**LAMPIRAN LISTING PROGRAM**

**Autentikasi Crawling**

import tweepy

from tweepy import OAuthHandler

import json

import datetime as dt

import time

import os

import sys

def load\_api():

consumer\_key = 'qCqIZvhVeDsw4gGqIRAuu1b83'

consumer\_secret = 'uEwknWBFV62CvCdkapwP1SLCog3BDHB1DEhj6A9q5Lhp3VA8pl'

access\_token = '569062312-FJgdeaQlnl5GP7v0pSeYyLPqvjKY8CF6YGQfalKI'

access\_secret = 'kiw2YYkt8BbNcEequ4tCsHhgPeYPaJ1ZJAF9on3rbGldy'

auth = OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_secret)

return tweepy.API(auth)

def tweet\_search(api, query, max\_tweets, max\_id, since\_id, geocode):

searched\_tweets = []

while len(searched\_tweets) < max\_tweets:

remaining\_tweets = max\_tweets - len(searched\_tweets)

try:

new\_tweets = api.search(q=query, count=remaining\_tweets,

since\_id=str(since\_id),

max\_id=str(max\_id-1))

# geocode=geocode)

print('found',len(new\_tweets),'tweets')

if not new\_tweets:

print('no tweets found')

break

searched\_tweets.extend(new\_tweets)

max\_id = new\_tweets[-1].id

except tweepy.TweepError:

print('exception raised, waiting 15 minutes')

print('(until:', dt.datetime.now()+dt.timedelta(minutes=15), ')')

time.sleep(15\*60)

break # stop the loop

return searched\_tweets, max\_id

def get\_tweet\_id(api, date='', days\_ago=7, query='a'):

''' Function that gets the ID of a tweet. This ID can then be

used as a 'starting point' from which to search. The query is

required and has been set to a commonly used word by default.

The variable 'days\_ago' has been initialized to the maximum

amount we are able to search back in time (9).'''

if date:

# return an ID from the start of the given day

td = date + dt.timedelta(days=1)

tweet\_date = '{0}-{1:0>2}-{2:0>2}'.format(td.year, td.month, td.day)

tweet = api.search(q=query, count=1, until=tweet\_date)

else:

# return an ID from \_\_ days ago

td = dt.datetime.now() - dt.timedelta(days=days\_ago)

tweet\_date = '{0}-{1:0>2}-{2:0>2}'.format(td.year, td.month, td.day)

# get list of up to 10 tweets

tweet = api.search(q=query, count=10, until=tweet\_date)

print('search limit (start/stop):',tweet[0].created\_at)

# return the id of the first tweet in the list

return tweet[0].id

def write\_tweets(tweets, filename):

''' Function that appends tweets to a file. '''

with open(filename, 'a') as f:

for tweet in tweets:

json.dump(tweet.\_json, f)

f.write('\n')

def main():

''' This is a script that continuously searches for tweets

that were created over a given number of days. The search

dates and search phrase can be changed below. '''

''' search variables: '''

search\_phrases = ['kpuid', 'kapeu']

time\_limit = 1.5 # runtime limit in hours

max\_tweets = 100 # number of tweets per search (will be

# iterated over) - maximum is 100

min\_days\_old, max\_days\_old = 1, 10 # search limits e.g., from 7 to 8

# gives current weekday from last week,

# min\_days\_old=0 will search from right now

USA = '39.8,-95.583068847656,2500km' # this geocode includes nearly all American

# loop over search items,

# creating a new file for each

for search\_phrase in search\_phrases:

print('Search phrase =', search\_phrase)

''' other variables '''

name = search\_phrase.split()[0]

json\_file\_root = name + '/' + name

os.makedirs(os.path.dirname(json\_file\_root), exist\_ok=True)

read\_IDs = False

# open a file in which to store the tweets

if max\_days\_old - min\_days\_old == 1:

d = dt.datetime.now() - dt.timedelta(days=min\_days\_old)

day = '{0}-{1:0>2}-{2:0>2}'.format(d.year, d.month, d.day)

else:

d1 = dt.datetime.now() - dt.timedelta(days=max\_days\_old-1)

d2 = dt.datetime.now() - dt.timedelta(days=min\_days\_old)

day = '{0}-{1:0>2}-{2:0>2}\_to\_{3}-{4:0>2}-{5:0>2}'.format(

d1.year, d1.month, d1.day, d2.year, d2.month, d2.day)

json\_file = json\_file\_root + '\_' + day + '.json'

if os.path.isfile(json\_file):

print('Appending tweets to file named: ',json\_file)

read\_IDs = True

# authorize and load the twitter API

api = load\_api()

# set the 'starting point' ID for tweet collection

if read\_IDs:

# open the json file and get the latest tweet ID

with open(json\_file, 'r') as f:

lines = f.readlines()

max\_id = json.loads(lines[-1])['id']

print('Searching from the bottom ID in file')

else:

# get the ID of a tweet that is min\_days\_old

if min\_days\_old == 0:

max\_id = -1

else:

max\_id = get\_tweet\_id(api, days\_ago=(min\_days\_old-1))

# set the smallest ID to search for

since\_id = get\_tweet\_id(api, days\_ago=(max\_days\_old-1))

print('max id (starting point) =', max\_id)

print('since id (ending point) =', since\_id)

''' tweet gathering loop '''

start = dt.datetime.now()

end = start + dt.timedelta(hours=time\_limit)

count, exitcount = 0, 0

while dt.datetime.now() < end:

count += 1

print('count =',count)

# collect tweets and update max\_id

tweets, max\_id = tweet\_search(api, search\_phrase, max\_tweets,

max\_id=max\_id, since\_id=since\_id,

geocode=USA)

# write tweets to file in JSON format

if tweets:

write\_tweets(tweets, json\_file)

exitcount = 0

else:

exitcount += 1

if exitcount == 3:

if search\_phrase == search\_phrases[-1]:

sys.exit('Maximum number of empty tweet strings reached - exiting')

else:

print('Maximum number of empty tweet strings reached - breaking')

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Preprocessing Data**

#Cleaning Tweet

import re

from pandas import DataFrame

def clean(idx, text):

text = text.strip()

try:

text = text.split('%d\_efw7912jdqw['%idx)[1]

except:

pass

text = text.split(']\_932ruo')

if len(text) == 2:

idx += 1

text = text[0]

text = re.sub(r"http\S+", '', text)

text = re.sub(r"#\S+", '', text)

text = re.sub(r"RT @[^\s]+:", '', text)

text = re.sub(r"@", '', text)

text = re.sub(r"[^0-9a-zA-Z ]+", '', text)

text = re.sub(r"RT dan reply.\*$", '', text).lower().rstrip()

return idx, text

df = DataFrame(columns=['Tweet'])

idx = 0

concat = False

with open('kpu.txt', 'r') as f:

lines = f.readlines()

for line in lines:

if not line.isspace():

last\_idx = idx

if concat:

idx, next\_text = clean(last\_idx, line)

text += ' '+next\_text

else:

idx, text = clean(last\_idx, line)

if last\_idx == idx:

concat = True

else:

concat = False

print(idx)

df = df.append({'Tweet': text}, ignore\_index=True)

df.to\_csv(r"kpu.csv")

**Remove Double Tweet**

#%%

import pandas as pd

import csv

#%%

df = pd.read\_csv('kpu\_no\_double.csv')

#%%

df

#%%

df.drop\_duplicates(subset=1, inplace=True)

**Tokenizing, Labelling Lexicon**

#%%

import pandas as pd

import re

from tqdm import tqdm\_notebook as tqdm

#%%

pos = []

with open("poststem2.txt") as f:

lines = f.readlines()

for line in lines:

line = line.split(',')[0]

pos.append(line)

#%%

kata\_kata = {kata:1 for kata in pos}

#%%

with open('budi/negstem2.txt') as f:

lines = f.readlines()

for line in lines:

line = line.split(',')[0]

try:

if kata\_kata[line] != 1:

kata\_kata[line] = -1

except KeyError:

kata\_kata[line] = -1

#%%

df = pd.read\_csv('budi/stemmed\_kpu\_no\_jutsu.csv')

#%%

del df['Unnamed: 0']

#%%

words = df['Stemmed']

#%%

def stem(ab):

a = ab.split()

a = [re.sub(r'(mem|kan|ter|per)', '', b) for b in a]

return ' '.join(a)

#%%

for i, ws in enumerate(words):

words.iloc[i] = stem(ws)

#%%

scores = []

for ws in tqdm(words):

wss = ws.split()

sc = 0

i = 0

for w in wss:

try:

sc += kata\_kata[w]

i += 1

except:

continue

if i != 0:

scores.append(sc/i)

else:

scores.append(0.0)

#%%

i = 0

for s in scores:

if s == 0:

i += 1

#%%

mask = scores != 0.0

#%%

scores = scores[mask]

#%%

import numpy as np

#%%

scores = np.array(scores)

#%%

for i, s in enumerate(scores):

if s > 0:

scores[i] = 1

elif s < 0:

scores[i] = -1

else:

scores[i] = 0

#%%

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

from sklearn.model\_selection import GridSearchCV, train\_test\_split

#%%

vectorizer = TfidfVectorizer(min\_df=0.01, max\_df=3.0)

model = GridSearchCV(SVC(), {'gamma':[1, 0.1, 0.001, 0.0001, 0.00001],

'kernel':['linear', 'rbf']},

refit=True, verbose=0)

#%%

words\_f = vectorizer.fit\_transform(raw\_documents=words)

#%%

x\_train, x\_test, y\_train, y\_test = train\_test\_split(words\_f, scores,

test\_size=0.2, shuffle=True)

#%%

model.fit(x\_train, y\_train)

#%%

model = model.best\_estimator\_

#%%

model.fit(x\_train, y\_train)

#%%

y\_pred = model.predict(x\_test)

#%%

report = classification\_report(y\_test, y\_pred, output\_dict=True)

report

#%%

scores

pd.DataFrame({

"Tweet" : words,

"Scores" : scores

}).to\_csv("kpu\_scored1.csv")

df1 = pd.DataFrame({

"Tweet" : words,

"Scores" : scores

})

df1 = df1[df1["Scores"] != 0.0]

df1

df1.to\_csv("kpu\_scored\_no\_netral.csv")

dfpos = pd.read\_csv("kpu\_scored\_no\_netral.csv")

dfpos.rename({"Unnamed: 0":"a"}, axis="columns",inplace=True)

del dfpos["a"]

dfpos = dfpos[dfpos["Scores"] != 1.0]

dfpos.to\_csv("kpu\_negatif.csv")

dfpos

**Stemming**

#%%

from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory

import spacy

import pandas as pd

from tqdm import tqdm

#%%

remover = StopWordRemoverFactory().create\_stop\_word\_remover()

tokener = spacy.blank("id")

def stem\_and\_remove(ws):

temp = ""

ws = tokener(ws)

for w in ws:

temp += "%s "%w.lemma\_.strip()

temp = remover.remove(temp)

return temp

sw = open("stopword3.txt", 'r')

\_sw = sw.readlines()

for i, s in enumerate(\_sw):

\_sw[i] = s.strip()

remover.dictionary.add\_words(\_sw)

#if remover.dictionary.contains("yuhu"):

# print('exist')

sw.close()

#file\_name = ["pbw", "jkw"]

file\_name = ['kpu']

for fn in file\_name:

#filename = "stemmed\_%s\_no\_jutsu.csv"%fn

#original = "%s\_no\_jutsu.csv"%fn

filename = "budi/%s1\_ready\_to\_stem.csv"%fn

df = pd.read\_csv(filename)

#df\_ori = pd.read\_csv(original)

#print(len(df\_ori["Tweet"]))

#print(len(df["Tweet"]))

df["Tweet"] = df["Tweet"].str.replace("\d+", "")

tweets = []

#tweets\_ori = []

for tw in tqdm(list(zip(df["Tweet"]))):

clean\_tw = stem\_and\_remove(tw)

\_tw = clean\_tw.split(" ")

if len(\_tw) >= 4:

try:

tweets.append(clean\_tw)

except Exception:

continue

pd.DataFrame({

#"Tweet": tweets\_ori,

"Stemmed": tweets,

}).to\_csv("stemmed\_%s\_no\_jutsu.csv"%fn)

print("Done %s"%filename)