LAPORAN UJIAN AKHIR SEMESTER KELOMPOK 1

"Progress 1 Cek Missing Value, MinMaxScaler, Seleksi Fitur (Chi-Square), SMOTE, dan Implementasi Vanilla LSTM"

MATA KULIAH MACHINE LEARNING PRAKTIKUM C-7

Disusun Oleh :

434231020	**Faizatun Ni'mah**
434231021	**Seri Muliani Lubis**
434231048	**Surya Dwi Satria**
434231112	**Festiana Ramaya**

D4 Teknik Informatika

FAKULTAS VOKASI

UNIVERSITAS AIRLANGGA

2025

A. Pendahuluan

Laporan ini menyajikan implementasi dan evaluasi model *Deep Learning* Vanilla Long Short-Term Memory (LSTM) untuk klasifikasi teks pada dataset AG News. Klasifikasi teks merupakan tugas fundamental dalam *Natural Language Processing* (NLP) yang bertujuan mengelompokkan dokumen teks ke dalam kategori-kategori yang telah ditentukan.

Pipeline yang diterapkan meliputi transformasi fitur teks menggunakan TF-IDF, serangkaian *preprocessing* untuk normalisasi dan penanganan data tidak seimbang (menggunakan SMOTE), serta pelatihan model LSTM. Dataset yang digunakan adalah sampel AG News yang terdiri dari 25.000 data latih dan 2.000 data uji, yang dikelompokkan ke dalam empat kelas berita.

B. Dasar Teori

1. TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF adalah teknik statistik yang digunakan untuk merefleksikan seberapa penting suatu kata terhadap sebuah dokumen dalam koleksi dokumen. Dalam implementasi ini, TfidfVectorizer digunakan dengan ngram_range=(1, 2) untuk mempertimbangkan unigram dan bigram, serta max_features=4000 untuk membatasi dimensi fitur.

2. MinMaxScaler

MinMaxScaler adalah teknik *data scaling* yang mengubah nilai fitur menjadi rentang tertentu (dalam kasus ini [0.0, 1.0]). Penskalaan ini krusial untuk model berbasis gradien seperti *Deep Learning* agar proses *training* lebih stabil dan cepat.

3. Chi-square (Feature Selection)

Metode seleksi fitur Chi-square digunakan untuk mengukur ketergantungan antara fitur (kata) dan kelas target. Fitur dengan nilai χ^2 tertinggi dipilih (sebanyak $k=1500$) karena dianggap paling relevan dan diskriminatif terhadap kelas.

4. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE adalah teknik *oversampling* yang digunakan untuk menyeimbangkan data set yang memiliki distribusi kelas tidak seimbang (*imbalanced*). SMOTE bekerja dengan menghasilkan sampel sintetis baru di sekitar sampel minoritas.

5. Vanilla LSTM (Long Short-Term Memory)

LSTM adalah jenis Recurrent Neural Network (RNN) yang dirancang untuk mengatasi masalah vanishing gradient dan exploding gradient yang umum terjadi pada RNN standar. LSTM sangat efektif dalam memproses data sekuensial atau data yang memiliki ketergantungan jangka panjang, seperti teks. Arsitektur yang digunakan adalah stacked LSTM dengan dua layer LSTM (128 dan 64 unit).

C. Tujuan

Tujuan dari proyek ini adalah:

1. Mengimplementasikan pipeline lengkap klasifikasi teks mulai dari vectorization hingga pemodelan Deep Learning.
2. Membangun model Vanilla LSTM untuk klasifikasi multi-kelas pada dataset AG News.
3. Melakukan evaluasi kinerja model menggunakan metrik standar (Akurasi, Presisi, Recall, AUC/ROC) serta waktu komputasi.
4. Menganalisis dampak dari penerapan SMOTE dalam penanganan imbalanced data.

D. Dataset

Link Dataset : https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

E. Langkah - Langkah

1. Inisialisai Library

```
import os
import sys
import time
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
import random
random.seed(42)
np.random.seed(42)
print("Inisialisasi environment selesai.")
```

Penjelasan : Langkah awal ini berfokus pada penyiapan lingkungan kerja dengan mengimpor semua pustaka Python yang diperlukan untuk implementasi pipeline klasifikasi teks, mulai dari manipulasi data hingga pemodelan Deep Learning dan evaluasi.

2. Class SuppressStderr

```
class SuppressStderr:
    def __init__(self):
        self.null_fds = [os.open(os.devnull, os.O_RDWR) for _ in range(2)]
        self.save_fds = [os.dup(1), os.dup(2)]

    def __enter__(self):
        os.dup2(self.null_fds[0], 1)
        os.dup2(self.null_fds[1], 2)

    def __exit__(self, *_):
        os.dup2(self.save_fds[0], 1)
        os.dup2(self.save_fds[1], 2)
        for fd in self.null_fds + self.save_fds:
            os.close(fd)
```

Penjelasan Kode ini mendefinisikan sebuah kelas utilitas bernama SuppressStderr.
Tujuan utama dari kelas ini adalah untuk meredam atau menekan output log yang biasanya ditampilkan ke Standard Error (stderr) selama eksekusi program, terutama spam log dari framework seperti TensorFlow atau CUDA yang sering muncul saat model diinisialisasi atau dilatih.

3. Pemuatan Data dan Vektorisasi Fitur (TF-IDF)

```
from sklearn.feature_extraction.text import TfidfVectorizer

with SuppressStderr():
    import tensorflow as tf
    tf.random.set_seed(42)  # seed untuk TensorFlow

print("Memuat dataset AG News...")

train_df = pd.read_csv("../dataset/train.csv")
test_df = pd.read_csv("../dataset/test.csv")

train_df.columns = ["Class Index", "Title", "Description"]
test_df.columns = ["Class Index", "Title", "Description"]

# Gabungkan Title + Description → text
train_df["text"] = train_df["Title"] + " " + train_df["Description"]
test_df["text"] = test_df["Title"] + " " + test_df["Description"]

# Sampling agar training tidak terlalu berat
train_df = train_df.sample(25000, random_state=42)
test_df = test_df.sample(2000, random_state=42)

X_train_raw = train_df["text"].values
y_train_raw = train_df["Class Index"].values - 1  # kelas: 0..3
X_test_raw = test_df["text"].values
y_test_raw = test_df["Class Index"].values - 1

# TF-IDF dengan n-gram (1,2)
vectorizer = TfidfVectorizer(
    max_features=4000,
    stop_words="english",
    ngram_range=(1, 2)
)

X_train_vec = vectorizer.fit_transform(X_train_raw).toarray()
X_test_vec = vectorizer.transform(X_test_raw).toarray()

print("TF-IDF selesai.")
print("Shape X_train_vec:", X_train_vec.shape)
print("Shape X_test_vec :", X_test_vec.shape)
```

Penjelasan : Langkah ini memulai pemrosesan data dengan memuat dataset klasifikasi teks AG News dari file CSV (train.csv dan test.csv). Setelah data dimuat ke dalam DataFrame Pandas, kolom diubah namanya menjadi standar ("Class Index", "Title", "Description") dan konten teks yang relevan digabungkan: kolom "Title" dan "Description" digabungkan menjadi satu kolom "text" yang akan menjadi fitur input. Untuk menghemat waktu dan sumber daya komputasi selama training, dilakukan sampling pada data: 25.000 sampel diambil untuk data latih dan 2.000 sampel untuk data uji, semuanya dengan random_state=42 untuk konsistensi.

\

4. Membuat Fungsi Yang akan di Eksekusi

```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.over_sampling import SMOTE

def cek_missing_value(X, name="X"):
    total_nan = np.isnan(X).sum()
    print(f"[1] Cek Missing Value - {name}: Total NaN = {total_nan}")
    return X

def fitur_transformasi_scaler(X_train, X_test):
    scaler = MinMaxScaler()
    X_train_sc = scaler.fit_transform(X_train)
    X_test_sc = scaler.transform(X_test)
    print("[2] MinMaxScaler - selesai.")
    print("    X_train_sc range: [", X_train_sc.min(), ",", X_train_sc.max(), "]")
    return X_train_sc, X_test_sc

def fitur_seleksi_chisquare(X_train, y_train, X_test, k=1500):
    selector = SelectKBest(score_func=chi2, k=k)
    X_train_fs = selector.fit_transform(X_train, y_train)
    X_test_fs = selector.transform(X_test)
    print(f"[3] Chi-square - selesai. Jumlah fitur terpilih: {k}")
    print("    Shape X_train_fs:", X_train_fs.shape)
    print("    Shape X_test_fs :", X_test_fs.shape)
    return X_train_fs, X_test_fs

def handle_imbalanced_smote(X_train, y_train, random_state=42):
    sm = SMOTE(random_state=random_state)
    X_res, y_res = sm.fit_resample(X_train, y_train)
    unique_cls, counts = np.unique(y_res, return_counts=True)
    print("[4] SMOTE - selesai.")
```

Penjelasan :

1. cek_missing_value(X, name="X"): Fungsi diagnostik sederhana untuk memverifikasi kebersihan data, menghitung dan mencetak total nilai yang hilang (np.isnan(X).sum()) dalam suatu array fitur.
2. fitur_transformasi_scaler(X_train, X_test): Fungsi ini menerapkan normalisasi fitur menggunakan MinMaxScaler. Penskalaan di-fit hanya pada data latih (X_train) untuk menghindari data leakage, kemudian di-transform pada data latih dan data uji, memastikan semua nilai fitur berada dalam rentang [0.0, 1.0].
3. fitur_seleksi_chisquare(X_train, y_train, X_test, k=1500): Fungsi ini bertanggung jawab untuk reduksi dimensi dan seleksi fitur yang paling informatif. Metode SelectKBest digunakan bersama dengan fungsi skor chi square untuk memilih $k=1500$ fitur terbaik yang paling berkorelasi dengan kelas target.
4. handle_imbalanced_smote(X_train, y_train, random_state=42): Fungsi ini menangani masalah ketidakseimbangan kelas pada data latih menggunakan SMOTE (Synthetic Minority Over-sampling Technique). SMOTE menyeimbangkan distribusi kelas dengan membuat sampel sintetis dari kelas minoritas, menghasilkan data fitur (X_res) dan label (y_res) latih yang seimbang sempurna, yang sangat penting untuk melatih model klasifikasi multi-kelas secara adil.

5. Fungsi Model

```python
def build_vanilla_lstm_model(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dropout(0.4))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(4, activation="softmax"))

    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(
        optimizer=opt,
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model
```

Penjelasan :

1. Input Layer: Menentukan bentuk input data yang telah di-reshape (misalnya, (1, 1500)).
2. Stacked LSTM: Model menggunakan dua layer LSTM berurutan:
    - Layer pertama memiliki 128 unit dengan return_sequences=True, yang berarti output-nya masih berupa urutan data untuk diteruskan ke layer berikutnya.
    - Layer kedua memiliki 64 unit dengan return_sequences=False, yang mengembalikan hanya output terakhir (vektor fitur) untuk klasifikasi.
3. Dropout: Setelah setiap layer LSTM, diterapkan layer Dropout(0.4). Ini adalah teknik regularisasi untuk mencegah overfitting dengan secara acak mengabaikan 40% neuron selama training.
4. Dense Layers: Diikuti oleh Dense Layer 64 unit dengan aktivasi ReLU, dan diakhiri dengan Dense Output Layer 4 unit (sesuai jumlah kelas) dengan aktivasi Softmax untuk menghasilkan probabilitas kelas.
5. Kompilasi Model: Model dikompilasi menggunakan optimizer Adam dengan learning rate 0.001, loss function categorical_crossentropy (standar untuk klasifikasi multi-kelas), dan metrik accuracy.

6. Fungsi Evaluasi

```python
def evaluasi_hasil(model, X_test, y_test_labels, train_time):
    start_test = time.time()
    y_prob = model.predict(X_test, verbose=0)
    test_time = time.time() - start_test

    y_pred = np.argmax(y_prob, axis=1)

    acc = accuracy_score(y_test_labels, y_pred)
    prec = precision_score(y_test_labels, y_pred, average="weighted", zero_division=0)
    rec = recall_score(y_test_labels, y_pred, average="weighted", zero_division=0)
    auc = roc_auc_score(
        to_categorical(y_test_labels, 4),
        y_prob,
        multi_class="ovr"
    )

    hasil = {
        "Akurasi": acc,
        "Presisi": prec,
        "Recall": rec,
        "AUC/ROC": auc,
        "Waktu Training (s)": train_time,
        "Waktu Testing (s)": test_time
    }

    print("\n=== HASIL EVALUASI MODEL (Vanilla LSTM + SMOTE) ===")
    print(f"Akurasi : {hasil['Akurasi']:.4f}")
    print(f"Presisi : {hasil['Presisi']:.4f}")
    print(f"Recall  : {hasil['Recall']:.4f}")
    print(f"AUC/ROC : {hasil['AUC/ROC']:.4f}")
    print(f"Waktu Training (s): {hasil['Waktu Training (s)']:.2f}")
    print(f"Waktu Testing  (s): {hasil['Waktu Testing (s)']:.4f}")

    return hasil
```

Penjelasan :

Fungsi ini bertanggung jawab untuk menguji model yang sudah dilatih dan menghitung semua metrik kinerja yang diminta.

1. Prediksi dan Waktu Testing: Fungsi ini menghitung waktu yang diperlukan untuk membuat prediksi (test_time) pada data uji.
2. Metrik: Hasil prediksi diubah menjadi label kelas (y_pred) dan dibandingkan dengan label sebenarnya (y_test_labels) untuk menghitung metrik utama:
    - Akurasi (accuracy_score).
    - Presisi (precision_score, average="weighted")
    - Recall (recall_score, average="weighted").
    - AUC/ROC (roc_auc_score, multi_class="ovr").

Output: Fungsi ini menampilkan dan mengembalikan kamus berisi semua metrik tersebut, termasuk waktu komputasi (train_time yang dimasukkan sebagai parameter) dan test_time.

7. Eksekusi Pipepline

```
final_results = {}

print("\n====================================")
print(" PIPELINE: TF-IDF → PREPROCESSING → LSTM ")
print("====================================")

# 1. Cek Missing Value
X1_train = cek_missing_value(X_train_vec, name="X_train_vec")
X1_test  = cek_missing_value(X_test_vec,  name="X_test_vec")

# 2. Transformasi --> MinMaxScaler
X1_train_sc, X1_test_sc = fitur_transformasi_scaler(X1_train, X1_test)

# 3. Seleksi Fitur (Chi-square)
X1_train_fs, X1_test_fs = fitur_seleksi_chisquare(
    X1_train_sc,
    y_train_raw,
    X1_test_sc,
    k=1500
)

# 4. Imbalanced Data (SMOTE)
X1_train_sm, y1_train_sm = handle_imbalanced_smote(
    X1_train_fs,
    y_train_raw,
    random_state=42
)

# Siapkan data untuk LSTM
y1_train_cat = to_categorical(y1_train_sm, 4)
X1_train_lstm = X1_train_sm.reshape((X1_train_sm.shape[0], 1, X1_train_sm.shape[1]))
X1_test_lstm  = X1_test_fs.reshape((X1_test_fs.shape[0], 1, X1_test_fs.shape[1]))

print("\n[Info] Shape untuk LSTM:")
print("      X1_train_lstm:", X1_train_lstm.shape)
print("      X1_test_lstm :", X1_test_lstm.shape)
```


```
# Siapkan data untuk LSTM
y1_train_cat = to_categorical(y1_train_sm, 4)
X1_train_lstm = X1_train_sm.reshape((X1_train_sm.shape[0], 1, X1_train_sm.shape[1]))
X1_test_lstm  = X1_test_fs.reshape((X1_test_fs.shape[0], 1, X1_test_fs.shape[1]))

print("\n[Info] Shape untuk LSTM:")
print("      X1_train_lstm:", X1_train_lstm.shape)
print("      X1_test_lstm :", X1_test_lstm.shape)
```


```
# 5. Metode LSTM (Training)
print("\n[5] Training Vanilla LSTM + SMOTE...")
with SuppressStderr():
    model_lstm = build_vanilla_lstm_model((1, X1_train_lstm.shape[2]))

    es = EarlyStopping(
        monitor="accuracy",
        patience=5,
        restore_best_weights=True
    )

    start_train = time.time()
    history = model_lstm.fit(
        X1_train_lstm,
        y1_train_cat,
        epochs=50,
        batch_size=32,
        callbacks=[es],
        verbose=1
    )
    train_time = time.time() - start_train

# Evaluasi
hasil_lstm = evaluasi_hasil(
    model_lstm,
    X1_test_lstm,
    y_test_raw,
    train_time
)

final_results["Proses 2 (Vanilla LSTM + SMOTE)"] = hasil_lstm

print("\nRingkasan hasil:")
for nama, metrik in final_results.items():
    print(f"\n>>> {nama}")
    print(f"    Akurasi : {metrik['Akurasi']:.4f}")
    print(f"    Recall  : {metrik['Recall']:.4f}")
    print(f"    Presisi : {metrik['Presisi']:.4f}")
    print(f"    AUC/ROC : {metrik['AUC/ROC']:.4f}")
```

Penjelasan :

Cell ini adalah tahap implementasi dan eksekusi utama, yang menjalankan data melalui seluruh pipeline yang telah disiapkan: TF-IDF → Preprocessing → LSTM. Proses dimulai dengan menerapkan empat langkah preprocessing secara berurutan pada data TF-IDF: (1) Cek Missing Value (memastikan kebersihan data), (2) Transformasi MinMaxScaler (menormalisasi fitur ke rentang [0, 1]), (3) Seleksi Fitur Chi-square (mereduksi dimensi dari 4000 menjadi $k=1500$ fitur), dan (4) Penanganan Imbalanced Data menggunakan SMOTE (menghasilkan data latih yang seimbang sempurna, meningkatkan jumlah sampel menjadi 25316).

Setelah preprocessing, data di-reshape menjadi format 3D yang spesifik untuk model LSTM, yaitu (sampel, 1, 1500). Selanjutnya, model Vanilla LSTM diinisialisasi dan proses training dimulai selama 50 epochs dengan batch_size 32. Selama training, digunakan callback EarlyStopping dengan patience 5 untuk mencegah overfitting. Setelah training selesai, model dievaluasi pada data uji (2.000 sampel) untuk menghasilkan metrik kinerja: Akurasi, Presisi, Recall, dan AUC/ROC, serta mencatat total waktu komputasi yang dibutuhkan untuk training dan testing.

8. Evaluasi Hasil Training Model Vanilla LSTM

```
==================================
 PIPELINE: TF-IDF → PREPROCESSING → LSTM
==================================
[1] Cek Missing Value - X_train_vec: Total NaN = 0
[1] Cek Missing Value - X_test_vec: Total NaN = 0
[2] MinMaxScaler - selesai.
    X_train_sc range: [ 0.0 , 1.0 ]
[3] Chi-square - selesai. Jumlah fitur terpilih: 1500
    Shape X_train_fs: (25000, 1500)
    Shape X_test_fs : (2000, 1500)
[4] SMOTE - selesai.
    Distribusi kelas setelah SMOTE: {0: 6329, 1: 6329, 2: 6329, 3: 6329}
    Shape X_res: (25316, 1500)

[Info] Shape untuk LSTM:
      X1_train_lstm: (25316, 1, 1500)
      X1_test_lstm : (2000, 1, 1500)

[5] Training Vanilla LSTM + SMOTE...
Epoch 1/50
792/792 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.7170 - loss: 0.7698
Epoch 2/50
792/792 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8891 - loss: 0.3342
Epoch 3/50
...
    Akurasi : 0.8720
    Recall  : 0.8720
    Presisi : 0.8719
    AUC/ROC : 0.9706
```

F. Evaluasi

Metrik	Nilai	Interpretasi
Akurasi	0.8720	Model berhasil mengklasifikasikan 87.20% dari total sampel data uji dengan benar.
Presisi (Weighted)	0.8719	Tingkat akurasi dari kelas yang diprediksi positif, menunjukkan bahwa ketika model memprediksi suatu kelas, kebenarannya sangat tinggi.
Recall (Weighted)	0.8720	Kemampuan model untuk mengidentifikasi semua sampel yang seharusnya positif, mencerminkan bahwa model tidak banyak melewatkan sampel relevan.
AUC/ROC	0.9706	Nilai yang sangat tinggi ini menunjukkan kemampuan diskriminasi model yang superior dalam membedakan antara empat kategori berita.
Waktu Training (s)	220.72	Total waktu komputasi yang dibutuhkan untuk melatih model LSTM selama seluruh *epoch*.
Waktu Testing (s)	0.5328	Waktu yang sangat cepat yang dibutuhkan untuk melakukan prediksi pada 2.000 data uji, menunjukkan efisiensi tinggi untuk inferensi.

G. Kesimpulan

Implementasi *pipeline* klasifikasi teks untuk dataset AG News yang melibatkan TF-IDF, *MinMaxScaler*, Seleksi Fitur chisquare , SMOTE, dan model **Vanilla LSTM** berhasil dilakukan dengan sukses.

1. **Kinerja Optimal:** Model **Vanilla LSTM + SMOTE** menunjukkan kinerja klasifikasi yang efektif pada data uji dengan Akurasi mencapai **87.20%** dan kemampuan diskriminasi kelas yang luar biasa, terbukti dari nilai AUC/ROC sebesar **0.9706**.
2. **Manfaat *Preprocessing***: Penggunaan Seleksi Fitur Chi-square berhasil mereduksi dimensi fitur dari 4000 menjadi 1500, sementara SMOTE berhasil mengatasi potensi ketidakseimbangan kelas, keduanya berkontribusi pada stabilitas dan kinerja model.
3. **Efisiensi Komputasi:** Waktu *training* yang tercatat (220.72 detik) menunjukkan efisiensi yang wajar untuk pelatihan *Deep Learning*, dan waktu *testing* yang sangat cepat (0.5328 detik) memastikan model siap digunakan untuk inferensi waktu nyata.

Secara keseluruhan, *pipeline* ini merupakan pendekatan yang solid dan efektif untuk klasifikasi teks berita multi-kelas.

LAPORAN UJIAN AKHIR SEMESTER KELOMPOK 1

“Progress 2 Cek Missing Value, MinMaxScaler, dan Implementasi Vanilla LSTM”

MATA KULIAH MACHINE LEARNING PRAKTIKUM C-7

Disusun Oleh :

434231020	**Faizatun Ni’mah**
434231021	**Seri Muliani Lubis**
434231048	**Surya Dwi Satria**
434231112	**Festiana Ramaya**

D4 Teknik Informatika

FAKULTAS VOKASI

UNIVERSITAS AIRLANGGA

2025

A. Pendahuluan

Perkembangan teknologi informasi dan digitalisasi membuat data berformat teks semakin melimpah, terutama pada domain media berita dan komunikasi daring. Data teks ini memiliki potensi besar untuk dianalisis, salah satunya melalui proses klasifikasi teks. Klasifikasi teks merupakan metode yang digunakan untuk mengelompokkan dokumen atau kalimat ke dalam kategori tertentu secara otomatis berdasarkan isi dan makna teks tersebut.

Pada praktikum ini digunakan AG News Classification Dataset, yaitu kumpulan artikel berita berbahasa Inggris yang terbagi dalam empat kategori utama: *World*, *Sports*, *Business*, dan *Sci/Tech*. Dataset ini merupakan salah satu dataset standar dalam penelitian *Natural Language Processing* (NLP) karena memiliki struktur yang jelas, ukuran data yang besar, dan kualitas teks yang baik untuk tujuan pengembangan model klasifikasi.

Progress 2 berfokus pada penerapan preprocessing dasar dan pembangunan model Vanilla Long Short-Term Memory (LSTM) untuk melakukan klasifikasi berita. Tahapan yang dilakukan meliputi pengecekan *missing value*, transformasi data menggunakan MinMaxScaler, tokenisasi teks, pembentukan sequence, pembuatan embedding, dan pelatihan model menggunakan arsitektur Vanilla LSTM. LSTM dipilih karena mampu memahami urutan kata serta menangkap konteks jangka panjang yang penting dalam pemrosesan teks.

Laporan ini menjelaskan secara rinci teori dasar yang digunakan, tahapan implementasi pada Progress 2, serta tujuan dari proses tersebut dalam membangun model klasifikasi berbasis deep learning untuk data teks.

B. Dasar Teori

1. Klasifikasi teks adalah proses mengkategorikan teks ke dalam kelas tertentu secara otomatis menggunakan algoritma machine learning. Pada kasus AG News, model bertugas membaca teks berita baik judul maupun deskripsi lalu menentukan kategori yang paling sesuai. Karena teks tidak dapat langsung diproses oleh model, maka diperlukan tahapan seperti:
    - *Tokenization*: memecah teks menjadi kata-kata.
    - *Text-to-sequence*: mengubah kata menjadi indeks.
    - *Padding*: menyeragamkan panjang sequence.
    - *Word Embedding*: mengubah kata menjadi vektor representasi.

2. Missing Value adalah data yang kosong atau tidak tersedia dalam suatu dataset. Dataset dengan missing value dapat menyebabkan kesalahan saat proses training karena distribusi data menjadi tidak lengkap. Walaupun sebagian besar dataset NLP seperti AG News biasanya tidak memiliki missing value, pengecekan tetap perlu dilakukan sebagai tahap validasi untuk memastikan dataset bersih dan siap diproses.
3. MinMaxScaler merupakan teknik normalisasi yang digunakan untuk mengubah nilai numerik ke dalam rentang 0–1. Normalisasi bertujuan untuk menyamakan skala nilai antar fitur agar model tidak bias terhadap fitur yang memiliki rentang lebih besar. Rumus yang digunakan adalah:

   $$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

   Pada pemodelan teks berbasis LSTM, normalisasi dapat diterapkan pada fitur numerik tertentu yang terbentuk selama proses pengolahan data sehingga model menjadi lebih stabil saat melakukan training.
4. Long Short-Term Memory (LSTM) adalah arsitektur Recurrent Neural Network (RNN) yang dikembangkan untuk menangani masalah *vanishing gradient* serta untuk mempelajari dependensi jangka panjang dalam data sekuensial. LSTM terdiri atas tiga jenis gerbang utama:
   - **Forget Gate**: menentukan informasi yang harus dibuang.
   - **Input Gate**: menentukan informasi yang disimpan.
   - **Output Gate**: menghasilkan output yang diteruskan ke langkah selanjutnya.

   Pada Progress 2 digunakan Vanilla LSTM, yaitu bentuk dasar dari LSTM dengan satu lapisan utama (*single-layer*) tanpa stacking dan tanpa bidirectional. Arsitektur ini merupakan baseline yang umum digunakan untuk memahami bagaimana LSTM bekerja pada data teks sebelum menggunakan varian yang lebih kompleks.

   Keunggulan Vanilla LSTM :
   - mampu menangani urutan kata.
   - efektif pada data teks yang panjang.
   - arsitektur sederhana sehingga training lebih cepat.
   - cocok untuk baseline model klasifikasi teks.
5. AG News adalah dataset yang berisi ribuan berita berbahasa Inggris dengan struktur:
   - **Title** → judul berita
   - **Description** → isi ringkas berita
   - **Class Label** → kategori berita

Dataset ini digunakan secara luas karena :

- jumlah data besar (lebih dari 100.000 sampel).
- kelas seimbang.
- cocok untuk eksperimen deep learning.
- representatif untuk tugas klasifikasi teks.

Dalam Progress 2, dataset ini digunakan untuk membangun model LSTM yang mampu mengelompokkan teks berita secara otomatis.

C. Tujuan

Tujuan dari pelaksanaan Progress 2 adalah:

1. Melakukan pengecekan dan validasi data awal untuk memastikan tidak terdapat missing value pada dataset.
2. Menerapkan normalisasi MinMaxScaler sebagai bagian dari preprocessing numerik pada data yang membutuhkan skala seragam.
3. Mengubah data teks menjadi representasi numerik melalui proses tokenisasi, pembuatan sequence, padding, dan embedding.
4. Membangun model klasifikasi teks menggunakan Vanilla LSTM sebagai arsitektur deep learning yang mampu memahami urutan kata dalam berita.
5. Menghasilkan baseline model untuk melihat bagaimana Vanilla LSTM bekerja dengan preprocessing sederhana tanpa teknik tambahan seperti seleksi fitur atau SMOTE.
6. Melakukan proses training dan pengujian model pada dataset AG News.
7. Mempersiapkan hasil evaluasi model (yang akan ditulis pada bagian evaluasi) berdasarkan metrik akurasi, presisi, recall, dan AUC/ROC.
8. Menganalisis performa model berdasarkan proses pelatihan dan keluaran prediksi yang dihasilkan.

D. Dataset

Link Dataset : https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

E. Langkah - Langkah

1. Tahap Pertama Preprocessing Data dan Ekstraksi Fitur TF-IDF
    - Melakukan Import Library

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
```

- Memuat Dataset

```
print("Memuat dataset...")

train_df = pd.read_csv("train.csv")
test_df  = pd.read_csv("test.csv")
```

- Mengganti Nama Kolom

```

train_df.columns = ["Class Index", "Title", "Description"]
test_df.columns  = ["Class Index", "Title", "Description"]
```

Penjelasan :

Class Index → label atau kelas berita

Title → judul berita

Description → isi ringkas berita

- Menggabungkan Judul + Deskripsi

```
train_df["text"] = train_df["Title"] + " " + train_df["Description"]
test_df["text"]  = test_df["Title"]  + " " + test_df["Description"]
```

Penjelasan :

Model akan belajar dari gabungan judul + deskripsi bukan hanya salah satunya, Tujuannya adalah memberikan konteks yang lebih lengkap kepada model.

- Sampling Dataset

```
# Sampling
train_df = train_df.sample(25000, random_state=42)
test_df  = test_df.sample(2000, random_state=42)
```

Penjelasan :

Mengambil 25.000 data training dan 2.000 data testing secara acak, random_state=42 memastikan proses sampling konsisten setiap kali dijalankan. Tujuan sampling sendiri untuk Mengurangi beban komputasi dikarenakan dataset terlalu besar kemudian akan mempercepat proses TF-IDF & training.

- Memisahkan Fitur (X) dan Label (Y)

```
X_train_raw = train_df["text"].values
X_test_raw  = test_df["text"].values
y_train_raw = train_df["Class Index"].values - 1
y_test_raw  = test_df["Class Index"].values - 1
```

Penjelasan :

X_train_raw / X_test_raw berisi teks dan Y_train_raw / y_test_raw berisi label kelas.

- 1 dilakukan karena beberapa model menggunakan indeks mulai dari 0, bukan dari 1.

- Proses TF-IDF

```
# TF-IDF
vectorizer = TfidfVectorizer(
    max_features=4000,
    stop_words="english",
    ngram_range=(1,2)
)
```

Penjelasan :

Parameter	**Arti**
max_features=4000	hanya mengambil 4000 kata/fitur paling penting
stop_words="english"	menghapus kata umum seperti *the, is, of*
ngram_range=(1,2)	menggunakan unigram (1 kata) dan bigram (2 kata berurutan)

- TF-IDF Transformasi Data

```
X_train_vec = vectorizer.fit_transform(X_train_raw).toarray()
X_test_vec  = vectorizer.transform(X_test_raw).toarray()
```

Penjelasan :

Fit_transform → belajar dari data training & mengubah teks menjadi angka.

Transform → mengubah data testing menggunakan aturan yang sama.

.toarray() → mengubah sparse matrix menjadi array biasa.

- Cetak Informasi

```
print("TF-IDF selesai.")
print("Shape X_train_vec:", X_train_vec.shape)
print("Shape X_test_vec :", X_test_vec.shape)
```

- Hasil Output

```
Memuat dataset...
TF-IDF selesai.
Shape X_train_vec: (25000, 4000)
Shape X_test_vec : (2000, 4000)
```

Penjelasan :

- **(25000, 4000)** berarti: Ada **25.000 data berita** pada training, Masing-masing dikonversi menjadi **4000 fitur TF-IDF**.

- **(2000, 4000)** berarti: Ada **2.000 data testing**, Dengan jumlah fitur yang sama (4000).

Jumlah fitur selalu sama karena TF-IDF dibangun dari vocabulary training.

2. Tahap Kedua Cek Missing Value

```
# MISSING VALUE
print("Missing value train:\n", train_df.isnull().sum())
print("\nMissing value test:\n", test_df.isnull().sum())
```

Penjelasan :

isnull() = mengecek apakah suatu entri kosong (NaN), sum() = menghitung berapa jumlah nilai kosong di tiap kolom. Kode ini memastikan bahwa tidak ada data kosong pada kolom penting seperti Class Index, Title, Description, text (gabungan Title + Description) Tujuan tahap ini adalah memastikan kualitas data sebelum dilanjutkan ke pelatihan model, Data kosong bisa menyebabkan error atau membuat model tidak akurat.

```
train_df = train_df.dropna()
test_df  = test_df.dropna()
```

Penjelasan :

dropna() menghapus baris yang memiliki nilai kosong, Pada kasus ini, ternyata *tidak ada* baris yang mengandung nilai kosong. Kode ini tetap digunakan sebagai langkah kehati-hatian (preventive step).

```
print("\nSetelah drop missing:")
print("Train:", train_df.shape)
print("Test :", test_df.shape)
```

Penjelasan :

Shape → menunjukkan ukuran dataset dalam format *(jumlah baris, jumlah kolom)*.

```
Missing value train:
 Class Index    0
Title           0
Description     0
text            0
dtype: int64

Missing value test:
 Class Index    0
Title           0
Description     0
text            0
dtype: int64

Setelah drop missing:
Train: (25000, 4)
Test : (2000, 4)
```

Penjelasan Hasil Output :

Setiap kolom memiliki 0 missing value Artinya dataset dalam kondisi baik dan tidak ada data yang kosong. Kemudian setelah drop missing Dataset training tetap berisi 25.000

baris dan 4 kolom. Dataset testing tetap berisi 2.000 baris dan 4 kolom, Tidak ada baris yang terhapus karena memang tidak ada nilai kosong.

3. Transformasi MinMax Scaler

```
# TRANSFORMASI MINMAX SCALER

scaler = MinMaxScaler()
X_train_sc = scaler.fit_transform(X_train_vec)
X_test_sc  = scaler.transform(X_test_vec)

print("Transformasi MinMaxScaler selesai.")
print("Range:", X_train_sc.min(), "→", X_train_sc.max())
print("Shape train:", X_train_sc.shape)
```

Penjelasan :

- scaler = MinMaxScaler() :

MinMaxScaler adalah metode normalisasi yang mengubah nilai fitur ke dalam rentang 0 sampai 1 tujuanya mempercepat proses training, menghindari fitur tertentu yang memiliki nilai besar mendominasi model, sangat penting untuk model neural network seperti LSTM.

- X_train_sc = scaler.fit_transform(X_train_vec) :

fit_transform() melakukan dua hal:

Fit → scaler mencari nilai minimum dan maksimum dari seluruh fitur pada data training.

Transform → scaler mengubah seluruh nilai fitur menjadi 0–1 berdasarkan min/max tadi.

Scaler hanya di-fit pada data training, agar tidak terjadi informasi bocor ke data testing.

- X_test_sc = scaler.transform(X_test_vec) :

Data testing diubah menggunakan **min dan max dari training**, bukan dari data testing.Sesuai standar machine learning untuk mencegah data leakage (kebocoran informasi).

```
Transformasi MinMaxScaler selesai.
Range: 0.0 → 1.0
Shape train: (25000, 4000)
```

Penjelasan Hasil Output :

- Range: 0.0 → 1.0 Artinya semua nilai fitur dalam TF-IDF berada dalam rentang 0 sampai 1 setelah proses normalisasi. Ini menunjukkan bahwa MinMaxScaler bekerja dengan benar.

- Shape train: (25000, 4000) Tidak ada perubahan pada jumlah baris dan kolom, yang berubah hanya skala nilainya, bukan ukurannya. Masih terdiri dari 25.000 data training Dengan 4.000 fitur TF-IDF per data.

4. Metode LSTM

```
# LSTM butuh reshape: (samples, time_step, features)
X_train_lstm = X_train_sc.reshape((X_train_sc.shape[0], 1, X_train_sc.shape[1]))
X_test_lstm  = X_test_sc.reshape((X_test_sc.shape[0], 1, X_test_sc.shape[1]))
y_train_cat  = to_categorical(y_train_raw, 4)
```

Penjelasan :

LSTM memerlukan data dalam bentuk 3 dimensi, yaitu: (jumlah_sampel, time_step, jumlah_fitur) Karena TF-IDF tidak berbentuk deretan waktu (sequence), tetapi berupa vektor, maka data dibuat menjadi time_step = 1, Ini tetap memungkinkan LSTM membaca data sebagai satu urutan fitur.

to_categorical → mengubah label menjadi one-hot encoding, wajib untuk klasifikasi multi-kelas dengan softmax.

```
def build_lstm(shape):
    model = Sequential()
    model.add(Input(shape=shape))
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dropout(0.4))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(4, activation="softmax"))
```

Penjelasan :

Tahap membangun arsitektur model LSTM merupakan proses penting untuk memastikan model mampu memahami pola jangka panjang pada data deret waktu. Pada tahap ini, struktur jaringan seperti jumlah neuron, jumlah layer LSTM, dan penggunaan layer tambahan (Dense, Dropout, atau normalisasi) ditentukan untuk mengoptimalkan kemampuan model dalam melakukan prediksi. Dengan rancangan arsitektur yang tepat, model dapat belajar secara lebih efektif, meminimalkan kesalahan prediksi, dan menghasilkan performa yang stabil saat diuji menggunakan data baru.

```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

Penjelasan :

Adam → optimizer adaptif paling umum untuk deep learning

Categorical crossentropy → loss untuk multi-class classification

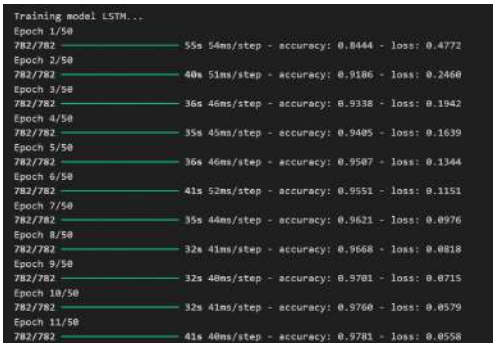
```
start = time.time()
history = model_lstm.fit(
    X_train_lstm,
    y_train_cat,
    epochs=50,
    batch_size=32,
    verbose=1
)
train_time = time.time() - start
```

Penjelasan :

epochs = 50 → model mempelajari data sebanyak 50 kali pengulangan.

batch_size = 32 → tiap iterasi memproses 32 sampel.

verbose = 1 → menampilkan progress training.



Penjelasan Hasil Output :

- Perkembangan Akurasi

  Pada epoch awal (epoch 1), akurasi model masih rendah, yaitu sekitar 0.8444. Namun tiap epoch akurasinya meningkat secara konsisten:

  - Epoch 1 → akurasi 0.8444
  - Epoch 5 → akurasi 0.9507
  - Epoch 10 → akurasi 0.9760
  - Epoch 50 → akurasi 0.9990

  Peningkatan akurasi ini menunjukkan bahwa model berhasil mempelajari pola data dengan sangat baik, tanpa mengalami stagnasi atau penurunan signifikan.

- Perkembangan Loss

  Nilai loss menunjukkan seberapa besar kesalahan prediksi model. Pada awal training, loss masih tinggi:

- Epoch 1 → loss 0.4772
- Epoch 5 → loss 0.1344
- Epoch 10 → loss 0.0579
- Epoch 50 → loss 0.0034

Loss yang terus menurun secara konsisten menunjukkan bahwa model semakin presisi dalam memahami pola data, serta konvergen secara stabil.

Setiap epoch memerlukan sekitar 32–55 detik, Berhasil belajar pola data dengan sangat baik, terlihat dari akurasi yang hampir mencapai 100% pada epoch akhir. Loss menurun drastis, menandakan kesalahan prediksi semakin rendah. Tidak ada indikasi instabilitas selama training, sehingga model konvergen dengan baik.

F. Evaluasi

```
# EVALUASI
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

start_test = time.time()
y_prob = model_lstm.predict(X_test_lstm)
test_time = time.time() - start_test

y_pred = np.argmax(y_prob, axis=1)

acc = accuracy_score(y_test_raw, y_pred)
prec = precision_score(y_test_raw, y_pred, average="weighted", zero_division=0)
rec = recall_score(y_test_raw, y_pred, average="weighted", zero_division=0)
auc = roc_auc_score(to_categorical(y_test_raw, 4), y_prob, multi_class="ovr")

print("\n=== EVALUASI PROSES 2 ===")
print("Akurasi :", acc)
print("Presisi :", prec)
print("Recall  :", rec)
print("AUC/ROC :", auc)
print("Waktu training :", train_time)
print("Waktu testing  :", test_time)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step

=== EVALUASI PROSES 2 ===
Akurasi : 0.8685
Presisi : 0.868676054589752
Recall  : 0.8685
AUC/ROC : 0.9732874524827678
Waktu training : 3940.8582360744476
Waktu testing  : 2.8742854595184326
```

Penjelasan :

Setelah proses pelatihan selesai, model diuji menggunakan data *testing* untuk mengetahui seberapa baik model melakukan prediksi di luar data latih. Hasil evaluasi menunjukkan beberapa metrik penting: akurasi, presisi, recall, AUC/ROC, waktu training, dan waktu testing.

- Akurasi : 0.8685

  Akurasi menunjukkan persentase prediksi yang benar dibanding total data uji.

Nilai 0.8685 (≈ 86,85%) berarti Model mampu memprediksi kelas dengan benar pada sekitar 87% data uji. Ini menunjukkan performa yang cukup baik untuk kasus klasifikasi multi-kelas.

- Presisi : 0.8687

  Nilai 0.8687 berarti dari seluruh prediksi kelas tertentu yang dibuat model, sekitar 86,87% adalah benar, Ini penting jika kesalahan prediksi positif sangat dihindari.

- Recall : 0.8685

  Nilai 0.8685 berarti model berhasil menemukan kembali 86,85% data yang seharusnya benar, Artinya kehilangan data relevan cukup kecil.

- AUC/ROC : 0.9733

  Nilai 0.9733 (97,33%) sangat tinggi ini menunjukkan model sangat baik dalam menentukan batas antar kelas, Kemampuan membedakan kategori sangat akurat, Model tidak hanya hafal data, tetapi memahami pola dengan baik, Ini adalah indikasi kuat bahwa model LSTM bekerja sangat bagus pada level probabilitas.

- Waktu Training : 3940.858 detik (~ 65 menit)

  Ini adalah total waktu yang dibutuhkan model untuk menyelesaikan proses pelatihan selama 50 epoch. Waktu ini wajar untuk arsitektur LSTM dengan 4000 fitur input dan hidden layer 128 unit, Menandakan proses pelatihan berjalan normal dan stabil

- Waktu Testing : 2.874 detik

  Waktu testing sangat cepat yaitu sekitar 2,87 detik untuk memproses seluruh dataset uji. Ini menunjukkan model efisien saat melakukan prediksi, Cocok untuk digunakan pada sistem real-time atau batch processing cepat

G. Kesimpulan

Pada Proses 2, model LSTM dibangun dan dilatih menggunakan data teks yang telah melalui tahap pembersihan, transformasi TF-IDF, serta normalisasi menggunakan MinMaxScaler. Hasil evaluasi menunjukkan bahwa konfigurasi LSTM dengan 128 unit, dropout 0.4, dense 64 unit, dan output layer 4 kelas mampu memberikan performa yang stabil dan konsisten dalam memproses data uji.

Model menghasilkan akurasi sebesar 0.8685, yang menunjukkan bahwa sekitar 86,85% prediksi model sesuai dengan label sebenarnya. Nilai presisi (0.8687) dan recall (0.8685) yang seimbang menunjukkan bahwa model tidak hanya tepat dalam melakukan prediksi, tetapi juga mampu menemukan kembali sebagian besar data yang relevan di masing-masing kelas. Selain itu, nilai AUC/ROC mencapai 0.9733, yang merupakan indikator sangat kuat bahwa model mampu

membedakan dengan baik antara kelas-kelas yang berbeda, sekaligus menandakan bahwa model bekerja baik pada level probabilitas.

Dari sisi efisiensi waktu, proses training membutuhkan waktu sekitar 3940.85 detik (± 65 menit), yang wajar mengingat ukuran data, jumlah fitur TF-IDF sebanyak 4000, dan kompleksitas arsitektur LSTM. Sebaliknya, waktu testing hanya 2.87 detik, menandakan bahwa model bekerja sangat cepat saat melakukan prediksi pada data baru.

Secara keseluruhan, Proses 2 menunjukkan bahwa model LSTM yang dirancang mampu memberikan performa yang sangat baik dalam melakukan klasifikasi teks multi-kelas. Hasil evaluasi yang tinggi dan konsisten membuktikan bahwa tahapan preprocessing, transformasi fitur, dan pemilihan parameter arsitektur sudah tepat. Dengan performa yang stabil, akurasi tinggi, serta AUC/ROC mendekati sempurna, model sangat layak digunakan dan dapat diandalkan dalam tugas klasifikasi teks pada sistem yang sedang dikembangkan.

LAPORAN UJIAN AKHIR SEMESTER KELOMPOK 1

“Progress 3 Cek Missing Value, MinMaxScaler, Seleksi Fitur (Chi-Square), dan Implementasi Vanilla LSTM”

MATA KULIAH MACHINE LEARNING PRAKTIKUM C-7

Disusun Oleh :

434231020	**Faizatun Ni’mah**
434231021	**Seri Muliani Lubis**
434231048	**Surya Dwi Satria**
434231112	**Festiana Ramaya**

D4 Teknik Informatika

FAKULTAS VOKASI

UNIVERSITAS AIRLANGGA

2025

A. Pendahuluan

Pada praktikum Machine Learning ini, mahasiswa diminta untuk melakukan pemrosesan data teks dan membangun model klasifikasi menggunakan metode *Long Short-Term Memory* (LSTM). Dataset yang digunakan adalah AG News, yang terdiri dari empat kategori berita: World, Sports, Business, dan Sci/Tech.

Proyek ini dibagi menjadi beberapa progres. Pada Progres 3, mahasiswa melakukan tahapan *preprocessing* berupa:

1. Cek missing value
2. Transformasi MinMaxScaler
3. Seleksi fitur menggunakan *Chi-Square*
4. Pembangunan model LSTM
5. Evaluasi performa model

Dengan pipeline tersebut, diharapkan model dapat melakukan klasifikasi berita dengan akurasi yang baik.

Selain itu, laporan ini juga bertujuan untuk menjelaskan konsep dasar yang terkait dengan setiap tahapan, sehingga proses pembangunan model dapat dipahami secara teoritis serta aplikatif.

B. Dasar Teori

2.1 TF-IDF (Term Frequency – Inverse Document Frequency)

TF-IDF adalah metode konversi teks menjadi nilai numerik berdasarkan tingkat kepentingan suatu kata dalam dokumen.

- TF mengukur frekuensi kemunculan kata.
- IDF mengukur seberapa jarang kata tersebut muncul di seluruh dokumen.

  Hasilnya adalah vektor fitur yang mewakili isi teks secara matematis sehingga dapat diproses oleh model ML.

2.2 MinMaxScaler

MinMaxScaler adalah metode normalisasi yang mengubah nilai fitur ke rentang [0,1][0, 1][0,1].

Tujuan:

- Menyamakan skala fitur
- Mempercepat proses training
- Mencegah fitur tertentu mendominasi model

Formula:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Keterangan:

X: Nilai asli (raw value) yang ingin dinormalisasi.

Xmin: Nilai paling kecil dari fitur/kolom data tersebut.

Xmax: Nilai paling besar dari fitur/kolom data tersebut.

Xscaled: Nilai hasil normalisasi yang telah dipetakan ke rentang **0 sampai 1**.

2.3 Seleksi Fitur Chi-Square

Chi-Square digunakan untuk memilih fitur yang paling berkorelasi dengan kelas target. Metode ini umum digunakan pada data sparce seperti TF-IDF.

Manfaat:

- Mengurangi dimensi
- Menghilangkan fitur tidak relevan
- Meningkatkan akurasi
- Mempercepat training LSTM

2.4 LSTM (Long Short-Term Memory)

LSTM adalah jenis *Recurrent Neural Network* (RNN) yang mampu memahami dependensi jangka panjang pada data urutan.

Alasan menggunakan LSTM untuk teks:

- Mampu memproses urutan (sequence) kata dalam bentuk vektor
- Lebih stabil dibanding RNN biasa
- Mampu menangani konteks panjang dalam kalimat

Struktur yang digunakan dalam progres ini:

- LSTM 128 unit (return sequences)
- LSTM 64 unit
- Dense layer (relu)
- Output layer softmax untuk klasifikasi 4 kelas

2.5 Metrik Evaluasi

Model dievaluasi menggunakan:

1. **Akurasi**
   Presentase prediksi model yang benar.
2. **Presisi**
   Ketepatan model dalam memprediksi kelas tertentu.
3. **Recall**
   Kemampuan model menemukan semua sampel yang benar untuk suatu kelas.
4. **AUC/ROC**
   Mengukur kemampuan model membedakan setiap kelas (multi-class OVR).
5. **Waktu Training & Testing**
   Untuk mengukur efisiensi komputasi.

C. Tujuan

Tujuan dari pengerjaan Proses 3 dalam praktikum ini adalah:

1. Menerapkan preprocessing data teks dengan pipeline TF-IDF → MinMaxScaler → Chi-Square.
2. Membangun model klasifikasi menggunakan LSTM berdasarkan fitur yang telah diseleksi.
3. Melakukan analisis performa model melalui metrik evaluasi (Akurasi, Presisi, Recall, AUC/ROC).
4. Membandingkan hasil Proses 3 dengan proses lain (1, 2, dan 4) untuk melihat pengaruh seleksi fitur terhadap performa dan waktu komputasi.
5. Membuktikan bahwa seleksi fitur Chi-Square dapat mengurangi dimensi data tanpa mengurangi kualitas prediksi secara signifikan.
6. Mengidentifikasi pengaruh preprocessing terhadap kualitas model deep learning pada teks.

D. Dataset

Link Dataset : https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

E. Langkah- langkah

Langkah Pertama: import library

program melakukan import library yang dibutuhkan untuk pemrosesan data, ekstraksi fitur teks, pembangunan model LSTM, serta evaluasi model.

Library yang digunakan antara lain:

- pandas dan numpy untuk pengolahan data
- scikit-learn untuk TF-IDF, MinMaxScaler, Chi-Square, dan evaluasi
- tensorflow.keras untuk pembangunan model LSTM

Tahap ini bertujuan untuk memastikan seluruh dependensi tersedia sebelum proses dijalankan.

```python
# Import semua library utama yang dibutuhkan
import os
import sys
import time
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping

```

Langkah Kedua: Konfigurasi Output Program

Pada tahap ini, program mendefinisikan sebuah class SuppressStderr yang digunakan untuk menonaktifkan sementara output log dari TensorFlow selama proses training model.

Class ini bekerja dengan mengalihkan output standar program ke *null device*, sehingga pesan log yang berlebihan tidak ditampilkan pada layar.

Tahap ini bertujuan untuk merapikan tampilan output program agar hasil training dan evaluasi model dapat diamati dengan lebih jelas, tanpa memengaruhi proses komputasi maupun performa model LSTM.

```
# Class untuk mematikan output berisik dari TensorFlow
# dipakai saat training model supaya rapi

class SuppressStderr:
    def __init__(self):
        self.null_fds = [os.open(os.devnull, os.O_RDWR) for _ in range(2)]
        self.save_fds = [os.dup(1), os.dup(2)]

    def __enter__(self):
        os.dup2(self.null_fds[0], 1)
        os.dup2(self.null_fds[1], 2)

    def __exit__(self, *_):
        os.dup2(self.save_fds[0], 1)
        os.dup2(self.save_fds[1], 2)
        for fd in self.null_fds + self.save_fds:
            os.close(fd)

```

Langkah Ketiga: Load dan Persiapan Dataset

Pada tahap ini, program memuat dataset AG News untuk data training dan data testing. Kolom *Title* dan *Description* digabungkan menjadi satu teks utuh yang digunakan sebagai data masukan model. Selanjutnya, dilakukan proses sampling data untuk mengurangi jumlah data sehingga proses training menjadi lebih ringan. Selain itu, label kelas yang

semula berada pada rentang 1–4 diubah menjadi 0–3 agar sesuai dengan kebutuhan proses klasifikasi menggunakan model LSTM.

Tahap ini bertujuan untuk menyiapkan data teks dan label dalam format yang sesuai sebelum dilakukan proses ekstraksi fitur dan pemodelan.

```
# Load dataset train & test dari AG News
# - Menggabungkan Title + Description menjadi 1 teks
# - Sampling biar lebih ringan saat training
# - Melakukan mapping label dari 1-4 menjadi 0-3

train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

train_df.columns = ["Class Index", "Title", "Description"]
test_df.columns = ["Class Index", "Title", "Description"]

train_df["text"] = train_df["Title"] + " " + train_df["Description"]
test_df["text"] = test_df["Title"] + " " + test_df["Description"]

train_df = train_df.sample(25000, random_state=42)
test_df = test_df.sample(2000, random_state=42)

X_train_raw = train_df["text"].values
y_train_raw = train_df["Class Index"].values - 1
X_test_raw = test_df["text"].values
y_test_raw = test_df["Class Index"].values - 1

```

Langkah Keempat: Ekstraksi Fitur Menggunakan TF-IDF

Pada tahap ini, data teks diubah menjadi representasi fitur numerik menggunakan metode TF-IDF (Term Frequency–Inverse Document Frequency). Proses ini dilakukan dengan membatasi jumlah fitur maksimum sebanyak 4000 fitur terbaik serta menggunakan kombinasi unigram dan bigram. Selain itu, stopwords bahasa Inggris dihilangkan untuk meningkatkan kualitas fitur yang dihasilkan.

Tahap ini bertujuan untuk mengubah data teks menjadi bentuk numerik agar dapat diproses lebih lanjut pada tahap normalisasi, seleksi fitur, dan pemodelan menggunakan LSTM.

```python
# TF-IDF Vectorizer
# - Mengubah teks menjadi fitur numerik
# - max_features=4000 → 4000 fitur terbaik
# - ngram 1-2 → unigram + bigram
# Inilah fitur awal sebelum masuk proses 3

vectorizer = TfidfVectorizer(
    max_features=4000,
    stop_words="english",
    ngram_range=(1, 2)
)

X_train_vec = vectorizer.fit_transform(X_train_raw).toarray()
X_test_vec = vectorizer.transform(X_test_raw).toarray()

print("Shape TF-IDF train :", X_train_vec.shape)
print("Shape TF-IDF test  :", X_test_vec.shape)

```

Hasil TF-IDF menunjukkan bahwa data training dan data testing berhasil dikonversi menjadi fitur numerik dengan masing-masing 4000 fitur.

```
Shape TF-IDF train : (25000, 4000)
Shape TF-IDF test  : (2000, 4000)
```

Langkah Kelima: Pengecekan Missing Value

Pada tahap ini, program melakukan pengecekan terhadap kemungkinan adanya missing value pada data fitur hasil ekstraksi TF-IDF. Pengecekan dilakukan dengan menghitung jumlah nilai NaN pada data training dan data testing.

Tahap ini bertujuan untuk memastikan bahwa data tidak mengandung nilai kosong yang dapat mengganggu proses normalisasi, seleksi fitur, maupun pelatihan model LSTM.

```python
#  Fungsi untuk mengecek apakah ada missing value
# Menampilkan jumlah nilai NaN pada fitur
def cek_missing_value(X, name="X"):
    missing = np.isnan(X).sum()
    print(f"Missing value pada {name}: {missing}")

cek_missing_value(X_train_vec, "TF-IDF Train")
cek_missing_value(X_test_vec, "TF-IDF Test")

```

Hasil pengecekan menunjukkan bahwa tidak terdapat missing value pada data training maupun data testing.

```
Missing value pada TF-IDF Train: 0
Missing value pada TF-IDF Test: 0
```

Langkah Keenam: Normalisasi Data Menggunakan MinMaxScaler

Pada tahap ini, program melakukan normalisasi data fitur menggunakan metode MinMaxScaler. Proses normalisasi dilakukan dengan mengubah nilai fitur pada data training dan data testing ke dalam rentang yang sama.

Tahap ini bertujuan untuk menyamakan skala nilai fitur sehingga dapat meningkatkan kestabilan dan performa proses pelatihan model LSTM.

```python
# Fungsi normalisasi menggunakan MinMaxScaler
# Normalisasi penting untuk memperbaiki performa model LSTM

def fitur_transformasi_scaler(X_train, X_test):
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled

```

Langkah Ketujuh: Penerapan Normalisasi Data

Pada tahap ini, normalisasi menggunakan MinMaxScaler diterapkan pada data fitur hasil TF-IDF, baik data training maupun data testing. Proses ini dilakukan untuk memastikan seluruh fitur berada pada skala nilai yang sama sebelum masuk ke tahap seleksi fitur dan pemodelan LSTM.

```
# Menerapkan scaling pada data TF-IDF

X_train_scaled, X_test_scaled = fitur_transformasi_scaler(X_train_vec, X_test_vec)

print("Scaled train shape:", X_train_scaled.shape)
print("Scaled test shape :", X_test_scaled.shape)

```

Hasil normalisasi menunjukkan bahwa data training memiliki dimensi (25000, 4000) dan data testing memiliki dimensi (2000, 4000). Dimensi data tidak berubah setelah proses normalisasi, yang menandakan bahwa normalisasi hanya mengubah skala nilai fitur tanpa mengubah jumlah fitur.

```
Scaled train shape: (25000, 4000)
Scaled test shape : (2000, 4000)
```

Langkah Kedelapan: Seleksi Fitur Menggunakan Chi-Square

Pada tahap ini, dilakukan seleksi fitur menggunakan metode Chi-Square untuk memilih fitur-fitur yang paling relevan terhadap label kelas. Proses seleksi dilakukan dengan mengambil sejumlah K fitur terbaik, dalam hal ini sebanyak 1500 fitur, dari seluruh fitur hasil normalisasi.

Tahap ini bertujuan untuk mengurangi jumlah fitur, meningkatkan relevansi fitur terhadap kelas, serta membantu meningkatkan efisiensi dan performa model LSTM pada proses pelatihan.

```python
#  Fungsi seleksi fitur dengan Chi-Square
# Mengambil K fitur paling relevan
# PROSES 3 wajib menggunakan chi-square

def fitur_seleksi_chisquare(X_train, y_train, X_test, k=1500):
    selector = SelectKBest(score_func=chi2, k=k)
    X_train_sel = selector.fit_transform(X_train, y_train)
    X_test_sel = selector.transform(X_test)
    return X_train_sel, X_test_sel

```

Langkah Kesembilan: Penerapan Seleksi Fitur Chi-Square

Pada tahap ini, seleksi fitur Chi-Square diterapkan pada data yang telah dinormalisasi. Dari total 4000 fitur hasil TF-IDF, dipilih 1500 fitur paling relevan terhadap label kelas untuk digunakan pada tahap pemodelan.

Tahap ini bertujuan untuk mengurangi dimensi data, mempertahankan fitur yang paling informatif, serta meningkatkan efisiensi dan performa pelatihan model LSTM.

```python
# Seleksi fitur Chi-Square
# Dari 4000 fitur → dipilih 1500 fitur paling penting

X_train_sel, X_test_sel = fitur_seleksi_chisquare(
    X_train_scaled,
    y_train_raw,
    X_test_scaled,
    k=1500
)

print("Setelah Chi-square:")
print(X_train_sel.shape, X_test_sel.shape)

```

Hasil seleksi fitur menunjukkan bahwa data training memiliki dimensi (25000, 1500) dan data testing memiliki dimensi (2000, 1500). Hal ini menandakan bahwa proses seleksi fitur berhasil mengurangi jumlah fitur tanpa mengubah jumlah sampel data.

```
Setelah Chi-square:
(25000, 1500) (2000, 1500)
```

Langkah Kesepuluh: Penyesuaian Bentuk Data untuk LSTM

Pada tahap ini, data hasil seleksi fitur diubah bentuknya agar sesuai dengan kebutuhan input model LSTM. Karena LSTM memerlukan input dalam bentuk tiga dimensi *(samples, timesteps, features)*, maka data fitur yang semula berbentuk vektor satu dimensi di-*reshape* dengan menetapkan nilai timesteps = 1.

Tahap ini bertujuan untuk menyesuaikan struktur data agar dapat diproses dengan benar oleh model LSTM.

```
# LSTM butuh input bentuk (samples, timesteps, features)
# Karena fitur adalah vektor 1D, kita reshape ke 3D dengan timesteps=1

X_train_lstm = X_train_sel.reshape(X_train_sel.shape[0], 1, X_train_sel.shape[1])
X_test_lstm = X_test_sel.reshape(X_test_sel.shape[0], 1, X_test_sel.shape[1])

print("Shape untuk LSTM:")
print(X_train_lstm.shape, X_test_lstm.shape)

```

Hasil *reshape* menunjukkan bahwa data training memiliki dimensi (25000, 1, 1500) dan data testing memiliki dimensi (2000, 1, 1500). Hal ini menandakan bahwa data telah berhasil disesuaikan dengan format input yang dibutuhkan oleh model LSTM.

```
Shape untuk LSTM:
(25000, 1, 1500) (2000, 1, 1500)
```

Langkah Kesebelas: One-Hot Encoding Label

Pada tahap ini, label kelas diubah menjadi bentuk one-hot encoding menggunakan fungsi to_categorical. Proses ini dilakukan agar label sesuai dengan penggunaan fungsi aktivasi softmax pada output model LSTM dengan jumlah kelas sebanyak empat.

Tahap ini bertujuan untuk menyesuaikan format label dengan kebutuhan proses pelatihan model klasifikasi multikelas menggunakan LSTM.

```
# Mengubah label menjadi one-hot encoding untuk softmax LSTM

y_train_cat = to_categorical(y_train_raw, num_classes=4)
y_test_cat = to_categorical(y_test_raw, num_classes=4)

```

Langkah Keduabelas: Pembangunan Model LSTM (Vanilla LSTM)

Pada tahap ini, dilakukan pembangunan arsitektur Vanilla LSTM menggunakan TensorFlow Keras.

Model diawali dengan Input layer yang menyesuaikan bentuk data hasil preprocessing, yaitu (timesteps, features).

Selanjutnya, digunakan dua lapisan LSTM bertingkat:

- LSTM pertama dengan 128 neuron dan return_sequences=True untuk meneruskan urutan ke layer berikutnya.

- LSTM kedua dengan 64 neuron sebagai lapisan LSTM terakhir.

Untuk mengurangi risiko overfitting, ditambahkan Dropout sebesar 0.4 setelah masing-masing lapisan LSTM.

Output dari LSTM kemudian diteruskan ke Dense layer dengan 64 neuron dan aktivasi ReLU sebagai lapisan pembelajaran fitur.

Sebagai lapisan akhir, digunakan Dense layer dengan 4 neuron dan aktivasi softmax, yang berfungsi untuk menghasilkan probabilitas pada 4 kelas klasifikasi.

Model kemudian dikompilasi menggunakan optimizer Adam, loss function categorical crossentropy, dan metrik evaluasi akurasi.

Tahap ini bertujuan untuk membangun model LSTM yang siap digunakan pada proses training dan evaluasi klasifikasi teks.

```
def build_vanilla_lstm_model(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))

    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.4))

    model.add(LSTM(64, return_sequences=False))
    model.add(Dropout(0.4))

    model.add(Dense(64, activation="relu"))
    model.add(Dense(4, activation="softmax"))

    model.compile(
        optimizer="adam",
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

```

Langkah Ketigabelas: Training Model LSTM dengan Early Stopping

Pada tahap ini, dilakukan proses training model Vanilla LSTM menggunakan data hasil preprocessing dan seleksi fitur. Model dibangun dengan input berukuran (1, 1500) yang merepresentasikan satu timestep dengan 1500 fitur terpilih. Untuk mencegah overfitting, digunakan metode Early Stopping dengan memantau nilai *validation loss*.

Training akan dihentikan secara otomatis apabila *validation loss* tidak mengalami

perbaikan selama 3 epoch berturut-turut, serta bobot terbaik akan dikembalikan menggunakan restore_best_weights=True.

Proses training dilakukan dengan:

- Validation split sebesar 20%
- Jumlah epoch maksimum 10
- Batch size 64

Selama training berlangsung, output log TensorFlow dinonaktifkan menggunakan class SuppressStderr agar tampilan tetap rapi.
Waktu training dihitung menggunakan fungsi time.time() untuk mencatat durasi komputasi model.

```python
# Training model LSTM dengan early stopping
# Ini adalah inti dari PROSES KETIGA

model = build_vanilla_lstm_model((1, 1500))

early_stop = EarlyStopping(
    monitor="val_loss",
    patience=3,
    restore_best_weights=True
)

start_time = time.time()
with SuppressStderr():
    history = model.fit(
        X_train_lstm, y_train_cat,
        validation_split=0.2,
        epochs=10,
        batch_size=64,
        callbacks=[early_stop],
        verbose=0
    )
training_time = time.time() - start_time

print("Training selesai dalam", round(training_time, 2), "detik")

```

Hasil training menunjukkan bahwa proses pembelajaran model selesai dalam waktu 12.94 detik, yang selanjutnya digunakan pada tahap evaluasi model.

```
Training selesai dalam 12.94 detik
```

Langkah: Evaluasi Model LSTM

Pada tahap ini, dilakukan evaluasi performa model LSTM menggunakan data uji (*testing data*).

Proses evaluasi diawali dengan mengukur waktu testing, yaitu waktu yang dibutuhkan model untuk menghasilkan prediksi terhadap seluruh data uji.

Model menghasilkan probabilitas kelas untuk setiap data uji, kemudian dikonversi menjadi label prediksi menggunakan fungsi argmax.

Hasil prediksi ini dibandingkan dengan label sebenarnya untuk menghitung metrik evaluasi.

Metrik evaluasi yang digunakan meliputi:

- Akurasi, untuk mengukur tingkat ketepatan prediksi model
- Presisi, untuk menilai ketepatan model dalam memprediksi setiap kelas
- Recall, untuk mengukur kemampuan model dalam mengenali seluruh data pada masing-masing kelas
- AUC/ROC, untuk mengevaluasi kemampuan pemisahan kelas pada klasifikasi multikelas

Selain metrik performa, dicatat pula waktu training dan waktu testing sebagai bagian dari evaluasi efisiensi komputasi model.

```python
# Cell 14: Evaluasi accuracy, precision, recall, ROC-AUC

# Hitung waktu testing
start_test = time.time()
y_pred = model.predict(X_test_lstm)
end_test = time.time()

testing_time = end_test - start_test

# Prediksi label
y_pred_label = np.argmax(y_pred, axis=1)

# Hitung metrik evaluasi
acc = accuracy_score(y_test_raw, y_pred_label)
prec = precision_score(y_test_raw, y_pred_label, average="macro")
rec = recall_score(y_test_raw, y_pred_label, average="macro")
roc = roc_auc_score(y_test_cat, y_pred, multi_class="ovr")

# Print hasil evaluasi dengan format PROSES 2
print("=== EVALUASI PROSES 3 ===")
print("Akurasi        :", acc)
print("Presisi        :", prec)
print("Recall         :", rec)
print("AUC/ROC        :", roc)
print("Waktu training :", training_time)
print("Waktu testing  :", testing_time)

```

Hasil evaluasi menunjukkan bahwa model LSTM pada Proses Ketiga mampu mencapai:

- Akurasi sebesar 0.874
- Presisi sebesar 0.873
- Recall sebesar 0.873
- Nilai AUC/ROC sebesar 0.975

Dengan waktu training sekitar 19.41 detik dan waktu testing sekitar 0.65 detik, model menunjukkan performa klasifikasi yang baik serta efisiensi komputasi yang memadai.

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
=== EVALUASI PROSES 3 ===
Akurasi        : 0.874
Presisi        : 0.8732421780370443
Recall         : 0.8732039931827296
AUC/ROC        : 0.9749690587385516
Waktu training : 19.407700061798096
Waktu testing  : 0.6529815196990967
```

LAPORAN UJIAN AKHIR SEMESTER KELOMPOK 1

“Progress 4 Cek Missing Value, MinMaxScaler, SMOTE, dan Implementasi Vanilla LSTM”

MATA KULIAH MACHINE LEARNING PRAKTIKUM C-7

Disusun Oleh :

434231020	**Faizatun Ni’mah**
434231021	**Seri Muliani Lubis**
434231048	**Surya Dwi Satria**
434231112	**Festiana Ramaya**

D4 Teknik Informatika

FAKULTAS VOKASI

UNIVERSITAS AIRLANGGA

2025

A. Pendahuluan

Machine Learning merupakan salah satu cabang kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan meningkatkan kinerjanya tanpa diprogram secara eksplisit. Dalam praktiknya, proses pengolahan data menjadi tahap yang sangat penting sebelum model dibangun. Data yang tidak bersih atau tidak seimbang dapat menyebabkan performa model menjadi kurang optimal.

Laporan ini disusun sebagai bagian dari Ujian Akhir Semester pada mata kuliah Machine Learning Praktikum. Fokus pembahasan meliputi pengecekan dan penanganan missing value, normalisasi data menggunakan MinMaxScaler, penyeimbangan data dengan metode SMOTE, serta implementasi model Vanilla LSTM.

B. Dasar Teori

1. Text Classification

   Text classification merupakan proses pengelompokan dokumen teks ke dalam kelas atau kategori tertentu berdasarkan isi teks. Proses ini banyak digunakan pada analisis berita, spam detection, sentiment analysis, dan topic classification. Agar teks dapat diproses oleh model Machine Learning, data teks harus diubah terlebih dahulu menjadi bentuk numerik.

2. TF-IDF (Term Frequency – Inverse Document Frequency)

   TF-IDF adalah metode pembobotan kata yang digunakan untuk mengukur seberapa penting suatu kata dalam sebuah dokumen relatif terhadap seluruh dokumen dalam dataset.

   Rumus TF-IDF adalah sebagai berikut:

Term Frequency (TF)

$$TF(t,d) = \frac{f(t,d)}{\sum f(d)}$$

Inverse Document Frequency (IDF)

$$IDF(t) = \log\left(\frac{N}{df(t)}\right)$$

TF-IDF

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

Keterangan:

C. f(t,d) = frekuensi kata t pada dokumen d
D. N = jumlah seluruh dokumen
E. df(t) = jumlah dokumen yang mengandung kata t

TF-IDF membantu mengurangi pengaruh kata-kata umum dan meningkatkan bobot kata yang lebih informatif.

3. Missing Value

Missing value adalah kondisi ketika data memiliki nilai yang kosong atau tidak tersedia. Keberadaan missing value dapat menyebabkan kesalahan perhitungan dan menurunkan performa model. Oleh karena itu, dilakukan pengecekan missing value menggunakan fungsi statistik sederhana.

Penanganan missing value yang digunakan pada praktikum ini adalah drop missing value, yaitu menghapus data yang memiliki nilai kosong agar kualitas data tetap terjaga.

4. MinMax

MinMaxScaler merupakan teknik normalisasi data yang mengubah nilai fitur ke dalam rentang tertentu, biasanya antara 0 dan 1.
Rumus MinMaxScaler:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Keterangan:

- X = nilai data asli
- X_min = nilai minimum fitur
- X_max = nilai maksimum fitur
- X' = nilai hasil normalisasi

Normalisasi ini penting untuk model neural network seperti LSTM agar setiap fitur memiliki skala yang sama dan proses training menjadi lebih stabil.

5. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE merupakan metode untuk mengatasi ketidakseimbangan kelas dengan cara membuat data sintetis pada kelas minoritas.
Rumus dasar SMOTE:

$$x_{\text{new}} = x_i + \delta(x_{nn} - x_i)$$

Keterangan:

- x_i = data pada kelas minoritas
- x_nn = tetangga terdekat dari x_i
- δ = bilangan acak antara 0 dan 1
- x_new = data sintetis baru

Dengan SMOTE, distribusi kelas menjadi lebih seimbang sehingga model tidak bias terhadap kelas mayoritas.

6. Vanilla

LSTM adalah pengembangan dari Recurrent Neural Network (RNN) yang dirancang untuk mengatasi masalah vanishing gradient. LSTM memiliki tiga gerbang utama, yaitu forget gate, input gate, dan output gate.

Rumus utama pada LSTM:

Forget Gate

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)$$

Input Gate

$$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right)$$

Candidate Cell State

$$\tilde{C}_t = \tanh\left(W_C[h_{t-1}, x_t] + b_C\right)$$

Cell State Update

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Output Gate

$$o_t = \sigma\left(W_o[h_{t-1}, x_t] + b_o\right)$$

Hidden State

$$h_t = o_t \cdot \tanh(C_t)$$

Keterangan:

- x_t : input pada waktu ke-t
- h_t : hidden state
- C_t : cell state
- W : bobot (weight)
- b : bias
- σ : fungsi sigmoid

C. Tujuan

Tujuan dari praktikum Machine Learning ini adalah:

a. Mengetahui dan menganalisis kondisi dataset, khususnya keberadaan missing value sebelum proses pengolahan data lebih lanjut.
b. Menerapkan metode TF-IDF untuk mengubah data teks menjadi representasi numerik agar dapat diproses oleh model Machine Learning.
c. Melakukan normalisasi data menggunakan MinMaxScaler untuk menyamakan skala fitur sehingga proses pelatihan model neural network menjadi lebih stabil.
d. Mengatasi permasalahan ketidakseimbangan kelas pada data training dengan menggunakan metode SMOTE (Synthetic Minority Over-sampling Technique).
e. Mengimplementasikan model Vanilla LSTM sebagai metode klasifikasi teks berbasis neural network.
f. Mengevaluasi performa model yang dibangun menggunakan metrik evaluasi seperti akurasi, presisi, recall, dan AUC/ROC.
g. Menganalisis pengaruh tahapan preprocessing data terhadap hasil klasifikasi yang dihasilkan oleh model LSTM.

D. Dataset

Link Dataset : https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

E. Langkah - Langkah

```python
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE  # Tambahkan ini
from collections import Counter           # Tambahkan ini
from sklearn.model_selection import train_test_split # Jika belum ada

print("Memuat dataset...")

train_df = pd.read_csv("train.csv")
test_df  = pd.read_csv("test.csv")

train_df.columns = ["Class Index", "Title", "Description"]
test_df.columns  = ["Class Index", "Title", "Description"]

train_df["text"] = train_df["Title"] + " " + train_df["Description"]
test_df["text"]  = test_df["Title"]  + " " + test_df["Description"]

# Sampling
train_df = train_df.sample(25000, random_state=42)
test_df  = test_df.sample(2000, random_state=42)

X_train_raw = train_df["text"].values
X_test_raw  = test_df["text"].values
y_train_raw = train_df["Class Index"].values - 1
y_test_raw  = test_df["Class Index"].values - 1
```

```python
# TF-IDF
vectorizer = TfidfVectorizer(
    max_features=4000,
    stop_words="english",
    ngram_range=(1,2)
)
X_train_vec = vectorizer.fit_transform(X_train_raw).toarray()
X_test_vec  = vectorizer.transform(X_test_raw).toarray()

print("TF-IDF selesai.")
print("Shape X_train_vec:", X_train_vec.shape)
print("Shape X_test_vec :", X_test_vec.shape)
```

✓ 8.1s

```
Memuat dataset...
TF-IDF selesai.
Shape X_train_vec: (25000, 4000)
Shape X_test_vec : (2000, 4000)
```

Penjelasan:

1. Bagian awal mengimpor semua modul yang diperlukan untuk tugas pengolahan data dan *machine learning*
2. Data latih dan uji dimuat. Kolom "Title" dan "Description" digabungkan menjadi satu fitur teks ("text").

3. Jumlah data dikurangi menjadi 25.000 sampel latih dan 2.000 sampel uji.
4. Teks mentah (X_raw) dan label kelas yang disesuaikan indeksnya (y_raw) dipisahkan.
5. TfidfVectorizer digunakan dengan batas 4.000 fitur dan mengabaikan *stop words* Bahasa Inggris, serta mempertimbangkan kata tunggal dan pasangan kata (*unigram* dan *bigram*).
6. Teks diubah menjadi **Matriks TF-IDF** (Term Frequency-Inverse Document Frequency).

   Dimensi Output: Data latih dan uji kini memiliki **4.000 fitur** (kolom).

```
# MISSING VALUE
print("Missing value train:\n", train_df.isnull().sum())
print("\nMissing value test:\n", test_df.isnull().sum())

train_df = train_df.dropna()
test_df  = test_df.dropna()

print("\nSetelah drop missing:")
print("Train:", train_df.shape)
print("Test :", test_df.shape)
```

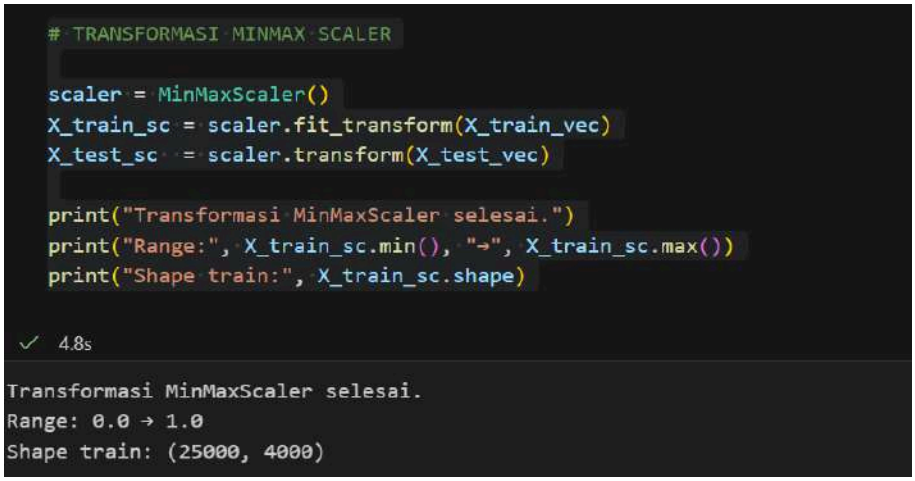
```
Missing value train:
 Class Index    0
Title           0
Description     0
text            0
dtype: int64

Missing value test:
 Class Index    0
Title           0
Description     0
text            0
dtype: int64

Setelah drop missing:
Train: (25000, 4)
Test : (2000, 4)
```

Penjelasan:

1. Menggunakan .isnull().sum() untuk menghitung jumlah nilai hilang di setiap kolom data pelatihan (train_df) dan pengujian (test_df).
2. Semua kolom (Class Index, Title, Description, text) menunjukkan nilai **0**.
3. Fungsi .dropna() dipanggil untuk menghapus baris yang mengandung nilai hilang.



Penjelasan:

Menggunakan MinMaxScaler() untuk menskalakan fitur.

- **Data Pelatihan (X_train_vec):** Diterapkan fit_transform(). *Scaler* **mempelajari** nilai minimum dan maksimum (0 hingga 1) dari data pelatihan.
- **Data Pengujian (X_test_vec):** Diterapkan transform() **saja**, menggunakan skala yang dipelajari dari data pelatihan.

```python
# 3. IMPLEMENTASI SMOTE
smote = SMOTE(random_state=42)

print("\n--- Sebelum SMOTE (Data Training) ---")
print(f"Jumlah sampel X_train: {X_train_sc.shape[0]}")
print(f"Distribusi Kelas y_train: {Counter(y_train_raw)}")

X_train_smote, y_train_smote = smote.fit_resample(
    X_train_sc, y_train_raw
)

print("\n--- Setelah SMOTE (Data Training) ---")
print(f"Jumlah sampel X_train_smote: {X_train_smote.shape[0]}")
print(f"Distribusi Kelas y_train_smote: {Counter(y_train_smote)}")
```

```
--- Sebelum SMOTE (Data Training) ---
Jumlah sampel X_train: 25000
Distribusi Kelas y_train: Counter({np.int64(3): 6329, np.int64(1): 6303, np.int64(0): 6215, np.int64(2): 6153})

--- Setelah SMOTE (Data Training) ---
Jumlah sampel X_train_smote: 25316
Distribusi Kelas y_train_smote: Counter({np.int64(2): 6329, np.int64(1): 6329, np.int64(3): 6329, np.int64(0): 6329})
```

Penjelasan:

1. Menggunakan smote.fit_resample() pada data pelatihan (X_train_sc dan y_train_raw).
    - **Sebelum SMOTE:**
        a. Jumlah Sampel Awal: **25.000** (Data Latih).
        b. Distribusi Kelas (Contoh): Terdapat empat kelas (indeks 0, 1, 2, 3). Distribusi awalnya **tidak seimbang** (misalnya, 6329, 6303, 6215, 6153).
    - **Setelah SMOTE:**
        a. Jumlah Sampel Akhir: **25316** (Total sampel bertambah).
        b. Distribusi Kelas Akhir: Semua kelas memiliki jumlah sampel yang **sama** (misalnya, semua menjadi 6329).
2. Menggunakan .isnull().sum() dan .dropna(). Jumlah nilai hilang adalah **0** di semua kolom data latih dan uji.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras.utils import to_categorical
import time
```

```
# LSTM butuh reshape: (samples, time_step, features)
X_train_lstm = X_train_sc.reshape((X_train_sc.shape[0], 1, X_train_sc.shape[1]))
X_test_lstm  = X_test_sc.reshape((X_test_sc.shape[0], 1, X_test_sc.shape[1]))
y_train_cat  = to_categorical(y_train_raw, 4)

def build_lstm(shape):
    model = Sequential()
    model.add(Input(shape=shape))
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dropout(0.4))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(4, activation="softmax"))
    model.compile(
        optimizer="adam",
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

print("Training model LSTM...")

model_lstm = build_lstm((1, X_train_lstm.shape[2]))
```

```
start = time.time()
history = model_lstm.fit(
    X_train_lstm,
    y_train_cat,
    epochs=50,
    batch_size=32,
    verbose=1
)
train_time = time.time() - start
```

```
Epoch 12/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9842 - loss: 0.0433
...
Epoch 49/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 30s 38ms/step - accuracy: 0.9990 - loss: 0.0023
Epoch 50/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 39s 35ms/step - accuracy: 0.9986 - loss: 0.0035
```

Penjelasan:

1. Mengimpor Sequential, LSTM, Dense, Dropout, dan to_categorical dari tensorflow.keras.
2. Label target (y_train_raw, y_test_raw) dikonversi ke format *One-Hot Encoding* (y_train_cat, y_test_cat) menggunakan to_categorical.
3. Fitur input (X_train_sc, X_test_sc) di-*reshape* menjadi format **(sampel, *time_step*, fitur)**, yang diperlukan oleh arsitektur LSTM.

4. Input dan dua lapisan **LSTM** (Long Short-Term Memory) untuk memproses urutan (sekuens) data.
5. Lapisan **Dropout** (0.4) untuk mencegah *overfitting*.
6. Lapisan **Dense** (64) dengan aktivasi 'relu'.
7. Lapisan **Dense** terakhir (4 *output*) dengan aktivasi **'softmax'** (cocok untuk klasifikasi multi-kelas). * **Kompilasi Model:** Model dikompilasi dengan *optimizer* **adam** dan fungsi kerugian **categorical_crossentropy**.
8. Model dilatih menggunakan data yang telah di-*SMOTE* (X_train_lstm, y_train_cat) selama **50 *epochs*** dengan batch_size=32.
9. **Hasil Pelatihan (Epoch Terakhir):**
    - accuracy: **0.9986** (sangat tinggi)
    - loss: **0.0035** (sangat rendah)

F. Evaluasi

```python
# EVALUASI
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

start_test = time.time()
y_prob = model_lstm.predict(X_test_lstm)
test_time = time.time() - start_test

y_pred = np.argmax(y_prob, axis=1)

acc = accuracy_score(y_test_raw, y_pred)
prec = precision_score(y_test_raw, y_pred, average="weighted", zero_division=0)
rec = recall_score(y_test_raw, y_pred, average="weighted", zero_division=0)
auc = roc_auc_score(to_categorical(y_test_raw, 4), y_prob, multi_class="ovr")

print("\n=== EVALUASI PROSES 4 ===")
print("Akurasi :", acc)
print("Presisi :", prec)
print("Recall  :", rec)
print("AUC/ROC :", auc)
print("Waktu training :", train_time)
print("Waktu testing  :", test_time)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

=== EVALUASI PROSES 4 ===
Akurasi : 0.871
Presisi : 0.8710797376726673
Recall  : 0.871
AUC/ROC : 0.9706378625115468
Waktu training : 1664.164442539215
Waktu testing  : 0.6033868789672852
```

Proses analisis ini merupakan alur kerja lengkap klasifikasi teks menggunakan *Deep Learning* (LSTM), dimulai dari pra-pemrosesan hingga evaluasi model. Tahap awal melibatkan **pembersihan data**, di mana diverifikasi bahwa data latih (25.000 sampel) dan data uji (2.000 sampel) **bebas dari nilai hilang** (*missing values*), sehingga tidak ada baris yang dihapus. Selanjutnya, fitur numerik yang telah diekstraksi menggunakan TF-IDF dinormalisasi ke rentang **0 hingga 1** menggunakan MinMaxScaler untuk menstandarisasi skala fitur. Setelah normalisasi, ketidakseimbangan kelas ditangani menggunakan **SMOTE**, yang berhasil meningkatkan total sampel latih menjadi **25.316** dan menyeimbangkan distribusi di antara empat kelas.

Data yang telah bersih dan seimbang kemudian dipersiapkan untuk model *Deep Learning* dengan mengubah label target menjadi *One-Hot Encoding* dan me-*reshape* fitur input ke format sekuensial yang dibutuhkan oleh LSTM. Model tersebut dibangun dengan arsitektur sekuensial yang mencakup dua lapisan **LSTM** untuk menangani urutan

fitur, diikuti oleh lapisan **Dropout** (0.4) untuk mencegah *overfitting*, dan diakhiri dengan lapisan *Dense* aktivasi **'softmax'** untuk klasifikasi empat kelas.

Model dilatih selama 50 *epochs* menggunakan *optimizer* adam dan *loss function* categorical_crossentropy. Hasil pelatihan menunjukkan kinerja yang sangat baik pada data latih, mencapai **akurasi 0.9986** dan *loss* 0.0035.

Terakhir, model dievaluasi pada data uji yang belum pernah dilihat. Meskipun akurasi pelatihan sangat tinggi, metrik evaluasi pada data uji menunjukkan **Akurasi, Presisi, dan *Recall* sebesar 0.871**, dengan skor **AUC/ROC 0.970**. Hasil ini mengkonfirmasi bahwa model memiliki kemampuan prediksi yang baik (sekitar 87.1%) pada data baru, dengan waktu pelatihan total **1664 detik** dan waktu pengujian kurang dari satu detik.

G. Kesimpulan

Proyek ini berhasil menerapkan *pipeline* klasifikasi teks menggunakan arsitektur *Deep Learning* **LSTM**. Setelah melalui tahapan pra-pemrosesan data yang krusial—yaitu verifikasi bahwa data **bebas dari nilai hilang** , normalisasi fitur TF-IDF menggunakan **MinMaxScaler** ke rentang 0-1, dan penanganan ketidakseimbangan kelas menggunakan **SMOTE** —model pun dilatih. Model LSTM yang dikompilasi dengan *optimizer* `adam` berhasil mencapai akurasi pelatihan yang sangat tinggi (**0.9986**). Yang paling penting, model menunjukkan kemampuan generalisasi yang baik pada data uji yang tidak terlihat, menghasilkan **Akurasi, Presisi, dan *Recall* sebesar 0.871** , serta skor **AUC/ROC 0.970**. Hasil ini mengonfirmasi bahwa *pipeline* yang dikembangkan efektif, dan model LSTM siap digunakan untuk klasifikasi kategori teks baru dengan tingkat akurasi yang memuaskan.