)
                self.pre_max_neg.append(self.max_neg)
                # print self.pre_max_pos, self.pre_max_neg
                if plural != '': term = plural
                self.sentence_text += ' {}'.format(term)
                if self.score != 0: term = "{}
[{}]".format(term, self.score)
                self.sentence_score.append(term)

            self.sentences_text.append(self.sentence_text)
            self.sentences_score.append("
".join(self.sentence_score))
            if self.is_tanya:
                self.max_neg = -1
            self.sentences_max_pos = self.max_pos if
self.max_pos > self.sentences_max_pos else
self.sentences_max_pos
            self.sentences_max_neg = self.max_neg if
self.max_neg < self.sentences_max_neg else
self.sentences_max_neg
            # print self.sentences_max_pos,
self.sentences_max_neg
        sentence_result = self.classify()
        # self.dfObj = self.dfObj.append({'Reviews' :
self.sentences_text , 'Result' : self.sentences_score,
'Max_Pos' : self.sentences_max_pos, 'Max_Neg' :
self.sentences_max_neg, 'Kelas' : sentence_result} ,
ignore_index=True)
        self.dfObj = self.dfObj.append(
            {'Reviews': self.sentences_text, 'Result':
self.sentences_score, 'Max_Pos': self.sentences_max_pos},
            ignore_index=True)
        print(self.dfObj);
        self.dfObj.to_csv('D:/BahanCoreRestoran/Restoran-
Jogja-Sebelum-Masa-Pandemi-PreProcessing-SS-
SemiPositive.csv', header=True, index=True);
```

```
        return {"classified_text": ".
".join(self.sentences_score), "tweet_text": ".
".join(self.sentences_text),
                "sentence_score": self.sentences_score,
"max_positive": self.sentences_max_pos,
                "max_negative": self.sentences_max_neg,
"kelas": sentence_result}
        # return {"classified_text":".
".join(self.sentences_score)}
        # print self.sentences_text
        # return {"classified_text":".
".join(self.sentences_score),"tweet_text":".
".join(self.sentences_text),"sentence_score":self.sentences_
score,"max_positive":self.sentences_max_pos,"max_negative":s
elf.sentences_max_neg,"kelas":sentence_result}

    # return {"RESULT : ": self.sentences_max_pos +
self.sentences_max_neg,
    #                     "max_positif":
self.sentences_max_pos,
    #                     "neg_positif":
self.sentences_max_neg,
    #                     "kelas": sentence_result}
    # return {"classified_text":".
".join(self.sentences_score)}
    # return{sentece_result}


config = dict()
config["negation"] = False
config["booster"] = False
config["ungkapan"] = False
config["consecutive"] = False
config["repeated"] = False
config["emoticon"] = False
config["question"] = False
config["exclamation"] = False
config["punctuation"] = False
senti = sentistrength(config)
# print(senti.main("agnezmo pintar dan cantik sekali tetapi
lintah darat :)"))

# print(senti.main("Tempatnya strategis, dekat dengan mall.
Kamarnya bersih. Saya suka menginap di royal ambarukmo.
Resepsionisnya namanya Wigati, sangat welcome dan membantu
kami, memberi info destinasi wisata maupun kuliner"))
# file_new = open("D:/reviews.csv","r");
list2 = [line.strip() for line in
         open("D:/BahanCoreRestoran/Restoran-Jogja-Sebelum-
Masa-Pandemi-PreProcessing-SS-SemiPositive.csv", "r",
errors='ignore')];
for p in list2: print(senti.main(p))
#                         file_new.write(senti.main(p));

# file_new.write(list2);
# with open("D:/reviews.csv","w") as writeFile:
```

```
# writer = csv.writer(writeFile)
# writer.writerow(list2)

# writeFile.close();
# file_new.close();
```

DAFTAR PUSTAKA

[1] D. P. D. I. Yogyakarta, "Statistik Kepariwisataan 2018," no. 0274.

[2] Samsir, Ambiyar, U. Verawardina, F. Edi, and R. Watrianthos, "Analisis Sentimen Pembelajaran Daring Pada Twitter di Masa Pandemi COVID-19 Menggunakan Metode Naïve Bayes," *J. Media Inform. Budidarma*, vol. 5, pp. 157–163, 2021.

[3] K. Suryadinata Putra, "Penerapan Protokol Kesehatan Pencegahan Covid-19 Di Restoran Naughty Nuri'S Dalam Perspektif Hukum Perlindungan Konsumen," *J. Kertha Wicara*, vol. 10, no. 3, pp. 262–272, 2021.

[4] F. Sodik and I. Kharisudin, "Analisis Sentimen dengan SVM , NAIVE BAYES dan KNN untuk Studi Tanggapan Masyarakat Indonesia Terhadap Pandemi Covid-19 pada Media Sosial Twitter," *Prisma*, vol. 4, pp. 628–634, 2021.

[5] R. Sari, "Analisis Sentimen Review Restoran menggunakan Algoritma Naive Bayes berbasis Particle Swarm Optimization," *J. Inform.*, vol. 6, no. 1, pp. 23–28, 2019.

[6] V. M. Ngo, "MEASURING CUSTOMER SATISFACTION : A LITERATURE REVIEW MEASURING CUSTOMER SATISFACTION : A LITERATURE REVIEW Vu Minh Ngo," no. January, 2018.

[7] J. Kandampully and D. Suhartanto, "Customer loyalty in the hotel industry: The role of customer satisfaction and image," *Int. J. Contemp. Hosp. Manag.*, vol. 12, no. 6, pp. 346–351, 2000.

[8] K. Cahyani and G. Rahanatha, "Pengaruh Kualitas Layanan Terhadap Kepuasan Dan Dampakanya Terhadap Kepercayaan Serta Loyalitas," *E-Jurnal Manaj. Univ. Udayana*, vol. 3, no. 10, p. 250199, 2014.

[9] D. A. Muthia, “Komparasi Algoritma Klasifikasi Text Mining Untuk Analisis Sentimen Pada Review Restoran,” *J. PILAR Nusa Mandiri*, vol. 14, no. 1, pp. 69–74, 2018.

[10] R. Siringoringo, “Text Mining dan Klasterisasi Sentimen Pada Ulasan Produk Toko Online,” pp. 1–6.

[11] P. Antinasari, R. S. Perdana, and M. A. Fauzi, “Analisis Sentimen Tentang Opini Film Pada Dokumen Twitter Berbahasa Indonesia Menggunakan Naive Bayes Dengan Perbaikan Kata Tidak Baku,” vol. 1, no. 12, pp. 1733–1741, 2017.

[12] U. Rofiqoh, R. S. Perdana, and M. A. Fauzi, “Analisis Sentimen Tingkat Kepuasan Pengguna Penyedia Layanan Telekomunikasi Seluler Indonesia Pada Twitter Dengan Metode Support Vector Machine dan Lexion Based Feature,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 1, no. 12, pp. 1725–1732, 2017.

[13] D. A. Muthia, “Analisis Sentimen Pada Review Restoran Dengan Teks Bahasa Indonesia Mengunakan Algoritma Naive Bayes,” *Jurnalilmu Pengetah. Dan Teknol. Komput.*, vol. 2, no. 2, pp. 39–45, 2017.

[14] A. Taufik, “Optimasi Particle Swarm Optimization Sebagai Seleksi Fitur Pada Analisis Sentimen Review Hotel Berbahasa Indonesia Menggunakan Algoritma Naïve Bayes,” *J. Tek. Komput. AMIK BSI*, vol. III, no. 2, pp. 40–47, 2017.

[15] A. Nugroho, “Analisis Sentimen Pada Media Sosial Twitter Menggunakan Naive Bayes Classifier Dengan Ekstrasi Fitur N-Gram,” *J-SAKTI (Jurnal Sains Komput. dan Inform.*, vol. 2, no. 2, p. 200, 2018.

[16] R. Wati, “Penerapan Algoritma Genetika Untuk Seleksi Fitur Pada Analisis Sentimen Review Jasa Maskapai Penerbangan,” *J. Evolusi*, vol. 4, no. 1, pp. 25–31, 2016.

[17] E. Guzman and W. Maalej, “How do users like this feature? A fine grained

sentiment analysis of App reviews," *2014 IEEE 22nd Int. Requir. Eng. Conf. RE 2014 - Proc.*, pp. 153–162, 2014.

[18] E. G. Kim and S. H. Chun, "Analyzing online car reviews using text mining," *Sustain.*, vol. 11, no. 6, 2019.

[19] K. Berezina, A. Bilgihan, C. Cobanoglu, and F. Okumus, "Understanding Satisfied and Dissatisfied Hotel Customers: Text Mining of Online Hotel Reviews," *J. Hosp. Mark. Manag.*, vol. 25, no. 1, pp. 1–24, 2016.

[20] C. Engko and P. Usmany, "Dampak Pandemi Covid-19 Terhadap Proses Pembelajaran Online," *J. Akunt.*, vol. 6, no. 1, pp. 23–38, 2020.

[21] B. A. B. Ii, "School Hotel Administration," pp. 10–35, 2005.

[22] A.-H. Tan, "Text Mining: The state of the art and the challenges," *Proc. PAKDD 1999 Work. Knowl. Disocovery from Adv. Databases*, vol. 8, pp. 65–70, 1999.

[23] M. J. Kim, K. Ohk, and C. S. Moon, "Trend analysis by using text mining of journal articles regarding consumer policy," *New Phys. Sae Mulli*, vol. 67, no. 5, pp. 555–561, 2017.

[24] A. Kao and S. Poteet, "Text mining and natural language processing," *ACM SIGKDD Explor. Newsl.*, vol. 7, no. 1, pp. 1–2, 2005.

[25] A. D'Andrea, F. Ferri, P. Grifoni, and T. Guzzo, "Approaches, Tools and Applications for Sentiment Analysis Implementation," *Int. J. Comput. Appl.*, vol. 125, no. 3, pp. 26–33, 2015.

[26] M. Thelwall, "Gender bias in sentiment analysis," *Online Inf. Rev.*, vol. 42, no. 1, pp. 45–57, 2018.

[27] H. Saif, Y. He, M. Fernandez, and H. Alani, "Contextual semantics for sentiment analysis of Twitter," *Inf. Process. Manag.*, vol. 52, no. 1, pp. 5–19, 2016.

[28] F. Previtali, A. F. Arrieta, and P. Ermanni, "Double-walled corrugated

structure for bending-stiff anisotropic morphing skins," *J. Intell. Mater. Syst. Struct.*, vol. 26, no. 5, pp. 599–613, 2015.

[29] R. B. Vukmir, "Customer satisfaction," *Int. J. Health Care Qual. Assur.*, vol. 19, no. 1, pp. 8–31, 2006.

[30] C. B. Liat, S. Mansori, G. C. Chuan, and B. C. Imrie, "Hotel Service Recovery and Service Quality: Influences of Corporate Image and Generational Differences in the Relationship between Customer Satisfaction and Loyalty," *J. Glob. Mark.*, vol. 30, no. 1, pp. 42–51, 2017.

[31] "How Too Build Brand Awareness And Customer Engagement," 2015.

[32] M. Syarifuddin, "Analisis Sentimen Opini Publik Mengenai Covid-19 Pada Twitter Menggunakan Metode Naïve Bayes Dan Knn," *Inti Nusa Mandiri*, vol. 15, no. 1, pp. 23–28, 2020.

[33] C. N. Kurniawan, "Review Integratif Mengenai Pandemi Covid-19 Dan Dampaknya Terhadap Industri Minuman Kopi," *Semin. Nas. Adm. Bisnis dan Manaj.*, vol. 6, no. October, pp. 21–30, 2020.