LAMPIRAN A. SOURCE CODE

1. SOURCE CODE TEXT PROCESSING

```
# Load necessary library and module
import warnings
warnings.filterwarnings('ignore')

import sys
import pandas as pd
import seaborn as sns
import numpy as np
import re
import string
import unicodedata
import nltk

nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from textblob import TextBlob, Word
# Apply text pre-processing to DataFrame
from tqdm._tqdm_notebook import tqdm_notebook
tqdm_notebook.pandas()

from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
snowball = nltk.stem.SnowballStemmer('english')
porter = PorterStemmer()
nltk.download('words')

# Read and show sample data
mlbb = pd.read_csv('hasil/Billy-GameDatasetBaruSemhas03082022-7200data-
allgame.csv', encoding='ISO-8859-1')
mlbb = mlbb[mlbb.score != 3]
mlbb['sortOrder'].value_counts()
mlbb['score'].value_counts()
sns.countplot(mlbb.score)
mlbb['appId'].value_counts()

def text_filter(text):
  text = text.lower # Lowercase all sentences
  text = re.sub('[-+]?[0-9]+', ' ', text)                # Remove numbers
  text = re.sub(r'https?://\S+|www\.\S+', ' ', text)     # Remove URLs
  text = re.sub('\S*@\S*\s?', ' ', text)                 # Remove email
  text = re.sub(r'[^\w\s]', ' ', text)                   # Remove punctuation
  text = re.sub(r'[!$%^&*@#()_+|~=`{}\[\]%\-:";\'<>?,.\/]', ' ', text)  # Tahap-
5: simbol
  text = re.sub(r'[0-9]+',' ', text)                      # Tahap-6: angka
  text = re.sub(r'([a-zA-Z])\1\1','\\1', text)           # Tahap-7: koreksi
duplikasi tiga karakter beruntun atau lebih (contoh. yukkk)
  text = re.sub(' +',' ', text)                          #remove multiple
whitespace
  text = re.sub(r'^[ ]|[ ]$',' ', text)                   # Tahap-9: spasi di
awal dan akhir kalimat

  # text = re.sub('\b[a-zA-Z0-9]{3}\b','',text)
  text = unicodedata.normalize('NFKD', text).encode('ascii',
'ignore').decode('utf-8', 'ignore') # Remove non-ascii character
  word_tokens = word_tokenize(text) # Word tokenize
  return text
```

```
# def lemmatization()
def stopword_removal(text):
    word_tokens = word_tokenize(text) # Word tokenize
    # Define Indonesian stopwords removal
    stop_words = stopwords.words('english')  # NLTK Indonesian stopwords
    clean_words = [word for word in word_tokens if word not in stop_words] #
stopwords removal
    clean_token = ' '.join(clean_words)
    return clean_token

def filter_english(text):
    word_tokens = word_tokenize(text) # Word tokenize
    words = set(nltk.corpus.words.words())
    cleaner_words = [w for w in word_tokens if w.lower() in words or not
w.isalpha()] #remove non english
    cleaner = ' '.join(cleaner_words)
    return cleaner

def stemming_porter(text):
    #stemming with porter update 07072022 stemmer terakhir
    porter_token=word_tokenize(text)
    porter_words = [porter.stem(w) for w in porter_token]
    porter_clean= ' '.join(porter_words)
    return porter_clean

def stemming_snowball(text):
    #stemming with snowball update 07072022 stemmer terakhir
    stem_token=word_tokenize(text)
    stem_words = [snowball.stem(w) for w in stem_token]
    stem_clean= ' '.join(stem_words)
    return stem_clean

%%time
mlbb['content_filter'] = mlbb['content'].progress_apply(lambda x:
text_filter(x))
%%time
mlbb['content_stopword'] = mlbb['content_filter'].progress_apply(lambda x:
stopword_removal(x))

mlbb['content_stopword'][1]
'rate higher problem everytime hold heros ability always disappears press
problem pressing skill track predict enemys movement match projectile hero
ability think cause aspect ratio phone sort hope issue resolved give star edit
years later still fixed nice job developers star best rate'
%%time

mlbb['content_english'] = mlbb['content_stopword'].progress_apply(lambda x:
filter_english(x))
%%time
# import snowballstemmer


mlbb['content_snowball'] = mlbb['content_english'].progress_apply(lambda x:
stemming_snowball(x))
mlbb['content_porter'] = mlbb['content_english'].progress_apply(lambda x:
stemming_porter(x))
mlbb.to_csv(
"hasil/1_datasetbilly_clean_content_porterVSsnowball_7200data_03082022.csv",
index=False, encoding='utf-8-sig')
```

2. SOURCE CODE CLEAN CONTENT SENTIMENT LABELING

```
# Load necessary library and module
import warnings
warnings.filterwarnings('ignore')

import sys
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

#NOW
from datetime import datetime
now = datetime.now() # current date and time
d = now.strftime("%d%m%Y_%H%M")

#Load Dataset
all =
pd.read_csv('hasil/1_datasetbilly_clean_content_porterVSsnowball_7200data_030820
22.csv', encoding='ISO-8859-1')
datasets='7200'
all['clean_content']=all['content_snowball']
all=all[['clean_content','score','appId']]
all.head()
all.shape
# score to polarity
def to_polarity(rating):
  rating = int(rating)
  if rating <= 2:
    return 'negative'
  elif rating == 3:
    return 'neutral'
  else:
    return 'positive'

# score to sentiment
def to_sentiment(rating):
  rating = int(rating)
  if rating <= 2:
    return 0
  elif rating == 3:
    return 1
  else:
    return 2
# score to sentiment
def to_appname(appId):

  if appId == 'com.mobile.legends':
    return 'mlbb'
  elif appId =='com.dts.freefiremax':
    return 'ffm'
  else:
    return 'hdi'
def show_values(axs, orient="v", space=.02):
    def _single(ax):
        if orient == "v":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() / 2
                _y = p.get_y() + p.get_height() + (p.get_height()*0.02)
```

```
                value = '{:.1f}'.format(p.get_height())
                ax.text(_x, _y, value, ha="center")
        elif orient == "h":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() + float(space)
                _y = p.get_y() + p.get_height() - (p.get_height()*0.5)
                value = '{:.1f}'.format(p.get_width())
                ax.text(_x, _y, value, ha="left")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _single(ax)
    else:
        _single(axs)

class_names = ['negative', 'positive']

all['sentiment'] = all.score.apply(to_sentiment)
all['polarity'] = all.score.apply(to_polarity)
all['appName'] = all.appId.apply(to_appname)

all.to_csv( "dataset/2_afterlabel_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')
all.shape
g = all.groupby(['appName','sentiment','score'])
#['clean_content'].value_counts()
all.to_csv(
"dataset/2_afterlabel_groupby_appname_sentiment_score_"+datasets+"data_"+d+".csv
", index=True,header=True, encoding='utf-8-sig')
g.apply(lambda x: x.sample())
# g.head()
# g.size().reset_index(name='jml')
gdf=pd.DataFrame(g.size().reset_index(name='jml'))
gdf
gdf.T

score=pd.DataFrame(all['score'].value_counts())
score
score.to_csv( "dataset/2_statistik_score_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

appname=pd.DataFrame(all['appName'].value_counts())
appname
appname.to_csv( "dataset/2_statistik_apps_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

polarity=pd.DataFrame({'jml':all['polarity'].value_counts()})
# polarity.columns=['pol','jml']
# polarity.restart_index()
polarity
polarity.to_csv( "dataset/2_statistik_polarity_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

sentiment=pd.DataFrame(all['sentiment'].value_counts())
sentiment
sentiment.to_csv( "dataset/2_statistik_sentiment_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

#CHECKING FOR MISSING VALUES
missingvalue=pd.DataFrame(all.isnull().sum(),columns=['before'])
```

```
all.dropna(inplace=True)
missingvalue['after']=pd.DataFrame(all.isnull().sum())
missingvalue
missingvalue.to_csv( "dataset/2_statistik_isnull_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')
all.describe()

#Class Count after remove missing value
score['afterdropna']=all['score'].value_counts()

score

score.to_csv( "dataset/2_statistik_score_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

scoreplot=sns.countplot(all.score)

for i in scoreplot.containers:
    scoreplot.bar_label(i,)

polarity['afterdropna']=all['polarity'].value_counts()

polarity

polarity.to_csv( "dataset/2_statistik_polarity_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')
polarplot=sns.countplot(all.polarity)
for i in polarplot.containers:
    polarplot.bar_label(i,)

sentiment['afterdropna']=all['sentiment'].value_counts()

sentiment

sentiplot=sns.countplot(all.sentiment)

for i in sentiplot.containers:
    sentiplot.bar_label(i,)

sentiment.to_csv( "dataset/2_statistik_sentiment_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

appname['afterdropna']=all['appName'].value_counts()
appname
all['appId'].value_counts()
appplot=sns.countplot(all.appId)

for i in appplot.containers:
    appplot.bar_label(i,)

appname.to_csv( "dataset/2_statistik_appname_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')
all['string_len'] = all['clean_content'].astype(str).apply(len)
all['word_count'] = all['clean_content'].apply(lambda x: len(str(x).split()))
all.to_csv(
"dataset/2_datasetbilly_clean_content_aftermissingvaluestrlen_snowvsporter_"+dat
asets+"data_"+d+".csv", index=False, encoding='utf-8-sig')
```

3. SOURCE CODE SENTIMENT ANALYSIS

```
import sys
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
TfidfVectorizer,CountVectorizer,TfidfTransformer
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.svm import SVC

nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('omw-1.4')
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS


from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# Model Evaluation metrics
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, recall_score, precision_score, f1_score
from datetime import datetime
now = datetime.now() # current date and time
d = now.strftime("%d%m%Y_%H%M")

#Load Dataset
allgame=pd.read_csv(
"dataset/2_datasetbilly_clean_content_aftermissingvaluestrlen_snowvsporter_7200d
ata_04082022_1054.csv",encoding='utf-8')

allgame = allgame[['clean_content','score','polarity','appName','sentiment']])
allgame.info()
allgame.isnull().sum()
allgame.dropna(inplace=True)
allgame.info()
# all= allgame.sample(n=6000,replace=False,random_state=42)
allgame['string_len'] = allgame['clean_content'].astype(str).apply(len)
allgame['word_count'] = allgame['clean_content'].apply(lambda x:
len(str(x).split()))
allgame

#Remove Score "3"
#remove neutral and sentiment 1 from dataset
allgame=allgame[allgame.sentiment != 1]

#All Game Data Describe
allgame.describe()

#All Game Data Statistics
#AppName Stats
allgame['appName'].value_counts()
sns.countplot(allgame.appName)
```

```
#Score Stats
sns.countplot(allgame.score)
allgame['score'].value_counts()

#Sentiment Stats
sns.countplot(allgame.sentiment)
allgame['sentiment'].value_counts()
allgame.shape

#NOW Date Time
# d = datetime.now().strftime("%d%m%Y_%H%M")

#Sampling For Test And Training
nsamples=4000
all= allgame.sample(n=nsamples,replace=True,random_state=1)
all

#Grouping Dataset
g = all.groupby(['sentiment','score','appName'])

g.describe()

g.size()

#Group Sampling to get balanced dataset
# gg=g.sample(g.size().min()) #373
ngsamples=200
gg=g.sample(ngsamples,replace=True,random_state=1)
groupsample = gg[['clean_content','score','polarity','appName','sentiment']]
groupsample.groupby(groupsample.columns.tolist(),as_index=False).size()
groupsample.describe()


#Make Directory if does not exists
import os
datasets="7200_allgame_"
direktori="7200_groupsample"+str(ngsamples)+"_from"+str(nsamples)+"samples_repla
cetrue_randomstate42cv10"
# direktori="5400data_"+str(nsamples)+"sample_no_option"
if not os.path.exists("hasil/"+direktori):
    os.makedirs("hasil/"+direktori)

#Save Group and Grouping Samples
groupsample.to_csv(
"hasil/"+direktori+"/4_1_grouping_sampling"+str(ngsamples)+"_from"+str(nsamples)
+"_"+d+".csv", index=False,header=True, encoding='utf-8-sig')

#Group Samples Statistics
score=pd.DataFrame(groupsample['score'].value_counts())

score.to_csv(
"hasil/"+direktori+"/4_groupsample_score_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

sentiment=pd.DataFrame(groupsample['sentiment'].value_counts())
sentiment.to_csv(
"hasil/"+direktori+"/4_groupsample_sentiment_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')
sentiment
```

```
appname=pd.DataFrame(groupsample['appName'].value_counts())
appname.to_csv(
"hasil/"+direktori+"/4_groupsample_appname_"+datasets+"data_"+d+".csv",
index=True,header=True, encoding='utf-8-sig')

appname
d = datetime.now().strftime("%d%m%Y_%H%M")

#DATA TRAINING & TESTING SEPARATION
dataset=groupsample #all
folder=direktori
dtlatih=75
dtuji=100-dtlatih
X = dataset['clean_content']     # Define feature matriX
y = dataset['sentiment']         # Define target feature matriX
jmluji=float(dtuji/100)
print(jmluji)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=jmluji,
random_state=42, stratify=y)

datalatih=('Dataset {} {}:{} \n [INFO] Sebaran kelas pada training  set: \n
negative: \t {} \n neutral: \t{} \n positive:\t{} \n'.format(dataset,int((1-
jmluji)*100),int(jmluji*100),sum(y_train==0), sum(y_train==1), sum(y_train==2)))
datauji=('[INFO] Sebaran kelas pada testing set:\n negative: \t{}\n neutral:
\t{}\n positive:\t{} \n'.format(sum(y_test==0), sum(y_test==1), sum(y_test==2)))

dimensi_data=('[INFO] Shape Data: \n X_train: \t {} \n X_test: \t {} \n y_train:
\t {} \n y_test: \t {} \n'.format(X_train.shape, X_test.shape, y_train.shape,
y_test.shape))

with open('hasil/'+folder+'/4_1_LatihdanUji_gridsearch_'+str(dtlatih)+'-
'+str(dtuji)+'_'+d+'.txt', 'a', encoding='utf-8') as f:
    f.writelines(''.join(datalatih))
    f.writelines(''.join(datauji))
    f.writelines(''.join(dimensi_data))


X_train

#SAVE DATASET
X_train.to_csv( "hasil/"+folder+"/4_1_train_data_"+str(dtlatih)+"_"+d+".csv",
index=True, header=True,encoding='utf-8-sig')
X_test.to_csv( "hasil/"+folder+"/4_1_test_data_"+str(dtuji)+"_"+d+".csv",
index=True, header=True,encoding='utf-8-sig')

#POSITIVE AND NEGATIVE TRAIN DATA
# xtrain_pos=X_train[allgame.sentiment != 1]
4   FEATURE EXTRACTION TFIDF
ngram = (1,3) #trigram
# min_df = 5 means "ignore terms that appear in less than 5 documents".
mindf = 1

featmax=3000

fs="unitrigram_max"+str(featmax)

tfidf =
TfidfVectorizer(analyzer='word',ngram_range=ngram,min_df=mindf,max_features=feat
max)
```

```
vectors = tfidf.fit_transform(dataset.clean_content).toarray()
fs
'unitrigram_max3000'
all.to_csv(
"hasil/"+direktori+"/4_1_sampling_all_"+str(nsamples)+"_randsvc42_randsampling1_
"+str(dtlatih)+'-'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, encoding='utf-8-
sig')

xtrain_vectors=tfidf.fit_transform(X_train).toarray()
xtest_vectors=tfidf.fit_transform(X_test).toarray()

X_train_to_df = tfidf.fit_transform(X_train).toarray()

train_words_df =
pd.DataFrame(X_train_to_df,columns=tfidf.get_feature_names_out())

train_words_df.to_csv("hasil/"+folder+"/4_1_tfidf_training_data_"+str(dtlatih)+'
-'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, encoding='utf-8-sig')

train_words_df.head()

train_words_df.describe()

X_test_to_df = tfidf.transform(X_test).toarray()
test_words_df = pd.DataFrame(X_test_to_df,
columns=tfidf.get_feature_names_out())
test_words_df.to_csv( "hasil/"+folder+"/4_1_tfidf_testing_data_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, encoding='utf-8-sig')

#Word Visualization
from nltk.corpus import stopwords
from collections import Counter
from collections import defaultdict
# Code Snippet for Top Non-Stopwords Barchart
def plot_top_non_stopwords_barchart(text):
    stop = set(stopwords.words('english'))

    new = text.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]

    counter = Counter(corpus)
    most = counter.most_common()
    x, y = [], []
    for word, count in most[:20]:
        if (word not in stop):
            x.append(word)
            y.append(count)

    sns.barplot(x=y, y=x)
def plot_top_ngrams_barchart(text, n=2):
    stop = set(stopwords.words('english'))

    new = text.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]

    def _get_top_ngram(corpus, n=None):
        vec = CountVectorizer(ngram_range=(n, n)).fit(corpus)
        bag_of_words = vec.transform(corpus)
        sum_words = bag_of_words.sum(axis=0)
```

```
        words_freq = [(word, sum_words[0, idx])
                      for word, idx in vec.vocabulary_.items()]
        words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
        return words_freq[:10]

    top_n_bigrams = _get_top_ngram(text, n)[:10]
    x, y = map(list, zip(*top_n_bigrams))
    sns.barplot(x=y, y=x)
plot_top_non_stopwords_barchart(X_train)

plot_top_ngrams_barchart(X_train, 2)

plot_top_ngrams_barchart(X_train, 1)

def plot_wordcloud(text):
    corpus=[]

    stop=set(stopwords.words('english'))

    def _preprocess_text(text):
        corpus=[]
        # stem=SnowballStemmer()
        lem=WordNetLemmatizer()
        for news in text:
            words=[w for w in word_tokenize(news) if (w not in stop)]

            words=[lem.lemmatize(w) for w in words if len(w)>2]

            corpus.append(words)
        return corpus

    corpus=_preprocess_text(text)

    wordcloud = WordCloud(
        background_color='white',
        stopwords=set(STOPWORDS),
        max_words=100,
        max_font_size=30,
        scale=3,
        random_state=1)

    wordcloud=wordcloud.generate(str(corpus))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')

    plt.imshow(wordcloud)
    plt.show()

plot_wordcloud(X_train)

plot_wordcloud(X_test)

#PIPELINE & GRIDSEARCH TEST
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from pprint import pprint
from time import time
from sklearn.decomposition import PCA
```

```
pipe = Pipeline([
    ('bag_of_words', tfidf),
    ('estimator', SVC(random_state=42))])

#create GridSearchCV object with set of possible parameters
Cs = [0.001, 0.01, 0.1, 1, 10, 100]
gammas = [0.001, 0.01, 0.1, 1, 10,100]
kernel = ['linear','rbf','poly','sigmoid']
param_grid={
    'estimator__C': Cs,
    'estimator__gamma': gammas,
    'estimator__kernel': kernel,
    # 'bag_of_words__ngram_r?ange': ((1,1),(1,2),(1,3)),
    # 'bag_of_words__max_features': (1000,2000),

}
print(pipe)
Pipeline(steps=[('bag_of_words',
                 TfidfVectorizer(max_features=3000, ngram_range=(1, 3))),
                ('estimator', SVC(random_state=42))])

%%time
from sklearn.metrics import make_scorer
scorer = {"accuracy":
"accuracy","recall":"recall_macro","precision":"precision_macro","f1":"f1_macro"
}

grid = GridSearchCV(pipe, param_grid=param_grid,cv=10,refit="accuracy",verbose =
3,scoring=scorer,n_jobs=-1) #,return_train_score=True)


%%time
grid.fit(X_train, y_train)
y_pred = grid.predict(X_test)

# print(classification_report(, y_test )) #print classification report
print("Performing grid search...")
print("pipeline:", [name for name, _ in pipe.steps])
print("param_grid:")
pprint(param_grid)
t0 = time()

print("done in %0.3fs" % (time() - t0))
timelapse=("Time : done in %0.3fs" % (time() - t0))
print()

print("Best score: %0.3f" % grid.best_score_)
bestscore=("Best score: %0.3f" % grid.best_score_)
print("Best parameters set:")
best_parameters = grid.best_estimator_.get_params()
print(best_parameters)
bestparam=[]
for param_name in sorted(param_grid.keys()):
    print("\t%s: %r" % (str(param_name), str(best_parameters[param_name])))

bestparam=bestparam+[{'Parameter':str(param_name),'value':str(best_parameters[pa
ram_name]),'training':str(dtlatih)}]

#GS Results
# grid.cv_results_
```

```
resultgs=pd.concat([
        pd.DataFrame(grid.cv_results_["params"]),
        pd.DataFrame(grid.cv_results_["mean_test_accuracy"],
columns=["accuracy"]),
        pd.DataFrame(grid.cv_results_["mean_test_recall"], columns=["recall"]),
        pd.DataFrame(grid.cv_results_["mean_test_precision"],
columns=["precision"]),
        pd.DataFrame(grid.cv_results_["mean_test_f1"], columns=["f1"]),
        ],
    axis=1)
resultgs.to_csv(
"hasil/"+folder+"/4_1_resultgs_all_classifier_accuracy_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False,header=True, encoding='utf-8-sig')
resultgs.head()

#Define Best Parameter for Feature Extraction, Training and Testing Session
# parammax=bestparam[0]['value']
# paramngram=bestparam[0]['value']
paramc=bestparam[0]['value']
paramg=bestparam[1]['value']
# paramngram,
paramc,paramg
('1', '1')

#Best Parameters Results on GS
bestpara=pd.DataFrame(bestparam)
bestpara.head()

bestpara.to_csv( "hasil/"+folder+"/4_1_bestparameter_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, encoding='utf-8-sig')
with open('hasil/'+folder+'/4_1_gridsearch_'+str(dtlatih)+'-
'+str(dtuji)+'_'+fs+'_'+d+'.txt', 'a', encoding='utf-8') as f:
    f.writelines('\n')
    f.writelines((timelapse)+'\n')
    f.writelines((bestscore)+'\n')
    f.writelines('Best Parameter set:'+'\n'+(str(bestparam)))
grid_predictions = grid.predict(X_test)

# print classification report
# print(classification_report(y_test, grid_predictions,output_dict=True))
report=pd.DataFrame(classification_report(y_test,
grid_predictions,output_dict=True)).T
def classification_report_csv(report):
    report_data = []
    lines = report.split('\n')
    for line in lines[2:-3]:
        row = {}
        row_data = line.split('      ')
        row['class'] = row_data[0]
        row['precision'] = float(row_data[1])
        row['recall'] = float(row_data[2])
        row['f1_score'] = float(row_data[3])
        row['support'] = float(row_data[4])
        report_data.append(row)
    dataframe = pd.DataFrame.from_dict(report_data)
    dataframe.to_csv('classification_report.csv', index = False)
print(report)
report.to_csv( "hasil/"+folder+"/4_1_testreport_gridsearch_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=True,header=True, encoding='utf-8-sig')
```

```
print("Detailed classification report:")
print()
print("The model is trained on the full development set.")
print("The scores are computed on the full evaluation set.")
print()
y_true, y_pred = y_test, grid.predict(X_test)
class_report=classification_report(y_true, y_pred)
matrix=confusion_matrix(y_true, y_pred)
print(class_report)
print()
print(matrix)
print()

label_names = pd.Series(['negative', 'positive'])
eva=pd.DataFrame(matrix,
     columns='Predicted ' + label_names,
     index='Is ' + label_names)
eva.to_csv( "hasil/"+folder+"/4_1_test_evaluation_gridsearch_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, encoding='utf-8-sig')

eva

#CLASSIFICATION
C=float(paramc)
gam=float(paramg)
models = [
          ('SVC linear kernel', SVC(kernel='linear',C=C,max_iter=10000)),
          ('SVC RBF kernel', SVC(kernel='rbf',gamma=gam,C=C)),
          ('SVC Polynomial (degree 3)', SVC(kernel='poly',degree=3,C=C)),
          ('SVC Sigmoid ', SVC(kernel='sigmoid',C=C,gamma=gam)),
        ]
# from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


#Training Session
pelatihan=[]
# %%time
for name, clf in models:
  clf.fit(X_train_to_df, y_train)
  # clf.fit(x_train_minmax, y_train) #with minmax normalization
  train_acc = accuracy_score(y_train, clf.predict(X_train_to_df))
  # train_acc = accuracy_score(y_train, clf.predict(x_test_minmax))
  latih=('Dataset {}, data split : {}:{} \n'.format(folder,(1-
jmluji)*100,jmluji*100))

pelatihan=pelatihan+[{'classifier':name,'akurasi':train_acc,'C':C,'Gamma':gam,'t
raining':str(dtlatih)}]
  printed_dataset=('Dataset: {} \t'.format(folder))
  printed=('[INFO] Training Menggunakan {}, akurasi pada training set: {}
\n'.format(name, train_acc))
  with open('hasil/'+folder+'/4_1_training_svm_'+str(dtlatih)+'-
'+str(dtuji)+'_'+fs+'_'+d+'.txt', 'a', encoding='utf-8') as f:
    f.writelines('\n'+(latih))
    f.writelines(''.join(printed_dataset))
```

```
    f.writelines(''.join(printed))

  print(printed)



df_pelatihan=pd.DataFrame(pelatihan)
print(df_pelatihan)
df_pelatihan.to_csv( "hasil/"+folder+"/4_1_training_svm_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, header=True,encoding='utf-8-sig')
#Testing Session

label_names = pd.Series(['negative','positive'])
pengujian=[]

for name, clf in models:
  pred=clf.predict(X_test_to_df)
  test_acc = accuracy_score(y_test, pred)
  mae = mean_absolute_error(y_test, pred)
  rmse=mean_squared_error(y_test, pred, squared = False)
  latih=('Dataset {}, data split : {}:{} \n'.format(folder,(1-
jmluji)*100,jmluji*100))

pengujian=pengujian+[{'classifier':name,'akurasi':test_acc,'C':C,'Gamma':gam,'te
sting':str(dtuji),'mae':mae,'rmse':rmse,'ngram':ngram,'max_feature':featmax}]
  printed_dataset=('Dataset: {} \t'.format(folder))
  printed=('[INFO] Testing Menggunakan {}, akurasi pada testing set: {}
\n'.format(name, test_acc))
  print_mae=('MAE {} : {} \n'.format(name,mae))
  print_rmse=('RMSE {} : {} \n'.format(name,rmse))
  with open('hasil/'+folder+'/4_1_testing_svm_'+str(dtlatih)+'-
'+str(dtuji)+'_'+fs+'_'+d+'.txt', 'a', encoding='utf-8') as f:
    f.writelines('\n'+(latih))
    f.writelines(''.join(printed_dataset))
    f.writelines(''.join(printed))
    f.writelines(''.join(print_mae))

  test_matrix=(confusion_matrix(pred,y_test))
  df_test=pd.DataFrame(test_matrix,
     columns='Predicted ' + label_names,
     index='Is ' + label_names)

  cm = confusion_matrix(y_test, pred, labels=clf.classes_)
  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)


  y_score = clf.decision_function(X_test_to_df)
  fpr, tpr, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
  roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr) #.plot()
  # auc = round(roc_auc_score(y_test, y_pred), 4)

  prec, recall, _ = precision_recall_curve(y_test, y_score,
pos_label=clf.classes_[1])
  pr_display = PrecisionRecallDisplay(precision=prec, recall=recall) #.plot()

  # plt.title(name)
  fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24, 8))
  roc_display.plot(ax=ax2)
  ax2.set_title('ROC '+name)
  pr_display.plot(ax=ax3)
  ax3.set_title('Precision Recal '+name)
```

```
  disp.plot(ax=ax1)
  ax1.set_title('Confusion Matrix '+name)

  plt.savefig("hasil/"+folder+"/4_9_testing_svm_"+name+"_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".png", bbox_inches='tight')

df_pengujian=pd.DataFrame(pengujian)
print(df_pengujian)
df_pengujian.to_csv( "hasil/"+folder+"/4_1_testing_svm_"+str(dtlatih)+'-
'+str(dtuji)+"_"+fs+"_"+d+".csv", index=False, header=True, encoding='utf-8-
sig')
```