APPENDIX

SCRAPING TRIBUNNEWS

```
from urllib2 import urlopen
import pandas as pd
#Scrapping Tribun News
URL_tribun = urlopen('https://www.tribunnews.com/news').read()
from bs4 import BeautifulSoup
soup_tribun = BeautifulSoup(URL_tribun,'lxml')
title_tribun=[]
text_tribun=[]
news_tribun = soup_tribun.find_all('li','p1520 art-list pos_rel')
i=1
for x in news_tribun:
         title_tribun.append(x.find('a','f20  ln24  fbo  txt-oev-
2').get_text())
     text_tribun.append(x.find('div','grey2 pt5 f13 ln18 txt-oev-
3').get_text())
data_tribun={'title_tribun':title_tribun,'text_tribun':text_tribun
}
df_tribun = pd.DataFrame(data_tribun)
```

SCRAPING LIPUTAN 6

```
#Scrapping Liputan 6
URL_liputan6 = urlopen('https://www.liputan6.com/news').read()
# page_liputan6 = requests.get(URL_liputan6)
soup_liputan6 = BeautifulSoup(URL_liputan6,'lxml')
title_liputan6=[]
text_liputan6=[]
news_liputan6    =    soup_liputan6.find_all('article','articles--
iridescent-list--item articles--iridescent-list--text-item')
for x in news_liputan6:
        title_liputan6.append(x.find('span','articles--iridescent-
list--text-item__title-link-text').get_text())
    text_liputan6.append(x.find('div','articles--iridescent-list--
text-item__summary    articles—iridescent-list—text-item__summary-
seo').get_text())
```

SCRAPING BBC

```
#Scrapping BBC.com
URL_bbc                                                          =
urlopen('https://www.bbc.com/indonesia/topics/cjgn7k8yx4gt').read(
)
# page_kompas = requests.get(URL_kompas)
soup_bbc = BeautifulSoup(URL_bbc,'lxml')
title_bbc=[]
text_bbc=[]
data_bbc={'title_bbc':title_bbc,'text_bbc':text_bbc}
df_bbc = pd.DataFrame(data_bbc).head(3)
```

RESULT SCRAPING

```
title_news=[df_bbc['title_bbc'],df_liputan6['title_liputan6'],df_t
ribun['title_tribun']]
text_news=[df_bbc['text_bbc'],df_liputan6['text_liputan6'],df_trib
un['text_tribun']]
a = pd.concat(title_news)
b = pd.concat(text_news)
result_join_news = pd.DataFrame(a,columns=['title_news'])
result_join_news['text_news']                                    =
pd.DataFrame(b,columns=['text_news'])
result_join_news
```

EXPORT RESULT SCRAPING TO JSON

```
#to JSON Local
# result_join_news
ag = result_join_news.to_json(orient='values')
with open('Doc_News_28April2020.json', 'w') as f:
    f.write(ag)
```

EXPORT RESULT SCRAPING TO CSV

```
# CSV
result_join_news.to_csv('Doc_News_28April2020.csv', encoding='utf-
8')
```

IMPORT LIBRARY

```
import requests
import pandas as pd
import numpy as np
import re
import json
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from pprint import pprint
```

GETTING DATA TURNBACKHOAX

```
# # #News turnbackhoax.id
response_hoax                                                    =
requests.get('https://yudistira.turnbackhoax.id/Antihoax/title/sal
ah/abae27d5f0fcf30308365b8ceeaf1fa4')
response_real                                                    =
requests.get('https://yudistira.turnbackhoax.id/Antihoax/title/kla
rifikasi/abae27d5f0fcf30308365b8ceeaf1fa4')

# News Tribun liputan6 BBC
response_news                                                    =
pd.read_csv("document/doc_berita.csv",encoding='utf8')
```

LOAD DATA

```
def load_read_data(response_hoax,response_real,response_news):
    #TITLE
```

```
    #Loads Data
    loads_hoax = json.loads(response_hoax.text)
    loads_json_hoax = json.dumps(loads_hoax)
    loads_real = json.loads(response_real.text)
    loads_json_real = json.dumps(loads_real)
    #Read DATA

    ##HOAX Turnbackhoax
    data_hoax=pd.read_json(loads_json_hoax)
    data_hoax_head= data_hoax
    data_hoax_title = data_hoax_head['title']
    data_hoax_content = data_hoax_head['content']

    ##REAL Turnbackhoax
    data_real=pd.read_json(loads_json_real)
    data_real_head= data_real
    data_real_title = data_real_head['title']
    data_real_content = data_real_head['content']
    ##TURNBACK RAW DATA
                                    data_hoax_turnback          =
{'title':data_hoax_title,'content':data_hoax_content}
                                    data_real_turnback          =
{'title':data_real_title,'content':data_real_content}

    ##News Tribun,liputan6,BBC
                                      data_real_title           =
pd.concat([          data_real_title,response_news['title_news']],
ignore_index=True)


    #CONTENT
    data_hoax_content = data_hoax_content
                                    data_real_content           =
pd.concat([data_real_content,response_news['text_news']],
ignore_index=True)
                                                              return
data_hoax_turnback,data_real_turnback,data_hoax_title,data_real_ti
tle,data_hoax_content,data_real_content
```

MAKING TYPE ASCII

```
def
to_ascii(data_hoax_title,data_hoax_content,data_real_title,data_re
al_content):
    #ASCII HOAX

    ascii_title_hoax=[]
    for sentences in data_hoax_title:
        text = sentences.encode('ascii','ignore')
        ascii_title_hoax.append(text)

    ascii_content_hoax=[]
    for sentences in data_hoax_content:
        text = sentences.encode('ascii','ignore')
```

```
        ascii_content_hoax.append(text)



    #ASCII REAL
    ascii_title_real=[]
    for sentences in data_real_title:
        text = sentences.encode('ascii','ignore')
        ascii_title_real.append(text)

    ascii_content_real=[]
    for sentences in data_real_content:
        text = sentences.encode('ascii','ignore')
        ascii_content_real.append(text)

return
ascii_title_hoax,ascii_content_hoax,ascii_title_real,ascii_content
_real
```

STORE DATA

```
def
store_data(ascii_title_hoax,ascii_content_hoax,ascii_title_real,as
cii_content_real):

                                                    data_hoax=
{'title':ascii_title_hoax,'content':ascii_content_hoax}
                                                    data_real=
{'title':ascii_title_real,'content':ascii_content_real}

    title_content_hoax = pd.DataFrame(data_hoax)
    title_content_real = pd.DataFrame(data_real)

    return title_content_hoax,title_content_real

title_content_hoax,title_content_real=store_data(ascii_title_hoax,
ascii_content_hoax,ascii_title_real,ascii_content_real)
```

REMOVE [...] IN THE TITLE

```
def remove_char(title_content_hoax,title_content_real):

    # Print Data for title
    title_data_hoax = title_content_hoax['title']
    content_data_hoax = title_content_hoax['content']
    title_data_real = title_content_real['title']
    content_data_real =  title_content_real['content']

    #Remove character
    import re
    import string
    rem = string.punctuation
    pattern = r".+]".format(rem)
    pattern2 = r"\(.+I.".format(rem)
```

```
    result_hoax =title_data_hoax.str.replace(pattern,'')
    result_data_hoax_title =result_hoax.str.replace(pattern2,'')

    result_hoax2 =content_data_hoax.str.replace(pattern,'')
                                        result_data_hoax_content
=result_hoax2.str.replace(pattern2,'')
    result_real =title_data_real.str.replace(pattern,'')
    result_data_real_title =result_real.str.replace(pattern2,'')

    result_real2 =content_data_real.str.replace(pattern,'')
                                        result_data_real_content
=result_real2.str.replace(pattern2,'')



    title_content_hoax['title'] = result_data_hoax_title
    title_content_real['title'] = result_data_real_title
    title_content_hoax['content'] = result_data_hoax_content
    title_content_real['content'] = result_data_real_content


    return title_content_hoax,title_content_real
```

FOLDING

```
def folding(title_content_hoax,title_content_real):
    #Folding HOAX
                folding_title_hoax=    [x.lower()    for    x    in
title_content_hoax['title']]
               folding_content_hoax=    [x.lower()    for    x    in
title_content_hoax['content']]

     #Folding Real
                folding_title_real=    [x.lower()    for    x    in
title_content_real['title']]
               folding_content_real=    [x.lower()    for    x    in
title_content_real['content']]


    title_content_hoax['title'] = folding_title_hoax
    title_content_real['title'] = folding_title_real
    title_content_hoax['content'] = folding_content_hoax
    title_content_real['content'] = folding_content_real

    return title_content_hoax,title_content_real
```

STEMMING

```
def stemming_title(title_content_hoax,title_content_real):
    #Stemming HOAX
    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    output1 = []
```

```
    for sentence1 in title_content_hoax['title']:
        output1.append(stemmer.stem(sentence1))



    #Stemming REAL
    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    output2 = []
    for sentence2 in title_content_real['title']:
        output2.append(stemmer.stem(sentence2))


    title_content_hoax['title'] = output1
    title_content_real['title'] = output2

    return title_content_hoax,title_content_real

def stemming_content(title_content_hoax,title_content_real):
    #Stemming HOAX
    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    output1 = []
    for sentence1 in title_content_hoax['content']:
        output1.append(stemmer.stem(sentence1))

    #Stemming REAL
    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    output2 = []
    for sentence2 in title_content_real['content']:
        output2.append(stemmer.stem(sentence2))


    title_content_hoax['content'] = output1
    title_content_real['content'] = output2

    return title_content_hoax,title_content_real

title_content_hoax,title_content_real                                =
stemming_title(title_content_hoax,title_content_real)
title_content_hoax,title_content_real                                =
stemming_content(title_content_hoax,title_content_real)
```

STOPWORDS

```
from    Sastrawi.StopWordRemover.StopWordRemoverFactory    import
StopWordRemoverFactory
```

```
factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

def stopword_title(title_content_hoax,title_content_real):
    #Stopword HOAX
    hasil_stop_hoax_title=[]
    for title in title_content_hoax['title']:
        title = stopword.remove(title)
        hasil_stop_hoax_title.append(title)

    #Stopword REAL
    hasil_stop_real_title=[]
    for title in title_content_real['title']:
        title = stopword.remove(title)
        hasil_stop_real_title.append(title)

    title_content_hoax['title'] = hasil_stop_hoax_title
    title_content_real['title'] = hasil_stop_real_title

    return title_content_hoax,title_content_real

def stopword_content(title_content_hoax,title_content_real):
    hasil_stop_hoax_content=[]
    for content in title_content_hoax['content']:
        content = stopword.remove(content)
        hasil_stop_hoax_content.append(content)

    hasil_stop_real_content=[]
    for content in title_content_real['content']:
        content = stopword.remove(content)
        hasil_stop_real_content.append(content)

    title_content_hoax['content'] = hasil_stop_hoax_content
    title_content_real['content'] = hasil_stop_real_content

    return title_content_hoax,title_content_real

title_content_hoax,title_content_real                                    =
stopword_title(title_content_hoax,title_content_real)
title_content_hoax,title_content_real                                    =
stopword_content(title_content_hoax,title_content_real)
```

DATA ALL PRE-PROCESSING

```
def data_all_preprocessing(title_content_hoax,title_content_real):
    #HOAX
    title_content_hoax['label'] = 'HOAX'

    #REAL
    title_content_real['label'] = 'REAL'

    #JOIN REAL & HOAX
```

```
                                        data_all_prepos          =
pd.concat([title_content_hoax,title_content_real],ignore_index=Tru
e)
    #     data_all_prepos.head(10)

    return data_all_prepos
```

EXPORT TO CSV RESULT PRE-PROCESSING

```
# CSV DATA (STORE PreProcessing)
data_PreProcessing= pd.DataFrame(data_all_prepos)
data_PreProcessing.to_csv('data_PreProcessing.csv',index=False)
```

RESULT PRE-PROCESSING 1100 DATA

```
data_all_prepos=pd.read_csv("document/
data_PreProcessing.csv",encoding='utf8').astype(str)

data_all_prepos_hoax = data_all_prepos[:500] #HOAX 500
data_all_prepos_real = data_all_prepos[1936:2536]#REAL 600

data_all_prepos=
pd.concat([data_all_prepos_hoax,data_all_prepos_real])
data_all_prepos                                          =
data_all_prepos.reset_index().drop(columns=['index'],axis=1)
```

SPLIT DATA HOAX

```
def split_data_title(data_all_prepos):
    all_tot_title=[]
    for x in data_all_prepos['title'].values:
        split_mentah = x.split(" ")
        for y in split_mentah:
            all_tot_title.append(y)

    all_tot_title = set(all_tot_title)
    return all_tot_title

def split_data_content(data_all_prepos):
    all_tot_content=[]
    for x in data_all_prepos['content'].values:
        split_mentah = x.split(" ")
        for y in split_mentah:
            all_tot_content.append(y)

    all_tot_content=set(all_tot_content)
    return all_tot_content

all_tot_title =split_data_title(data_all_prepos)
all_tot_content =split_data_content(data_all_prepos)
```

TERM FREQUENCY

```
def term_frekuensi_title(all_tot_title,data_all_prepos):
    term_info_frek =[]
    info = []
```

```
    for term in all_tot_title:
        i = 0;
        jumlah = 0
        for lists in data_all_prepos['title']:
                count = lists.count(term)
                jumlah += count
                info1 = {"doc": i, "count" : count, "term":term}
                info.append(info1)
                i +=1
        term_info = {"term": term, "info":info, "df" : jumlah}
        term_info_frek.append(term_info)

    df_term_info_title = pd.DataFrame(term_info_frek)
    # df_term_info['info']

    df_count_doc_term_title = pd.DataFrame(info)
#     df_count_doc_term

                                    group_term_count_title        =
df_count_doc_term_title.groupby(['term','doc'])
#     group_term_count.first()

                                                                  return
df_term_info_title,df_count_doc_term_title,group_term_count_title


def term_frekuensi_content(all_tot_content,data_all_prepos):
    term_info_frek =[]
    info = []

    for term in all_tot_content:
        i = 0;
        jumlah = 0
        for lists in data_all_prepos['content']:
            count = lists.count(term)
            jumlah += count
            info1 = {"doc": i, "count" : count, "term":term}
            info.append(info1)
            i +=1
        term_info = {"term": term, "info":info, "df" : jumlah}
        term_info_frek.append(term_info)

    df_term_info_content = pd.DataFrame(term_info_frek)
    # df_term_info['info']

    df_count_doc_term_content = pd.DataFrame(info)
#     df_count_doc_term
                                    group_term_count_content        =
df_count_doc_term_content.groupby(['term','doc'])
#     group_term_count.first()
```

```
                                                                    return
df_term_info_content,df_count_doc_term_content,group_term_count_co
ntent

df_term_info_title,df_count_doc_term_title,group_term_count_title
= term_frekuensi_title(all_tot_title,data_all_prepos)
df_term_info_content,df_count_doc_term_content,group_term_count_co
ntent = term_frekuensi_content(all_tot_content,data_all_prepos)
```

INVERSE DOCUMENT FREQUENCY

```
def idf_title(df_term_info_title,data_all_prepos):

    #Document Frekuensi

    import math
    idf = []

    for tf in df_term_info_title['df']:
        a= math.log1p(len(data_all_prepos['title'])/tf)
        idf.append(a)

    # pd.DataFrame([df_term['term']],columns=['idf'])
    df_term = pd.DataFrame(df_term_info_title['term'])
    df_idf = pd.DataFrame(idf, columns=['idf'])

                                            df_term_idf          =
pd.DataFrame({'term':df_term['term'],'idf':df_idf['idf']})
    group_idf_title =  df_term_idf.groupby(['term'])


    return group_idf_title


def idf_content(df_term_info_content,data_all_prepos):

    #Document Frekuensi

    import math
    idf = []

    for tf in df_term_info_content['df']:
        a= math.log1p(len(data_all_prepos['content'])/tf)
        idf.append(a)

    # pd.DataFrame([df_term['term']],columns=['idf'])
    df_term = pd.DataFrame(df_term_info_content['term'])
    df_idf = pd.DataFrame(idf, columns=['idf'])

                                              df_term_idf          =
pd.DataFrame({'term':df_term['term'],'idf':df_idf['idf']})
    group_idf_content =  df_term_idf.groupby(['term'])
```

```
    return group_idf_content

group_idf_title = idf_title(df_term_info_title,data_all_prepos)
group_idf_content                                                =
idf_content(df_term_info_content,data_all_prepos)
```

TF-IDF

```
def
tf_idf_title(group_idf_title,group_term_count_title,data_all_prepo
s):
                                                    tf_idf            =
group_idf_title['idf'].first()*group_term_count_title.first()
['count']
    tf_idf_frame = pd.DataFrame(tf_idf,columns=['tf_idf'])

    #SUM each DOC
    tf_idf_doc = tf_idf_frame.groupby(['doc'])['tf_idf'].sum()
    df_tf_idf_doc = pd.DataFrame(tf_idf_doc)
                                                          df_tf_idf_doc=
df_tf_idf_doc.reset_index().drop(columns=['doc'])


    hasil_tfidf_title = data_all_prepos.join(df_tf_idf_doc)

    return hasil_tfidf_title

def
tf_idf_content(group_idf_content,group_term_count_content,data_all
_prepos):
                                                    tf_idf            =
group_idf_content['idf'].first()*group_term_count_content.first()
['count']
    tf_idf_frame = pd.DataFrame(tf_idf,columns=['tf_idf'])

    #SUM each DOC
    tf_idf_doc = tf_idf_frame.groupby(['doc'])['tf_idf'].sum()
    df_tf_idf_doc = pd.DataFrame(tf_idf_doc)
                                                          df_tf_idf_doc=
df_tf_idf_doc.reset_index().drop(columns=['doc'])
    hasil_tfidf_content = data_all_prepos.join(df_tf_idf_doc)
    return hasil_tfidf_content
hasil_tfidf_title                                                =
tf_idf_title(group_idf_title,group_term_count_title,data_all_prepo
s)
hasil_tfidf_content                                              =
tf_idf_content(group_idf_content,group_term_count_content,data_all
_prepos)
```

CATEGORY TF-IDF

```
def category_tfidf_title(hasil_tfidf_title):

    result_category = []
    for x in hasil_tfidf_title['tf_idf']:
```

```
        if x <= 0 :
            result_category.append(5)
        elif 0 < x <= 50 :
            result_category.append(10)
        elif 50 < x <= 100 :
            result_category.append(15)
        elif 100 < x <= 150 :
            result_category.append(20)
        elif 150 < x <= 200 :
            result_category.append(25)
        elif 200 < x <= 250 :
            result_category.append(30)
        elif 250 < x <= 300 :
            result_category.append(35)
        elif 300 < x <= 350 :
            result_category.append(40)
        elif 350 < x <= 400 :
            result_category.append(45)
        elif 400 < x <= 450 :
            result_category.append(50)
        elif 450 < x <= 500 :
            result_category.append(55)
        elif 500 < x <= 550 :
            result_category.append(60)
        else:
            result_category.append(65)

                                                category            =
pd.DataFrame(result_category,columns=['kategori_tfidf_title'])
    hasil_tfidf_title= hasil_tfidf_title.join(category)

    return hasil_tfidf_title

def category_tfidf_content(hasil_tfidf_content):
    result_category = []
    for x in hasil_tfidf_content['tf_idf']:
        if x <= 0 :
            result_category.append(5)
        elif 0 < x <= 50 :
            result_category.append(10)
        elif 50 < x <= 100 :
            result_category.append(15)
        elif 100 < x <= 150 :
            result_category.append(20)
        elif 150 < x <= 200 :
            result_category.append(25)
        elif 200 < x <= 250 :
            result_category.append(30)
        elif 250 < x <= 300 :
            result_category.append(35)
        elif 300 < x <= 350 :
            result_category.append(40)
        elif 350 < x <= 400 :
            result_category.append(45)
```

```
        elif 400 < x <= 450 :
            result_category.append(50)
        elif 450 < x <= 500 :
            result_category.append(55)
        elif 500 < x <= 550 :
            result_category.append(60)
        else:
            result_category.append(65)

                                                    category           =
pd.DataFrame(result_category,columns=['kategori_tfidf_content'])
    hasil_tfidf_content= hasil_tfidf_content.join(category)

    return hasil_tfidf_content

hasil_tfidf_title = category_tfidf_title(hasil_tfidf_title)
hasil_tfidf_content = category_tfidf_content(hasil_tfidf_content)
```

RESULT TF-IDF FRAME

```
df_content                                                          =
{'tf_idf_content':hasil_tfidf_content['tf_idf'],'kategori_tfidf_co
ntent':hasil_tfidf_content['kategori_tfidf_content']}
df_content = pd.DataFrame(df_content)

result_hasil_tfidf                                                  =
hasil_tfidf_title.join(df_content).rename(columns={'tf_idf':'tf_id
f_title'})
```

TRAINING DATA AND TESTING DATA

```
def Train_test(result_hasil_tfidf):
    # #Split Train & Testing TITLE

    X= result_hasil_tfidf

    num_random = np.random.rand(X.shape[0])
    # split = num_random < np.percentile(arr_rand, 80)
    split = num_random < 0.6

    X_train =  X[split] #TRUE
    X_test  =  X[~split] #FALSE

    return X_train,X_test
```

READ CSV RESULT TF-IDF

```
#                           hasil_stem                                =
pd.read_csv("document/data_prepos.csv",encoding='utf8')
hasil_tfidf                                                         =
pd.read_csv("document/analisis/hasil_tfidf.csv",encoding='utf8')
```

READ CSV TRAINING DATA 60% AND TESTING DATA 40%

```
#60% 40%
```

```
data_X_train6040                                                    =
pd.read_csv("document/analisis/X_train6040.csv",encoding='utf8')#
0.6 Train 644 data
data_X_test6040                                                     =
pd.read_csv("document/analisis/X_test6040.csv",encoding='utf8')  #
456 data
```

READ CSV TRAINING DATA 70% AND TESTING DATA 30%

```
#70% 30%
data_X_train7030                                                    =
pd.read_csv("document/analisis/X_train7030.csv",encoding='utf8')#
0.8 Train 768 data
data_X_test7030                                                     =
pd.read_csv("document/analisis/X_test7030.csv",encoding='utf8')
#332 data
```

READ CSV TRAINING DATA 80% AND TESTING DATA 20%

```
#80% 20%
data_X_train8020                                                    =
pd.read_csv("document/analisis/X_train8020.csv",encoding='utf8')#
0.8 Train 886 data
data_X_test8020                                                     =
pd.read_csv("document/analisis/X_test8020.csv",encoding='utf8')
#214 data
```

READ CSV TRAINING DATA 90% AND TESTING DATA 10%

```
#90% 10%
data_X_train9010                                                    =
pd.read_csv("document/analisis/X_train9010.csv",encoding='utf8')#
0.8 Train 899 data
data_X_test9010                                                     =
pd.read_csv("document/analisis/X_test9010.csv",encoding='utf8')
#201 data
```

REMOVE COLUMN SVM (TITLE)

```
def remove_col(data_X_train,data_X_test):

    # data X remove column label
                                              X_train_title         =
data_X_train.drop(columns='label').drop(columns='content').drop(co
lumns='kategori_tfidf_content').drop(columns='tf_idf_content').gro
upby(['title']).first().values
                                               X_test_title         =
data_X_test.drop(columns='label').drop(columns='content').drop(col
umns='kategori_tfidf_content').drop(columns='tf_idf_content').grou
pby(['title']).first().values

    # data Y
    kategori = {'HOAX':0,'REAL':1}
          data_X_train['kategori']  =  [kategori[r]  for  r  in
data_X_train['label']]
```

```
          data_X_test['kategori']   =   [kategori[r]   for   r   in
data_X_test['label']]

    y_train_kategori = data_X_train.groupby(['title'])
    y_train_title = y_train_kategori.first()['kategori'].values

    y_test_kategori = data_X_test.groupby(['title'])
    y_test_title = y_test_kategori.first()['kategori'].values

    return X_train_title,X_test_title,y_train_title,y_test_title

X_train_title,X_test_title,y_train_title,y_test_title            =
remove_col(data_X_train9010,data_X_test9010)
```

REMOVE COLUMN SVM (CONTENT)

```
def remove_col(data_X_train,data_X_test):

    # data X remove column label
                                        X_train_content          =
data_X_train.drop(columns='label').drop(columns='content').drop(co
lumns='kategori_tfidf_title').drop(columns='tf_idf_title').groupby
(['title']).first().values
                                         X_test_content          =
data_X_test.drop(columns='label').drop(columns='content').drop(col
umns='kategori_tfidf_title').drop(columns='tf_idf_title').groupby(
['title']).first().values

    # data Y
    kategori = {'HOAX':0,'REAL':1}
          data_X_train['kategori']   =   [kategori[r]   for   r   in
data_X_train['label']]
          data_X_test['kategori']   =   [kategori[r]   for   r   in
data_X_test['label']]

    y_train_kategori = data_X_train.groupby(['title'])
    y_train_content = y_train_kategori.first()['kategori'].values

    y_test_kategori = data_X_test.groupby(['title'])
    y_test_content = y_test_kategori.first()['kategori'].values

                                                                 return
X_train_content,X_test_content,y_train_content,y_test_content
```

SUPPORT VECTOR MACHINE

```
import numpy as np

class SVM :
                                                                def
__init__(self,learning_rate=0.0001,lambda_parameter=0.01,n_iter=10
0):
        #constructors
        self.learning_rate=learning_rate
```

```
        self.lambda_parameter = lambda_parameter
        self.n_iter=n_iter
        self.w = None
        self.b = None

    def fit(self,X,y):
        n_sample=len(X)
        n_feature=2
        y = np.where(y<=0,-1,1)
        self.w = np.zeros(n_feature)
        self.b = 0

        for x in range(self.n_iter):
            for i,xi in enumerate(X):
                kondisi= y[i] * (np.dot(self.w,xi)-self.b)>=1
#                 print(y[i])
                if kondisi:
                    self.w = self.w - self.learning_rate * (2 *
self.lambda_parameter * self.w)
                else:
                    self.w = self.w - self.learning_rate * (2 *
self.lambda_parameter * self.w - np.dot(y[i],xi))
                    self.b = self.b - self.learning_rate * y[i]
    def predict(self,X):
        hasil = np.dot(X,self.w)-self.b
        return np.where(hasil<=0,-1,1)
```

TESTING TITLE SECTION USING SUPPORT VECTOR MACHINE

```
def
SVM_test(X_train_title,y_train_title,X_test_title,y_test_title):

    y_train_title = np.where(y_train_title <= 0, -1, 1)
    y_test_title = np.where(y_test_title <= 0, -1, 1)

    svm=SVM()
    svm.fit(X_train_title, y_train_title)
    prediksi_svm = svm.predict(X_test_title)

#     print(svm.w, svm.b)
    # print(prediksi)

    return prediksi_svm,svm,y_train_title,y_test_title

prediksi_svm,svm,y_train_title,y_test_title                    =
SVM_test(X_train_title,y_train_title,X_test_title,y_test_title)
```

TESTING CONTENT SECTION USING SUPPORT VECTOR MACHINE

```
def
SVM_test(X_train_content,y_train_content,X_test_content,y_test_con
tent):

    y_train_content = np.where(y_train_content <= 0, -1, 1)
    y_test_content = np.where(y_test_content <= 0, -1, 1)
```

```
    svm=SVM()
    svm.fit(X_train_content, y_train_content)
    prediksi_svm = svm.predict(X_test_content)

    # print(svm.w, svm.b)
    # print(prediksi)

    return prediksi_svm,svm,y_train_content,y_test_content

prediksi_svm,svm,y_train_content,y_test_content                    =
SVM_test(X_train_content,y_train_content,X_test_content,y_test_con
tent)
```

HYPERPLANE LINEAR SVM

```
def get_hyperplane(x, w, b, offset):
    return (-w[0] * x + b + offset ) / w[1]
```

VISUALIZATION HYPERPLANE LINEAR SVM

```
def visualisasi_svm(X):

    color = {-1:'r',1:'g'}
    fig = plt.figure(figsize=(20,10))
    ax = fig.add_subplot(1,1,1)

     plt.scatter(X[:,0], X[:,1], marker='o',c=[color[x] for x in
prediksi_svm ])

    # calculate X_test[:,0] min & max
    kolom0_minimal = np.amin(X[:,0])
    kolom0_maksimal = np.amax(X[:,0])
    # print(kolom0_minimal,kolom0_maksimal)

    ##################################################

    # calculate hyperplane value offset 0
    x01 = get_hyperplane(kolom0_minimal, svm.w, svm.b, 0)
    x02 = get_hyperplane(kolom0_maksimal, svm.w, svm.b, 0)
    #print(x01,x02)

    # calculate hyperplane value offset -1
    x11_negatif = get_hyperplane(kolom0_minimal, svm.w, svm.b, -1)
     x12_negatif = get_hyperplane(kolom0_maksimal, svm.w, svm.b, -
1)
#     print(x11_negatif,x12_negatif)

    # calculate hyperplane value offset 1
    x11_positif = get_hyperplane(kolom0_minimal, svm.w, svm.b, 1)
    x12_positif = get_hyperplane(kolom0_maksimal, svm.w, svm.b, 1)
    # print(x11_positif,x12_positif)

    ##################################################
```

```
    ax.plot([kolom0_minimal, kolom0_maksimal],[x01, x02], 'b--')
          ax.plot([kolom0_minimal,  kolom0_maksimal],[x11_negatif,
x12_negatif], 'black')
          ax.plot([kolom0_minimal,  kolom0_maksimal],[x11_positif,
x12_positif], 'black')

#     ax.plot([x0,sampai x1],[y0,sampai y1], 'black')

#     kolom1_minimal = np.amin(X_test_title[:,1])
#     kolom1_maksimal = np.amax(X_test_title[:,1])

#     ax.set_ylim([kolom1_minimal+2,kolom1_maksimal+2])

    ax.margins(0.05)
    plt.show()
```

MATRIX PREDICT SVM (TITLE)

```
def matriks_prediksi(y_test_title,prediksi_svm):
    matrix = confusion_matrix(y_test_title, prediksi_svm)

    return matrix

matrix= matriks_prediksi(y_test_title,prediksi_svm)
```

MATRIX PREDICT SVM (CONTENT)

```
def matriks_prediksi(y_test_content,prediksi_svm):
    matrix = confusion_matrix(y_test_content, prediksi_svm)

    return matrix

matrix= matriks_prediksi(y_test_content,prediksi_svm)
```

VISUALIZATION CONFUSION MATRIX SVM (TITLE)

```
import seaborn as sn
plt.figure(figsize=(12,10))
sn.heatmap(confusion_matrix(y_test_title,
prediksi_svm),xticklabels=['HOAX','FAKTA'],
yticklabels=['HOAX','FAKTA'],annot=True,fmt='d')
plt.title('Support Vector Machine Title (Linear Kernel)')
plt.xlabel('Y ASLI')
plt.ylabel('Y PREDIKSI')
plt.show()
```

VISUALIZATION CONFUSION MATRIX SVM (CONTENT)

```
import seaborn as sn
plt.figure(figsize=(12,10))
sn.heatmap(confusion_matrix(y_test_content,
prediksi_svm),xticklabels=['HOAX','FAKTA'],
yticklabels=['HOAX','FAKTA'],annot=True,fmt='d')
plt.title('Support Vector Machine Content (Linear Kernel)')
plt.xlabel('Y ASLI')
plt.ylabel('Y PREDIKSI')
```

```
plt.show()
```

ACCURACY, PRECISION, RECALL, F1-SCORE SVM (TITLE)

```
def performance(y_test_title, prediksi_svm):
    tp = confusion_matrix(y_test_title, prediksi_svm).item(0) #TP
    fp = confusion_matrix(y_test_title, prediksi_svm).item(1) #FP
    fn = confusion_matrix(y_test_title, prediksi_svm).item(2) #FN
    tn = confusion_matrix(y_test_title, prediksi_svm).item(3) #TN

    # (TP + TN ) / (TP+FP+FN+TN)
    accuracy_svm = (tp+tn)/float((tp+fp+fn+tn))

    # (TP) / (TP+FP)
    precision_svm = tp/float((tp+fp))

    # (TP) / (TP + FN)
    recall_svm = tp/float((tp+fn))

    #  2 * (Recall*Precission) / (Recall + Precission)
                                        f1score_svm=           2
*(recall_svm*precision_svm)/float((recall_svm+precision_svm))

                                                          return
tp,fn,fp,tn,accuracy_svm,precision_svm,recall_svm,f1score_svm
tp,fn,fp,tn,accuracy_svm,precision_svm,recall_svm,f1score_svm    =
performance(y_test_title, prediksi_svm)
# PRINT

print('Accuracy :',accuracy_svm*100)
print('Precision :',precision_svm*100)
print('Recall :',recall_svm*100)
print('F1-Score :',f1score_svm*100)
```

ACCURACY, PRECISION, RECALL, F1-SCORE SVM (CONTENT)

```
#matrix
tp = confusion_matrix(y_test_content, prediksi_svm).item(0) #TP
fp = confusion_matrix(y_test_content, prediksi_svm).item(1) #FP
fn = confusion_matrix(y_test_content, prediksi_svm).item(2) #FN
tn = confusion_matrix(y_test_content, prediksi_svm).item(3) #TN


# (TP + TN ) / (TP+FP+FN+TN)
accuracy_svm = (tp+tn)/float((tp+fp+fn+tn))

# (TP) / (TP+FP)
precision_svm = tp/float((tp+fp))

# (TP) / (TP + FN)
recall_svm = tp/float((tp+fn))

#  2 * (Recall*Precission) / (Recall + Precission)
f1score_svm=                                                    2
*(recall_svm*precision_svm)/float((recall_svm+precision_svm))
```

```
# PRINT

print('Accuracy :',accuracy_svm*100)
print('Precision :',precision_svm*100)
print('Recall :',recall_svm*100)
print('F1-Score :',f1score_svm*100)
```

COMPARE RESULT PREDICT SVM AND TESTING DATA (TITLE)

```
sum(y_test_title == prediksi_svm)
```

COMPARE RESULT PREDICT SVM AND TESTING DATA (CONTENT)

```
sum(y_test_title == prediksi_svm)
```

REMOVE COLUMN (RANDOM FOREST ALGORITHM (TITLE))

```
def remove_col(data_X_train,data_X_test):

    # data X remove column label
                                        X_train_title          =
data_X_train.drop(columns='content').drop(columns='kategori_tfidf_
content').drop(columns='tf_idf_content').groupby(['title']).first(
)
                                         X_test_title          =
data_X_test.drop(columns='content').drop(columns='kategori_tfidf_c
ontent').drop(columns='tf_idf_content').groupby(['title']).first()

    y_train_kategori = data_X_train.groupby(['title'])
    y_train_title = y_train_kategori.first()['label'].values

    y_test_kategori = data_X_test.groupby(['title'])
    y_test_title = y_test_kategori.first()['label'].values

    return X_train_title,X_test_title,y_train_title,y_test_title
```

REMOVE COLUMN (RANDOM FOREST ALGORITHM (CONTENT))

```
def remove_col(data_X_train,data_X_test):

    # data X remove column label
                                       X_train_content          =
data_X_train.drop(columns='content').drop(columns='kategori_tfidf_
title').drop(columns='tf_idf_title').groupby(['title']).first()
                                        X_test_content          =
data_X_test.drop(columns='content').drop(columns='kategori_tfidf_t
itle').drop(columns='tf_idf_title').groupby(['title']).first()

    y_train_kategori = data_X_train.groupby(['title'])
    y_train_content = y_train_kategori.first()['label'].values

    y_test_kategori = data_X_test.groupby(['title'])
    y_test_content = y_test_kategori.first()['label'].values

                                                            return
X_train_content,X_test_content,y_train_content,y_test_content
```

CHECK DATA PURE

```
#Check Data Pure
def check_data_pure(data):
    label = data[:,0]

    unique_label = np.unique(label)

    if len(unique_label) == 1:
        return True
    else:
        return False
```

CLASSIFICATION

```
def klasifikasi(data):
    label = data[:,0]

    unique_label,jml_label = np.unique(label,return_counts=True)

    index = np.argmax(jml_label)

    klasifikasi_label = unique_label[index]

    return klasifikasi_label
```

POTENSIAL SPLIT

```
def get_potensial_split(data):
    potential_splits = {}
    x, n_columns = data.shape
    for column_index in range(1,n_columns):
        potential_splits[column_index] = []
        values = data[:, column_index]
        unique_values = np.unique(values)

        for index in range(len(unique_values)):
            if index != 0:
                current_value = unique_values[index]
                previous_value = unique_values[index - 1]
                potential_split = (current_value + previous_value)
/ 2

                potential_splits[column_index].append(potential_sp
lit)
    return potential_splits
```

VISUAL POTENSIAL SPLIT

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.lmplot(data=X_train_title,x='tf_idf_title',y='kategori_tfidf_t
itle',hue='label',fit_reg=False,size=6,aspect=2)

# potensial_split[1] columns tfidf
```

```
plt.vlines(x=potensial_split[1],ymin=0,ymax=60)
```

FUNCTION SPLIT DATA

```
def split_data(data,split_column,split_value):
    split_data = data[:,split_column]

    data_min =  data[split_data<=split_value]
    data_max =  data[split_data>split_value]

    return data_min,data_max
```

FUNCTION ENTROPY

```
def hitung_entropy(data):
    label = data[:,0]
    unique_label,count_label = np.unique(label,return_counts=True)
    probabilitas =  count_label / float(sum(count_label))
    entropy = sum(probabilitas * -np.log2(probabilitas))

    return entropy
```

FUNCTION OVERALL ENTROPY

```
def hitung_overall_entropy(data_min,data_max):
    jml_data = len(data_min)+len(data_max)

    jml_data_min = len(data_min)/float(jml_data)
    jml_data_max = len(data_max)/float(jml_data)

     overall_entropy = jml_data_min * hitung_entropy(data_min) +
jml_data_max * hitung_entropy(data_max)

    return overall_entropy
```

FUNCTION BEST SPLIT

```
def best_split(data, potential_splits):
    high_entropy = 999 # High Entropy
    for col_index in potential_splits:
        for value in potential_splits[col_index]:
                        data_min,  data_max  =  split_data(data,
split_column=col_index, split_value=value)
                overall_entropy = hitung_overall_entropy(data_min,
data_max)
            if overall_entropy < high_entropy:
                overall_entropy = overall_entropy
                split_column = col_index
                split_value = value


    return split_column, split_value
```

DECISION TREE METHOD

```
def decisionTree_fit(df, i=0, min_samples=2, max_depth=2):

    # prepare data
    if i == 0:
        global column_headers
        column_headers = df.columns
        data = df.values

    else:
        data = df

    # check data pure, or panjang data less then sample or i same
max_depth
     if (check_data_pure(data) or len(data)<min_samples) or (i ==
max_depth):
        classification = klasifikasi(data)
        return classification


    # i is not 1
    else:

        i += 1
        # function before algorithm
        potential_splits = get_potensial_split(data)
                split_column,  split_value  =  best_split(data,
potential_splits)
             data_min,  data_max  =  split_data(data,  split_column,
split_value)



        # inisiasi sub-tree
        feature_name = column_headers[split_column]
        question = "{} <= {}".format(feature_name, split_value)
#         print(question)

        sub_tree = {question: []}
#         print(sub_tree)

        # recursif to find question
            small  =  decisionTree_fit(data_min,  i,  min_samples,
max_depth)
              big  =  decisionTree_fit(data_max,  i,  min_samples,
max_depth)

        if small == big:
            sub_tree = small
        else:
            sub_tree[question].append(small)
            sub_tree[question].append(big)

        return sub_tree
```

```
def predict_X(X, tree):
    question = tree.keys()[0].encode('utf-8')
#     print(question)
    feature_name, operator, value = question.split(" ")

    # ask question
    if operator == "<=":
        if X[feature_name] <= float(value):
            answer = tree[question][0] #answer
        else:
            answer = tree[question][1]


    # recursive
    if isinstance(answer, dict):
        sisa_tree = answer
#         print(predict_X(X, sisa_tree))
        return predict_X(X, sisa_tree)
    else:
        return answer


def decisionTree_predict(X_test, tree):
    DT_prediksi = X_test.apply(predict_X, args=(tree,), axis=1)

    return DT_prediksi
```

RANDOM FOREST

```
# random data
def split_random_data(X, n_data):
          index_data  =  np.random.randint(low=0,  high=len(X)-1,
size=n_data)
    split_random_df = X.iloc[index_data]
    return split_random_df

def randomforest_fit(X, n_trees, n_data, max_depth):
    rf = []
    for i in range(n_trees):
        split_random_df = split_random_data(X, n_data)
#         print(split_random_df)
                    tree   =   decisionTree_fit(split_random_df,
max_depth=max_depth)
        rf.append(tree)

    return rf
rf   =   randomforest_fit(X_train_title,   n_trees=3,   n_data=50,
max_depth=2)
```

PREDICT RANDOM FOREST

```
def randomforest_predict(X_test, rf):
    data_predict = {}
```

```
    for i in range(len(rf)):
        column_name = "{}".format(i)

        if isinstance(rf[i], dict):
#             print(rf[i])
            column_name = "i{}".format(i)
            predict = decisionTree_predict(X_test, tree=rf[i])
            data_predict[column_name] = predict
            data_prediksi = pd.DataFrame(data_predict)
#             print(data_prediksi)
        else:
            continue

    random_forest_predictions = data_prediksi.mode(axis=1)[0]
#     print(random_forest_predictions)
    return random_forest_predictions
prediksi_rf = randomforest_predict(X_test_title, rf)
```

MATRIX PREDICT RANDOM FOREST

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics

print(confusion_matrix(y_test_title, prediksi_rf))
print(confusion_matrix(y_test_content, prediksi_rf))
```

VISUALIZATION CONFUSION MATRIX RANDOM FOREST (TITLE)

```
import seaborn as sn
plt.figure(figsize=(12,10))
sn.heatmap(confusion_matrix(y_test_title,
prediksi_rf),xticklabels=['Hoax','Fakta'],
yticklabels=['Hoax','Fakta'],annot=True,fmt='d')
plt.title('Random Forest')
plt.xlabel('Y ASLI')
plt.ylabel('Y PREDIKSI')
plt.show()
```

VISUALIZATION CONFUSION MATRIX RANDOM FOREST (CONTENT)

```
import seaborn as sn
plt.figure(figsize=(12,10))
sn.heatmap(confusion_matrix(y_test_title,
prediksi_rf),xticklabels=['Hoax','Fakta'],
yticklabels=['Hoax','Fakta'],annot=True,fmt='d')
plt.title('Random Forest')
plt.xlabel('Y ASLI')
plt.ylabel('Y PREDIKSI')
plt.show()
```

ACCURACY, PRECISION, RECALL, F1-SCORE (RANDOM FOREST (TITLE))

```
#matrix
```

```
tp = confusion_matrix(y_test_title, prediksi_rf).item(0) #TP
fp = confusion_matrix(y_test_title, prediksi_rf).item(1) #FP
fn = confusion_matrix(y_test_title, prediksi_rf).item(2) #FN
tn = confusion_matrix(y_test_title, prediksi_rf).item(3) #TN

# (TP + TN ) / (TP+FP+FN+TN)
accuracy_rf = (tp+tn)/float((tp+fp+fn+tn))

# (TP) / (TP+FP)
precision_rf = tp/float((tp+fp))

# (TP) / (TP + FN)
recall_rf = tp/float((tp+fn))

#  2 * (Recall*Precission) / (Recall + Precission)
f1score_rf=                                                    2
*(recall_rf*precision_rf)/float((recall_rf+precision_rf))

print('Accuracy :',accuracy_rf*100)
print('Precision :',precision_rf*100)
print('Recall :',recall_rf*100)
print('F1-Score :',f1score_rf*100)
```

ACCURACY, PRECISION, RECALL, F1-SCORE (RANDOM FOREST (CONTENT))

```
#matrix
tp = confusion_matrix(y_test_content, prediksi_rf).item(0) #TP
fp = confusion_matrix(y_test_content, prediksi_rf).item(1) #FP
fn = confusion_matrix(y_test_content, prediksi_rf).item(2) #FN
tn = confusion_matrix(y_test_content, prediksi_rf).item(3) #TN

# (TP + TN ) / (TP+FP+FN+TN)
accuracy_rf = (tp+tn)/float((tp+fp+fn+tn))

# (TP) / (TP+FP)
precision_rf = tp/float((tp+fp))

# (TP) / (TP + FN)
recall_rf = tp/float((tp+fn))

#  2 * (Recall*Precission) / (Recall + Precission)
f1score_rf=                                                    2
*(recall_rf*precision_rf)/float((recall_rf+precision_rf))

print('Accuracy :',accuracy_rf*100)
print('Precision :',precision_rf*100)
print('Recall :',recall_rf*100)
print('F1-Score :',f1score_rf*100)
```

COMPARING RANDOM FOREST AND TESTING DATA (TITLE)

```
sum(y_test_title == prediksi_rf)
```

COMPARING RANDOM FOREST AND TESTING DATA (CONTENT)

```
sum(y_test_content== prediksi_rf)
```

COMPARING 2 ALGORITHM BASED ON ACCURACY, PRECISION, RECALL, F1-SCORE BY SHOWING VISUALIZATION OF HISTOGRAM CHART

```
df_title = pd.DataFrame([['Random Forest','Accuracy',accuracy_rf],
['SVM','Accuracy',accuracy_svm],
                         ['Random Forest','Precision',precision_rf],
['SVM','Precision',precision_svm],
                                ['Random Forest','Recall',recall_rf],
['SVM','Recall',recall_svm],
                           ['Random Forest','F1 Score',f1score_rf],
['SVM','F1
Score',f1score_svm]],columns=['Algoritma','Analisis','val'])

df_title.pivot("Analisis", "Algoritma", "val").plot(kind='bar')

plt.title('Performa TITLE Algoritma')
plt.show()
```

CHART PIE (TITLE)

```
# the amount of similarity data based on accuracy

svm_mirip= accuracy_svm * len(y_test_title)
rf_mirip = accuracy_rf * len(y_test_title)

labels = 'Random Forest', 'SVM'
sizes = [rf_mirip, svm_mirip]
explode = (0, 0)  # only the 2nd slice

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn
as a circle.

plt.title('Level of prediction similarity to original data
(Title)')
plt.show()
```

CHART PIE (CONTENT)

```
# the amount of similarity data based on accuracy

svm_mirip= accuracy_svm * len(y_test_content)
rf_mirip = accuracy_rf * len(y_test_content)

labels = 'Random Forest', 'SVM'
sizes = [rf_mirip, svm_mirip]
explode = (0, 0)  # only the 2nd slice
```

```
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%
%',shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn
as a circle.

plt.title('Level of prediction similarity to original data
(Content)')
plt.show()
```

PREDICTION RESULTS USING RANDOM FOREST ALGORITHM AND SUPPORT VECTOR MACHINE ALGORITHM

```
prediksi_label_svm = {-1:'HOAX',1:'REAL'}
prediksi_svm_rf = {'predict_svm':[prediksi_label_svm[x] for x in
prediksi_svm],'predict_rf':prediksi_rf}
pd.DataFrame(prediksi_svm_rf)
```

PLAGIARISM CHECK.ORG

8.1% PLAGIARISM APPROXIMATELY

Report #11051300

Introduction Background Before internet era people communicated directly by involving more senses (Meilinda, 2018). Besides that humans can only rely on television, radio and newspapers as information media. But with the changing times of the Information Technology that is currently developing. Media - media are now a lot of convenience in terms of communication, not limited by time and distance to communicate with each other. The media is also a forum for disseminating information that is very influential on society (Juditha, 2018). So that Online Media greatly affects the mindset of the public in consuming information received. In terms of disseminating information it is very easy to get through social media including facebook, twitter, instagram, whatsapp, and others. So that with the internet, it is very easy for humans to spread information as well as get information received. But with the ease of disseminating this information many individuals or groups are not responsible for spreading fake or often referred to as hoaxes. A hoax is information or news that contains things that have not been proven true or are uncertain. In Indonesia, Kementerian Komunikasi dan Informasi (Kominfo) in April 2019 identified 486 hoaxes, the highest number of hoaxes since August 2018 (Kementerian Komunikasi dan Informasi, 'Temuan Kominfo: Hoax Paling Banyak Beredar