

Platinum Challenge

SENTIMENT ANALYSIS NEURAL NETWORK DAN LSTM

KELOMPOK 1



Binar Academy
Data Science

Analisis Sentimen

Analisis sentimen adalah proses mendapatkan data textual dengan mengekstrak informasi subjektif yang berguna untuk memahami sentimen sosial tentang suatu isu yang membantu dalam peningkatan bisnis dan quality control.

Twitter (X) adalah layanan jejaring sosial di mana pengguna dapat menulis, me-retweet, dan membaca postingan. Twitter adalah salah satu platform paling populer untuk mengekspresikan pendapat atau mengomentari isu-isu yang sedang berlangsung dari berbagai sumber.

Tujuan utama kami adalah mengklasifikasikan sentimen ke dalam berbagai kategori.

Mengapa Machine Learning?

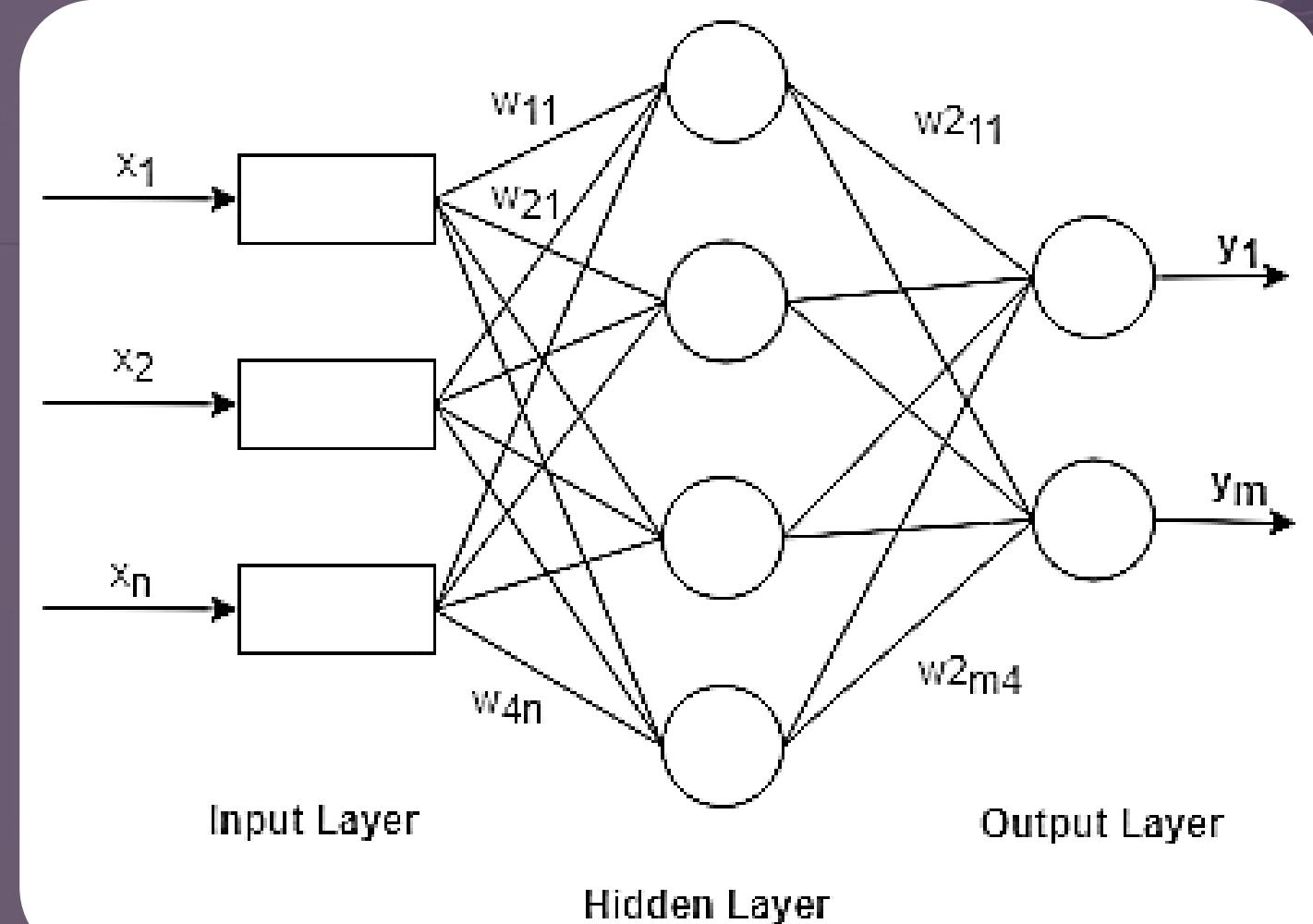
Machine learning merupakan bidang yang terus berkembang secara signifikan dalam beberapa tahun terakhir untuk memroses dan menganalisis data yang lebih besar dan rumit dengan waktu yang lebih singkat. Berbagai algoritme dan library yang memudahkan pengembangan model. Beberapa library populer yang sering digunakan adalah Scikit-Learn, Keras, TensorFlow, Pytorch, dll. Ada juga kasus dimana developer menggabungkan dua atau lebih dari library tersebut dengan tujuan untuk membuat model terbaik.

WHAT IS MLPCLASSIFIER?

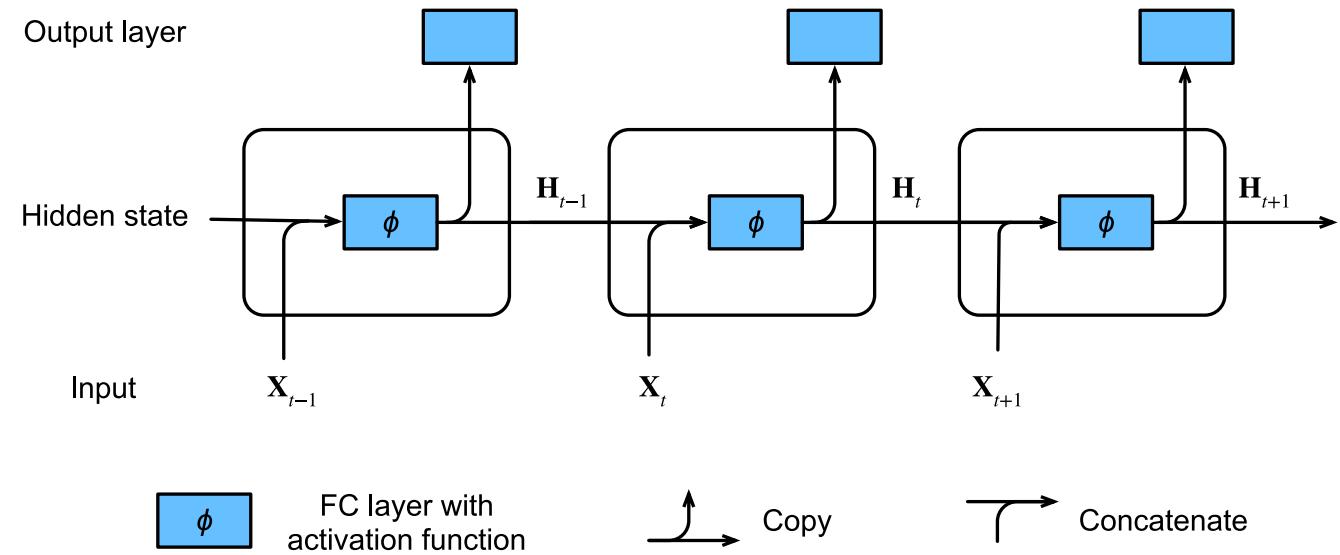
MLPClassifier merupakan salah satu algoritma neural network yang dapat digunakan untuk melakukan klasifikasi. Algoritma ini didasarkan pada arsitektur multilayer perceptron (MLP), yang merupakan jenis neural network feedforward.

ARCHITECTURE OF MLPCLASSIFIER

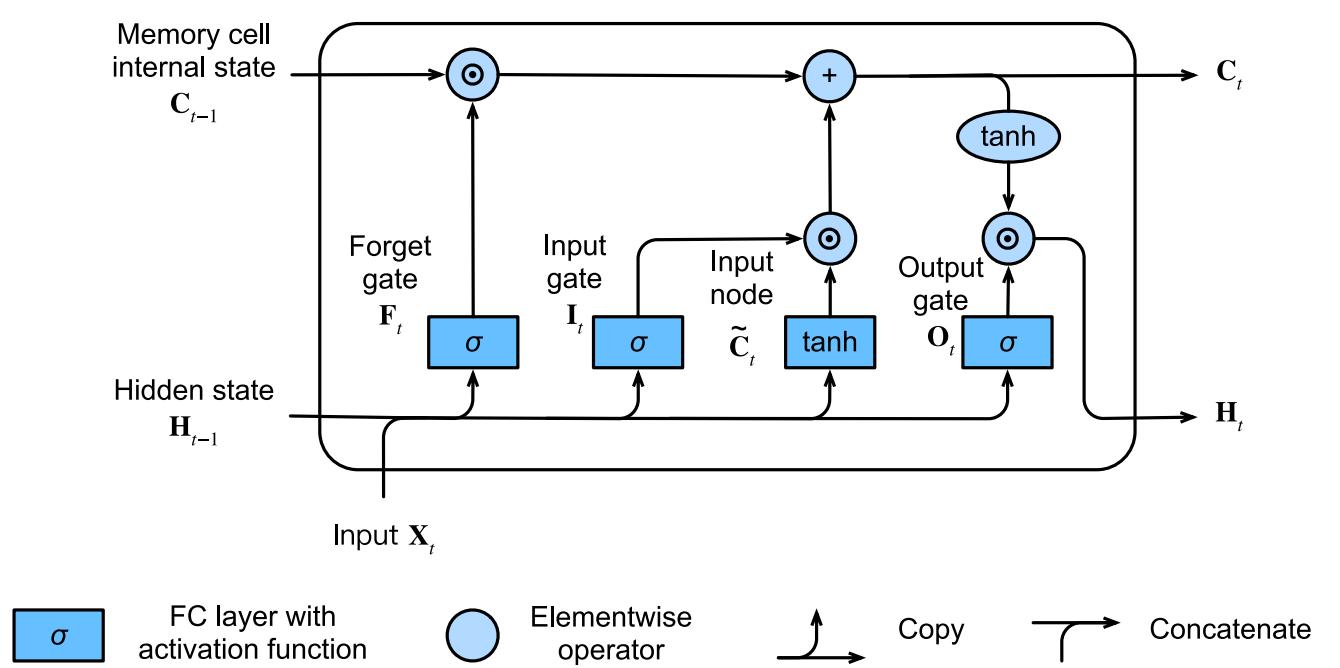
Algoritma MLPClassifier terdiri dari layer input, **satu atau lebih hidden layers**, dan **layer output**. **layer input bertanggung jawab untuk menerima input ke model**, yang kemudian diteruskan ke hidden layers. **Hidden layers** melakukan komputasi pada data input, dan **layer output menghasilkan output akhir dari model**.



Gambar 1: Struktrur MLP classifier
(Sumber: [MLP Architecture](#)).



Gambar 2: Arsitektur dari RNN Unit
(Sumber: [RNN Architecture](#))



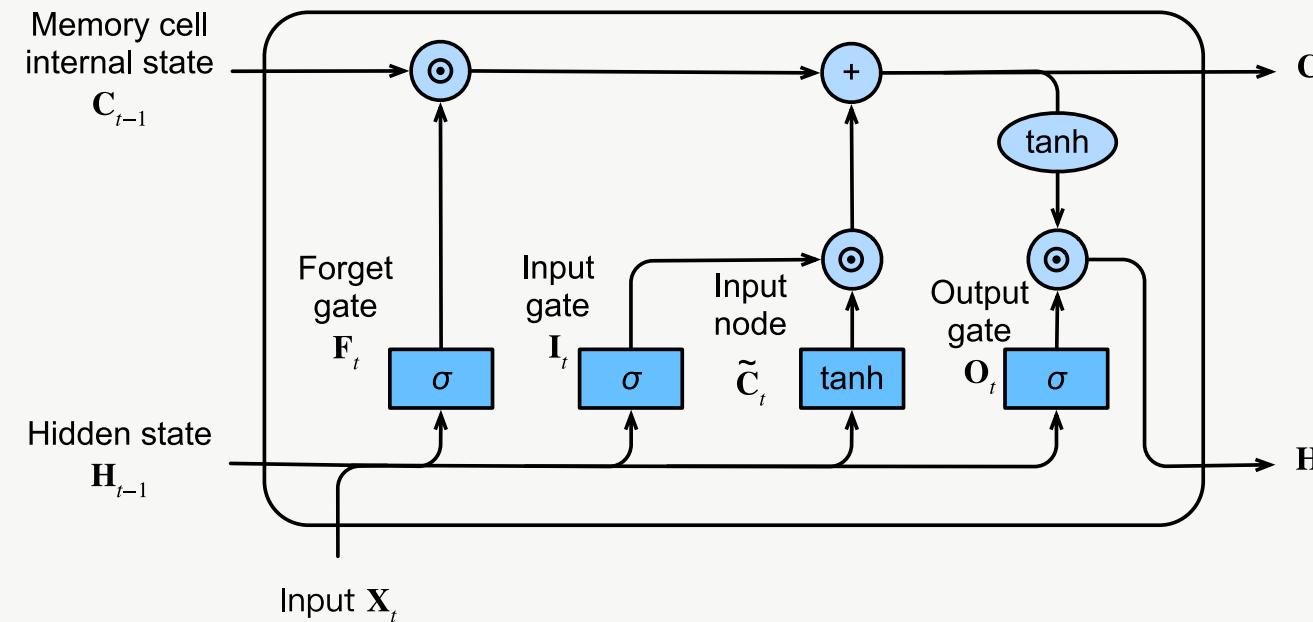
Gambar 3: Arsitektur dari LSTM Unit
(Sumber: [LSTM Architecture](#))

RNN DAN LSTM

Recurrent Neural Network (RNN) secara teoritis **mengklaim dapat mempertahankan long-term dependencies**, namun pada praktiknya mengalami kegagalan seperti yang dibahas oleh Bengio dkk. (1994).

Untuk mengatasi masalah ini, Hochreiter dkk (1997) mempresentasikan Long Short-Term Memory yang biasanya disebut LSTM. **LSTM menyerupai RNN, tetapi di sini setiap simpul berulang bisa digantikan oleh sel memori.**

Istilah “long short-term memory” berasal dari intuisi berikut. **RNN memiliki memori jangka panjang dalam bentuk bobot. Bobotnya berubah secara perlahan selama pelatihan.**



σ FC layer with activation function ○ Elementwise operator $\xrightarrow{\quad}$ Copy $\xrightarrow{\quad}$ Concatenate

$$\begin{aligned}
 I_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i), \\
 F_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f), \\
 O_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o), \\
 \tilde{C}_t &= \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c), \\
 C_t &= F_t \odot C_{t-1} + I_t \odot \tilde{C}_t.
 \end{aligned}$$

$$\mathbf{H}_t = O_t \odot \tanh(C_t).$$

```

import tensorflow as tf

class LSTMScratch(tf.Module):
  def __init__(self, num_inputs, num_hiddens, sigma=0.01):
    super().__init__()
    self.save_hyperparameters()

    init_weight = lambda *shape: tf.Variable(tf.random.normal(shape) * sigma)
    triple = lambda: (init_weight(num_inputs, num_hiddens),
                      init_weight(num_hiddens, num_hiddens),
                      tf.Variable(tf.zeros(num_hiddens)))

    self.W_xi, self.W_hi, self.b_i = triple() # Input gate
    self.W_xf, self.W_hf, self.b_f = triple() # Forget gate
    self.W_xo, self.W_ho, self.b_o = triple() # Output gate
    self.W_xc, self.W_hc, self.b_c = triple() # Input gate

  def forward(self, inputs, H_C=None):
    if H_C is None:
      # Initial state with shape: (batch_size, num_hiddens)
      H = tf.zeros((inputs.shape[1], self.num_hiddens))
      C = tf.zeros((inputs.shape[1], self.num_hiddens))
    else:
      H, C = H_C

    outputs = []
    for X in inputs:
      I = tf.sigmoid(tf.matmul(X, self.W_xi) +
                     tf.matmul(H, self.W_hi) + self.b_i)
      F = tf.sigmoid(tf.matmul(X, self.W_xf) +
                     tf.matmul(H, self.W_hf) + self.b_f)
      O = tf.sigmoid(tf.matmul(X, self.W_xo) +
                     tf.matmul(H, self.W_ho) + self.b_o)
      c_tilde = tf.tanh(tf.matmul(X, self.W_xc) +
                        tf.matmul(H, self.W_hc) + self.b_c)

      C = F * C + I * c_tilde
      H = O * tf.tanh(C)
      outputs.append(H)

    return outputs, (H, C)
  
```

DATA CLEANING

Pada tahapan ini data akan dibersihkan dengan beberapa tahapan



Lowercase & Remove Unnecessary Character

mengubah semua huruf menjadi **huruf kecil** (**lowercase**), menghilangkan **tanda baca** dan **karakter** yang tidak diperlukan menggunakan **regex**



Tokenization

mengubah **kalimat** menjadi **unit daftar (list) kata** contoh: ‘aku suka makan’ menjadi ‘aku’, ‘suka’, ‘makan’



Normalization

mengubah kata tertentu yang **tidak baku** menjadi **kata baku**, mengubah singkatan menjadi kata baku, proses ini diperlukan untuk **menghindari variasi kata** yang memiliki arti yang sama



Remove Stopword

Menghapus kata yang tidak memiliki tendensi tertentu atau **umum** dan **sering digunakan** seperti kata penghubung, kata tunjuk, dsb.



Drop Duplicated & Missing Value

Menghapus data yang memiliki **kesamaan value (duplikat)** dan data yang **tidak memiliki value (missing)**

LOWERCASE REMOVE UNNECESSARY CHARACTER TOKENIZATION

```
def Clean(text):
    #Lowercase for every word
    text = text.lower()

    #Clean Pattern
    #remove USER
    text = re.sub(r'user', ' ', text)
    #remove 'RT'
    text = re.sub(r'rt', ' ', text)
    #remove 'URL'
    text = re.sub(r'url', ' ', text)
    #remove HTTPS
    text = re.sub(r'https', ' ', text)
    #remove HTTP
    text = re.sub(r'http', ' ', text)
    #remove &
    text = re.sub(r'&', ' ', text)

#Clean_Unnecessary_Character
#remove \n or every word afte '\' with space
text = re.sub(r'\\\+[a-zA-Z0-9]+', ' ', text)
#remove text emoji
text = re.sub(r'^[a-zA-Z0-9\s]{2,}:[a-zA-Z0-9]{0,}', ' ', text)
#remove all unnecessary character
text = re.sub(r'^[0-9a-zA-Z\s]+', ' ', text)
#remove all number
text = re.sub(r'[0-9]+', ' ', text)
#remove extra space
text = re.sub(r' +', ' ', text)
#remove space at the start or the end of string
text = re.sub(r'^ +| +$', ' ', text)

return text

def tokenization(text):
    text = re.split('\W+', text)
    return text
```

NORMALIZATION AND REMOVE STOPWORD

```
kamus_alay = pd.read_csv(r"E:\BINAR\Binar-Gold-Challenge\Dataset\new_kamusalay.csv",
                         encoding = 'ISO-8859-1', header = None)
kamus_alay = kamus_alay.rename(columns={0: 'kata alay', 1: 'arti kata'})

#Create dictionary from kamus_alay
kamus_alay_dict = dict(zip(kamus_alay['kata alay'], kamus_alay['arti kata']))

#normalization function to convert every word tha contain 'kata alay' to 'arti kata'
def normalization(text):
    newlist = []
    for word in text:
        if word in kamus_alay_dict:
            text = kamus_alay_dict[word]
            newlist.append(text)
        else:
            text = word
            newlist.append(text)
    return newlist
```

```
stopword_list = ['yang', 'untuk', 'pada', 'ke', 'para', 'namun', 'menurut', 'antara',
                 'dia', 'dua', 'ia','ia', 'seperti', 'jika', 'sehingga', 'kembali', 'dan',
                 'ini', 'karena', 'kepada', 'oleh', 'saat', 'sementara', 'setelah', 'kami',
                 'sekitar', 'bagi', 'serta', 'di', 'dari', 'telah', 'sebagai', 'masih', 'hal',
                 'ketika', 'adalah', 'itu', 'dalam', 'bahwa', 'atau', 'kita', 'dengan', 'akan',
                 'juga', 'ada', 'mereka', 'sudah', 'saya', 'terhadap', 'secara', 'agar', 'lain',
                 'anda', 'begitu', 'mengapa', 'kenapa', 'yaitu', 'yakni', 'daripada', 'itulah',
                 'lagi', 'maka', 'tentang', 'demi', 'dimana', 'kemana', 'pula', 'sambil', 'sebelum',
                 'sesudah', 'supaya', 'guna', 'kah', 'pun', 'sampai', 'sedangkan', 'selagi', 'sementara',
                 'tetapi', 'apakah', 'kecuali', 'sebab', 'seolah', 'seraya', 'seterusnya', 'dsb', 'dst',
                 'dll', 'dahulu', 'dulunya', 'anu', 'demikian', 'mari', 'nanti', 'oh', 'ok', 'setiap',
                 'sesuatu', 'saja', 'toh', 'walau', 'amat', 'apalagi', 'dengan', 'bahwa', 'oleh']

stopword_list.extend(["yg", "dg", "rt", "dgn", "ny", "d", 'klo',
                      'kalo', 'amp', 'biar', 'bikin', 'bilang',
                      'gak', 'ga', 'krn', 'nya', 'nih', 'sih',
                      'si', 'tau', 'tdk', 'tuh', 'utk', 'ya',
                      'jd', 'jgn', 'sdh', 'aja', 'n', 't',
                      'nyg', 'hehe', 'pen', 'u', 'nan', 'loh', 'rt',
                      'gue', 'yah', 'kayak'])

stopword_list = set(stopword_list)

def remove_stopwords(text):
    text = [word for word in text if word not in stopword_list]
    return text
```

MLPCLASSIFIER

Pada tahapan ini data akan dibersihkan dengan beberapa tahapan

Cleaning & handling Imbalanced Data

proses membersihkan data dan melihat ratio data, apabila terdapat indikasi ratio data yang tidak seimbang maka perlu dilakukan proses menambah jumlah sample untuk menyeimbangkan jumlah data.

Split Data Train and Test

membagi data menjadi 80% Data Training dan 20% Data Testing

Feature Extraction (TF-IDF)

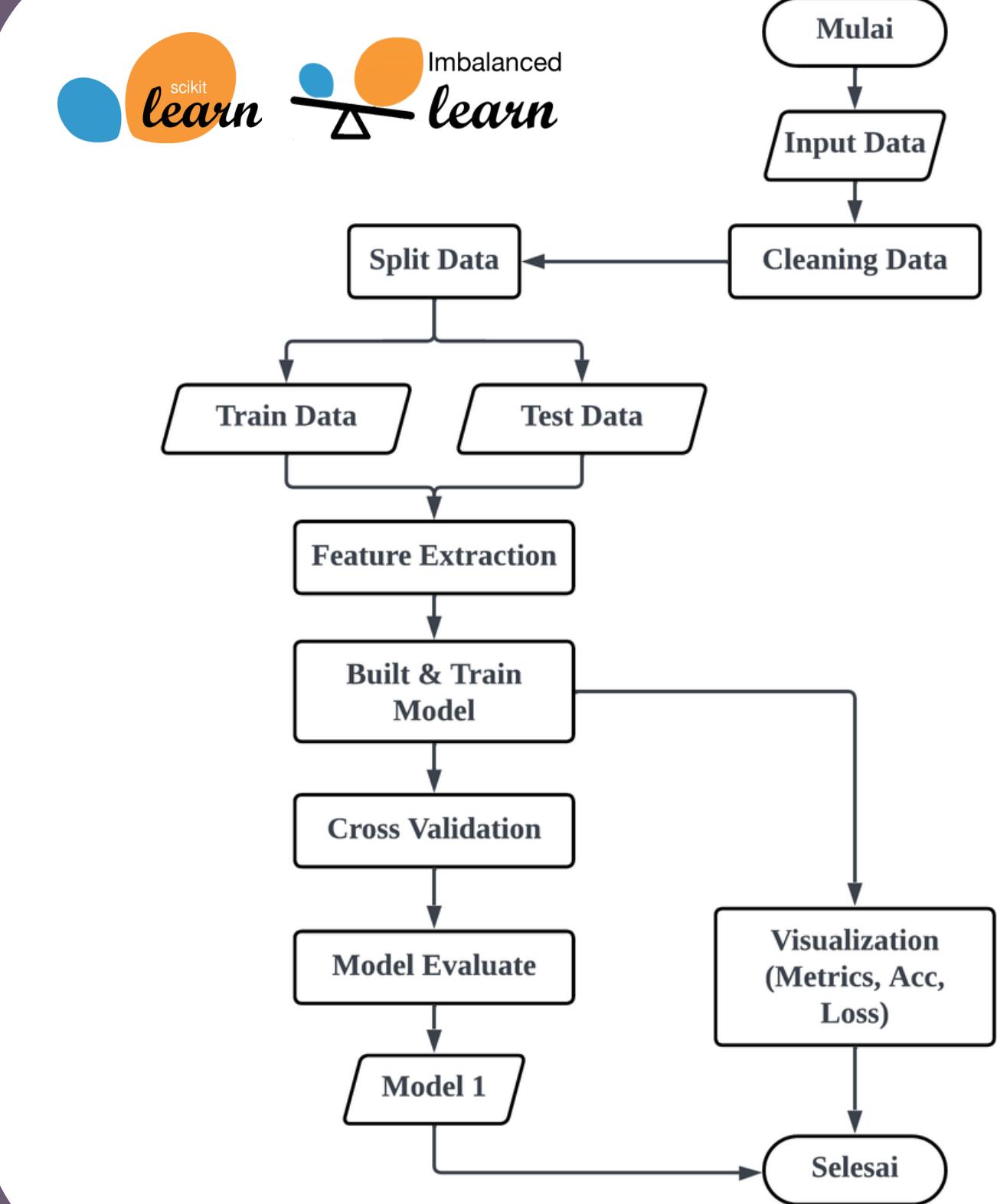
merubah data mentah menjadi data yang siap di training dan mudah dimengerti oleh sistem. Dalam kasus ini kata akan diubah menjadi numerical yang dapat dibaca oleh mesin (komputer)

Train, Evaluation, & Visualization Model

Train - proses untuk membangun machine learning yang bertujuan 'melatih' model yang diterapkan dalam sistem

Evaluation - untuk menghitung akurasi yang dihasilkan oleh model

Visualization - confusion matrix, loss curve, accuracy curve



Gambar 4: Tahapan Pembuatan Model menggunakan MLPClassifier

SPLIT DATA TRAIN & TEST FEATURE EXTRACTION (TF-IDF)

1

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_text,
                                                    labels,
                                                    test_size=0.2, random_state=42)

from collections import Counter
from imblearn.over_sampling import RandomOverSampler

ROS = RandomOverSampler()
X_train_ros, y_train_ros = ROS.fit_resample(X_train, y_train)

#Convert X_train_text into list form
Train_preprocessed = X_train_ros.text_clean.tolist()
#Convert X_test_text into list form
Test_preprocessed = X_test.text_clean.tolist()
```

2

```
from sklearn.feature_extraction.text import TfidfVectorizer
#Proses Feature Extraction
count_vect = TfidfVectorizer()
count_vect.fit_transform(Train_preprocessed)

#inisiasi value
X_train = count_vect.fit_transform(Train_preprocessed)
print("Feature Extraction For Train Data Has Successfully Completed")

X_test = count_vect.transform(Test_preprocessed)
print("Feature Extraction For Test Data Has Successfully Completed")
```

MODEL TRAINING & VISUALIZATION

1

```
#Loss curve
plt.plot(model_MLP.loss_curve_, label='loss curve')
plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('loss')
plt.legend()
plt.show()

#validation score
plt.plot(model_MLP.validation_scores_, label= 'validation score')
plt.title("Validation Score", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

```
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

model_MLP = MLPClassifier( early_stopping=True, validation_fraction=0.25)
model_MLP.fit(X_train, y_train)

print('Training selesai')
```

2

MODEL EVALUATION & VISUALIZATION

1

```
from sklearn.metrics import classification_report

#model evaluation
test = model_MLP.predict(X_test)
print("Testing selesai")

print(classification_report(y_test, test))
```

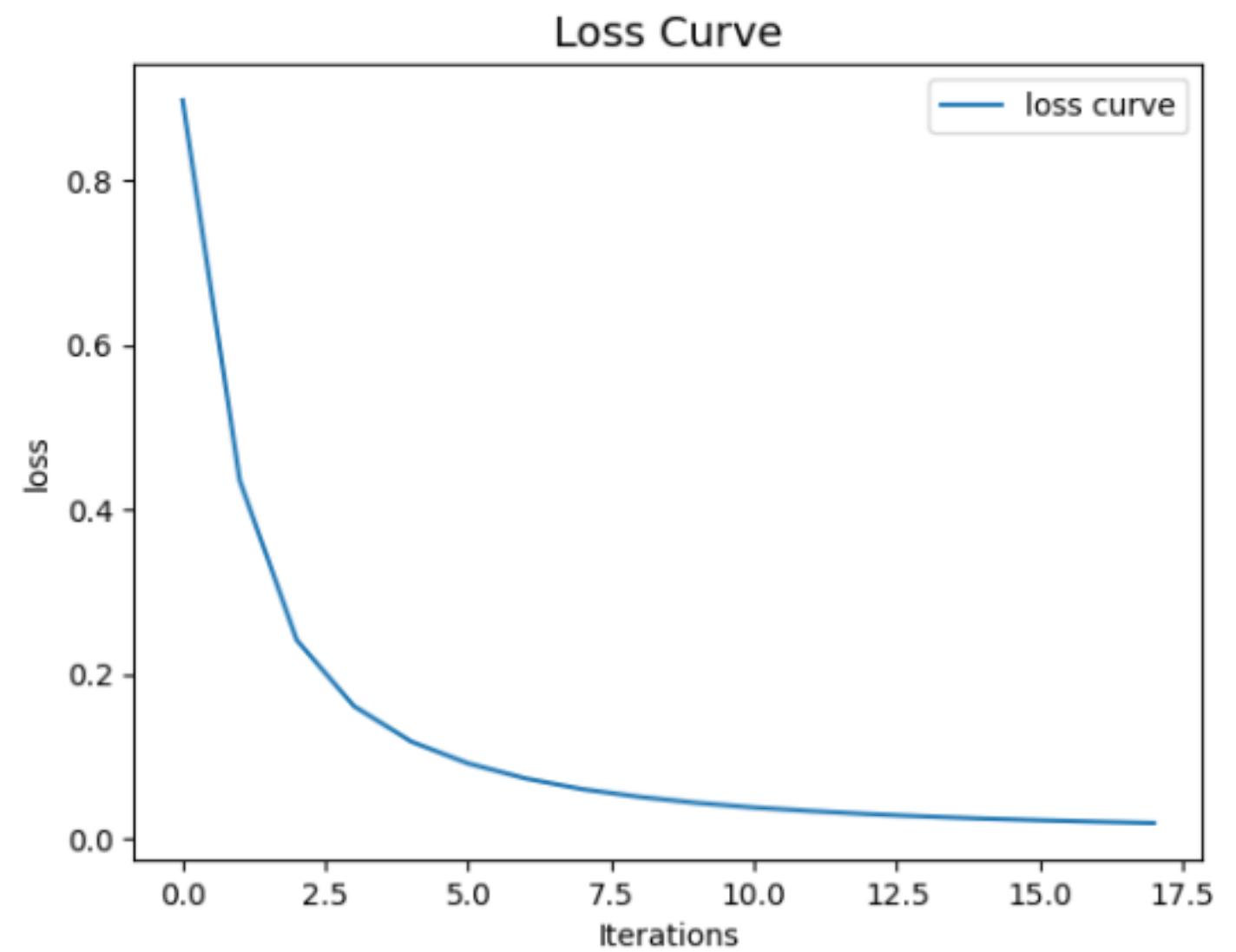
2

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

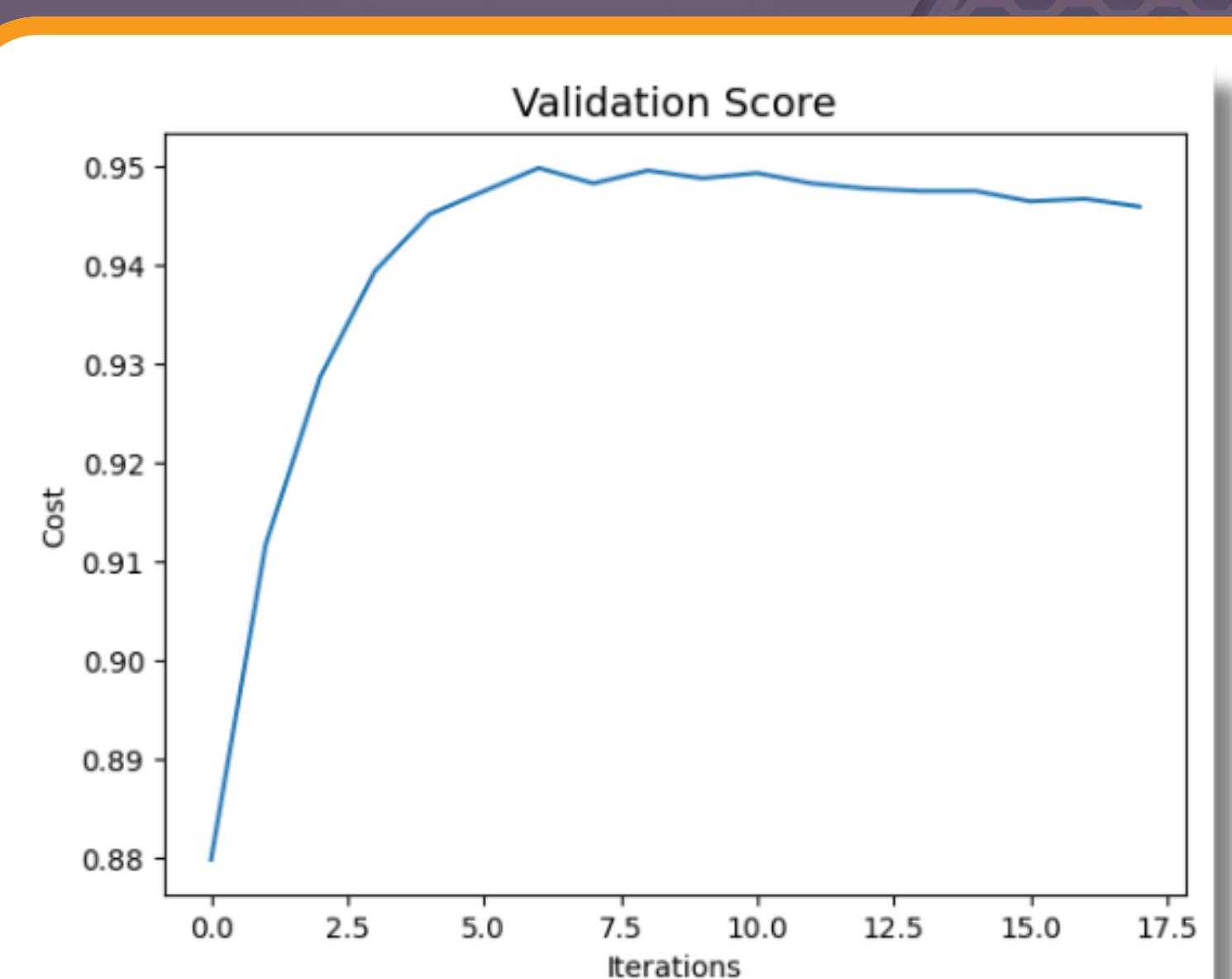
#Matrix confusion
cm = confusion_matrix(y_test, test, labels=model_MLP.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                             display_labels=model_MLP.classes_)

disp.plot()
plt.show()
```

LOSS CURVE & VALIDATION SCORE



Gambar 5: Kurva Loss dari Model MLPClassifier

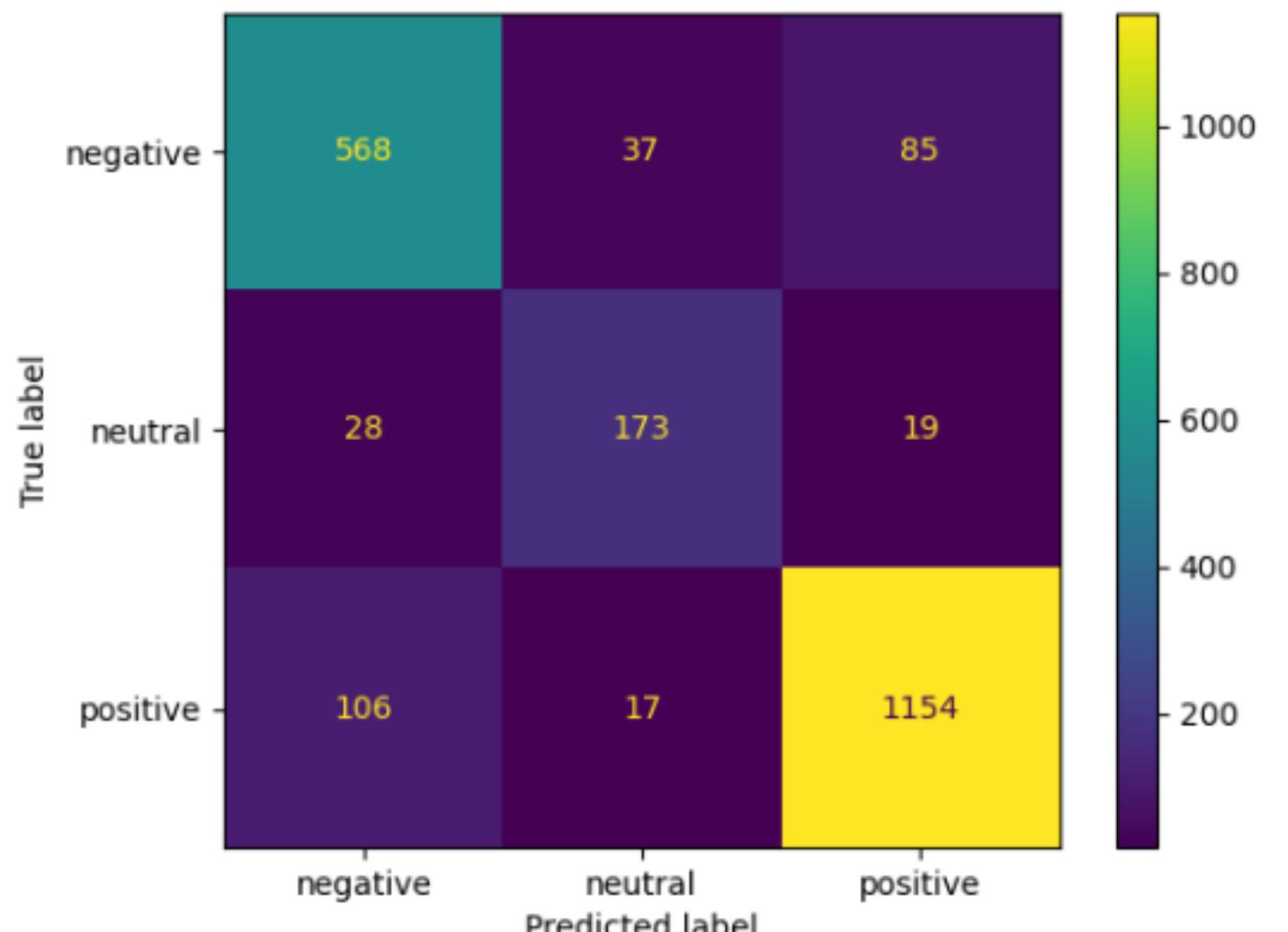


Gambar 6: Kurva Validation Score dari Model MLPClassifier

CLASSIFICATION REPORT & CONFUSION MATRIX

	precision	recall	f1-score	support
negative	0.81	0.82	0.82	690
neutral	0.76	0.79	0.77	220
positive	0.92	0.90	0.91	1277
accuracy			0.87	2187
macro avg	0.83	0.84	0.83	2187
weighted avg	0.87	0.87	0.87	2187

Gambar 7: Classification Report dari Model
MLPClassifier



Gambar 8: Confusion Matrix dari Hasil Evaluasi
Model MLPClassifier

LSTM

Pada tahapan ini data akan dibersihkan dengan beberapa tahapan

Cleaning & handling Imbalanced Data

proses membersihkan data dan melihat ratio data, apabila terdapat indikasi ratio data yang tidak seimbang maka perlu dilakukan proses menambah jumlah sample untuk menyeimbangkan jumlah data

Split Data Train and Test

membagi data menjadi 60% Data Training, 20% Data Validation, dan 20% Data Testing

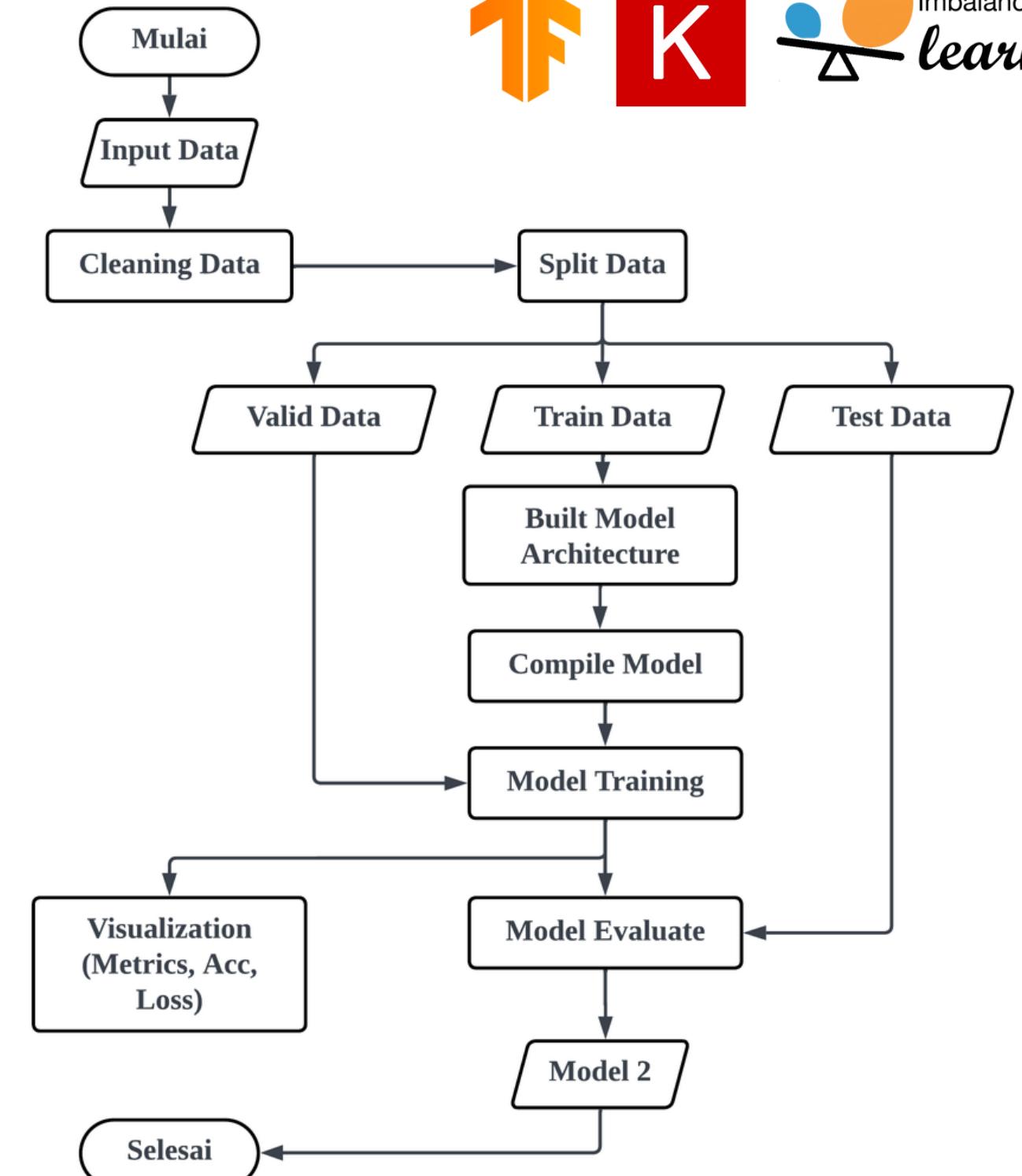
Architecture, Compile & Training Data

proses untuk membangun machine learning yang bertujuan 'melatih' model yang diterapkan dalam sistem

Visualization & Evaluation

Evaluation - untuk menghitung akurasi yang dihasilkan oleh model

Visualization - confusion matrix, loss curve, accuracy curve



Gambar 9: Tahapan Pembuatan Model menggunakan LSTM



Tensorflow & Keras

Tensorflow dan Keras adalah neural network yang populer untuk digunakan dalam model deep learning. Library ini didesain secara user-friendly, easy debugging, dan high performance sehingga cocok digunakan untuk training model dataset yang berjumlah besar.

Di dalam bagian challenge ini Tensorflow dan Keras akan digunakan mulai dari tahap Text Vectorization, Model Architecture, Model Compiling, hingga Model Training.

SPLIT DATA TRAIN, VALID & TEST TEXTVECTORIZATION LAYER

```
from tensorflow.keras.layers import TextVectorization
import numpy as np

max_features = 10000

#Create Layer
encoder = TextVectorization(max_tokens=max_features, split='whitespace', standardize='lower')
encoder.adapt(X_train_ros.text_clean.tolist())

vocab = np.array(encoder.get_vocabulary())
```

1

```
from sklearn.model_selection import train_test_split

X_train, X_ass, y_train, y_ass = train_test_split(data_text, labels,
                                                test_size=0.4, random_state=42)
X_test, X_valid, y_test, y_valid = train_test_split(X_ass, y_ass,
                                                test_size=0.5, random_state=42)

from collections import Counter
from imblearn.over_sampling import RandomOverSampler

ROS = RandomOverSampler()
X_train_ros, y_train_ros = ROS.fit_resample(X_train, y_train)
```

2

MODEL ARCHITECTURE

1

```
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM
from tensorflow.keras.utils import plot_model

length_input = len(encoder.get_vocabulary())
length_output = 100

model = Sequential()
model.add(encoder)
model.add(Embedding(input_dim=length_input,
                     output_dim=length_output))
model.add(LSTM(64, dropout=0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(3, activation='softmax'))
```

2

COMPILER & MODEL TRAINING

1

```
from tensorflow.keras import optimizers  
from tensorflow.keras.losses import CategoricalCrossentropy  
from tensorflow.keras.metrics import Accuracy  
  
model.compile(optimizer = optimizers.Adam(learning_rate=0.001),  
               loss= CategoricalCrossentropy(),  
               metrics= ['Accuracy'])  
  
print(model.summary())
```

2

```
from tensorflow.keras.callbacks import EarlyStopping  
  
my_callbacks = [  
    EarlyStopping(monitor='val_loss', patience = 5, verbose=1)]  
  
history = model.fit(X_train,  
                     y_train,  
                     batch_size = 32,  
                     epochs=20,  
                     validation_data=(X_valid, y_valid),  
                     verbose=1,  
                     callbacks=my_callbacks)
```

TRAINING & VALIDATION VISUALIZATION

1

```
def plot_history(history):
    plt.style.use('ggplot')

    acc = history.history['Accuracy']
    val_acc = history.history['val_Accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
```

2

```
plt.subplot(1, 2, 1)
plt.plot(x, acc, 'b', label='Training acc')
plt.plot(x, val_acc, 'r', label='Validation acc')
plt.title('Training & Validation Accuracy')
plt.ylim([0, 1])
plt.legend()
```

3

```
plt.subplot(1, 2, 2)
plt.plot(x, loss, 'b', label='Training loss')
plt.plot(x, val_loss, 'r', label='Validation loss')
plt.title('Training & Validation Loss')
plt.ylim([0, 1])
plt.legend()

plt.savefig('image.png')
```

MODEL EVALUATION & VISUALIZATION

1

```
from sklearn.metrics import classification_report

prediction = model.predict(X_test)
y_pred = prediction

matrix_test = classification_report(y_test.argmax(axis=1), y_pred.argmax(axis=1))
print('Testing Selesai')
print(matrix_test)
```

2

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

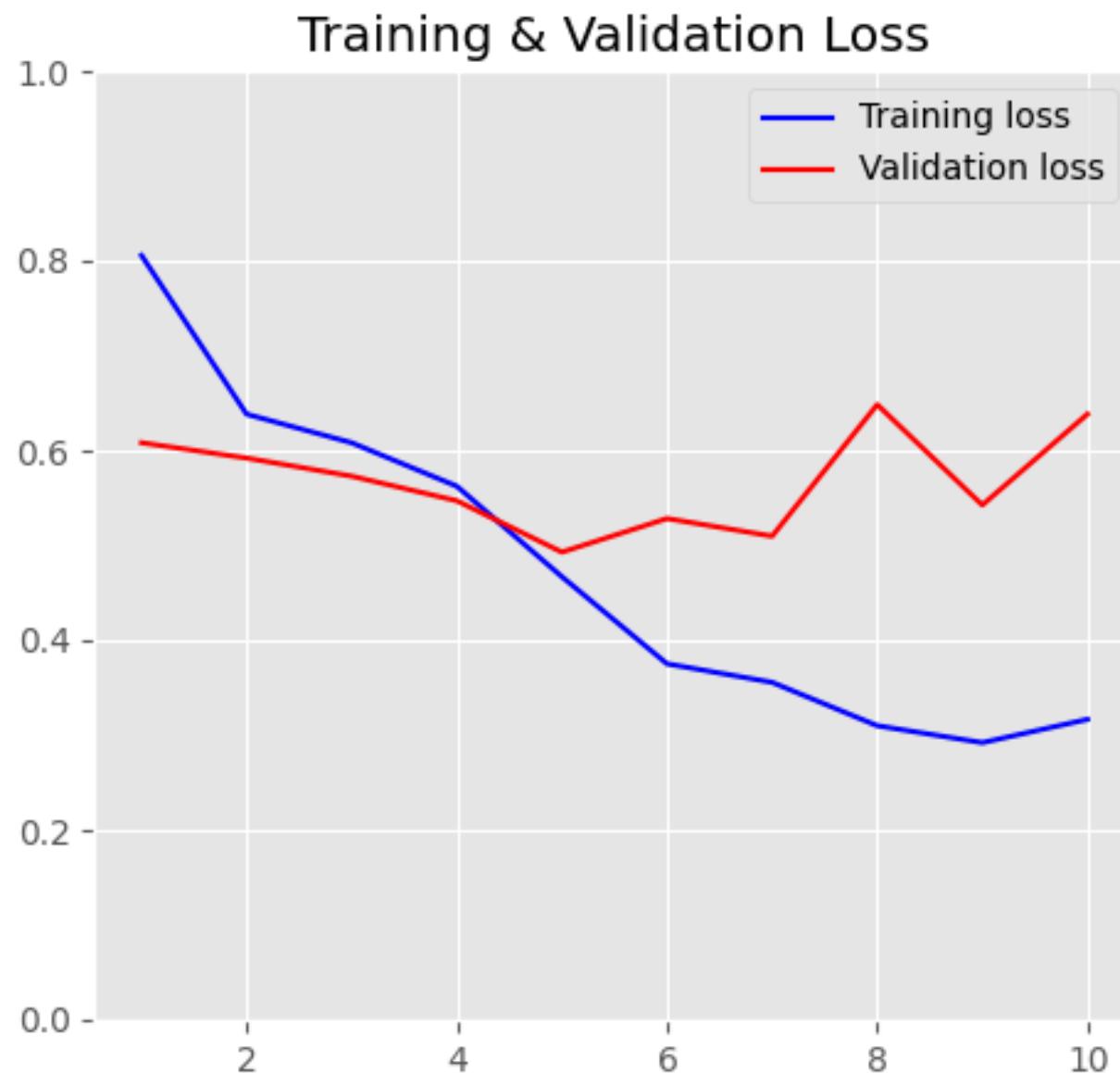
class_names = ['Negative', 'Neutral', 'Positive']

prediction = model.predict(X_test)
y_pred = prediction
cm = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))

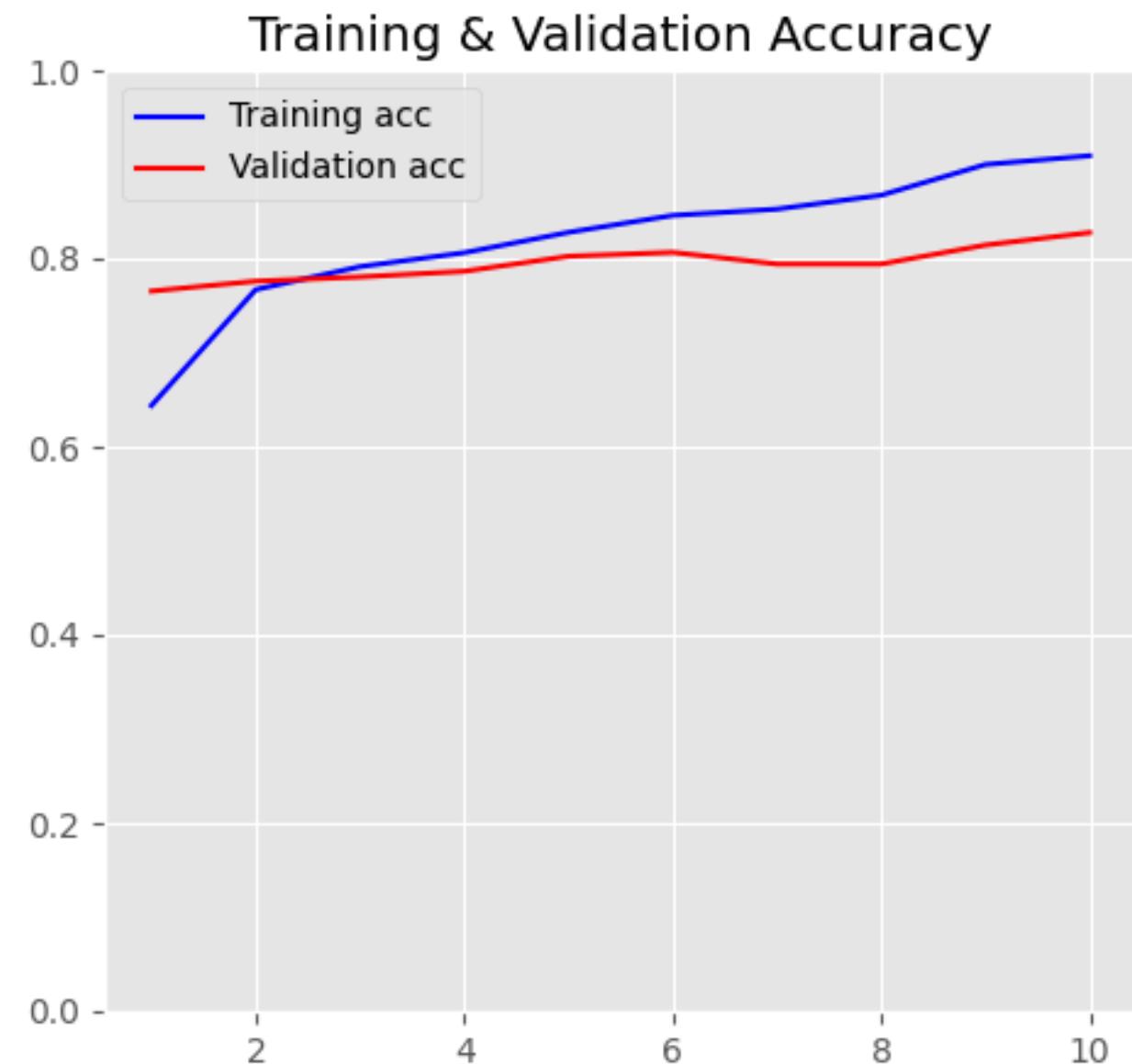
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                             display_labels=class_names)

disp.plot()
plt.grid(False)
plt.savefig('matrix_10.png')
plt.show()
```

ACCURACY & LOSS CURVE



Gambar 10: Kurva Akurasi Data Train dan Valid
dari Model LSTM

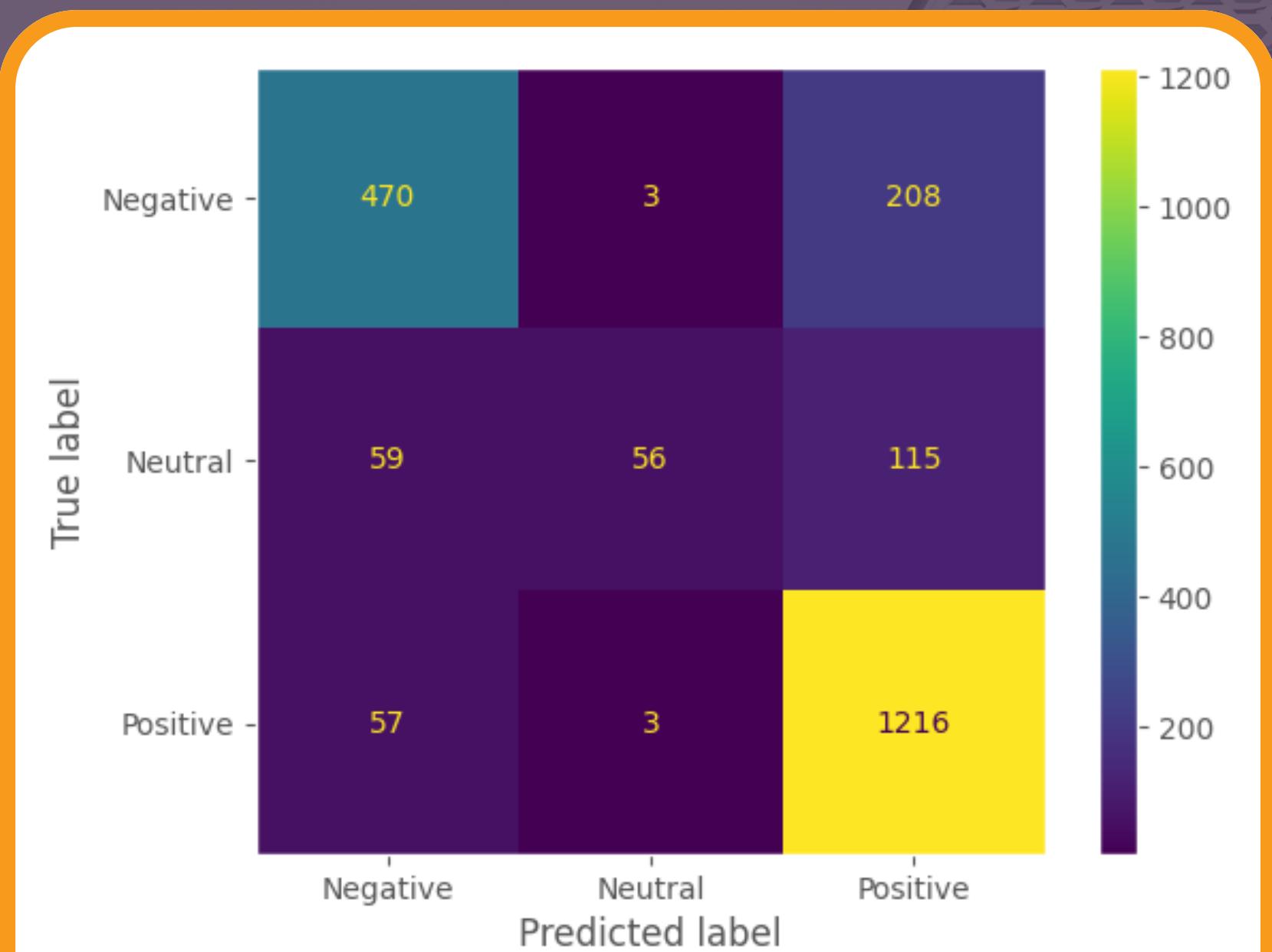


Gambar 11: Kurva Loss Data Train dan Valid
dari Model LSTM

CLASSIFICATION REPORT & CONFUSION MATRIX

	precision	recall	f1-score	support
0	0.80	0.69	0.74	681
1	0.90	0.24	0.38	230
2	0.79	0.95	0.86	1276
accuracy			0.80	2187
macro avg	0.83	0.63	0.66	2187
weighted avg	0.81	0.80	0.78	2187

Gambar 12: Classification Report dari Hasil Evaluasi Model LSTM



Gambar 13: Confusion Matrix dari Hasil Evaluasi Model LSTM

MODUL MLP & LSTM MODELL

```
1 import os
import json
import pickle
import numpy as np
import pandas as pd
from keras.models import load_model

# ----- Define Current Directory -----
current_directory = os.path.dirname(os.path.abspath(__file__))

# Vectorizer
file = open(current_directory + "\Model\\count_vect.p" , "rb")
vectorizer = pickle.load(file)
file.close()

# MLP Model
file = open(current_directory + "\Model\Model_MLP.p", "rb")
model_MLP = pickle.load(file)
file.close()

list_sentiment = ['negative', 'neutral', 'positive']

model_LSTM = load_model(current_directory + "\Model\Model_LSTM.keras")
```

```
def text_prediction(text: str, model: str):
    if model == 'MLP':
        vect_count = vectorizer.transform([text])
        sentiment = model_MLP.predict(vect_count)[0]

    elif model == 'LSTM':
        prediction = model_LSTM.predict([text])
        polarity = np.argmax(prediction[0])
        sentiment = list_sentiment[polarity]

    return sentiment
```

```
def CSV_prediction(df: pd.core.frame.DataFrame, model: str, json_out=True):
    if model == 'MLP':
        Data = df.Tweet.tolist()
        count_v = vectorizer.transform(Data)
        #prediction
        label = model_MLP.predict(count_v)

    elif model == 'LSTM':
        Data = np.array(df.Tweet)
        prediction = model_LSTM.predict(Data, batch_size=32)

        #prediction looping
        label = []
        for i in range(len(prediction)):
            polarity = np.argmax(prediction[i])
            sentiment = list_sentiment[polarity]
            label.append(sentiment)

    return label
```

API

ENDPOINT MLPCLASSIFIER

```
# ----- ENDPOINT MLP -----
@swag_from("docs/MLP_text_processing.yml", methods=['POST'])
@app.route('/1_MLP-text', methods=['POST'])
def text_processing_MLP():
    #request to input text
    text = request.form['text']

    #preprocessing text
    clean_text = dc.clean_text(text)
    #prediction
    sentiment = pred.text_prediction(clean_text, model='MLP')

    json_response = {
        'Sentiment': sentiment,
        'original_text': text,
        'clean_text': clean_text
    }

    response_data = jsonify(json_response)
    return response_data
```

```
@swag_from("docs/MLP_processing_file.yml", methods=['POST'])
@app.route('/2_MLP-file-processing', methods=['POST'])
def upload_processing_file_MLP():

    #CSV File
    #Upload single CSV File
    file = request.files['file']
    #read CSV file
    df_fileInput = pd.read_csv(file, encoding='latin1')

    #preprocessing
    #Filter column Tweet column
    df_tweet= df_fileInput[['Tweet']]
    #apply data cleaning function from DataCleaning
    df_tweet['Tweet'] = df_tweet['Tweet'].apply(dc.clean_data)
    #Drop duplicates
    df_tweet= df_tweet.drop_duplicates()
    #Drop Missing Value
    df_tweet = df_tweet.dropna()

    #Prediction
    json_pred = pred.csv_prediction(df_tweet, model='MLP', json_out=True)

    return json_pred
```

API

ENDPOINT LSTM

```
# ----- ENDPOINT LSTM -----
@swag_from("docs/LSTM_text_processing.yml", methods=['POST'])
@app.route('/3_LSTM-text', methods=['POST'])
def text_processing_LSTM():
    #request to input text
    text = request.form['text']

    #preprocessing text
    clean_text = dc.clean_text(text)
    #prediction
    sentiment = pred.text_prediction(clean_text, model='LSTM')

    json_response = {
        'Sentiment': sentiment,
        'original_text': text,
        'clean_text': clean_text
    }

    response_data = jsonify(json_response)
    return response_data
```

```
@swag_from("docs/LSTM_processing_file.yml", methods=['POST'])
@app.route('/4_LSTM-file-processing', methods=['POST'])
def upload_processing_file_LSTM():

    #CSV File
    #Upload single CSV File
    file = request.files['file']
    #read CSV file
    df_fileInput = pd.read_csv(file, encoding='latin1')

    #preprocessing
    #Filter column Tweet column
    df_tweet= df_fileInput[['Tweet']]
    #apply data cleaning fucntion from DataCleaning
    df_tweet['Tweet'] = df_tweet['Tweet'].apply(dc.clean_data)
    #Drop duplicates
    df_tweet= df_tweet.drop_duplicates()
    #Drop Missing Value
    df_tweet = df_tweet.dropna()

    #Prediction
    json_pred = pred.CSV_prediction(df_tweet, model='LSTM', json_out=True)

    return json_pred
```

HASIL PREDIKSI SENTIMEN

The screenshot shows the Swagger UI for a sentiment analysis API. The top navigation bar includes the Swagger logo, the URL '/docs.json', and a 'Explore' button. The main title is 'Dokumentasi API untuk Proses Sentiment Analysis 1.0.0'. Below the title, there is a brief introduction and a list of four main processing methods:

- 1. MLP - Text Processing
- 2. MLP - Process File CSV
- 3. LSTM - Text Processing
- 4. LSTM - Process File CSV

Each method is represented by a green button labeled 'POST' followed by the endpoint name. For example, '1. MLP - Text Processing' has a button labeled 'post_1_MLP_text'. At the bottom right of the interface, it says '[Powered by Flasgger 0.9.7.1]'. The entire screenshot is framed by a thick orange border.

Gambar 14: UI Dokumentasi API

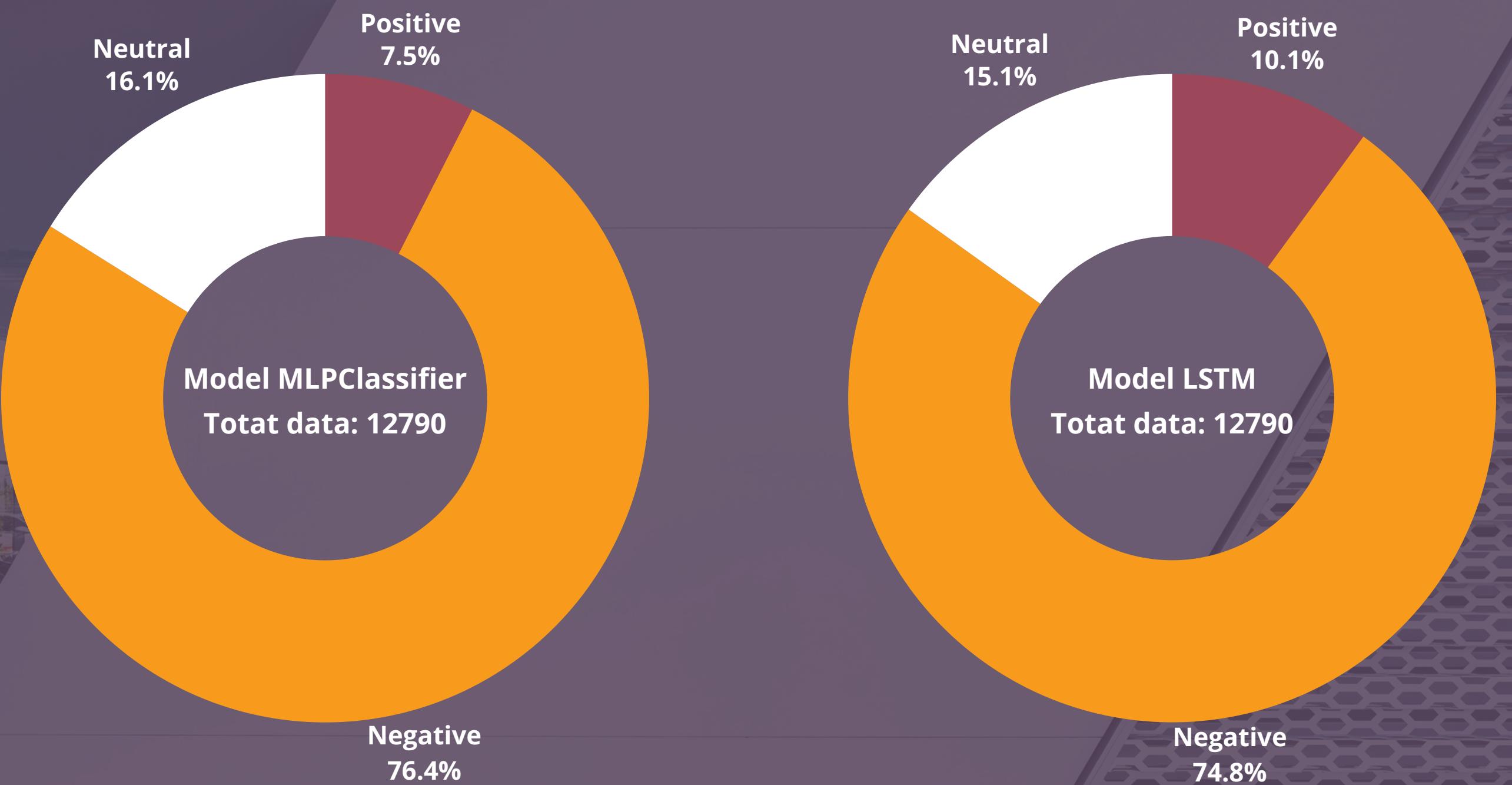
HASIL PREDIKSI SENTIMEN

```
{  
    "Tweet": "ajakan menolak berita hoaks sukseskan pilihan kepala daerah wilayah kota kediri",  
    "Label": "positive"  
},  
{  
    "Tweet": "presiden ri jokowi luar biasa kerja tulus ikhlas rakyatb rsamajkw",  
    "Label": "positive"  
},  

```

Gambar 15: Contoh Hasil Prediksi Sentiment
(diambil dari Response Body Swagger UI)

PERSENTASE PERBANDINGAN HASIL



Gambar 16: Perbandingan Hasil MLPClassifier dan LSTM

MODEL MLPCLASSIFIER

+

Sederhana, mudah digunakan, dan proses trainig cepat.

+

cocok digunakan sebagai basis model menggunakan dataset kecil sebelum dikembangkan model klasifikasi yang lain.

+

Sebagian besar dirancang untuk machine learning ‘tradisional’, sehingga cocok digunakan oleh pemula.

keterbatasan untuk membuat model learning dan neural networks yang lebih kompleks.

-

Keterbatasan performance saat menangani dataset dan tugas yang lebih kompleks dan berskala besar.

-

Model MLP memiliki suatu ‘batasan’ dimana dia hanya bisa merekam short-term memory tergantung dari bentuk input yang disiapkan.

-

MODEL LSTM

+

Flexibility dan scalability untuk membuat neural networks yang kompleks

+

Dukungan yang lebih luas untuk membuat model deep learning

+

LSTM memiliki model dengan pengelolaan memori yang lebih baik

Membutuhkan pemahaman mendalam konsep yang

-

Kurang cocok digunakan untuk dataset skala kecil

-

Proses training cukup rumit dengan waktu training yang lama

-

KESIMPULAN

MLP (SKLEARN)

1

Fokus pada tugas machine learning secara traditional

2

Efisien untuk dataset berukuran kecil hingga menengah

3

Keterbatasan untuk membuat model neural networks yang lebih kompleks

4

Ideal untuk pemula atau mereka yang ingin melakukan implementasi secara cepat

LSTM (TENSORFLOW)

1

Focus pada model neural network yang lebih kompleks dan deep learning

2

Lebih unggul dalam menangani data besar dan komputasi yang kompleks

3

Menyediakan kerangka kerja yang fleksibel untuk membangun neural networks yang kompleks

4

Lebih cocok digunakan apabila memiliki pengalaman/pemahaman yang lebih dalam tentang machine learning

REFERENSI

- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157–166. <https://doi.org/10.1109/72.279181>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Nova. (2023, March 8). A Comprehensive Guide to Using Scikit-Learn's MLPClassifier. AI TechTrend. <https://aitechtrend.com/a-comprehensive-guide-to-using-scikit-learns-mlpclassifier/>
- Srinivas, A. C. M. V., Satyanarayana, Ch., Divakar, Ch., & Sirisha, K. P. (2021). Sentiment Analysis using Neural Network and LSTM. *IOP Conference Series: Materials Science and Engineering*, 1074(1), 012007. <https://doi.org/10.1088/1757-899X/1074/1/012007>

TIM TERBAIK KAMI



Ahmad Fauzi



Prasa Fakhriyah
Mumtaz



Fachry Ramadhan
Wachdin