LAMPIRAN

Lampiran 1. Riwayat Hidup

RIWAYAT HIDUP

Perkenalkan nama saya Komang Hokky Aryasta, saat ini, saya tinggal di Seririt, Buleleng, Bali. Perjalanan pendidikan saya dimulai di SD N 1 Seririt, tempat saya menuntaskan pendidikan dasar. Setelah menyelesaikan Sekolah Dasar, pendidikan saya dilanjutkan ke SMP N 1 Seririt. Mengikuti jejak pendidikan menengah pertama, saya melanjutkan ke SMA N 1 Seririt, 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. Seluruh perjalanan pendidikan ini mencerminkan tekad dan keseriusan saya dalam mengejar ilmu, serta keyakinan bahwa setiap fase pendidikan adalah langkah menuju pencapaian tujuan dan kesuksesan di masa depan.

Lampiran 2. Kode *Scraping Data*

```
# mengimport library google_play_scraper
from google_play_scraper import app, Sort, reviews,
reviews_all

# mengatur score yang akan diambil (1-5)
score = 1
filename = f'score-{score}'
result = reviews_all(
    'com.instagram.barcelona',
    sleep_milliseconds=0,
    lang='id',
    country='id',
    sort=Sort.MOST_RELEVANT,
    filter_score_with=score,
)

# simpan file hasil scraping masing-masing score
result_df.to_csv(f'{filename}.csv', index=False,
escapechar='\\')

df1 = pd.read_csv('score-1.csv')
df2 = pd.read_csv('score-2.csv')
df3 = pd.read_csv('score-3.csv')
```

```
df4 = pd.read_csv('score-4.csv')
df5 = pd.read_csv('score-5.csv')

combined_df = pd.concat([df1, df2, df3, df4, df5],
ignore_index=True)

# gabungkan file dari hasil scraping masing-masing score
combined_df.to_csv('dataset-mentah.csv', index=False,
sep=';')
```

Lampiran 3. Kode Data *Selection*

```
# menghapus data duplikat dan data null
df = df.drop_duplicates()
df = df.dropna()
```

Lampiran 4. Kode Data *Labeling*

```
# memberikan label sesuai dengan score
df['sentiment'] = df['score'].apply(lambda x: 'negatif' if x
in [1, 2] else ('netral' if x == 3 else 'positif'))
```

Lampiran 5. Kode *Case Folding*

```
# mengubah text menjadi lowercase
df['text'] = df['content'].str.lower()
```

Lampiran 6. Kode *Cleansing*

```
import re
import string

# hapus special character (emoticon, simbol, dan lainnya)
def remove_special_char(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://",
           " ")
df['text'] = df['text'].apply(remove_special_char)

# hapus angka
def remove_number(text):
    return  re.sub(r"\d+", "", text)
df['text'] = df['text'].apply(remove_number)

# hapus tanda baca
def remove_punctuation(text):
    return
text.translate(str.maketrans("","",string.punctuation))
df['text'] = df['text'].apply(remove_punctuation)
```

```
# hapus whitespace leading & trailing
def remove_whitespace_LT(text):
    return text.strip()
df['text'] = df['text'].apply(remove_whitespace_LT)

# hapus whitespace multiple
def remove_whitespace_multiple(text):
    return re.sub('\s+',' ',text)
df['text'] = df['text'].apply(remove_whitespace_multiple)

# hapus single char
def remove_single_char(text):
    return re.sub(r"\b[a-zA-Z]\b", "", text)
df['text'] = df['text'].apply(remove_single_char)
```

Lampiran 7. Kode *Tokenizing*

```
# mengimport library nltk dan word_tokenize
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
def word_tokenize_wrapper(text):
    return word_tokenize(text)

df['text'] = df['text'].apply(word_tokenize_wrapper)
```

Lampiran 8. Kode *Stopword Removal*

```
# mengimport library nltk dan stopwords
from nltk.corpus import stopwords
nltk.download('stopwords')

stopwords_indonesia = set(stopwords.words('indonesian'))
stopwords_inggris = set(stopwords.words('english'))
list_stopwords = stopwords_indonesia.union(stopwords_inggris)

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

df['text'] = df['text'].apply(stopwords_removal)
```

Lampiran 9. Kode *Stemming*

```
# mengimport library Sastrawi
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

factory = StemmerFactory()
stemmer = factory.create_stemmer()
```

```
# df['text'] = df['text'].apply(lambda x: '
               '.join([stemmer.stem(word) for word in
               x.split()]))
```

Lampiran 10. Kode *Normalization*

```
# mengimport corpus dari github (riochr17)
normalizad_word = pd.read_csv('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]

def normalized_term(document):
    return [normalizad_word_dict[term] if term in
    normalizad_word_dict else term for term in document]
df['text'] = df['text'].apply(normalized_term)

df['text'] = (
    df['text']
        .transform(
            lambda x: " ".join(map(str,x))
        )
    )

df = df.loc[:, ['text', 'sentiment']]
df = df.drop_duplicates()
df = df.dropna()
df.to_csv('threads-reviews.csv', index=False, sep=';')
```

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, 'netral': 1, 'negatif': 2}
    INDEX2LABEL = {0: 'positif', 1: 'netral', 2: 'negatif'}
    NUM_LABELS = 3

    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

df = pd.read_csv('/content/threads-reviews.csv', sep=';')

model =
BertForSequenceClassification.from_pretrained('indobenchmark
/indobert-base-p1', config=config)
optimizer = optim.Adam(model.parameters(), lr=3e-6)
model = model.cuda()

n_epochs = 3
```

```
k_folds = 10

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

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)))

    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)

# 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', 'netral', '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)
```

Lampiran 12. Kode Multinomial Naïve Bayes

```
import time
import itertools
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix,
classification_report, accuracy_score
from sklearn.model_selection import KFold
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

df = pd.read_csv('/content/threads-reviews.csv', sep=';')
df = df.dropna()
df = df.drop_duplicates()
data_x = df['text'].values
data_y = df['sentiment'].values
vectorizer = CountVectorizer()
data_x = vectorizer.fit_transform(data_x)
class_names = df['sentiment'].unique()
```

```
fold_accuracies = []

def evaluate_model(data_x, data_y):
    k_fold = KFold(10, shuffle=True, random_state=1)

    predicted_targets = np.array([])
    actual_targets = np.array([])

    total_training_time = 0

    for train_ix, test_ix in k_fold.split(data_x):
        train_x, train_y, test_x, test_y = data_x[train_ix],
data_y[train_ix], data_x[test_ix], data_y[test_ix]

        start_time = time.time()

        # Fit the classifier
        classifier = MultinomialNB().fit(train_x, train_y)

        end_time = time.time()
        total_training_time += end_time - start_time

        # Predict the labels of the test set samples
        predicted_labels = classifier.predict(test_x)

        predicted_targets = np.append(predicted_targets,
                                      predicted_labels)
        actual_targets = np.append(actual_targets, test_y)

        accuracy = accuracy_score(test_y, predicted_labels)
        fold_accuracies.append(accuracy)

    return predicted_targets, actual_targets,
           total_training_time

def plot_confusion_matrix(predicted_labels_list,
y_test_list, class_names):
    cnf_matrix = confusion_matrix(y_test_list,
                  predicted_labels_list, labels=class_names)
    np.set_printoptions(precision=2)

    plt.figure(figsize=(8, 6))
    generate_confusion_matrix(cnf_matrix,
                              classes=class_names,
                              title='Confusion Matrix
                              Multinomial Naive Bayes')
    plt.show()

def generate_confusion_matrix(cnf_matrix, classes,
    normalize=False, title='Confusion matrix'):
    sns.heatmap(cnf_matrix, annot=True, fmt='d',
                cmap='Blues', xticklabels=classes,
                yticklabels=classes)
```

```
    plt.title('Confusion Matrix Multinomial Naive Bayes')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')

def print_evaluation_metrics(actual_targets,
    predicted_targets):
    report = classification_report(actual_targets,
                                   predicted_targets,
                                   target_names=class_names)
    print(report)

predicted_target, actual_target, total_training_time =
evaluate_model(data_x, data_y)
plot_confusion_matrix(predicted_target, actual_target,
class_names)
print_evaluation_metrics(actual_target, predicted_target)
```