DAFTAR PUSTAKA

Azhari SN, dkk. 2014. *Analisis sentiment dan klasifikasi kategori terhadap toko public pada twitter* Skripsi. Bandung: Universitas Gajah Mada.

Agussalim,dkk. 2021. *Analisis sentiment hastag kuliner Indonesia menggunakan Naïve Bayes*. Jurnal. Malang: UPN Vetran Jawa Timur

Faiz, Kamal Azam. 2017. *Text Mining untuk analisa Sentimen ekspedisi jasa pengiriman barang menggunakan metode naive bayes pada aplikasi J&T Express.* Helsinki University Dian Nuswantoro.

Wahyu Setyobudi,dkk. 2019. *Sentimen Analisis twitter terhadap penyelanggaraan GOJEK TRAVELOKA liga 1 Indonesia*. Makassar: Universitas Muhammadiyah Penorogo.

Rahman, Arief Mochammad. 2016. *Implementasi Naïve Bayes Classifier pada sistem analisis sentiment twitter (Studi Kasus: BPJS)*. Skripsi Sistem Informasi.

Ratnawati, Fajar 2017. *Analisis sentiment opini pada film menggunakan algoritma dynamic convoncional neural network*. Jurnal: Universitas Gajah Mada

Widianto, Teguh Puji, Agus Sihabuddin. 2016. *Teknik realtime data processing untuk monitoring tweet menggunakan analisis sentiment dengan algoritma naïve bayes studi kasus tweet yang terkait dengan UGM*. Universitas Gajah Mada.

Priyani, Rina. 2016. *Analisis sentimen pengguna Twitter terhadap acara Standup Comedy dengan menggunakan Support Vector Machine dan Naïve Bayes Classifier*. Universitas Komputer Indonesia.

Adawiyah, R. 2018. *Analisis sentiment pada aplikasi Mobile Banking menggunakan metode naïve bayes dan asosiasi*. Yogyakarta: Universitas Islam Indonesia.

LAMPIRAN

LAMPIRAN 1

- Data Cleansing

```
import nltk
import string
import re #regex library
nltk.download('punkt')

# import word_tokenize & FreqDist from NLTK
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist

# ------ Proses Tokenisasi ---------

def remove_tweet_special(text):
    # remove tab, new line, ans back slice
    text = text.replace('\\t'," ").replace('\\n'," ").replace('\\u'," ").replace('\\',"")
    # remove non ASCII (emoticon, chinese word, .etc)
    text = text.encode('ascii', 'replace').decode('ascii')
    # remove mention, link, hashtag
    text = ' '.join(re.sub("([@#][A-Za-z0-9]+)|(\w+:\/\/\S+)"," ", text).split())
    # remove incomplete URL
    return text.replace("http://", " ").replace("https://", " ")

TWEET_DATA['content'] = TWEET_DATA['content'].apply(remove_tweet_special)
```

```
#remove number
def remove_number(text):
   return  re.sub(r"\d+", "", text)

TWEET_DATA['content'] = TWEET_DATA['content'].apply(remove_number)

#remove punctuation
def remove_punctuation(text):
   return text.translate(str.maketrans("","",string.punctuation))

TWEET_DATA['content'] = TWEET_DATA['content'].apply(remove_punctuation)

#remove whitespace leading & trailing
def remove_whitespace_LT(text):
   return text.strip()

TWEET_DATA['content'] = TWEET_DATA['content'].apply(remove_whitespace_LT)

#remove multiple whitespace into single whitespace
def remove_whitespace_multiple(text):
   return re.sub('\s+',' ',text)

TWEET_DATA['content'] =
TWEET_DATA['content'].apply(remove_whitespace_multiple)

# remove single char
def remove_singl_char(text):
   return re.sub(r"\b[a-zA-Z]\b", "", text)
```

```
TWEET_DATA['content'] = TWEET_DATA['content'].apply(remove_singl_char)


# NLTK word rokenize
def word_tokenize_wrapper(text):
   return word_tokenize(text)


TWEET_DATA['content_tokens'] =
TWEET_DATA['content'].apply(word_tokenize_wrapper)


print('Tokenizing Result : \n')
print(TWEET_DATA['content_tokens'].head())
print('\n\n\n')


# NLTK calc frequency distribution
def freqDist_wrapper(text):
   return FreqDist(text)


TWEET_DATA['content_tokens_fdist'] =
TWEET_DATA['content_tokens'].apply(freqDist_wrapper)


print('Frequency Tokens : \n')
print(TWEET_DATA['content_tokens_fdist'].head().apply(lambda x :
x.most_common()))
TWEET_DATA.head()


nltk.download('stopwords')
```

```
from nltk.corpus import stopwords


# ---------------------- get stopword from NLTK stopword -----------------------------
# get stopword indonesian
list_stopwords = stopwords.words('indonesian')



# --------------------------- manualy add stopword  ------------------------------------
# append additional stopword
list_stopwords.extend(['jg', 'josss', "yg", "dg", "rt", "dgn", "ny", "d", 'klo',
                       'kalo', 'amp', 'biar', 'bikin', 'bilang',
                       'gak', 'ga', 'krn', 'nya', 'nih', 'sih',
                       'si', 'tau', 'tdk', 'tuh', 'utk', 'ya',
                       'jd', 'jgn', 'sdh', 'aja', 'n', 't',
                       ])
# ---------------------- add stopword from txt file ------------------------------------
# read txt stopword using pandas
txt_stopword = pd.read_csv("datashopee.csv", names= ["stopwords"], header =
None)


# convert stopword string to list & append additional stopword
list_stopwords.extend(txt_stopword["stopwords"][0].split(' '))


# ----------------------------------------------------------------------------------


# convert list to dictionary
list_stopwords = set(list_stopwords)
```

```
#remove stopword pada list token

def stopwords_removal(words):

   return [word for word in words if word not in list_stopwords]


TWEET_DATA['content_tokens_WSW'] =
TWEET_DATA['content_tokens'].apply(stopwords_removal)


print(TWEET_DATA['content_tokens_WSW'].head())


#Normalizazion

normalizad_word = pd.read_csv("datashopee.csv")


normalizad_word_dict = {}


for index, row in normalizad_word.iterrows():

   if row[0] not in normalizad_word_dict:

      normalizad_word_dict[row[0]] = row[1]


def normalized_term(document):

   return [normalizad_word_dict[term] if term in normalizad_word_dict else term
for term in document]


TWEET_DATA['content_normalized'] =
TWEET_DATA['content_tokens_WSW'].apply(normalized_term)


TWEET_DATA['content_normalized'].head(7)
```

```
# ------ Proses Stemming --------
# import Sastrawi package
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
import swifter



# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()




# stemmed
def stemmed_wrapper(term):
   return stemmer.stem(term)




term_dict = {}




for document in TWEET_DATA['content_normalized']:
   for term in document:
      if term not in term_dict:
         term_dict[term] = ' '
```

```
for term in term_dict:
    term_dict[term] = stemmed_wrapper(term)
    print(term,":" ,term_dict[term])


print(term_dict)
print("------------------------")



# apply stemmed term to dataframe
def get_stemmed_term(document):
    return [term_dict[term] for term in document]


TWEET_DATA['content_tokens_stemmed'] =
TWEET_DATA['content_normalized'].swifter.apply(get_stemmed_term)
print(TWEET_DATA['content_tokens_stemmed'])
```

- Analisis Data

```
#menghitung jumlah anggota tiap label
label, count = np.unique(df['sentiment'], return_counts=True)
print(label, count)
#menghitung persentasi tiap kelas
negatif = count[0]/len(df)
netral  = count[1]/len(df)
positif = count[2]/len(df)

#plot pie
size = [negatif, netral, positif]
plt.figure(figsize=(5,3), dpi=120)
plt.pie(size, explode=[0, 0, 0.1], labels=['Negatif', 'Netral', 'Positif'],
autopct='%.0f%%');

#konversi Label ke polaritas

def convert(polarity):
   if polarity == 'positif':
      return 1
   elif polarity == 'netral':
      return 0
   else:
      return – 1

   df['Polarity'] = df['sentiment'].apply(convert)

x = df['content']
y = df['Polarity']
```

```
#TF-IDF
#vectorizer = CountVectorizer()
#x = vectorizer.fit_transform(df['content'])
bow_transformer = CountVectorizer()
print(df['content'].shape)
X = bow_transformer.fit_transform(df['content'])

print(X.toarray())
print('Shape of Sparse Matrix: ', X.shape)
print('Amount of Non-Zero occurrences: ', X.nnz)
#save the Count Vectorized to disk
filename1 = 'count_vectorized1.pkl'
pickle.dump(bow_transformer, open(filename1, 'wb'))
#TF IDF transformer
tf_transform = TfidfTransformer(use_idf=False).fit(X)
X = tf_transform.transform(X)
print(X.shape)

#save the TFIDF to disk
filename1 = 'tfid_transform1.pkl'
pickle.dump(tf_transform, open(filename1, 'wb'))
density = (100.0 * X.nnz / (X.shape[0] * X.shape[1]))
print('Density: {}'.format((density)))

#splitting data
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2)
print(X_test)
```

```
#classifier data
nb = MultinomialNB()
nb.fit(X_train, y_train)
nb.score(X_train, y_train)
preds = nb.predict(X_train)


preds = nb.predict(X_train)


#print(classification_report(y_train, preds))
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_train, preds))


#training report
plot_confusion_matrix(nb, X_train, y_train)


#print(classification_report(y_train, preds))
print(classification_report(y_train, nb.predict(X_train)))
from io import StringIO
classification = classification_report(y_train, preds)
s = StringIO(classification)
with open('classification.csv', 'w') as f:
   for line in s:
      f.write(line)
print(accuracy_score(y_train, preds))
```

```
accuracy = accuracy_score(y_train, preds)

a = np.asarray([accuracy])

np.savetxt("accuracy.csv", a, delimiter=",", fmt='%s')


Final = df[['content', 'sentiment', 'Polarity']]

Final =
Final.rename(columns={'content':'Tweet','sentiment':'sentiment','Polarity':'Polarity'
})

#fit the model on training set

model = LogisticRegression()

model.fit(X_test, y_test)


#save the model disk

filename = 'model_analisis.pkl'

pickle.dump(model, open(filename, 'wb'))


#load the model from disk

loaded_model = pickle.load(open(filename, 'rb'))

result = loaded_model.score(X_test, y_test)

print(result)


loaded_model.predict(X_test)
```