Lampiran 1 Source Code

Crawling Data Instagram

```
!pip install selenium bs4
!pip install beautifulsoup4
!pip install pandas
!apt-get update
!apt-get install -y chromium-browser
!apt install chromium-chromedriver
```

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from bs4 import BeautifulSoup
import time
import pandas as pd
from pandas import ExcelWriter
import os.path
import csv
```

```
# path ke Chromedriver
chromedriver_path = '/usr/lib/chromium-browser/chromedriver'

# Set opiton Chrome untuk webdriver
options = webdriver.ChromeOptions()
options.add_argument('--no-sandbox')
options.add_argument('--headless')
options.add_argument('--disable-gpu')
options.add_argument('--disable-dev-shm-usage')

# Inisialisasi driver Chrome
driver = webdriver.Chrome(options=options)
```

```
url='https://www.instagram.com/accounts/login/'
driver.get(url)
usernameInput='LOGIN-ANDA'
passwordInput='PASSWORD'
# mencari username input
username=WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, '//input[@name="username"]'))) .send_keys(usernameInput)
# mencari password input
password=WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, '//input[@name="password"]'))).send_keys(passwordInput)
# press ENTER untuk login
WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, '//input[@name="password"]'))).send_keys(Keys.ENTER)
```

Preprocessing Data

```
!pip install Sastrawi
import re
import pandas as pd
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
```

```
data_crawling = 'data.csv'
df = pd.read_csv(data_crawling)
# Menampilkan DataFrame
print(df)
```

```
# case folding
df['comment'] = df['comment'].str.lower()
print('case Folding Result : \n')
print(df['comment'])
print('\n\n\n')
```

```
def clean_text(text):
    # Cleansing
    # Remove tab, new line, and back slash
    text = text.replace('\\t',' ').replace('\n',' ').replace('\\n',' ').replace('\\','')
    # Remove non-ASCII characters
    text = text.encode('ascii', 'replace').decode('ascii')
    # Remove mention, link, and hashtag
    text = ' '.join(re.sub("([@#][A-Za-z0-9]+)|(\w+:\/\/\S+)"," ", text).split())
    # remove URLs
    text = re.sub(r'http\S+', ' ', text)
    # remove incomplete URL
    text = text.replace('http://', ' ').replace('https://', ' ')
    # remove numbers
    text = re.sub(r"\d+", "", text)
    # remove punctuation
    text = re.sub(r'[^\w\s]', ' ', text)
    # remove leading and trailing whitespace
    text = text.strip()
    # remove multiple whitespace into single whitespace
    text = re.sub('\s+',' ',text)
    # remove single character
    text = re.sub(r"\b[a-zA-Z]\b", "", text)
    return text

# Hapus baris dengan nilai NaN atau string kosong pada kolom 'comment'
df = df[df['comment'].notna() & (df['comment'] != '')]

# Cleansing text
df['cleansing'] = df['comment'].apply(lambda x: clean_text(x))

print('Cleansing text \n')
print(f"{df['cleansing'].head()}\n\n")
```

```
factory = StemmerFactory()
stemmer = factory.create_stemmer()
def stem_text(teks):
    return stemmer.stem(teks)
df['stemming'] = df['cleansing'].apply(stem_text)
print('Stemming text:\n')
print(f"{df['stemming'].head()}\n\n")


from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory
factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()
```

```
def remove_stopwords(text):
    return stopword.remove(text)
df['stopwords'] = df['stemming'].apply(remove_stopwords)
print('Stowords Text:\n')
print(f"{df['stopwords'].head()}\n\n")
```

```
# Load data slangwords dari file csv
slangwords_df = pd.read_csv("slangwords_indonesia.csv")

# Buat kamus slangwords dari dataframe
slangwords_dict = dict(zip(slangwords_df['slang'], slangwords_df['formal']))

# Fungsi untuk mengganti slangwords dalam teks
def replace_slangwords(text):
    words = text.split()
    normalized_words = [slangwords_dict[word] if word in slangwords_dict else word for word in words]
    return ' '.join(normalized_words)

# Terapkan fungsi pada kolom 'stopwords'
df['slangword'] = df['stopwords'].apply(replace_slangwords)

# Simpan hasil
df.head()
```

Translate Data

```
!pip install translate
```

```
import pandas as pd
from translate import Translator

df = pd.read_csv('hasil1.csv')

def translate_to_english(text):
    if isinstance(text, str):
        translator = Translator(to_lang="en", from_lang="id")
        translation = translator.translate(text)
        return translation
    else:
        return text

df['english_translation'] = df['slangword'].apply(translate_to_english)
df['english_translation'].to_csv('translate_1.csv', index=False)
```

```
import glob
import pandas as pd

# List semua file CSV
csv_files = glob.glob('translate_*.csv')

# Inisialisasi empty dataframe
combined_df = pd.DataFrame()

# Menggabungkan semua file menjadi satu
for csv_file in csv_files:
    df = pd.read_csv(csv_file)
    combined_df = pd.concat([combined_df, df], ignore_index=True)

# Save to new csv
combined_df.to_csv('translate_combined.csv', index=False)
```

```
# membaca file csv
df = pd.read_csv('translate_combined.csv')

# Hapus baris di mana 'english_translation' kosong
df_filtered = df[df['english_translation'].str.contains('[a-zA-Z]', na=False)]

# Simpan data yang telah difilter
df_filtered.to_csv('filtered_translate.csv', index=False)
```

Labelling Data

```
# Read CSV
df = pd.read_csv('filtered_translate.csv')
# Print 'slangword' column
print(df['English_Translation'])
```

```
!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
```

```
# Membuat Score setiap Comment
df['Compound Score'] = df['English_Translation']
scores = [analyser.polarity_scores(c) for c in df['Compound Score']]
print(scores)

df['Compound_Score'] = [s['compound'] for s in scores]
```

```
hasilAnalisis = []
for score in df['Compound_Score']:
    sentimen = {}
    if score < 0:
        sentimen['Sentiments'] = 'Negatif'
        hasilAnalisis.append(sentimen)
    elif score > 0:
        sentimen['Sentiments'] = 'Positif'
        hasilAnalisis.append(sentimen)
    else:
        # Jika score tepat 0
        sentimen['Sentiments'] = 'Netral'
        hasilAnalisis.append(sentimen)

# membuat df baru dari hasilAnalisis
result_df = pd.DataFrame(hasilAnalisis)

# Menggabungkan dataframe asli dengan dataframe hasil
df = pd.concat([df, result_df], axis=1)

df.head()
```

Konversi sentiment kedalam data index

```
column_to_delete = 'Compound_Score'
df = df.drop(column_to_delete, axis=1)
```

```
labels = df.Sentiments.unique()
```

```
labels_dict = {}
for index, labels in enumerate(labels):
  labels_dict[labels] = index
```

```
df['labels'] = df.Sentiments.replace(labels_dict)
print(df)
```

Split Data

```
from sklearn.model_selection import train_test_split

x_train, x_val, y_train, y_val = train_test_split(df.index.values,
                                                  df.labels.values,
                                                  test_size=0.10,
                                                  random_state=2020,
                                                  stratify=df.labels.values)
```

```
df['data_type'] = ['not_set']*df.shape[0]
df.loc[x_train, 'data_type'] = 'train'
df.loc[x_val, 'data_type'] = 'val'
df.groupby(['Sentiments', 'labels','data_type']).count()
```

BERT Tokenizer dan Encode Data

```
!pip install transformers
```

```
from transformers import BertTokenizer
from torch.utils.data import TensorDataset

tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-uncased',
                                          do_lower_case=True)
```

```
# Proses tokenisasi dan konversi ke token IDs untuk setiap kalimat
tokenized_sentences = [tokenizer.tokenize(sentence) for sentence in sentences]
token_ids_list = [tokenizer.convert_tokens_to_ids(tokenized_sentence) for tokenized_sentence in tokenized_sentences]

# mendapatkan panjang maksimum dari seluruh data
max_sentence_length = max(len(token_ids) for token_ids in token_ids_list)

print("Original Sentences: ", sentences[0])
print("Tokenized Sentences: ", tokenized_sentences[0])
print("Token IDs: ", token_ids_list[0])
print("Max Sentence Length: ", max_sentence_length)
```

```
#Encode training data
encoded_data_train = tokenizer.batch_encode_plus(
    df[df.data type== 'train'].English Translation.values,
    add_special_tokens=True,
    return_attention_mask True,
    padding=True,
    max_length=32,
    return_tensors='pt',
    truncation True
)
#Encode validation data
encoded_data_val = tokenizer.batch_encode_plus(
    df[df.data type== 'val'].English Translation.values,
    add_special_tokens=True,
    return_attention_mask=True,
    padding=True,
    max_length=32,
    return_tensors='pt',
    truncation=True
)
# Extract input_ids, attention_mask, and labels untuk training data
input_ids_train = encoded_data_train['input_ids']
attention_mask_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(df[df.data_type=='train'].labels.values)

# Extract input_ids, attention_mask, and labels untuk validation data
input_ids_val = encoded_data_val['input_ids']
attention_mask_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(df[df.data type== 'val'].labels.values)
```

```
# membuat TensorDataset untuk training dan validation data
dataset_train = TensorDataset(input_ids_train, attention_mask_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_mask_val, labels_val)
```

BERT Pretrained Model

```
from transformers import BertForSequenceClassification
model = BertForSequenceClassification.from_pretrained('bert-base-multilingual-uncased',
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

```
params = list(model.named_parameters())

print('The BERT model has {:} different named parameters.\n'.format(len(params)))

print('==== Embedding Layer ====')
for p in params[0:5]:
  print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====')
for p in params[5:21]:
  print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====')
for p in params[-4:]:
  print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

# Set batch size untuk data loader
batch_size = 32

# Membuat dataloader untuk train dan validasi
dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=batch_size)

dataloader_validation = DataLoader(dataset_val,
                                   sampler=SequentialSampler(dataset_val),
                                   batch_size=32)
```

```
from torch.optim import AdamW
from transformers import get_linear_schedule_with_warmup

optimizer = AdamW(model.parameters(), lr=1e-5, eps=1e-8)

epochs = 10

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0,
                                            num_training_steps = len(dataloader_train)*epochs)
```

```
import numpy as np
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
```

```
#define untuk calculate f1 score
def f1_score_func(preds, labels):
  preds_flat = np.argmax(preds, axis=1).flatten()
  labels_flat = labels.flatten()
  return f1_score(labels_flat, preds_flat, average='weighted')

#define untuk calculate precision
def precision_score_func(preds, labels):
  preds_flat = np.argmax(preds, axis=1).flatten()
  labels_flat = labels.flatten()
  return precision_score(labels_flat, preds_flat, average='macro')

#define untuk calculate recall
def recall_score_func(preds, labels):
  preds_flat = np.argmax(preds, axis=1).flatten()
  labels_flat = labels.flatten()
  return recall_score(labels_flat, preds_flat, average='macro')

#define untuk calculate accuracy
def accuracy_score_func(preds, labels):
  preds_flat = np.argmax(preds, axis=1).flatten()
  labels_flat = labels.flatten()
  return accuracy_score(labels_flat, preds_flat)
```

```
#define untuk calculate accuracy per class
def accuracy_per_class(preds, labels):
# [illegible]
  label_dict_inverse = {v: k for k, v in label_dict.items()}

  preds_flat = np.argmax(preds, axis=1).flatten()
  labels_flat = labels.flatten()

  for label in np.unique(labels_flat):
# [illegible]
    y_preds = preds_flat[labels_flat==label]
    y_true = labels_flat[labels_flat==label]
    print(f'Class: {label_dict_inverse[label]}')
    print(f'Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

```
!pip install tqdm
from tqdm import tqdm
```

```
# [illegible]
training_stats = []
# [illegible]
for epoch_i in tqdm(range(1, epochs+1)):
    model.train()
    loss_train_total = 0
# [illegible]
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch_i), leave=False, disable=False)
    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {'input_ids': batch[0],
                  'attention_mask': batch[1],
                  'labels': batch[2],}
# [illegible]
        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
# [illegible]
        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})
# Save model
    torch.save(model.state_dict(), f'[illegible]_{epoch_i}.model')
# [illegible]
    tqdm.write(f'\nEpoch {epoch_i}')
# [illegible]
    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')
# [illegible]
    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    val_precision = precision_score_func(predictions, true_vals)
    val_recall = recall_score_func(predictions, true_vals)
    val_accuracy = accuracy_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score: {val_f1}')
    tqdm.write(f'Precision Score: {val_precision}')
    tqdm.write(f'Recall Score: {val_recall}')
    tqdm.write(f'Accuracy Score: {val_accuracy}')

    # [illegible]
    training_stats.append(
        {
            'epoch': epoch_i,
            'Training Loss': loss_train_avg,
            'Valid. Loss': val_loss,
            'F1 Score': val_f1,
            'Precision Score': val_precision,
            'Recall Score': val_recall,
            'Accuracy Score': val_accuracy,
        }
    )

# [illegible]
val_loss, predictions, true_vals = evaluate(dataloader_validation)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = true_vals
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# [illegible]
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', [illegible])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix [illegible]')
plt.show()
```

Tokenize

```
import re
import numpy as np
import pandas as pd
# NLTK
from nltk.stem import WordNetLemmatizer
# Sklearn
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

```
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize

df = pd.read_csv('labelling.csv')

# Unduh model tokenisasi punkt
nltk.download('punkt')

column_name = 'English Translation'

# Tokenisasi
df[column_name] = df[column_name].apply(lambda x: word_tokenize(str(x)))

# Tampilkan hasil tokenisasi
print(df[column_name])
```

TF-IDF

```
# Membuat TfidfVectorizer dengan parameter-parameter
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=500000)
# Melatih vectorizer
vectorizer.fit(df.English_Translation)
print("No. of feature_words: ", len(vectorizer.get_feature_names_out()))
```

```
# Membuat DataFrame untuk training data
df_train = pd.DataFrame()
df_train['English_Translation'] = X_train
df_train['Sentiments'] = y_train
```

```
# Membuat DataFrame untuk test data
df_test = pd.DataFrame()
df_test['English Translation'] = X_val
df_test['Sentiments'] = y_val
```

```
Melakukan transformasi data menggunakan TF-IDF Vectorizer
X_train_tfidf = vectorizer.transform(df_train['English Translation'])
X_val_tfidf  = vectorizer.transform(df_test['English Translation'])
```

```
# Ini untuk mengubah transformasi TF IDF ke dalam DataFrame
df_tfidf = pd.DataFrame(X_train_tfidf.toarray(), columns=vectorizer.get_feature_names_out())

# Cara tercepat dengan weight rata rata TF IDF
average_tfidf = df_tfidf.mean(axis=0)
top_terms = average_tfidf.sort_values(ascending=False)

print("10 Kata TF IDF Weight:")
print(top_terms.head(10))
```

Metode SVM

```
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC
for c in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
  svm = LinearSVC(C=c)
  svm.fit(x_train_tfidf,y_train)
  print('Akurasi C kernel Linear = %.2%%' %(c, accuracy_score(y_val, svm.predict(x_val_tfidf))))
```

```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1)
model.fit(x_train_tfidf,y_train)
```

```
# Prediksi pengujian SVM 2 (1:10)
from sklearn.metrics import accuracy_score

# Melakukan prediksi sentimen pada data validasi
predictions_SVM_1_100 = model.predict(x_val_tfidf)

test_prediction = pd.DataFrame()
test_prediction['Text'] = x_val
test_prediction['sentiments'] = predictions_SVM_1_100

#Menghitung akurasi prediksi
SVM_accuracy_TFIDF= accuracy_score(predictions_SVM_TFIDF, y_val)*100
SVM_accuracy_TFIDF= round(SVM_accuracy_TFIDF, )
```

```
#SVM Accuracy
SVM_accuracy_TFIDF
```

Confusion Matrix

```
# Accuracy, Precision, Recall, F1-score
from sklearn.metrics import classification_report
print ("Classification report:")
print (classification_report(y_val, predictions_SVM_TFIDF))
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# menghitung confusion matrix
cm = confusion_matrix(y_val, predictions_SVM_TFIDF)

# Visualisasi confusion matrix sebagai heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=['0', '1', '2'], yticklabels=['0', '1', '2'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```