LAMPIRAN

Lampiran 1. Riwayat Hidup

RIWAYAT HIDUP

Perkenalkan nama saya I Gusti Ngurah Agung Pernata, saya saat ini tinggal di Desa Tinga-tinga, Kecamatan Gerokgak, Kabupaten Buleleng, Bali. Perjalanan pendidikan saya berawal dari SD N 2 Tinga-tinga, yang merupakan tempat saya menuntaskan pendidikan dasar. Setelah menyelesaikan Sekolah Dasar, pendidikan saya dilanjutkan ke SMP N 1 Gerokgak. Yang kemudian saya melanjutkan ke jenjang selanjutnya di SMA N 1 Seririt, dengan mengambil jurusan MIA (Matematika dan Ilmu Pengetahuan Alam). Pada tahun 2020, saya berhasil menyelesaikan pendidikan tingkat menengah atas dan melanjutkan perjalanan akademik saya ke tingkat perguruan tinggi, di mana saya diterima di Universitas Pendidikan Ganesha, Fakultas Teknik dan Kejuruan, Jurusan Teknik Informatika, Program Studi Sistem Informasi melalui jalur SBMPTN. Setiap tahapan pendidikan yang saya jalani mencerminkan tekad dan keseriusan saya dalam mengejar ilmu. Saya percaya bahwa setiap fase pendidikan merupakan langkah penting menuju pencapaian tujuan dan kesuksesan di masa yang akan datang..

Lampiran 2 : Kode Scrapping Ulasan Steam

```
from datetime import datetime as dt
import requests
import re
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup

current_year = dt.today().year
headers = {'User-Agent':'Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/13.1.2 Safari/605.1.15'}

# Fungsi untuk mengambil jumlah review
def get_review_count(id):
    url = 'https://store.steampowered.com/app/' +
str(id)
    html = requests.get(url, headers=headers).text
    soup = BeautifulSoup(html, 'html.parser')

    # Check if the element with 'for':
'review_language_mine' exists
    label_element = soup.find('label', {'for':
'review_language_mine'})
```

```
    if label_element:
        # Check if the 'span' element exists within
the 'label' element
        span_element = label_element.span
        if span_element:
            count = span_element.text
            count = count.strip('()').replace(',',
'')
            return int(count)

    return 0

# Fungsi untuk mencari ID dan nama game berdasarkan
kata kunci (search term) di Steam
def search_game_id(search_term, all_results=False):
    page = 1
    game = []
    id = []
    if not all_results:
        url =
f'https://store.steampowered.com/search/?category1=99
8&page={page}&term={search_term}'
        html = requests.get(url,
headers=headers).text
        soup = BeautifulSoup(html, 'html.parser')
        search_results =
soup.find(class_='search_result_row')
        # Menambahkan pengecekan apakah hasil
pencarian ditemukan atau tidak
        if search_results:
            game = search_results.find('span',
class_='title').text
            id = search_results['data-ds-appid']
            return pd.DataFrame({
                'game':[game],
                'id':[id]
            })
        else:
            print(f"No results found for
'{search_term}'.")
            return pd.DataFrame()

    else:
        while True:
            url =
f'https://store.steampowered.com/search/?category1=99
8&page={page}&term={search_term}'
```

```
            html = requests.get(url,
headers=headers).text
            soup = BeautifulSoup(html, 'html.parser')
            search_results =
soup.find_all(class_='search_result_row')
            if not search_results:
                break

            title = [result.find('span',
class_='title').text for result in search_results]
            appid = [result['data-ds-appid'] for
result in search_results]
            game.extend(title)
            id.extend(appid)
            page += 1

        return pd.DataFrame({
            'game':game,
            'id':id
        })

def get_game_ids(n, filter='topsellers'):
    page = 1
    game = []
    id = []
    while len(game) < n:
        url =
f'https://store.steampowered.com/search/?category1=99
8&page={page}&filter={filter}'
        html = requests.get(url,
headers=headers).text
        soup = BeautifulSoup(html, 'html.parser')
        search_results =
soup.find_all(class_='search_result_row')
        if not search_results:
            break

        title = [result.find('span',
class_='title').text for result in search_results]
        appid = [result['data-ds-appid'] for result
in search_results]
        game.extend(title)
        id.extend(appid)
        page += 1
    num = min(n, len(game))
    return pd.DataFrame({
        'game':game[:num],
        'id':id[:num]
    })
```

```
def clean_date(date):
    date = date.split(' ',1)[1]
    try:
        dt.strptime(date,'%B %d, %Y')
        pass
    except ValueError:
        date += ', ' +str(current_year)
    return date

def get_game_review(id, language='default'):
    user_name_list = []
    hour_list = []
    user_link_list = []
    post_date_list = []
    helpful_list = []
    comment_list = []
    title_list = []
    early_access_list = []

    cursor = ''
    i=0
    while True:

url=f'https://steamcommunity.com/app/{id}/homecontent
/'
        params = {
            'userreviewsoffset': i  * 10,
            'p': i + 1,
            'workshopitemspage': i + 1,
            'readytouseitemspage': i + 1,
            'mtxitemspage': i + 1,
            'itemspage': i + 1,
            'screenshotspage': i + 1,
            'videospage': i + 1,
            'artpage': i + 1,
            'allguidepage': i + 1,
            'webguidepage': i + 1,
            'integeratedguidepage': i + 1,
            'discussionspage': i + 1,
            'numperpage': 10,
            'browsefilter': 'toprated',
            'browsefilter': 'toprated',
            'appid': id,
            'appHubSubSection': 10,
            'l': 'english',
            'filterLanguage': language,
            'searchText': '',
            'forceanon':1,
```

```
            'maxInappropriateScore':50,
        }
        if i > 0:
            params['userreviewscursor'] = cursor
        html = requests.get(url, headers=headers,
params=params).text
        soup = BeautifulSoup(html, 'html.parser')
        reviews=soup.find_all('div', {'class':
'apphub_Card'})

        if not reviews:
            break

        users = [review.find('div', {'class':
'apphub_CardContentAuthorName'}) for review in
reviews]
        user_name = [user.find('a').text for user in
users]
        user_link = [user.find('a').attrs['href'] for
user in users]
        title = [review.find('div', {'class':
'title'}).text for review in reviews]
        hour = [float(review.find('div', {'class':
'hours'}).text.split(' ')[0].replace(',', '')) if
review.find('div', {'class': 'hours'})
            else np.nan for review in reviews]
        helpful = [review.find('div',{'class':
'found_helpful'}).get_text(strip=True).split(' ')[0]
for review in reviews]
        helpful = [0 if num == 'No' else
int(num.replace(',', '')) for num in helpful]  #
Perbaikan
        comment_section = [review.find('div',
{'class': 'apphub_CardTextContent'}) for review in
reviews]
        raw_post_date =
[x.find('div',{'class':'date_posted'}).get_text(strip
=True) for x in comment_section]
        post_date = [clean_date(date) for date in
raw_post_date]
        comment =
[''.join(review.find_all(string=True,
recursive=False)).strip() for review in
comment_section]
        early_access = [x.find('div',{'class':
'early_access_review'}).text if
x.find('div',{'class': 'early_access_review'})
                        else None for x in
comment_section]
```

```
        cursor =
soup.find_all('form')[0].find('input',{'name':
'userreviewscursor'})['value']

        user_name_list.extend(user_name)
        hour_list.extend(hour)
        user_link_list.extend(user_link)
        post_date_list.extend(post_date)
        helpful_list.extend(helpful)
        comment_list.extend(comment)
        title_list.extend(title)
        early_access_list.extend(early_access)
        i += 1

    review_df=pd.DataFrame({
        'user': user_name_list,
        'playtime': hour_list,
        'user_url': user_link_list,
        'post_date': post_date_list,
        'helpfulness': helpful_list,
        'review': comment_list,
        'recommend': title_list,
        'early_access_review': early_access_list
    })
    return review_df

# Input nama game
game_name = "LIRE"

# Mencari ID game berdasarkan nama
result_df = search_game_id(game_name)

# Memeriksa game id dan jumlah review
# Mengambil nilai pertama dari DataFrame sebagai
game_id
game_id = int(result_df.iloc[0]['id'])

# Menampilkan ID game
print(f"Game ID for '{game_name}': {game_id}")

# Mengambil review count
review_count = get_review_count(game_id)
print(f"Total Review '{game_name}': {review_count}")

# Mengambil review berdasarkan game_id
language = 'bahasa indonesia'
reviews = get_game_review(game_id, language)

# Membuat DataFrame dari review
```

```
df = pd.DataFrame(reviews)

# Mengganti karakter khusus dalam nama file
valid_filename = re.sub(r'[\\/*?:"<>|]', '_',
game_name)

# Menyimpan DataFrame ke file CSV dengan nama file
sesuai dengan nama game yang sudah diubah
csv_filename = f"{valid_filename.replace(' ', '_')}-
reviews.csv"
df.to_csv(csv_filename, index=False)

print(f"Reviews saved to '{csv_filename}'")
```

Lampiran 3 : Kode Data Selection

```
import pandas as pd
from langdetect import detect

# Baca file CSV
df = pd.read_csv('Dataset-Steam-Reviews.csv')

# Fungsi untuk mendeteksi bahasa
def detect_language(text):
    try:
        lang = detect(text)
        return lang
    except:
        return None

# Menambahkan kolom baru untuk menyimpan hasil
deteksi bahasa
df['language'] = df['review'].apply(detect_language)

# Filter hanya data dengan bahasa selain Inggris
df_filtered = df[df['language'] != 'en']

# Hapus kolom bahasa jika sudah tidak diperlukan
df_filtered = df_filtered.drop(columns=['language'])

# Simpan dataframe yang telah difilter ke file CSV
baru
df_filtered.to_csv('Dataset-Steam-Reviews-language-
filter.csv', index=False)
```

Lampiran 4 : Kode Data Labelling

```
import pandas as pd
```

```
# Baca file CSV
file_path = 'Result/preprocessing_results.csv'
df = pd.read_csv(file_path, sep=';')

# Hapus kolom yang tidak diperlukan
df = df[['review', 'recommend']]

# Tambahkan kolom label
df['label'] = df['recommend'].apply(lambda x:
'positif' if x == 'Recommended' else 'negatif')

# Tampilkan DataFrame hasil
print(df)

# Gantilah 'recommend' dengan nama kolom yang ingin
Anda ubah
nama_kolom_yang_akan_diubah = 'review'

# Gantilah 'text' dengan nama baru yang diinginkan
nama_kolom_baru = 'text'

# Mengubah nama kolom
df = df.rename(columns={nama_kolom_yang_akan_diubah:
nama_kolom_baru})
print(df)

# Simpan hasil ke file CSV baru (jika diperlukan)
output_file_path =
'Result/preprocessing_results_text.csv'
df.to_csv(output_file_path, sep=';', index=False)
```

Lampiran 5 : Kode Case Folding

```
# ------ Case Folding --------
REVIEW_DATA['review'] =
REVIEW_DATA['review'].str.lower()
REVIEW_DATA['review']
```

Lampiran 6 : Kode Cleansing

```
import re
import string

# ------ Cleansing ---------
# CLEANSING FUNCTION
# remove special character
```

```
def remove_review_special(text):
    text = text.replace('\\t', " ").replace('\\n', "
").replace('\\u', " ").replace('\\', "")
    text = text.encode('ascii',
'replace').decode('ascii')
    text = ' '.join(re.sub("([@#][A-Za-z0-
9]+)|(\w+:\/\/\S+)", " ", text).split())
    return text.replace("http://", "
").replace("https://", " ")

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

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

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

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

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

#CLEANSING

# Apply cleansing functions
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_review_special)
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_number)
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_punctuation)
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_whitespace_LT)
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_whitespace_multipl
e)
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(remove_single_char)
```

Lampiran 7 : Kode Tokenizing

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

#---------TOKENIZING-----------
def word_tokenize_wrapper(text):
    return word_tokenize(text)

REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(word_tokenize_wrapper)

REVIEW_DATA[['review']]
```

Lampiran 8 : Kode Stopword Removal

```
# Stopword Removal
from nltk.corpus import stopwords
nltk.download('stopwords')

list_stopwords = stopwords.words('indonesian')
list_stopwords.extend(['iya', 'yaa', 'kak', 'yang',
'yg', 'dengan', 'kan', 'nya', 'gimana', 'juga', 'ke',
'bikin', 'bilang', 'ingin', 'bisa','gimana', 'harus',
'nih', 'sih', 'kakak', 'terus', 'juga', 'demi',
'lah', 'luar', 'waktu', 'terus', 'pun', 'kenapa',
'si', 'tuh', 'untuk', 'n', 'tt', 'daripada', 'mana',
'tuh', 'siapa', 'tadi', 'pak', 'tanpa', 'atau', 'di',
'hehe', 'u', 'ni', 'loh', 'tu', 'bahkan', 'masi',
'masih', 'dulu', 'kali', 'nah', 'niii', 'gasi',
'amp', 'nih', 'seminggu', 'itupun', 'bawah', 'lu',
'gimana', 'bulan','yah', 'ka', 'tapi', 'pas', 'saat',
'satu', 'minggu', 'hari', 'orang', 'pada',
'ini','itu', 'hanya', 'akan',
'sini','sana','atau','dan', 'apa', 'loh', 'lo',
'kapan', 'untuk', 'akhir', 'baru', 'apalagi', 'kok',
'bagaimana', 'gimana', 'gapapa' ,'gk', 'ga','senin',
'selasa', 'rabu', 'kamis', 'jumat', 'sabtu',
'minggu', 'akan', 'depan', 'hari', 'tahun','ku',
'mu', 'dia', 'anda', 'kamu', 'mereka', 'kita',
'kami', 'gitu', 'gini', 'padahal', 'siang'])
list_stopwords = set(list_stopwords)

def stopwords_removal(words):
    return [word for word in words if word not in
list_stopwords]
```

```
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(stopwords_removal)

REVIEW_DATA[['review']]
```

Lampiran 9 : Kode Stemming

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

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

# Define a function for stemming
def sastrawi_stemming(tokens):
    return [stemmer.stem(term) for term in tokens]

# Apply stemming to the 'review' column
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(sastrawi_stemming)
REVIEW_DATA[['review']]
```

Lampiran 10 : Kode Normalization

```
## Reload the normalizad_word_dict
normalizad_word =
pd.read_csv("cleaning_source/singkatan.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]

# Define a function for normalization
def normalized_term(tokens):
    return [normalizad_word_dict[term] if term in
normalizad_word_dict else term for term in tokens]

# Apply normalization to the 'review' column
REVIEW_DATA['review'] =
REVIEW_DATA['review'].apply(normalized_term)

REVIEW_DATA[['review']]
```

Lampiran 11 : Kode Fine-tuning IndoBERT

```
import time
```

```
import random
import numpy as np
import pandas as pd
import torch
from torch import optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
from tqdm import tqdm
from transformers import
BertForSequenceClassification, BertConfig,
BertTokenizer
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix,
classification_report, accuracy_score, f1_score,
recall_score, precision_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# common functions
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

def count_param(module, trainable=False):
    if trainable:
        return sum(p.numel() for p in
module.parameters() if p.requires_grad)
    else:
        return sum(p.numel() for p in
module.parameters())

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def metrics_to_string(metric_dict):
    string_list = []
    for key, value in metric_dict.items():
        string_list.append('{}:{:.2f}'.format(key,
value))
    return ' '.join(string_list)

def forward_sequence_classification(model,
batch_data, i2w, is_test=False, device='cpu',
**kwargs):
    # Unpack batch data
```

```
    if len(batch_data) == 3:
        (subword_batch, mask_batch, label_batch) =
batch_data
        token_type_batch = None
    elif len(batch_data) == 4:
        (subword_batch, mask_batch, token_type_batch,
label_batch) = batch_data

    # Prepare input & label
    subword_batch = torch.LongTensor(subword_batch)
    mask_batch = torch.FloatTensor(mask_batch)
    token_type_batch =
torch.LongTensor(token_type_batch) if
token_type_batch is not None else None
    label_batch = torch.LongTensor(label_batch)

    if device == "cuda":
        subword_batch = subword_batch.cuda()
        mask_batch = mask_batch.cuda()
        token_type_batch = token_type_batch.cuda() if
token_type_batch is not None else None
        label_batch = label_batch.cuda()

    # Forward model
    outputs = model(subword_batch,
attention_mask=mask_batch,
token_type_ids=token_type_batch, labels=label_batch)
    loss, logits = outputs[:2]

    # generate prediction & label list
    list_hyp = []
    list_label = []
    hyp = torch.topk(logits, 1)[1]
    for j in range(len(hyp)):
        list_hyp.append(i2w[hyp[j].item()])
        list_label.append(i2w[label_batch[j][0].item(
)])

    return loss, list_hyp, list_label

def document_sentiment_metrics_fn(list_hyp,
list_label):
    metrics = {}
    metrics["ACC"] = accuracy_score(list_label,
list_hyp)
    metrics["F1"] = f1_score(list_label, list_hyp,
average='macro')
    metrics["REC"] = recall_score(list_label,
list_hyp, average='macro')
```

```
    metrics["PRE"] = precision_score(list_label,
list_hyp, average='macro')
    return metrics

# Set random seed
set_seed(26092020)

import sys
class DocumentSentimentDataset(Dataset):
    # Static constant variable
    LABEL2INDEX = {'positif': 0, 'negatif': 1}
    INDEX2LABEL = {0: 'positif', 1: 'negatif'}
    NUM_LABELS = 2

    def load_dataset(self, df):
        df.columns = ['text', 'sentiment']
        df['sentiment'] =
df['sentiment'].apply(lambda lab:
self.LABEL2INDEX[lab])
        return df

    def __init__(self, dataset_path, tokenizer,
no_special_token=False, *args, **kwargs):
        self.data = self.load_dataset(dataset_path)
        self.tokenizer = tokenizer
        self.no_special_token = no_special_token

    def __getitem__(self, index):
        data = self.data.iloc[index, :]
        text, sentiment = data['text'],
data['sentiment']
        subwords = self.tokenizer.encode(text,
add_special_tokens=not self.no_special_token)
        return np.array(subwords),
np.array(sentiment), data['text']

    def __len__(self):
        return len(self.data)

class DocumentSentimentDataLoader(DataLoader):
    def __init__(self, max_seq_len=512, *args,
**kwargs):
        super(DocumentSentimentDataLoader,
self).__init__(*args, **kwargs)
        self.collate_fn = self._collate_fn
        self.max_seq_len = max_seq_len

    def _collate_fn(self, batch):
        batch_size = len(batch)
```

```
        max_seq_len = max(map(lambda x: len(x[0]),
batch))
        max_seq_len = min(self.max_seq_len,
max_seq_len)

        subword_batch = np.zeros((batch_size,
max_seq_len), dtype=np.int64)
        mask_batch = np.zeros((batch_size,
max_seq_len), dtype=np.float32)
        sentiment_batch = np.zeros((batch_size, 1),
dtype=np.int64)

        seq_list = []
        for i, (subwords, sentiment, raw_seq) in
enumerate(batch):
            subwords = subwords[:max_seq_len]
            subword_batch[i,:len(subwords)] =
subwords
            mask_batch[i,:len(subwords)] = 1
            sentiment_batch[i,0] = sentiment

            seq_list.append(raw_seq)

        return subword_batch, mask_batch,
sentiment_batch, seq_list

# Load Tokenizer and Config
tokenizer =
BertTokenizer.from_pretrained('indobenchmark/indobert
-base-p1')
config =
BertConfig.from_pretrained('indobenchmark/indobert-
base-p1')
config.num_labels =
DocumentSentimentDataset.NUM_LABELS
w2i, i2w = DocumentSentimentDataset.LABEL2INDEX,
DocumentSentimentDataset.INDEX2LABEL

# Membaca data dari file CSV
df = pd.read_csv('/content/Review_data.csv', sep=';')

# Menghapus baris yang mengandung nilai NaN
df = df.dropna()

# Mengubah nilai kolom "sentiment"
df['sentiment'] = df['sentiment'].replace({0:
'negatif', 1: 'positif'})
```

```
model =
BertForSequenceClassification.from_pretrained('indobe
nchmark/indobert-base-p1', config=config)
optimizer = optim.Adam(model.parameters(), lr=3e-4)
model = model.cuda()

n_epochs = 8
k_folds = 10

kf = KFold(n_splits=k_folds, shuffle=True,
random_state=42)
all_list_label, all_list_hyp, fold_accuracies,
training_times = [], [], [], []

# List untuk menyimpan metrik pada setiap fold
fold_accuracies, fold_precisions, fold_recalls,
fold_f1_scores = [], [], [], []

for fold, (train_idx, valid_idx) in
enumerate(kf.split(df)):
    print(f"Fold {fold + 1}/{k_folds}")

    # Split data into train and validation sets
    train_data, valid_data = df.iloc[train_idx],
df.iloc[valid_idx]

    # Create datasets and loaders for the current
fold
    train_dataset =
DocumentSentimentDataset(train_data, tokenizer,
lowercase=True)
    valid_dataset =
DocumentSentimentDataset(valid_data, tokenizer,
lowercase=True)

    train_loader =
DocumentSentimentDataLoader(dataset=train_dataset,
max_seq_len=512, batch_size=32, num_workers=16,
shuffle=True)
    valid_loader =
DocumentSentimentDataLoader(dataset=valid_dataset,
max_seq_len=512, batch_size=32, num_workers=16,
shuffle=False)

    start_time = time.time()

    for epoch in range(n_epochs):
        model.train()
        torch.set_grad_enabled(True)
```

```
        total_train_loss = 0
        list_hyp, list_label = [], []

        train_pbar = tqdm(train_loader, leave=True,
total=len(train_loader))

        for i, batch_data in enumerate(train_pbar):
            # Forward model
            loss, batch_hyp, batch_label =
forward_sequence_classification(model, batch_data[:-
1], i2w=i2w, device='cuda')

            # Update model
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            tr_loss = loss.item()
            total_train_loss = total_train_loss +
tr_loss

            # Calculate metrics
            list_hyp += batch_hyp
            list_label += batch_label

            train_pbar.set_description("(Fold {})
(Epoch {}) TRAIN LOSS:{:.4f} LR:{:.8f}".format((fold
+ 1), (epoch + 1),
                total_train_loss / (i + 1),
get_lr(optimizer)))

        # Calculate train metric
        metrics =
document_sentiment_metrics_fn(list_hyp, list_label)
        print("(Fold {}) (Epoch {}) TRAIN LOSS:{:.4f}
{} LR:{:.8f}".format((fold + 1), (epoch + 1),
            total_train_loss / (i + 1),
metrics_to_string(metrics), get_lr(optimizer)))

    # Evaluate on validation
    model.eval()
    torch.set_grad_enabled(False)

    total_loss, total_correct, total_labels = 0, 0, 0
    list_hyp, list_label = [], []

    pbar = tqdm(valid_loader, leave=True,
total=len(valid_loader))
```

```
    for i, batch_data in enumerate(pbar):
        batch_seq = batch_data[-1]
        loss, batch_hyp, batch_label =
forward_sequence_classification(model, batch_data[:-
1], i2w=i2w, device='cuda')

        # Calculate total loss
        valid_loss = loss.item()
        total_loss = total_loss + valid_loss

        # Calculate evaluation metrics
        list_hyp += batch_hyp
        list_label += batch_label
        metrics =
document_sentiment_metrics_fn(list_hyp, list_label)

        pbar.set_description("(Fold {}) VALID
LOSS:{:.4f} {}".format((fold + 1), total_loss / (i +
1), metrics_to_string(metrics)))

    # Simpan metrik pada setiap fold
    fold_accuracies.append(metrics["ACC"])
    fold_precisions.append(metrics["PRE"])
    fold_recalls.append(metrics["REC"])
    fold_f1_scores.append(metrics["F1"])

    metrics = document_sentiment_metrics_fn(list_hyp,
list_label)
    all_list_hyp.extend(list_hyp)
    all_list_label.extend(list_label)
    fold_accuracies.append(metrics["ACC"])
    print("(Fold {}) (Epoch {}) VALID LOSS:{:.4f}
{}".format((fold + 1), (epoch + 1),
        total_loss / (i + 1),
metrics_to_string(metrics)))

    end_time = time.time()
    training_time = end_time - start_time
    training_times.append(training_time)


# Hitung rata-rata metrik
avg_accuracy = np.mean(fold_accuracies)
avg_precision = np.mean(fold_precisions)
avg_recall = np.mean(fold_recalls)
avg_f1_score = np.mean(fold_f1_scores)

# Tampilkan rata-rata performa
```

```
print("Average Accuracy:
{:.16f}".format(avg_accuracy))
print("Average Precision:
{:.16f}".format(avg_precision))
print("Average Recall: {:.16f}".format(avg_recall))
print("Average F1-Score:
{:.16f}".format(avg_f1_score))

# Convert list_hyp and list_label to numpy arrays
np_hyp = np.array(all_list_hyp)
np_label = np.array(all_list_label)

# Calculate confusion matrix
classes = ['positif', 'negatif']
conf_matrix = confusion_matrix(np_label, np_hyp,
labels=classes)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=classes,
yticklabels=classes)
plt.title('Confusion Matrix IndoBERT')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

report = classification_report(all_list_label,
all_list_hyp, target_names=classes)
print(report)

# Menggunakan numpy untuk membuat array dari nilai-
nilai akurasi
fold_indices = np.arange(1, len(fold_accuracies) + 1)

# Plot grafik
plt.figure(figsize=(10, 6))
plt.plot(fold_indices[:k_folds],
fold_accuracies[:k_folds], marker='o', linestyle='-',
color='b')
for i, txt in enumerate(fold_accuracies[:k_folds]):
    plt.annotate(f'{txt:.4f}', (fold_indices[i],
txt), textcoords="offset points", xytext=(0,5),
ha='center')
plt.title('Akurasi Setiap Fold pada Fine Tuning
IndoBERT')
plt.xlabel('Fold')
plt.ylabel('Akurasi')
plt.grid(True)
```

```
plt.show()
```

Lampiran 12 : Kode Fine-tuning IndoBERT **dengan Ekstraksi Fitur TF-IDF**

```
import numpy as np
import pandas as pd
import time
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder,
MinMaxScaler
from sklearn.feature_extraction.text import
TfidfVectorizer
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from transformers import BertTokenizer, BertModel
import torch
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score,
confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Baca dataset
df = pd.read_csv('/content/Review_data.csv', sep=';')
df = df.dropna(subset=['text'])

# Kolom teks ulasan
texts = df['text'].astype(str).tolist()

# Kolom label sentiment
labels = df['sentiment'].tolist()

# TF-IDF Vectorizer
tfidf_vectorizer =
TfidfVectorizer(max_features=1000)  # Sesuaikan
max_features sesuai kebutuhan
tfidf_matrix =
tfidf_vectorizer.fit_transform(texts).toarray()

# Label Encoding
label_encoder = LabelEncoder()
labels =
label_encoder.fit_transform(df['sentiment'].tolist())

# Integrasi dengan model BERT
```

```
tokenizer_bert =
BertTokenizer.from_pretrained('indobenchmark/indobert
-base-p1')
model_bert =
BertModel.from_pretrained('indobenchmark/indobert-
base-p1')

# Combine features from TF-IDF and BERT
combined_features = []

for i, text in enumerate(texts):
    # TF-IDF
    tfidf_embedding = tfidf_matrix[i]

    # BERT
    input_ids = tokenizer_bert.encode_plus(text,
return_tensors="pt", padding=True, truncation=True,
max_length=512)
    output = model_bert(**input_ids)
    pooler_output =
output.pooler_output.detach().numpy()

    # Combine embeddings
    combined_embedding =
np.concatenate((tfidf_embedding,
pooler_output.flatten()))
    combined_features.append(combined_embedding)

# Convert to numpy array
combined_features = np.array(combined_features)

# Normalize input features
scaler = MinMaxScaler()
combined_features_normalized =
scaler.fit_transform(combined_features)

# Initialize variables to store total confusion
matrix and true/pred labels
total_conf_matrix = np.zeros((2, 2), dtype=int)
all_true_labels, all_pred_labels = [], []

# K-Fold Cross Validation
k_folds = 10
kf = KFold(n_splits=k_folds, shuffle=True,
random_state=42)

fold_accuracies, fold_precisions, fold_recalls,
fold_f1_scores = [], [], [], []
```

```
# Initialize list to store average accuracies per
fold
avg_accuracies_per_fold = []

# Initialize list to store training times per fold
fold_training_times = []

for fold, (train_idx, test_idx) in
enumerate(kf.split(combined_features)):
    print(f"Fold {fold + 1}/{k_folds}")

    # Split data into training and testing sets
    X_train, X_test =
combined_features_normalized[train_idx],
combined_features_normalized[test_idx]
    labels_train, labels_test = labels[train_idx],
labels[test_idx]

    # Ensure labels are numpy arrays with shape
(num_samples, 1)
    labels_train = np.array(labels_train).reshape(-1,
1)
    labels_test = np.array(labels_test).reshape(-1,
1)

    # Model
    model = Sequential()
    model.add(Dense(128, activation='relu',
input_dim=combined_features.shape[1]))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=4e-6),
loss='binary_crossentropy', metrics=['accuracy'])

    # Training model
    start_time = time.time()
    model.fit(X_train, labels_train, epochs=2,
batch_size=16, validation_data=(X_test, labels_test))
    end_time = time.time()
    elapsed_time = end_time - start_time
    fold_training_times.append(elapsed_time)

    # Evaluation
    loss, accuracy = model.evaluate(X_test,
labels_test)
    print(f'Fold {fold + 1} - Test Loss: {loss:.4f},
Test Accuracy: {accuracy:.4f}')

    # Prediction using the model on the test set
    y_pred = model.predict(X_test)
```

```
    predictions_test = (y_pred > 0.5).astype(int)

    # Update total confusion matrix and true/pred
labels
    total_conf_matrix +=
confusion_matrix(labels_test, predictions_test)
    all_true_labels.extend(labels_test)
    all_pred_labels.extend(predictions_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(labels_test,
predictions_test)
    precision = precision_score(labels_test,
predictions_test)
    recall = recall_score(labels_test,
predictions_test)
    f1 = f1_score(labels_test, predictions_test)

    # Save metrics for each fold
    fold_accuracies.append(accuracy)
    fold_precisions.append(precision)
    fold_recalls.append(recall)
    fold_f1_scores.append(f1)

    # Append average accuracy to the list
    avg_accuracies_per_fold.append(accuracy)

    # Display metrics for each fold
    print(f'Fold {fold + 1} - Accuracy:
{accuracy:.4f}, Precision: {precision:.4f}, Recall:
{recall:.4f}, F1 Score: {f1:.4f}')

# Calculate average metrics across folds
avg_accuracy = np.mean(fold_accuracies)
avg_precision = np.mean(fold_precisions)
avg_recall = np.mean(fold_recalls)
avg_f1_score = np.mean(fold_f1_scores)

total_training_time = sum(fold_training_times)

# Display average metrics
print(f"\nTotal Training Time:
{total_training_time:.2f} seconds")
print("\nAverage Accuracy:
{:.4f}".format(avg_accuracy))
print("Average Precision:
{:.4f}".format(avg_precision))
print("Average Recall: {:.4f}".format(avg_recall))
```

```
print("Average F1-Score:
{:.4f}".format(avg_f1_score))

# Plot average accuracies per fold
plt.figure(figsize=(10, 6))
plt.plot(range(1, k_folds + 1),
avg_accuracies_per_fold, marker='o', linestyle='-',
color='b')
plt.title('Average Accuracy Per Fold')
plt.xlabel('Fold')
plt.ylabel('Average Accuracy')
plt.grid(True)
plt.show()

# Display total confusion matrix
print("\nTotal Confusion Matrix:")
print(total_conf_matrix)

# Display total classification report with custom
labels
unique_labels_str = ['negatif', 'positif']
report = classification_report(all_true_labels,
all_pred_labels, target_names=unique_labels_str)
print("\nTotal Classification Report:")
print(report)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(total_conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=unique_labels_str,
yticklabels=unique_labels_str)
plt.title('Total Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Lampiran 13 : Kode Fine-tuning IndoBERT **dengan Ekstraksi Fitur Word2Vec**

```
import numpy as np
import pandas as pd
import time
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder,
MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Dense, Embedding, Flatten,
concatenate
from keras.optimizers import Adam
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import
pad_sequences
from gensim.models import Word2Vec
from transformers import BertTokenizer, BertModel
import torch
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score,
confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Baca dataset
df = pd.read_csv('/content/Review_data.csv', sep=';')
df = df.dropna(subset=['text'])

# Kolom teks ulasan
texts = df['text'].astype(str).tolist()

# Kolom label sentiment
labels = df['sentiment'].tolist()

# Tokenisasi teks
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Padding sequences agar memiliki panjang yang sama
padded_sequences = pad_sequences(sequences)

# Word2Vec embedding
word2vec_model = Word2Vec(sentences=[text.split() for
text in texts], vector_size=100, window=5,
min_count=1, workers=4)
word_index = tokenizer.word_index

word2vec_vectors = []
for word, i in word_index.items():
    if word in word2vec_model.wv:
        word2vec_vectors.append(word2vec_model.wv[wor
d][:100])

# Membuat array 2D dari word2vec_vectors
word2vec_matrix = np.array(word2vec_vectors)
```

```
# Label Encoding
label_encoder = LabelEncoder()
labels =
label_encoder.fit_transform(df['sentiment'].tolist())

# Integrasi dengan model BERT
tokenizer_bert =
BertTokenizer.from_pretrained('indobenchmark/indobert
-base-p1')
model_bert =
BertModel.from_pretrained('indobenchmark/indobert-
base-p1')

# Combine features from Word2Vec and BERT
combined_features = []

for text in texts:
    # Word2Vec
    word2vec_embedding =
[word2vec_model.wv[word][:100] for word in
text.split() if word in word2vec_model.wv]
    word2vec_embedding = np.mean(word2vec_embedding,
axis=0) if word2vec_embedding else np.zeros(100)

    # BERT
    input_ids = tokenizer_bert.encode_plus(text,
return_tensors="pt", padding=True, truncation=True,
max_length=512)
    output = model_bert(**input_ids)
    pooler_output =
output.pooler_output.detach().numpy()

    # Combine embeddings
    combined_embedding =
np.concatenate((word2vec_embedding,
pooler_output.flatten()))
    combined_features.append(combined_embedding)

# Convert to numpy array
combined_features = np.array(combined_features)

scaler = MinMaxScaler()
combined_features_normalized =
scaler.fit_transform(combined_features)

# Initialize variables to store total confusion
matrix and true/pred labels
total_conf_matrix = np.zeros((2, 2), dtype=int)
all_true_labels, all_pred_labels = [], []
```

```
# K-Fold Cross Validation
k_folds = 10
kf = KFold(n_splits=k_folds, shuffle=True,
random_state=42)

fold_accuracies, fold_precisions, fold_recalls,
fold_f1_scores = [], [], [], []

# Initialize list to store average accuracies per
fold
avg_accuracies_per_fold = []

# Initialize list to store training times per fold
fold_training_times = []

for fold, (train_idx, test_idx) in
enumerate(kf.split(combined_features)):
    print(f"Fold {fold + 1}/{k_folds}")

    # Split data into training and testing sets
    X_train, X_test =
combined_features_normalized[train_idx],
combined_features_normalized[test_idx]
    labels_train, labels_test = labels[train_idx],
labels[test_idx]

    # Ensure labels are numpy arrays with shape
(num_samples, 1)
    labels_train = np.array(labels_train).reshape(-1,
1)
    labels_test = np.array(labels_test).reshape(-1,
1)

    # Model
    model = Sequential()
    model.add(Dense(128, activation='relu',
input_dim=combined_features.shape[1]))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=4e-6),
loss='binary_crossentropy', metrics=['accuracy'])

    # Training model
    start_time = time.time()
    model.fit(X_train, labels_train, epochs=2,
batch_size=16, validation_data=(X_test, labels_test))
    end_time = time.time()
    elapsed_time = end_time - start_time
    fold_training_times.append(elapsed_time)
```

```
    # Evaluation
    loss, accuracy = model.evaluate(X_test,
labels_test)
    print(f'Fold {fold + 1} - Test Loss: {loss:.4f},
Test Accuracy: {accuracy:.4f}')

    # Prediction using the model on the test set
    y_pred = model.predict(X_test)
    predictions_test = (y_pred > 0.5).astype(int)

    # Update total confusion matrix and true/pred
labels
    total_conf_matrix +=
confusion_matrix(labels_test, predictions_test)
    all_true_labels.extend(labels_test)
    all_pred_labels.extend(predictions_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(labels_test,
predictions_test)
    precision = precision_score(labels_test,
predictions_test)
    recall = recall_score(labels_test,
predictions_test)
    f1 = f1_score(labels_test, predictions_test)

    # Save metrics for each fold
    fold_accuracies.append(accuracy)
    fold_precisions.append(precision)
    fold_recalls.append(recall)
    fold_f1_scores.append(f1)

    # Append average accuracy to the list
    avg_accuracies_per_fold.append(accuracy)

    # Display metrics for each fold
    print(f'Fold {fold + 1} - Accuracy:
{accuracy:.4f}, Precision: {precision:.4f}, Recall:
{recall:.4f}, F1 Score: {f1:.4f}')

# Calculate average metrics across folds
avg_accuracy = np.mean(fold_accuracies)
avg_precision = np.mean(fold_precisions)
avg_recall = np.mean(fold_recalls)
avg_f1_score = np.mean(fold_f1_scores)

total_training_time = sum(fold_training_times)
```

```
# Display average metrics
print(f"\nTotal Training Time:
{total_training_time:.2f} seconds")

print("\nAverage Accuracy:
{:.4f}".format(avg_accuracy))
print("Average Precision:
{:.4f}".format(avg_precision))
print("Average Recall: {:.4f}".format(avg_recall))
print("Average F1-Score:
{:.4f}".format(avg_f1_score))

# Plot average accuracies per fold
plt.figure(figsize=(10, 6))
plt.plot(range(1, k_folds + 1),
avg_accuracies_per_fold, marker='o', linestyle='-',
color='b')
plt.title('Average Accuracy Per Fold')
plt.xlabel('Fold')
plt.ylabel('Average Accuracy')
plt.grid(True)
plt.show()

# Display total confusion matrix
print("\nTotal Confusion Matrix:")
print(total_conf_matrix)

# Display total classification report with custom
labels
unique_labels_str = ['negatif', 'positif']
report = classification_report(all_true_labels,
all_pred_labels, target_names=unique_labels_str)
print("\nTotal Classification Report:")
print(report)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(total_conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=unique_labels_str,
yticklabels=unique_labels_str)
plt.title('Total Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```