

TUGAS 1

**INTEGRASI DECISION TREE DAN LSTM UNTUK PREDIKSI STRES
DAN ANALISIS SENTIMEN KESEHATAN MENTAL DENGAN
DATASET “MENTAL HEALTH DATASET” DAN “SENTIMENT
ANALYSIS FOR MENTAL HEALTH”**

LAPORAN

Oleh:

Tim GCD

IF-A PAGI

- **211110347 - CINDY SINTIYA**
- **211110948 - GRACE HELENA HUTAGAOL**
- **211111930 - DAVID BATE'E**



**PROGRAM STUDI S-1
FAKULTAS INFORMATIKA
TEKNIK INFORMATIKA
UNIVERSITAS MIKROSKIL
2024/2025**

DAFTAR ISI

PENDAHULUAN.....	1
Latar Belakang	1
Tujuan.....	1
PEMBAHASAN	3
Detail Dataset.....	3
1. Prediksi Stres.....	3
1.1. Sumber Dataset	3
1.2. Fitur Independen (x)	3
1.3. Fitur Dependen (y).....	4
1.4. Ukuran Dataset.....	4
1.5. Preview Dataset	4
2. Analisis Sentimen Kesehatan Mental	4
2.1. Sumber Dataset	4
2.2. Fitur Independen (x)	4
2.3. Fitur Dependen (y).....	4
2.4. Ukuran Dataset.....	4
2.5. Preview Dataset	4
Preprocessing & Train-Test Split.....	5
1. Prediksi Stres.....	5
1.1. Data Cleaning.....	5
1.2. Feature & Label Encoding.....	6
1.3. Feature Selection.....	6
1.4. Train-Test Splitting	7
2. Analisis Sentimen.....	7
2.1. Data Cleaning.....	7
2.2. Data Balancing.....	8
2.3. Label Encoding	9
2.4. Feature Selection and Lower Casing	9
2.5. Train-Test Splitting	9
2.6. Tokenization.....	10

2.7. Pad Sequences.....	10
2.8. Batch Data Loading	10
Pemilihan Algoritma dan Evaluasi Model	11
1. Prediksi Stres.....	11
1.1. Algoritma Decision Tree.....	11
1.2. Evaluasi Model Decision Tree	11
1.3. Hyperparameter Tuning	12
2. Analisis Sentimen.....	14
2.1. Algoritma LSTM	14
2.2. Struktur Model LSTM	14
2.3. Evaluasi Model LSTM.....	15
PENGEMBANGAN APLIKASI	17
Requirements	17
1. Versi Notebook.....	17
2. Versi Aplikasi <i>Web (end-user)</i>	18
3. Versi Aplikasi <i>Mobile (end-user)</i>	18
User Interface.....	19
1. Aplikasi <i>Web</i>	19
2. Aplikasi <i>Mobile</i>	21
PENUTUP	23
Kesimpulan	23
Rekomendasi.....	23
Link Folder.....	23
Link Notebook	23
Link GitHub/ Source Code Lengkap	23

PENDAHULUAN

Latar Belakang

Latar belakang penelitian ini didasari oleh meningkatnya tantangan kesehatan mental di seluruh dunia, dimana tingkat stres dan kondisi mental lainnya terus meningkat secara signifikan. Dalam beberapa tahun terakhir, perubahan sosial yang cepat, tekanan lingkungan, dan dampak luas dari pandemi COVID-19 yang berkontribusi pada peningkatan stres juga membuat kebutuhan akan dukungan kesehatan mental yang mudah diakses. Namun, layanan kesehatan mental yang ada masih terbatas, terutama dalam hal alat yang dipersonalisasi dan berbasis data yang mudah diakses oleh masyarakat umum.

Pendekatan tradisional untuk memantau kesehatan mental biasanya bergantung pada penilaian berkala oleh profesional, yang cakupannya bisa terbatas dan kurang efisien untuk dilakukan. Dengan kemajuan *Machine Learning*, terdapat peluang untuk memanfaatkan data dari interaksi harian, seperti posting media sosial atau pengumpulan formulir, guna mendapatkan wawasan berkelanjutan tentang kondisi mental seseorang. Melalui integrasi prediksi stres menggunakan Decision Tree dan analisis sentimen menggunakan LSTM, penelitian ini bertujuan untuk menjembatani kesenjangan antara dukungan kesehatan mental klinis dan pelacakan kesejahteraan mental sehari-hari.

Pemilihan algoritma ini juga dilakukan dengan pertimbangan khusus, yaitu Decision Tree yang menawarkan ketahanan dalam menangani data terstruktur terkait stres, sementara LSTM yang efektif untuk data sekuensial, sehingga ideal untuk analisis sentimen dalam data teks. Dengan menggabungkan kedua pendekatan ini, penelitian ini berupaya menciptakan alat yang komprehensif, yang tidak hanya memprediksi tingkat stres tetapi juga memberikan wawasan lebih dalam tentang tren kesehatan mental, sehingga memfasilitasi intervensi dini dan mendukung perawatan kesehatan mental yang proaktif.

Tujuan

1. Memprediksi apakah seseorang mengalami stres atau tidak melalui beberapa variabel, seperti pekerjaan, sosialisasi, riwayat gangguan mental sebelumnya, dan sebagainya.
2. Memprediksi seberapa besar kemungkinan seseorang mengalami gangguan mental, seperti depresi, gangguan kecemasan, kecenderungan bunuh diri, dan sebagainya.

3. Menggabungkan kedua model dalam 1 aplikasi untuk membantu pengguna mendeteksi adanya stres ataupun gangguan kesehatan mental bagi mereka yang takut berkonsultasi langsung dengan orang lain ataupun ahli medis profesional.

PEMBAHASAN

Detail Dataset

Sumber dataset utama yang digunakan berasal dari kaggle. Dataset tambahan juga sudah dicari dari *website-website* lain untuk menyeimbangkan beberapa kelas data pada dataset “sentiment.csv”, namun ternyata dataset yang di kaggle sudah gabungan dari banyak *website* sehingga diputuskan untuk menggunakan dataset dari kaggle saja.

1. Prediksi Stres

1.1. Sumber Dataset

kaggle.com/datasets/bhavikjikadara/mental-health-dataset

1.2. Fitur Independen (x)

<i>Timestamp</i>	: waktu pengiriman form
<i>Gender</i>	: jenis kelamin (<i>Male/ Female</i>)
<i>Country</i>	: negara
<i>Occupation</i>	: pekerjaan (<i>Business/ Corporate/ Housewife/ Student/ Other</i>)
<i>self_employed</i>	: wiraswasta (<i>Yes/ No</i>)
<i>family_history</i>	: apakah ada anggota keluarga dengan riwayat gangguan mental (<i>Yes/ No</i>)
<i>treatment</i>	: apakah sudah pernah mencari pengobatan untuk masalah gangguan kesehatan mental (<i>Yes/ No</i>)
<i>Days_Indoors</i>	: sudah berapa lama <i>stay</i> dirumah saja (<i>Go out Every day/ 1-14 days/ 15-30 days/ 31-60 days/ More than 2 months</i>)
<i>Changes_Habits</i>	: perubahan kebiasaan (<i>Yes/ No/ Maybe</i>)
<i>Mental_Health_History</i>	: riwayat gangguan mental (<i>Yes/ No/ Maybe</i>)
<i>Mood_Swings</i>	: perubahan mood/ suasana hati (<i>Low/ Medium/ High</i>)
<i>Coping_Struggles</i>	: apakah berjuang/ berusaha mengatasinya (<i>Yes/ No</i>)

Work_Interest : sangat terikat dengan pekerjaan (*Yes/ No/ Maybe*)

Social_Weakness : lemah dalam bersosialisasi (*Yes/ No/ Maybe*)

mental_health_interview : apakah akan menceritakan terkait masalah kesehatan saat *interview* kerja (*Yes/ No/ Maybe*)

care_options : kepedulian terhadap masalah (*Yes/ No/ Not Sure*)

1.3. Fitur Dependen (y)

Growing_Stress (*Yes/ No/ Maybe*)

1.4. Ukuran Dataset

292.364 baris data, 17 kolom fitur

1.5. Preview Dataset

```
raw_data.sample(frac = 1).head()
```

Timestamp	Gender	Country	Occupation	self_employed	family_history	treatment	Days_Indoors	Growing_Stress	Changes_habits	Mental_health_history	Mood_Swings	Coping_Struggles	Work_Interest	Social_Weakness	mental_health_interview	Care_Options	
217345	02/27/2014 11:35	Male	United States	Business	No	Yes	Yes	1-14 days	Maybe	Yes	No	Medium	No	No	Yes	No	Yes
268646	02/27/2014 18:41	Male	New Zealand	Student	No	No	Yes	Go out Every day	Maybe	No	Yes	High	No	No	Yes	Maybe	No
174353	02/27/2014 12:33	Male	Canada	Housewife	No	No	Yes	1-14 days	No	Maybe	No	Low	No	Yes	Yes	No	Not sure
15649	02/27/2014 16:14	Female	United Kingdom	Student	No	No	Yes	31-60 days	Maybe	Yes	No	Low	Yes	Yes	Yes	No	Yes
39140	02/27/2014 21:54	Female	United States	Student	No	No	No	1-14 days	Maybe	Maybe	No	High	No	Maybe	Yes	Maybe	Not sure

2. Analisis Sentimen Kesehatan Mental

2.1. Sumber Dataset

[kaggle.com/datasets/suchintikasarkar/sentiment-analysis-for-mental-health](https://www.kaggle.com/datasets/suchintikasarkar/sentiment-analysis-for-mental-health)

2.2. Fitur Independen (x)

statement : sentimen dalam bentuk teks kalimat

2.3. Fitur Dependen (y)

status (*Normal/ Depression/ Suicidal/ Anxiety/ Stress/ Bipolar/ Personality disorder*)

2.4. Ukuran Dataset

53.043 baris data, 2 kolom fitur (+ 1 kolom *index*)

2.5. Preview Dataset

```
df.sample(frac = 1).head()
```

Total data (rows): 52681

Unnamed: 0	statement	status
4908	4908 planning	Normal
24088	24088 For want of connection we seek each other out!...	Suicidal
49431	49431 Can stress cause things like this? \nOnly meds...	Stress
43565	43565 is going to school to do dt	Normal
47359	47359 I can't handle the emotional ping pong of feel...	Depression

Preprocessing & Train-Test Split

1. Prediksi Stres

1.1. Data Cleaning

Pembersihan data adalah tahapan pertama yang wajib dilakukan untuk dataset yang sudah dikumpulkan. Untuk dataset “stress.csv” yang diperoleh dari kaggle berisi 292.364 baris data terlihat cukup mencurigakan karena jumlah data yang sangat banyak.

Bagian pertama yang dilakukan adalah membuang baris data yang berisi nilai kosong (*NaN*) karena data tersebut dianggap *invalid* dan akan mengganggu proses *build* model.

Selanjutnya, pengecekan baris data duplikat juga dilakukan untuk membuang data yang sama persis agar mencegah model *overfitting* dengan konsumsi data yang sama berulang kali. Dan hasilnya, hanya tersisa 92.920 baris data valid.

<i>Before</i>	<i>After</i>
<div>Timestamp0</div> <div>Gender0</div> <div>Country0</div> <div>Occupation0</div> <div>self_employed5202</div> <div>family_history0</div> <div>treatment0</div> <div>Days_Indoors0</div> <div>Growing_Stress0</div> <div>Changes_Habits0</div> <div>Mental_Health_History0</div> <div>Mood_Swings0</div> <div>Coping_Struggles0</div> <div>Work_Interest0</div> <div>Social_Weakness0</div> <div>mental_health_interview0</div> <div>care_options0</div> <div>dtype: int64</div>	<div>Timestamp0</div> <div>Gender0</div> <div>Country0</div> <div>Occupation0</div> <div>self_employed0</div> <div>family_history0</div> <div>treatment0</div> <div>Days_Indoors0</div> <div>Growing_Stress0</div> <div>Changes_Habits0</div> <div>Mental_Health_History0</div> <div>Mood_Swings0</div> <div>Coping_Struggles0</div> <div>Work_Interest0</div> <div>Social_Weakness0</div> <div>mental_health_interview0</div> <div>care_options0</div> <div>dtype: int64</div>
<div>raw_data.shape</div> <div>(292364, 17)</div>	<div>preprocessed_data.shape</div> <div>(92920, 16)</div>

Bagian terakhir yang dilakukan adalah mengecek apakah adanya ketidakseimbangan (*imbalanced*) kelas data agar mencegah model terlalu condong ke salah satu atau beberapa kelas dengan jumlah data mayoritas dan mengabaikan kelas yang memiliki jumlah data yang lebih sedikit.

Untuk dataset “stress.csv”, jumlah data untuk tiap kelas sudah seimbang, dengan masing-masing kelas lebih kurang 30.000 data sehingga *data balancing* tidak perlu dilakukan lagi.

```
class_counts = preprocessed_data['Growing_Stress'].value_counts()
class_counts
```

```

count
Growing_Stress
Yes      31974
Maybe   31571
No       29375

dtype: int64
```

1.2. Feature & Label Encoding

Model *machine learning* biasanya menerima data input dalam bentuk angka. Dengan data numerik pula, model dapat belajar lebih baik dan memprediksi dengan lebih akurat.

Dataset yang digunakan berisi nilai-nilai kategorikal yang perlu diubah terlebih dahulu ke dalam format numerik, yang akan dilakukan dengan mengkodekan (*encoding*) masing-masing kolom menjadi bentuk bilangan bulat (0, 1, 2, ...), termasuk juga fitur independen (y) yang dilabeli sebagai 1, 2, dan 3.

```
preprocessed_data.head()
```

	Gender	Country	Occupation	self_employed	family_history	treatment	Days_Indoors	Growing_Stress	changes_habits	Mental_health_history	Mood_Swings	Coping_Struggles	Work_Interest	Social_Weakness	Mental_health_interview	Care_Options
3	0	34	1	0	1	1	0	2	1	2	2	0	1	2	0	2
4	0	34	1	0	1	1	0	2	1	2	2	0	1	2	1	2
5	0	25	1	0	0	1	0	2	1	2	2	0	1	2	0	1
6	0	0	1	0	1	1	0	2	1	2	2	0	1	2	1	1
7	0	34	1	0	0	0	0	2	1	2	2	0	1	2	1	0

Mapping for column 'Mood_Swings':

Original	Encoded
0	High
1	Low
2	Medium

Mapping for column 'Social_Weakness':

Original	Encoded
0	Maybe
1	No
2	Yes

Mapping for column 'Days_Indoors':

Original	Encoded
0	1-14 days
1	15-30 days
2	31-60 days
3	Go out Every day
4	More than 2 months

1.3. Feature Selection

Diantara 17 kolom yang ada, kolom “*Growing_Stress*” dipilih sebagai fitur dependen (y) yang akan dijadikan kelas hasil prediksi model dengan nilai *Yes*, *No*, dan *Maybe* untuk menentukan apakah seseorang mengalami stres.

Untuk fitur independen (x) akan dipilih dari 16 kolom yang tersisa. Kolom *Timestamp* yang merupakan kolom waktu pengumpulan/pengiriman formulir akan dihapus karena dianggap tidak relevan.

Beberapa percobaan juga sudah dilakukan untuk memilih kombinasi kolom-kolom terbaik yang akan dijadikan fitur x, sehingga tersisa kolom-kolom berikut:

<i>Occupation</i>	: pekerjaan (<i>Business/ Corporate/ Housewife/ Student/ Other</i>)
<i>treatment</i>	: apakah sudah pernah mencari pengobatan untuk masalah gangguan kesehatan mental (<i>Yes/ No</i>)
<i>Days_Indoors</i>	: sudah berapa lama mengurung diri dirumah saja (<i>Go out Every day/ 1-14 days/ 15-30 days/ 31-60 days/ More than 2 months</i>)
<i>Changes_Habits</i>	: perubahan kebiasaan (<i>Yes/ No/ Maybe</i>)
<i>Mental_Health_History</i>	: riwayat gangguan mental (<i>Yes/ No/ Maybe</i>)
<i>Mood_Swings</i>	: perubahan mood/ suasana hati (<i>Low/ Medium/ High</i>)
<i>Coping_Struggles</i>	: apakah berjuang/ berusaha mengatasinya (<i>Yes/ No</i>)
<i>Work_Interest</i>	: sangat terikat dengan pekerjaan (<i>Yes/ No/ Maybe</i>)
<i>Social_Weakness</i>	: lemah dalam bersosialisasi (<i>Yes/ No/ Maybe</i>)

1.4. Train-Test Splitting

Dataset yang sudah bersih kemudian akan dibagi dengan rasio 7:3, di mana 70% data akan dipakai untuk *training* dan sisa 30% lainnya akan dipakai sebagai data *testing* dengan *random state* 1.000 untuk mempertahankan variasi pengacakan setiap kali *code* dijalankan.

2. Analisis Sentimen

2.1. Data Cleaning

Pembersihan data adalah tahapan pertama yang wajib dilakukan untuk dataset yang sudah dikumpulkan. Untuk dataset “sentiments.csv” yang diperoleh dari kaggle berisi 53.043 baris data.

Bagian pertama yang dilakukan adalah membuang baris data yang berisi nilai kosong (*NaN*) karena data tersebut dianggap *invalid* dan akan mengganggu proses *build* model.

Selanjutnya, pengecekan baris data duplikat juga dilakukan untuk membuang data yang sama persis agar mencegah model *overfitting* dengan konsumsi data yang sama berulang kali. Hasilnya, tidak terdapat data duplikat dan jumlah data tersisa 52.681 baris data valid.

<i>Before</i>	<i>After</i>
<code>raw_data.shape</code> (53043, 3)	<code>df.sample(frac = 1).head()</code> Total data (rows): 52681

2.2. Data Balancing

Pengecekan jumlah data tiap kelas dilakukan untuk mencegah model terlalu condong ke salah satu atau beberapa kelas dengan jumlah data mayoritas dan mengabaikan kelas yang memiliki jumlah data yang lebih sedikit, yang juga dapat membuat model *overfitting* dan menurunkan akurasi model.

Hasil pengecekan menunjukkan bahwa jumlah data dari tiap kelas memiliki *range* yang terlalu jauh mulai dari 16.000-an data untuk kelas *Normal* hingga hanya 1.000-an data untuk kelas *Personality disorder* yang menunjukkan bahwa terdapat *imbalanced* data untuk kelas *Anxiety*, *Bipolar*, *Stres*, dan *Personality disorder*.

<i>Before</i>	<i>After</i>
<pre> count status Normal 16343 Depression 15404 Suicidal 10652 Anxiety 3841 Bipolar 2777 Stress 2587 Personality disorder 1077 dtype: int64 </pre>	<pre> count status Normal 16343 Depression 15404 Suicidal 10652 dtype: int64 </pre>

Untuk menyeimbangkan data tiap kelas, salah satu cara yang dipilih adalah *Filtering* atau *Class Removal*, dengan menghapus data pada kelas minoritas dan menyisakan kelas mayoritas agar model tetap dapat memprediksi dari dataset dengan jumlah yang seimbang.

2.3. Label Encoding

Masing-masing kelas yang masih bersifat kategorikal akan di-*encode* menjadi label numerik dalam bentuk bilangan bulat dengan label 0 untuk *Normal*, 1 untuk *Depression*, dan 2 untuk *Suicidal*.

	Class Name	Value
0	Normal	0
1	Depression	1
2	Suicidal	2

Proses *encoding* dilakukan secara manual bukan menggunakan *library* karena hasil *training* menunjukkan bahwa akurasi model dengan *encoding* manual lebih bagus.

2.4. Feature Selection and Lower Casing

Kolom yang akan dijadikan sebagai label/ kelas menggunakan kolom *Status*. Maka tersisa kolom *statement* dan *Unnamed:0* (merupakan kolom *index*), sehingga fitur independen (x) hanya akan menggunakan kolom *statement*.

Semua data teks yang terdapat pada kolom *statement* juga perlu disamakan terlebih dahulu huruf besar kecilnya.

```
x_data = df['statement'].apply(lambda x: str(x).lower())
y_data = df['status']
```

2.5. Train-Test Splitting

Tahapan *preprocessing* data belum selesai, namun pembagian data *training* dan *testing* dilakukan terlebih dahulu dengan beberapa pertimbangan sebagai berikut:

- Mempertahankan distribusi data *training* dan *testing*
- Mencegah kebocoran data dimana data *training* dan *testing* akan ditokenisasi secara terpisah

```
x_train, x_test, y_train, y_test = train_test_split(
    x_data.values,
    y_data.values,
    test_size = 0.3,
    random_state = 1000
)
```

Dataset akan dibagi dengan rasio 7:3, di mana 70% data akan dipakai untuk *training* dan sisa 30% lainnya akan dipakai sebagai data *testing* dengan *random state* 1.000 untuk mempertahankan variasi pengacakan setiap kali *code* dijalankan.

2.6. Tokenization

Perlakukan data teks sentimen tentu saja berbeda dengan data teks yang bersifat kategorikal yang bisa langsung dilakukan *label encoding*. Proses tokenisasi perlu dilakukan untuk mengkonversi data teks menjadi rangkaian angka agar dapat diproses oleh model.

```
tokenizer = Tokenizer(oov_token='UNK', lower = True)
tokenizer.fit_on_texts(x_data.values)

x_train_tokenized = tokenizer.texts_to_sequences(x_train)
x_test_tokenized = tokenizer.texts_to_sequences(x_test)
```

Setiap kata yang ditokenisasi dapat diibaratkan sebagai kamus *vocabulary*. Sebagai contoh, kalimat “I feel so happy today” akan ditokenisasi menjadi [2, 15, 1, 37, 8] dimana setiap angka mewakili kata tertentu dalam kamus.

2.7. Pad Sequences

Karena model seringkali mengharapkan input sekuensial dengan panjang data yang sama, *padding* perlu ditambahkan agar ukuran data yang dimasukkan memiliki panjang yang seragam.

Proses *padding* akan menambahkan nilai tambahan (biasa berupa nilai 0) pada data dengan panjang data yang lebih pendek sesuai dengan panjang maksimal yang sudah ditentukan. Panjang maksimal dapat ditentukan dari data sekuensial hasil *tokenizer* yang memiliki jumlah token paling banyak atau ukuran paling panjang.

```
x_train_tokenized_padded = pad_sequences(x_train_tokenized, maxlen = max_len)
x_test_tokenized_padded = pad_sequences(x_test_tokenized, maxlen = max_len)
```

Sebagai contoh, [2, 15, 1, 37, 8] dan [3, 29, 15] akan dimodifikasi menjadi [2, 15, 1, 37, 8] dan [3, 29, 15, 0, 0] jika panjang sekuensial maksimal yang ditentukan adalah 5.

2.8. Batch Data Loading

Batch data loading dilakukan agar model dapat melakukan *training* pada beberapa data sekaligus dalam 1 (satu) *batch* untuk mengurangi dan mengoptimalkan memori dan waktu yang terpakai untuk *training* serta *testing* model.

Terkhusus untuk model yang dibangun dengan pytorch, pembagian *batch* harus dilakukan secara manual (jika di tensorflow, jumlah *batch* bisa langsung dimasukkan pada parameter yang tersedia).

```
batch_size = 24
train_dataloader = DataLoader(CustomizedDataset(x_train_tokenized_padded, y_train), shuffle = True, batch_size = batch_size)
test_dataloader = DataLoader(CustomizedDataset(x_test_tokenized_padded, y_test), shuffle = True, batch_size = batch_size)
```

Pemilihan Algoritma dan Evaluasi Model

1. Prediksi Stres

1.1. Algoritma Decision Tree

Percobaan dilakukan menggunakan beberapa algoritma, antara lain Random Forest, KNN, dan Decision Tree. Dari ketiga model yang dibangun, model Decision Tree dan Random Forest (setelah proses *hyperparameter tuning*) memberikan akurasi yang sama, sehingga diputuskan untuk memilih model Decision Tree karena memiliki keunggulan sebagai berikut:

- Mudah dipahami, divisualisasi, dan diinterpretasikan karena berbasis “*If-Then-Else*”
- Proses *build* dan *training* model lebih cepat karena lebih simpel dan tidak sekompleks model Random Forest
- Cocok untuk data yang memiliki korelasi antar variabel yang non-linear

1.2. Evaluasi Model Decision Tree

Hyperparameter dari model Decision Tree tidak sebanyak dan sekompleks model Random Forest. Percobaan pertama untuk penentuan kedalaman maksimal (*max depth*) dari pohon diambil dengan pertimbangan jumlah kolom/ fitur yang digunakan agar model tidak mengalami *overfitting* akibat model yang terlalu dalam dan kompleks. *Random state* 1.000 digunakan agar model memberikan hasil yang sama setiap kali *code* dijalankan ulang.

```
dectree_model = DecisionTreeClassifier(max_depth=10, random_state=1000)
dectree_model.fit(X_train, y_train)
```

DecisionTreeClassifier

DecisionTreeClassifier(max_depth=10, random_state=1000)

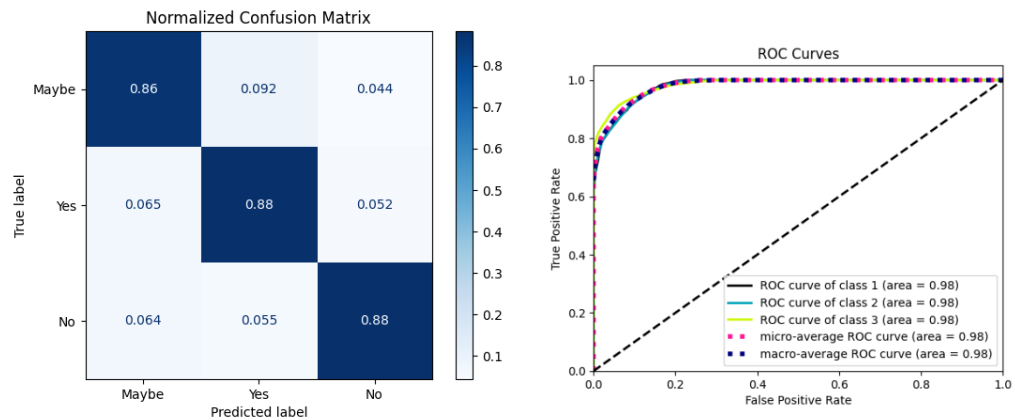
Hasil training dan testing model menunjukkan akurasi 87,57% dengan nilai *precision*, *recall*, dan F1 yang bisa dibilang cukup merata untuk masing-masing kelas data.

```
pd.DataFrame(metrics.classification_report(y_test, dectree_result, target_names = ["Maybe", "No", "Yes"], output_dict=True))
```

	Maybe	No	Yes	accuracy	macro avg	weighted avg
precision	0.873259	0.846843	0.907009	0.875951	0.875704	0.876637
recall	0.863819	0.883016	0.881379	0.875951	0.876071	0.875951
f1-score	0.868514	0.864551	0.894010	0.875951	0.875692	0.876102
support	9436.000000	8779.000000	9661.000000	0.875951	27876.000000	27876.000000

Dilihat dari grafik *Confusion Matrix* dibawah, jumlah kesalahan hasil prediksi dari data aktual juga cukup kecil, walau nilai akurasi

masih belum mencapai target yakni 90%. Nilai ROC juga sudah bagus yang ditunjukkan oleh garis grafik yang melengkung ke bagian kiri atas dengan nilai 0.98.



1.3. Hyperparameter Tuning

Hyperparameter Tuning dilakukan untuk mencari kombinasi *hyperparameter* terbaik untuk model agar dapat memberikan prediksi dengan akurasi terbaik.

```
param_grid = {
    'max_depth': [10, 15, 20, None],
    'random_state': [42, 1000],
}
```

Salah satu *tools* scikit-learn yang dapat digunakan untuk *Hyperparameter Tuning* model secara otomatis adalah GridSearchCV. Terdapat 3 pilihan *max_depth* (kedalaman pohon), 2 pilihan *random_state*, dan 3-fold *cross validation* sehingga terdapat total $6 \times 2 \times 3 = 24$ kombinasi model yang dibangun dan metrik pemilihan model terbaik akan ditentukan berdasarkan akurasi model tertinggi.

Hasilnya, *hyperparameter* terbaik yang dipilih adalah *max_depth* (kedalaman maksimal) pohon sebesar 20 level dan nilai *random_state* di angka 42.

Fitting 3 folds for each of 8 candidates, totalling 24 fits

```
GridSearchCV
├── best_estimator_: DecisionTreeClassifier
│   └── DecisionTreeClassifier
│       └── DecisionTreeClassifier(max_depth=20, random_state=42)
```

Akurasi model meningkat dari 87,59% menjadi 98,24% yang menunjukkan bahwa kombinasi *hyperparameter* hasil tuning sudah sesuai dan akurasi model juga sudah mencapai target (>90%).

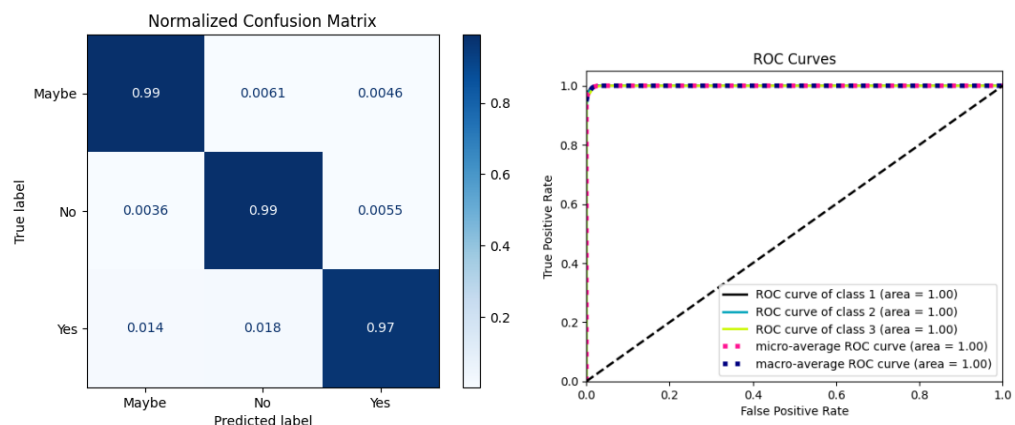
```
pd.DataFrame(metrics.classification_report(y_test, best_model_dt_result, target_names = ["Maybe", "No", "Yes"], output_dict=True))
```

	Maybe	No	Yes	accuracy	macro avg	weighted avg
precision	0.982115	0.974350	0.990363	0.982422	0.982276	0.982528
recall	0.989296	0.990887	0.968016	0.982422	0.982733	0.982422
f1-score	0.985692	0.982549	0.979062	0.982422	0.982435	0.982405
support	9436.000000	8779.000000	9661.000000	0.982422	27876.000000	27876.000000

Selain akurasi, presisi, dan *recall*, perhitungan *error* juga dilakukan pada model hasil *hyperparameter tuning* dengan beberapa metrik. Nilai *R-square* model berada di angka 0.98 atau 98,24% yang menunjukkan bahwa model dapat menjelaskan variasi pada variabel target dengan sangat baik dan fitur-fitur yang dipilih sebagai fitur independen (x) sudah sesuai. Untuk metrik lainnya, perhitungan MAPE juga menunjukkan *error* yang kecil di angka 1,2%.

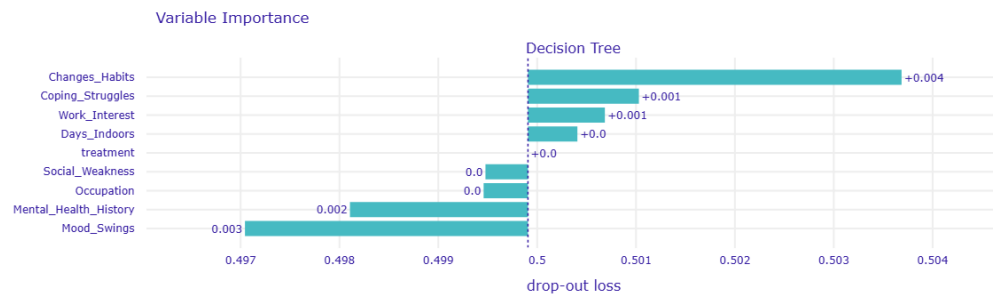
$R^2 = 0.9824221552590041$
 $MAE = 0.024070885349404506$
 $MSE = 0.03705696656622184$
 $RMSE = 0.19250186120196822$
 $MAPE = 1.1945759793370643 \%$

Grafik *Confusion Matrix* dibawah menunjukkan bahwa model sudah dapat memprediksi dengan sangat baik untuk masing-masing kelas dan tidak condong ke salah satu kelas karena data yang diinput sudah *balanced*. Nilai ROC juga naik menjadi 1.00 yang menunjukkan model sudah dapat mengklasifikasikan setiap kelas data dengan baik dan tepat.



Beberapa variabel yang berperan penting dalam prediksi menurut model yang sudah dibangun antara lain: *Changes_Habits* (perubahan kebiasaan akhir-akhir ini), *Coping_Struggles* (ada tidaknya keinginan untuk menghadapi dan mengatasi masalah mental yang dihadapi), *Work_Interest* (seberapa terikat dengan pekerjaan), dan *Days_Indoors* (sudah berapa lama berada atau “mengurung diri” dirumah saja), yang jika dipikirkan secara logika

cukup masuk akal karena fitur tersebut dapat untuk menjawab apakah seseorang mengalami stres atau tidak.



2. Analisis Sentimen

2.1. Algoritma LSTM

Algoritma yang paling efektif dalam pemrosesan bahasa alami (NLP) termasuk analisis sentimen adalah LSTM (*Long-Short Term Memory*), yang sudah digunakan secara luas karena kinerjanya dalam menangkap konteks data sekuensial seperti urutan kata dalam teks yang mempengaruhi maknanya pula. Setiap urutan kata mengandung emosi dan pernyataan sentimen, sehingga mekanisme *memory gate* pada LSTM dapat sangat bermanfaat dalam memfokuskan pada bagian teks yang relevan dan meningkatkan sensitivitas terhadap perubahan emosi dan mendukung klasifikasi sentimen yang kuat.

2.2. Struktur Model LSTM

```
SentimentAnalysisModel(
    (embd): Embedding(54741, 128, padding_idx=0)
    (lstm): LSTM(128, 32, bidirectional=True)
    (linear): Linear(in_features=403200, out_features=3, bias=True)
    (dropout): Dropout(p=0.35, inplace=False)
)
```

Model LSTM yang dibangun terdiri dari 4 lapisan, yaitu:

1. **Embedding**
Layer ini menerima input berupa token, dengan ukuran *vocab* 54.741 data token unik yang akan direpresentasikan dalam vektor 128 dimensi. Penambahan *padding* bernilai 0 untuk menyeragamkan ukuran sekuensial.
2. **LSTM**
Layer inti dari model yang akan menerima data sekuensial berdimensi 128, memiliki 32 unit *neuron cell* pada setiap *hidden layer* LSTM, dan diproses secara *forward* dan *backward* (*bidirectional*).

3. Linear

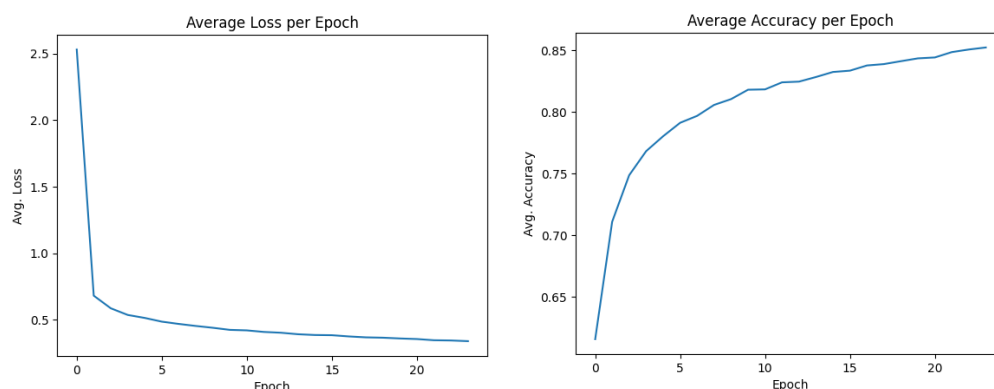
Disebut juga *Fully Connected* layer. *Layer* ini menerima input berukuran 403.200 (2x32x6300) yang merupakan hasil *Flattened* dari *layer* LSTM sebelumnya. Hasil output dari *layer* ini akan berada diantara 3 kelas, yaitu *Yes/ No/ Maybe*.

4. Dropout

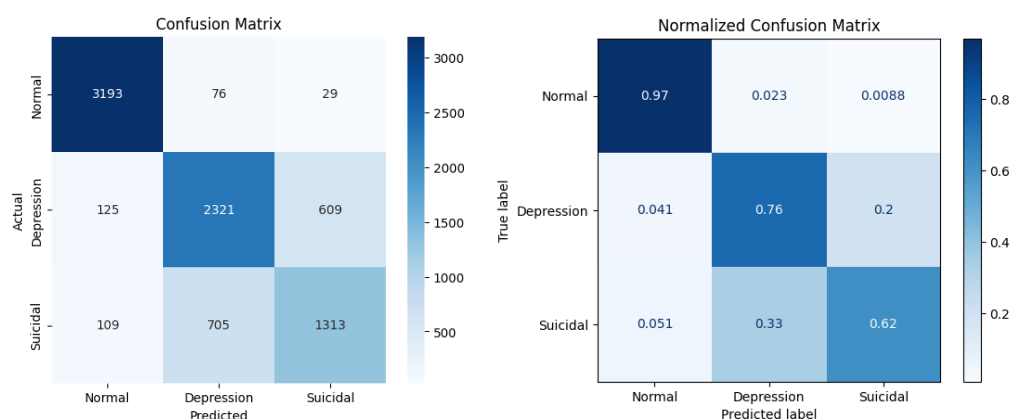
Untuk mencegah terjadinya *overfitting* selama proses *training* model, penambahan *layer* Dropout dilakukan dengan pembuangan 35% unit *neuron*.

2.3. Evaluasi Model LSTM

Setelah di-*looping* sebanyak 24 *epochs* untuk proses *training*, nilai *loss* model menurun drastis pada *epoch* ke-3 dan nilai akurasi model meningkat hingga 85,23% di akhir *epoch*. Bisa dilihat bahwa model belajar dengan cukup baik selama proses *training*.



Grafik *Confusion Matrix* menunjukkan lemahnya model dalam memprediksi label *Depression* dan *Suicidal* dengan tepat walaupun *imbalanced* data sudah ditangani di tahapan *preprocessing*, dimana masih terdapat *range* sekitar 6.000 data antara data berlabel *Normal* dengan *Suicidal*.



Hal ini juga tentu berdampak pada menurunnya akurasi model karena model cenderung memprediksi label *Depression* dan *Suicidal* secara terbalik satu sama lain. Sedangkan untuk label *Normal*, nilai *precision*, *recall*, dan F1 menunjukkan angka sekitar 95% yang berarti model mampu memprediksi dengan sangat baik.

Nilai akurasi model setelah di-*testing* adalah 80,5% yang masih belum mencapai target akurasi di angka 90%. Nilai akurasi model yang turun dari 85% pada proses training menjadi 80,5% pada proses testing menunjukkan bahwa model tidak terlalu *overfitting*, namun kinerja model masih kurang baik.

```
pd.DataFrame(metrics.classification_report(y_test, y_pred, target_names = classes, output_dict=True))
```

	Normal	Depression	Suicidal	accuracy	macro avg	weighted avg
precision	0.931719	0.748227	0.672988	0.805071	0.784311	0.800718
recall	0.968163	0.759738	0.617301	0.805071	0.781734	0.805071
f1-score	0.949591	0.753939	0.643943	0.805071	0.782491	0.802441
support	3298.000000	3055.000000	2127.000000	0.805071	8480.000000	8480.000000

PENGEMBANGAN APLIKASI

Requirements

1. Versi Notebook

Aplikasi yang dibangun membutuhkan model yang di-*generate* dari *notebook* agar mempercepat proses prediksi tanpa perlu melakukan proses *training* setiap kali prediksi dilakukan. Detail dan grafik evaluasi model juga dapat dilihat pada *file notebook* agar lebih rapi dan terstruktur. Berikut langkah-langkahnya:

1. Akses [link notebook Google Colab](#) atau bisa menggunakan *file* “ML_Mental_Health_PT.ipynb” (model dibangun menggunakan *pytorch*) atau “ML_Mental_Health_TF.ipynb” (model dibangun menggunakan *tensorflow*) yang ada di dalam folder “notebook”.
Disarankan untuk menggunakan *file Colab* agar menghemat waktu eksekusi dan *training* model menggunakan GPU yang telah disediakan oleh *Google Colab Compute Engine Backend*.
2. Siapkan API kaggle atau *file* dataset dalam format csv.
Untuk mengambil dataset dapat dilakukan pengunduhan secara manual melalui [link kaggle data stres](#) ataupun menggunakan *file* “stress.csv” untuk prediksi stres dan juga melalui [link kaggle data sentimen](#) ataupun menggunakan *file* “sentiments.csv” untuk analisis sentimen, yang sudah tersedia di folder “data”.
 - > Jika ingin mengunduh secara otomatis dari *notebook*, pastikan sudah menyiapkan API kaggle yang disimpan dalam *file* “kaggle.json”, lalu *upload* ke *runtime notebook* Colab.
 - > Jika menggunakan *file* yang sudah tersedia di folder “data”, pastikan sudah *upload file* dataset pada *runtime notebook* Colab dan melakukan *comment* dan *uncomment* pada *code* sesuai dengan video penjelasan yang ada.
3. Eksekusi 1 per 1 *cell* secara berurutan (opsional).
Jika ingin menggunakan model dari proses ini, pastikan sudah menjalankan semua *cell* (minimal hingga “*Model Evaluation*” untuk *save* keseluruhan model hasil *training* dan digunakan pada aplikasi *web* sebagai *pre-trained model*).
Step ini bersifat opsional karena *file pre-trained model* juga sudah tersedia pada folder “data” dan dapat langsung digunakan pada aplikasi *web*.

2. Versi Aplikasi *Web* (*end-user*)

Model yang sudah diperoleh dari *notebook* akan digunakan sebagai *pre-trained model* untuk mempercepat proses prediksi. Untuk memudahkan pengguna menggunakan aplikasi, model akan diimplementasikan dalam bentuk aplikasi *web* yang lebih ringan, ringkas, dan dapat diakses darimana saja. Berikut langkah-langkah menjalankan aplikasi *web*:

1. Lakukan instalasi *library* yang tercatat dalam file “*requirement.txt*”

Jalankan

```
pip install -r requirement.txt
```

pada terminal dan tunggu hingga instalasi selesai.

Jika menggunakan Anaconda, pastikan *base* conda sudah aktif dengan menjalankan perintah berikut:

```
conda activate base
```

atau

```
C:/Users/<device name>/anaconda3/Scripts/activate
```

(sesuaikan dengan lokasi Anaconda perangkat Anda)

lalu jalankan:

```
conda install Flask Jinja2 flask-cors numpy pandas  
tensorflow torch scikit-learn tqdm deep-translator joblib
```

dan tunggu hingga instalasi selesai.

Atau bisa juga membaca file “*README.md*” yang ada untuk detail instalasi *library*.

2. Jika ingin mengganti model dengan yang baru di-*save* dari *notebook*, pastikan *file* model dan konfigurasi diletakkan di folder yang sama dengan lokasi *file* model awalnya (folder “*data*”) agar aplikasi dapat berjalan tanpa *error*.

3. Jalankan aplikasi *web* dengan perintah

```
python app.py
```

Jika menggunakan Anaconda, pastikan *base* conda sudah aktif sebelum menjalankan aplikasi.

4. Tunggu hingga aplikasi berhasil *running*, kemudian akses <http://127.0.0.1:5000> di *browser* atau sesuaikan dengan *link* yang muncul di terminal.
5. Aplikasi *web* siap digunakan.

3. Versi Aplikasi *Mobile* (*end-user*)

Selain aplikasi *web*, aplikasi *mobile* juga semakin berkembang dan banyak digunakan karena mudah diakses dan tidak perlu menghafal *link* URL aplikasi agar dapat digunakan. Pengembangan aplikasi *mobile* sebagai lanjutan dari aplikasi *web* juga dilakukan agar lebih banyak

pengguna dapat menikmati fitur dari model yang sudah dibangun. Berikut langkah-langkah menjalankan aplikasi *mobile*:

1. Buka terminal (cmd/ powershell), lalu masuk ke folder mobile

```
cd mobile
```

kemudian jalankan perintah

```
flutter pub get
```

2. Untuk dapat menggunakan versi *mobile*, pastikan aplikasi *web* sudah aktif dan berjalan karena akan digunakan sebagai API untuk aplikasi *mobile*.
3. Jalankan aplikasi *mobile* dengan *mode debug* (tekan tombol F5 pada *keyboard*) atau eksekusi perintah

```
flutter run
```

pada terminal, lalu pilih *device* yang akan digunakan untuk *running* aplikasi *mobile* (selain *device* Android, bisa juga menggunakan *browser* seperti Chrome ataupun Edge untuk menjalankan aplikasi).

Jika menggunakan perangkat Android, maka dapat langsung menjalankan perintah

```
flutter install
```

untuk menginstal aplikasi ke perangkat Android Anda (disarankan).

4. Masukkan *IP Address/ IP Host* dari aplikasi *web* yang sudah berjalan (alamat *IP* dapat dilihat pada terminal aplikasi *web*) dengan menekan tombol *Setting* yang ada di bagian sudut kanan atas *interface*, lalu klik tombol “*Save and Try Again*”.
5. Tunggu hingga model selesai di-*fetch* dan di-*load* dari *backend* API *web*, dan aplikasi *mobile* siap digunakan.

User Interface

1. Aplikasi Web

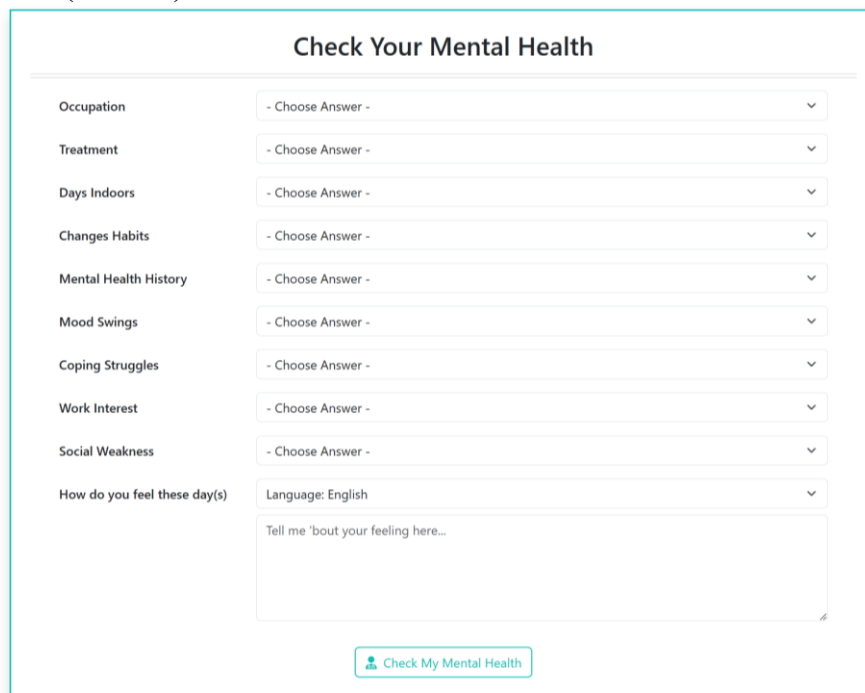
1. Saat pertama kali *website* dibuka, akan ada proses *loading* model *pre-trained*. Hal ini dilakukan agar pada saat proses prediksi, pengguna tidak perlu menunggu model *loading* terus menerus setiap kali *submit* formulir dan bisa langsung mendapatkan hasil prediksi dengan cepat.



We're loading Model for you! Please Wait...

- Setelah model selesai di-load, pengguna akan langsung melihat halaman formulir yang setiap *field*-nya wajib diisi.

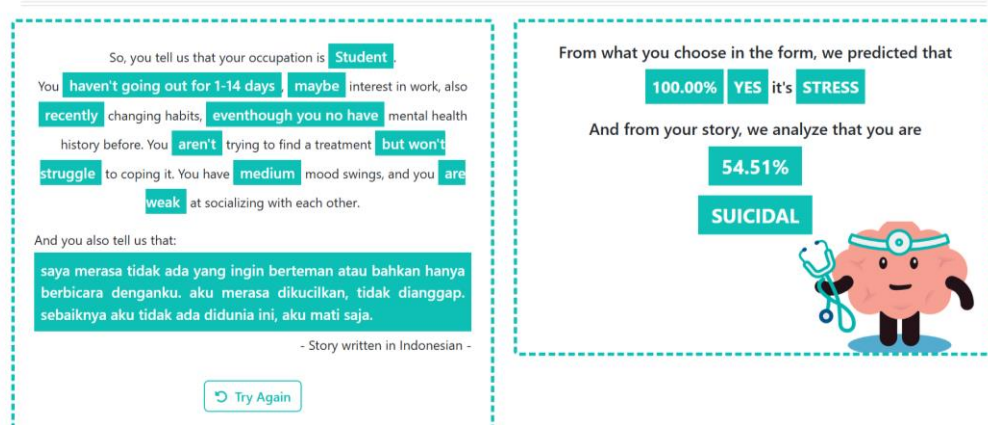
Bagian formulir yang berupa pilihan *dropdown* akan digunakan sebagai data input model prediksi stres (Decision Tree). Pada bagian “How do you feel these day(s)?”, pengguna dapat menceritakan (curhat) terkait perasaan yang akan digunakan sebagai data input model analisis sentimen (LSTM).



Copyright © 2024 - GCD

- Hasil prediksi dapat langsung dilihat setelah pengguna menekan tombol *submit* atau “Check My Mental Health”. Bagian ini terbagi atas 2 prediksi, yaitu stres dan analisis sentimen.

Your Mental Health Analysis Result



Copyright © 2024 - GCD

- Untuk melakukan pengisian formulir lagi, pengguna dapat menekan tombol “*Try Again*” yang terdapat dibagian bawah rangkuman formulir.

2. Aplikasi *Mobile*

1. Saat pertama kali aplikasi *mobile* dibuka, pengguna perlu memasukkan IP Address dari API yang diperoleh dari aplikasi *web* yang sudah dijalankan sebelumnya.

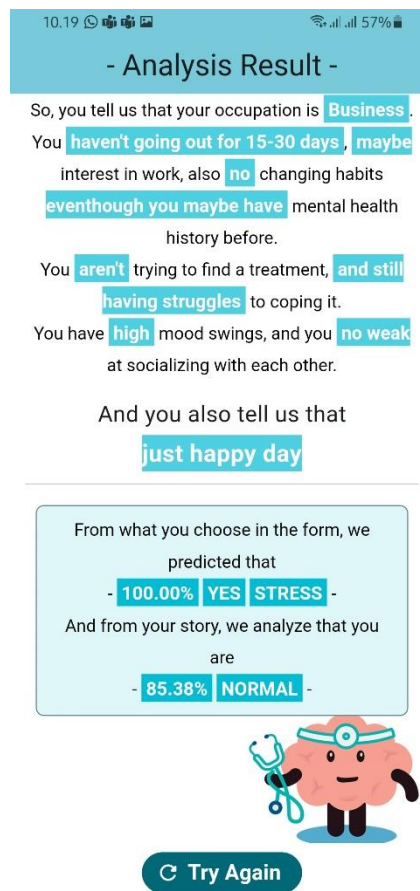
The first screenshot (10:15) shows the app's main screen with a brain character and the text: "Please fill the Host IP from setting for backend API fetching." The second screenshot (10:16) shows a modal for "Host IP Address" with a text input field and a "Save and Try Again" button. The third screenshot (10:27) shows an error message: "Error!!! Server off, wrong host ip, or maybe no internet connection. Please check on either 1 of it or change the Host IP on setting."

Jika IP yang dimasukkan ternyata salah atau server/ aplikasi *web* belum berjalan, pengguna akan mendapatkan *error Timeout* setelah 45 detik atau dapat langsung mengganti IP yang valid pada menu *Setting*.

2. Setelah itu, akan ada proses *load* model jika sebelumnya model belum di-load. Pengguna kemudian diarahkan ke halaman formulir.

The first screenshot (10:16) shows a loading screen with a brain character and the text: "We're loading Model for you! Please Wait...". The second screenshot (10:16) shows the main form titled "- Mental Health Form -" with a welcome message and a list of questions: "What is your occupation?", "Have you ever find a treatment for your mental health issue?", and "Do you have trouble socializing with others?". The third screenshot (10:16) shows the form with a "Submit" button.

3. Sama seperti aplikasi *web*, hasil prediksi dapat langsung dilihat setelah pengguna menekan tombol “*Submit*”. Bagian ini terbagi atas 2 prediksi, yaitu stres dan analisis sentimen.




10.19 57%

- Analysis Result -

So, you tell us that your occupation is **Business**.
You **haven't going out for 15-30 days**, maybe
interest in work, also **no** changing habits
eventhough you maybe have mental health
history before.
You **aren't** trying to find a treatment, **and still**
having struggles to coping it.
You have **high** mood swings, and you **no weak**
at socializing with each other.

And you also tell us that
just happy day

From what you choose in the form, we
predicted that
- **100.00% YES STRESS** -
And from your story, we analyze that you
are
- **85.38% NORMAL** -



[Try Again](#)

4. Untuk melakukan pengisian formulir lagi, pengguna dapat menekan tombol “*Try Again*” yang terdapat dibagian bawah hasil prediksi.

PENUTUP

Kesimpulan

Dari 3 model yang dicoba untuk prediksi stres, yakni Random Forest, KNN, dan Decision Tree, akurasi prediksi tertinggi ada pada model Random Forest dan Decision Tree dengan akurasi yang sama di angka 98,24%. Karena algoritma Decision Tree lebih sederhana, mudah dipahami dan diinterpretasikan, serta sesuai untuk dataset dengan kompleksitas rendah yang memiliki 9 fitur independen, model ini dipilih untuk melakukan prediksi stres karena sudah mampu memberikan hasil yang akurat dengan tingkat akurasi 98,24%.

Sementara itu, model LSTM yang digunakan untuk analisis sentimen masih perlu dilakukan perbaikan dan peningkatan akurasi yang hanya mentok di angka 80%.

Rekomendasi

Untuk analisis sentimen kesehatan mental, sebaiknya menambahkan data aktual ataupun mencari dataset dari sumber lain agar dapat menyeimbangkan data *imbalanced* untuk meningkatkan akurasi model. Eksplorasi teknik pemrosesan teks yang lebih lanjut seperti penggunaan *embeddings* yang lebih spesifik juga dapat dicoba untuk memperbaiki kualitas model, dan juga mencoba algoritma NLP yang lain seperti BERT mungkin bisa meningkatkan akurasi model dalam mengklasifikasikan sentimen.

Link Folder

[Akses folder lengkap \(video & dokumen\) GCD - Mental Health Analysis](#)

Link Notebook

[Akses notebook GCD - Stress Prediction Decision Tree](#)

[Akses notebook GCD - Sentiment Analysis LSTM - Pytorch](#)

[Akses notebook GCD - Sentiment Analysis LSTM - Tensorflow](#)

Link GitHub/ Source Code Lengkap

[Akses GitHub GCD - Mental Health Analysis disini](#)