**Input Data**

|  |
| --- |
| import pandas as pd  #Koneksi pada Google Drive  from google.colab import drive  drive.mount('/content/drive')  #Input Data  data = pd.read\_csv('/content/drive/MyDrive/SKRIPSI/Data Labelling.csv',sep=";")  data |

**Data Pre-Processing**

|  |
| --- |
| #Case Folding  data['review'] = data['review'].str.lower()  print('Case Folding Result : \n')  print(data['review'].head(5))  print('\n\n\n')  import string, re  import nltk  nltk.download('punkt')  from nltk.tokenize import word\_tokenize  import string, re  import nltk  nltk.download('punkt')  from nltk.tokenize import word\_tokenize  #Tokenizing  def remove\_comments\_special(text):  #menghapus karakter khusus dalam teks, seperti tab, newline, dan backslash  text = text.replace('\\t'," ").replace('\\n'," ").replace('\\u'," ").replace('\\'," ").replace('.'," ")  # remove non ASCII (emoticon, chinese word, .etc)  text = text.encode('ascii', 'replace').decode('ascii')  # remove mention, link, hashtag  text = ' '.join(re.sub("([@#][A-Za-z0-9]+)|(\w+:\/\/\S+)"," ", text).split())  # remove incomplete URL  return text.replace("http://", " ").replace("https://", " ")  data['review'] = data['review'].apply(remove\_comments\_special)  #menghapus angka dalam teks dengan menggunakan regular expression  def remove\_number(text):  return re.sub(r"\d+", " ", text)  data['review'] = data['review'].apply(remove\_number)  #menghapus emotikon dari teks menggunakan regular expression  def remove\_emoticons(text):  emoticon\_pattern = re.compile("["  u"\U0001F600-\U0001F64F" # emoticons  u"\U0001F300-\U0001F5FF" # simbol & piktogram  u"\U0001F680-\U0001F6FF" # transportasi & simbol peralatan  u"\U0001F1E0-\U0001F1FF" # bendera negara  u"\U00002702-\U000027B0" # simbol lainnya  u"\U000024C2-\U0001F251"  "]+", flags=re.UNICODE)  return emoticon\_pattern.sub(r'', text)  data['review'] = data['review'].apply(remove\_emoticons)  #data['review'] = data['review'].apply(remove\_number)  def remove\_punctuation(text):  return text.translate(str.maketrans("","",string.punctuation))  data['review'] = data['review'].apply(remove\_punctuation)  # menghapus spasi yang ada di awal dan akhir teks menggunakan str.strip()  def remove\_whitespace\_LT(text):  return text.strip()  data['review'] = data['review'].apply(remove\_whitespace\_LT)  #menghapus spasi berlebih dalam teks dengan menggantinya dengan satu spasi menggunakan regular expression  def remove\_extra\_spaces(text):  return re.sub(r'\s+', ' ', text)  data['review'] = data['review'].apply(remove\_extra\_spaces)  #mengganti multiple whitespace (spasi berturut-turut) dengan satu spasi menggunakan regular expression  def remove\_whitespace\_multiple(text):  return re.sub('\s+',' ',text)  data['review'] = data['review'].apply(remove\_whitespace\_multiple)  #menghapus huruf yang berulang dalam teks  def remove\_repeated\_letters(text):  return re.sub(r"(.)\1+", r"\1", text)  data['review'] = data['review'].apply(remove\_repeated\_letters)  #menghapus kata-kata yang terdiri dari satu huruf (single character) dalam teks menggunakan regular expression  def remove\_singl\_char(text):  return re.sub(r"\b[a-zA-Z]\b", " ", text)  data['review'] = data['review'].apply(remove\_singl\_char)  #mengubah kalimat menjadi token  def word\_tokenize\_wrapper(text):  return word\_tokenize(text)  data['comments\_tokens'] = data['review'].apply(word\_tokenize\_wrapper)  print('Tokenizing Result : \n')  print(data['comments\_tokens'].head())  print('\n\n\n')  #Filtering  #Spelling Normalization  normalizad\_word = pd.read\_csv("https://raw.githubusercontent.com/meisaputri21/Indonesian-Twitter-Emotion-Dataset/master/kamus\_singkatan.csv", sep=";", header=None)  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]  data['comments\_normalized'] = data['comments\_tokens'].apply(normalized\_term)  normalizad\_word2 = pd.read\_csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vRQS3tlUL5EcxYqbbYzFLHmHaqm2npjY-DLyz0dzwMIcUVhfoVWKuhR52P9YCqbAyY9zCgT66JVutWA/pub?output=csv",header=None)  normalizad\_word\_dict2 = {}  for index, row in normalizad\_word2.iterrows():  if row[0] not in normalizad\_word\_dict2:  normalizad\_word\_dict2[row[0]] = row[1]  def normalized\_term2(document):  return [normalizad\_word\_dict2[term] if term in normalizad\_word\_dict2 else term for term in document]  data['comments\_normalized'] = data['comments\_normalized'].apply(normalized\_term2)  #Stopword Removal  list\_stopwords= pd.read\_csv('/content/drive/MyDrive/SKRIPSI/stopwordbahasa.csv')  list\_stopwords = set(list\_stopwords)  def stopwords\_removal(words):  return [word for word in words if word not in list\_stopwords]  data['comments\_tokens\_sw'] = data['comments\_normalized'].apply(stopwords\_removal)  #Stemming  # import Sastrawi package  ! pip install Sastrawi  from Sastrawi.Stemmer.StemmerFactory import StemmerFactory  # create stemmer  factory = StemmerFactory()  stemmer = factory.create\_stemmer()  # stemmed  def stemmed\_wrapper(term):  return stemmer.stem(term)  term\_dict = {}  for document in data['Comments']:  for term in document:  if term not in term\_dict:  term\_dict[term] = ' '  print(len(term\_dict))  print("------------------------")  for term in term\_dict:  term\_dict[term] = stemmed\_wrapper(term)  print(term,":" ,term\_dict[term])  print(term\_dict)  print("------------------------")  # apply stemmed term to dataframe  def get\_stemmed\_term(document):  return [term\_dict[term] for term in document]  data['comments\_tokens\_stemmed'] = data['Comments'].apply(get\_stemmed\_term)  print(data['comments\_tokens\_stemmed']) |

***WordCloud***

|  |
| --- |
| #Import Library untuk WordCloud  import pandas as pd  import matplotlib.pyplot as plt  %matplotlib inline  from wordcloud import WordCloud  text = " ".join(title for title in data["Ulasan\_clean"])  word\_cloud = WordCloud(collocations = False, background\_color = 'white',  width = 2048, height = 1080).generate(text)  #Menyimpan Gambar WordCloud  word\_cloud.to\_file('wordcloud.png')  #Menampilkan Hasil WordCloud  plt.imshow(word\_cloud, interpolation='bilinear')  plt.axis("off")  plt.show() |

***Word Embedding***

|  |
| --- |
| df\_preprocessed = data.drop(labels=['wordcloud','review','label','comments\_tokens', 'comments\_normalized','comments\_tokens\_sw','Comments',  'comments\_tokens\_stemmed'], axis=1)  import gzip  from urllib.request import urlopen  import numpy as np  file = gzip.open(urlopen('https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.id.300.vec.gz'))  vocab\_and\_vectors = {}  # put words as dict indexes and vectors as words values  for line in file:  values = line.split()  word = values [0].decode('utf-8')  vector = np.asarray(values[1:], dtype='float32')  vocab\_and\_vectors[word] = vector  # more imports  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.utils import to\_categorical  # how many features should the tokenizer extract  features = 500  tokenizer = Tokenizer(num\_words = features)  # fit the tokenizer on our text  texts = df\_preprocessed.Ulasan\_clean  tokenizer.fit\_on\_texts(texts)  # get all words that the tokenizer knows  word\_index = tokenizer.word\_index  print('Found %s unique tokens.' % len(word\_index))  # put the tokens in a matrix  X = tokenizer.texts\_to\_sequences(df\_preprocessed["Ulasan\_clean"].tolist())  X = pad\_sequences(X)  # prepare the labels  Y = df\_preprocessed["sentiment"]  from keras.utils.np\_utils import to\_categorical  print(texts)  embedding\_matrix = np.zeros((len(word\_index) + 1, 300))  for word, i in word\_index.items():  embedding\_vector = vocab\_and\_vectors.get(word)  # words that cannot be found will be set to 0  if embedding\_vector is not None:  embedding\_matrix[i] = embedding\_vector  # Print sample of tokenized text  sample\_tokenized = texts.sample(3)  for i, tokenized in sample\_tokenized.items():  print(tokenized)  print(X[i])  print('\n') |

**Pembagian Data**

|  |
| --- |
| # Pembagian train (pelatihan) dan testing  Y = data["sentiment"]  Y  # One hot encoding label  from keras.utils.np\_utils import to\_categorical  Y = to\_categorical(Y, num\_classes= 3)  Y  # split in train and test 80:20  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size= 0.2, stratify=Y, random\_state=14)  print(X\_train.shape, Y\_train.shape)  print(X\_test.shape, Y\_test.shape)  print(X\_train.shape)  print(X\_test.shape)  print(Y\_train.shape)  print(Y\_test.shape)  # grafik data training dan testing  sns.countplot(Y\_train,label='count')  plt.ylabel('Frekuensi', fontsize=12)  plt.xlabel('Label', fontsize=12)  plt.show()  sns.countplot(Y\_test,label='count')  plt.ylabel('Frekuensi', fontsize=12)  plt.xlabel('Label', fontsize=12)  plt.show() |

**Random Over Sampling**

|  |
| --- |
| from collections import Counter  Counter(y\_train.argmax(axis=1))  from imblearn.over\_sampling import RandomOverSampler  y\_train.argmax(axis=1)  # transform the dataset  oversample = RandomOverSampler(sampling\_strategy='not majority')  X\_over, y\_over = oversample.fit\_resample(X\_train, y\_train.argmax(axis=1))  # summarize the new class distribution  counter = Counter(y\_over)  print(counter)  sns.countplot(y\_over,label='count')  plt.ylabel('Frekuensi', fontsize=12)  plt.xlabel('Label', fontsize=12)  plt.show()  # # One hot encoding the label  from keras.utils.np\_utils import to\_categorical  y\_over = to\_categorical(y\_over, num\_classes=3)  y\_over  print(X\_train.shape)  print(X\_test.shape)  print(y\_train.shape)  print(y\_test.shape)  print(X\_smote.shape)  print(y\_smote.shape)  print(X\_over.shape)  print(y\_over.shape) |

**Pemodelan Bidirectional Gated Recurrent Unit (BiGRU)**

|  |
| --- |
| # Hyperparameters range  learning\_rate = [0.001, 0.01, 0.1]  dropout\_rate = [0.1, 0.2, 0.3]  use\_batch\_norm = [True, False]  batch\_size = [16, 32, 64]  neurons = [32, 64, 128]  dense\_dim = [6,8,16,32,64]  # Randomly select hyperparameters  import random  selected\_dropout\_rate = random.choice(dropout\_rate)  selected\_use\_batch\_norm = random.choice(use\_batch\_norm)  selected\_batch\_size = random.choice(batch\_size)  selected\_neurons = random.choice(neurons)  selected\_learning\_rate = random.choice(learning\_rate)  selected\_dense\_dim = random.choice(dense\_dim)  #Build Model  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Embedding, Bidirectional, GRU, Dropout, BatchNormalization, Dense, Flatten  from tensorflow.keras.callbacks import EarlyStopping  from tensorflow.keras.optimizers import Adam  def create\_model(dropout\_rate, use\_batch\_norm, batch\_size, neurons, learning\_rate):  model = Sequential()  model.add(Embedding(len(word\_index) + 1, EMBEDDING\_DIM, input\_length=MAX\_SEQUENCE\_LENGTH, weights=[embedding\_matrix], trainable=False))  model.add(Bidirectional(GRU(neurons, kernel\_regularizer=tf.keras.regularizers.l1\_l2(l1=0.01, l2=0.01))))  if use\_batch\_norm:  model.add(BatchNormalization())  model.add(Dropout(dropout\_rate))  model.add(Dense(selected\_dense\_dim, activation='relu'),  model.add(Dense(3, activation='softmax'))  optimizer = Adam(learning\_rate= selected\_learning\_rate)  model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=[['accuracy', 'val\_loss']])    return model  # Create model with selected hyperparameters  model = create\_model(selected\_learning\_rate,selected\_dropout\_rate, selected\_use\_batch\_norm, selected\_batch\_size, selected\_neurons)  # Define early stopping  early\_stopping = EarlyStopping(patience=3, monitor='val\_loss', mode='min', restore\_best\_weights=True)  # Train the model  HistoryBiGRU = model.fit(X\_over, y\_over, batch\_size=selected\_batch\_size, epochs=10, validation\_data= (X\_test, y\_test), validation\_steps=30, callbacks=[early\_stopping]) |

**Evaluasi Model**

|  |
| --- |
| # Predict classes for test data  y\_pred = model.predict(X\_test)  y\_pred\_classes = np.argmax(y\_pred, axis=1)  y\_true = np.argmax(y\_test, axis=1)  # Compute confusion matrix  from sklearn.metrics import confusion\_matrix  cm = confusion\_matrix(y\_true, y\_pred\_classes)  # Plot confusion matrix  plt.figure(figsize=(8, 6))  sns.heatmap(cm, annot=True, fmt='d', cbar=False)  plt.xlabel('Predicted Labels')  plt.ylabel('True Labels')  plt.title('Confusion Matrix')  plt.show() |