

LAPORAN TUGAS PERBAIKAN AKURASI KLASIFIKASI MATA KULIAH TEXT MINING & NATURAL LANGUAGE PROCESSING



Tim Penyusun:

1. <5231811022> <Lathif Ramadhan>
2. <5231811029> <Andini Angel Meivita>
3. <5231811033> <Rama Panji Nararendra>
4. <5231811036> <Giffari Riyanda Pradithya>

**PROGRAM STUDI SAINS DATA PROGRAM SARJANA
FAKULTAS SAINS & TEKNOLOGI
UNIVERSITAS TEKNOLOGI YOGYAKARTA
2025**

DAFTAR ISI

DAFTAR ISI.....	2
1a. KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE (Menggunakan 2 Kategori Label: Positive dan Negative).....	4
1. Tulis/Screenshot Import Library yang digunakan.....	4
2. Tulis/screenshot import dataset yang digunakan.....	5
3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan..	6
a. Tulis/screenshot codingnya.....	6
b. Tulis/screenshot hasilnya.....	7
4. Tuning Decision Tree yang digunakan untuk pemodelan (max_depth, min_samples_split, criterion):.....	7
a. Tuliskan/screenshotkan codingnya.....	7
b. Tuliskan/screenshotkan hasil.....	9
5. Tampilkan hasil akurasi dan tabel confusion matrix nya.....	10
a. Akurasi.....	10
b. Confusion Matrix.....	11
6. Tampilkan pohon keputusannya.....	12
Adding: Feature Importances.....	13
1b. KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE (Menggunakan 3 Kategori Label: Positive dan Negative).....	14
1. Tulis/Screenshot Import Library yang digunakan.....	14
2. Tulis/screenshot import dataset yang digunakan.....	15
3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan	16
a. Tulis/screenshot codingnya.....	16
b. Tulis/screenshot hasilnya.....	16
4. Tuning Decision Tree yang digunakan untuk pemodelan (max_depth, min_samples_split, criterion):.....	17
a. Tuliskan/screenshotkan codingnya.....	17
b. Tuliskan/screenshotkan hasil.....	19
5. Tampilkan hasil akurasi dan tabel confusion matrix nya.....	20
a. Akurasi.....	20
b. Confusion Matrix.....	20
6. Tampilkan pohon keputusannya.....	22
2a. PERBAIKAN KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE DENGAN TUNING PARAMETER (Menggunakan 2 Kategori Label: Positive dan Negative).....	25
1. Tulis/Screenshot perintah/coding preprocessing yang digunakan:.....	25
a. Case Folding.....	25
b. Tokenizing (n-gram).....	25
c. Stopword removal dan Stemming.....	25
d. Lemmatization.....	26
e. Tulis/screenshot hasil preprocessing yang dilakukan.....	30
2. Tulis/Screenshot max_depth yang digunakan.....	30
3. Tulis/screenshot min_samples_split yang digunakan.....	31
4. Tulis/screenshot criterion yang digunakan.....	31

5. Tuliskan/screenshot metode validation yang digunakan:.....	32
a. Split-Validation.....	32
b. Cross-Validation.....	33
6. Tuliskan/screenshot feature selection yang digunakan:.....	35
7. Tuliskan/screenshot SMOTE yang digunakan:.....	38
2b. PERBAIKAN KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE DENGAN TUNING PARAMETER (Menggunakan 3 Kategori Label: Positive, Negative, dan Neutral).....	49
1. Tulis/Screenshot perintah/coding preprocessing yang digunakan:.....	49
a. Case Folding.....	49
b. Tokenizing (n-gram).....	49
c. Stopword removal dan Stemming.....	50
d. Lemmatization.....	50
e. Tulis/screenshot hasil preprosesing yang dilakukan.....	54
2. Tulis/Screenshot max_depth yang digunakan.....	55
3. Tulis/screenshot min_samples_split yang digunakan.....	56
4. Tulis/screenshot criterion yang digunakan.....	56
5. Tuliskan/screenshot metode validation yang digunakan:.....	57
a. Split-Validation.....	57
b. Cross-Validation.....	58
6. Tuliskan/screenshot feature selection yang digunakan:.....	60
7. Tuliskan/screenshot SMOTE yang digunakan:.....	63
3a. PROSES KLASIFIKASI MENGGUNAKAN ALGORITMA (Menggunakan 2 Kategori Label: Positive dan Negative).....	76
1. Tulis/Screenshot Import Library yang digunakan.....	76
2. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan.....	79
a. Tulis/screenshot codingnya.....	79
b. Tulis/screenshot hasilnya.....	80
3. Proses pemodelan algoritma Random Forest, SVM, Neural net,	80
a. Tulis/screenshot codingnya.....	80
b. Tulis/screenshot hasilnya.....	84
4. Tampilkan hasil akurasi dan tabel confusion matrix nya.....	86
KESIMPULAN (Menggunakan 3 label {'Positive', 'Negative', 'Neutral'}.....	89
3b. PROSES KLASIFIKASI MENGGUNAKAN ALGORITMA (Menggunakan 3 Kategori Label: Positive, Negative, dan Neutral).....	91
1. Tulis/Screenshot Import Library yang digunakan.....	91
2. Tulis/screenshot import dataset yang digunakan.....	93
3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan.....	94
a. Tulis/screenshot codingnya.....	94
b. Tulis/screenshot hasilnya.....	95
4. Proses pemodelan algoritma Random Forest, SVM, Neural net, Logistic Regression.....	95
a. Tulis/screenshot codingnya.....	95
b. Tulis/screenshot hasilnya.....	99
5. Tampilkan hasil akurasi dan tabel confusion matrix nya.....	100
KESIMPULAN (Menggunakan 3 label {'Positive', 'Negative', 'Neutral'}.....	104
Tabel Kesimpulan Akhir.....	106

1a. KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE (Menggunakan 2 Kategori Label: Positive dan Negative)

1. Tulis/Screenshot Import Library yang digunakan

Jawab:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_graphviz
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    ConfusionMatrixDisplay,
    roc_auc_score,
    roc_curve
)
import matplotlib.pyplot as plt
import seaborn as sns # Untuk visualisasi yang lebih menarik
import graphviz # Untuk visualisasi graphviz
import nltk # Untuk pra-pemrosesan teks lebih lanjut
from nltk.stem import WordNetLemmatizer # Untuk lemmatization
from nltk.corpus import stopwords as nltk_stopwords # Untuk custom stop words

# Download resource NLTK yang mungkin dibutuhkan
# Catch the LookupError directly when nltk.data.find fails
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    print("NLTK resource 'wordnet' not found. Downloading...")
    nltk.download('wordnet')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'wordnet': {e}")

try:
```

```

    nltk.data.find('corpora/omw-1.4')
except LookupError:
    print("NLTK resource 'omw-1.4' not found. Downloading...")
    nltk.download('omw-1.4') # WordNet multilingual resource
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'omw-1.4': {e}")

try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    print("NLTK resource 'stopwords' not found. Downloading...")
    nltk.download('stopwords')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'stopwords': {e}")

# Pengaturan umum untuk plot agar lebih menarik
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis") # Atau palet lain seperti 'pastel', 'muted'

```

2. Tulis/screenshot import dataset yang digunakan

Jawab:

```

path =
'https://raw.githubusercontent.com/LatiefDataVisionary/text-mining-and-natural-
language-processing-college-task/refs/heads/main/datasets/ramadan_labeled_senti-
ment.csv'
df = pd.read_csv(path)

print(f"Dataset berhasil di-load dari: {path}")
print(f"Jumlah baris: {df.shape[0]}, Jumlah kolom: {df.shape[1]}")
df.head(10)

```

Output:

Dataset berhasil di-load dari:
https://raw.githubusercontent.com/LatiefDataVisionary/text-mining-and-natural-language-processing-college-task/refs/heads/main/datasets/ramadan_labeled_sentiment.csv
v
Jumlah baris: 836, Jumlah kolom: 8

	tweet_clean	Tweet	sentiment	sentiment_scores	neg	neu	pos	compound
0	['abraj', 'al', 'bait', 'clock', 'tower', 'bea...]	abraj al bait clock tower beams indicating com...	negative	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...}	0.000	1.000	0.000	0.0000
1	['accounts', 'recognised', 'ramadan', 'none', ...]	accounts recognised ramadan none recognised be...	negative	{'neg': 0.147, 'neu': 0.853, 'pos': 0.0, 'comp...	0.147	0.853	0.000	-0.4767
2	['admin', 'post', 'peaceful', 'ramadan', 'cele...]	admin post peaceful ramadan celebrations east ...	positive	{'neg': 0.0, 'neu': 0.714, 'pos': 0.286, 'comp...	0.000	0.714	0.286	0.4939
3	['admin', 'post', 'ramadan', 'norway']	admin post ramadan norway	negative	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...}	0.000	1.000	0.000	0.0000
4	['admin', 'post', 'ramadan', 'usual', 'peacefu...]	admin post ramadan usual peaceful start englan...	positive	{'neg': 0.0, 'neu': 0.775, 'pos': 0.225, 'comp...	0.000	0.775	0.225	0.4939

3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan

a. Tulis/screenshot codingnya

Jawab:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25, # 25% data digunakan untuk testing
    random_state=42, # Untuk reproduktifitas hasil
    stratify=y # Mempertahankan proporsi kelas sentimen pada data
    training dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train      shape:      {X_train.shape},      y_train      shape:
{y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
```

```
print(y_test.value_counts())
```

b. Tulis/screenshot hasilnya

Jawab:

Ukuran Data Setelah Pembagian:

X_train shape: (627, 937), y_train shape: (627,)

X_test shape: (209, 937), y_test shape: (209,)

Distribusi kelas pada data training (proporsi):

sentiment

1 0.532695

0 0.467305

Name: proportion, dtype: float64

sentiment

1 334

0 293

Name: count, dtype: int64

Distribusi kelas pada data testing (proporsi):

sentiment

1 0.535885

0 0.464115

Name: proportion, dtype: float64

sentiment

1 112

0 97

Name: count, dtype: int64

4. Tuning Decision Tree yang digunakan untuk pemodelan (max_depth, min_samples_split, criterion):

a. Tuliskan/screenshotkan codingnya

Jawab:

```
# Mendefinisikan grid parameter yang akan diuji, dengan rentang yang lebih luas
# dan penambahan class_weight untuk menangani potensi imbalance
param_grid_dt = {
```

```

        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20, 30, 40, 50], # Perluas sedikit
        'min_samples_split': [2, 5, 10, 15, 20], # Perluas sedikit
        'min_samples_leaf': [1, 2, 4, 6, 8], # Perluas sedikit
        'class_weight': [None, 'balanced'] # Untuk menangani imbalance kelas
        # 'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa
        dicoba
    }

dt_model = DecisionTreeClassifier(random_state=42)

grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Gunakan semua core CPU
    verbose=1 # Tampilkan log
)

print("Memulai GridSearchCV untuk Decision Tree...")
grid_search_dt.fit(X_train, y_train)

best_dt_model = grid_search_dt.best_estimator_
print("\nGridSearchCV selesai.")

print("Parameter terbaik yang ditemukan untuk Decision Tree:")
print(grid_search_dt.best_params_)

print(f"\nSkor akurasi cross-validation terbaik untuk Decision Tree:
{grid_search_dt.best_score_:.4f}")

print("\nPenjelasan Terkait Pemilihan Split pada Decision Tree:")
print("Model Decision Tree yang dilatih menggunakan kriteria impurity untuk
menentukan split terbaik pada setiap node.")
print(f"Kriteria impurity yang dipilih oleh GridSearchCV untuk model terbaik
ini adalah: '{best_dt_model.criterion}'.")
if best_dt_model.criterion == 'entropy':
    print("    - Dengan kriteria 'entropy', model bertujuan untuk
memaksimalkan Information Gain.")
    print("    - Information Gain mengukur pengurangan ketidakpastian setelah
dataset di-split berdasarkan sebuah atribut.")
    print("    - Dihitung sebagai:  $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$ .")
elif best_dt_model.criterion == 'gini':
    print("    - Dengan kriteria 'gini', model bertujuan untuk meminimalkan
Gini Impurity.")

```



```

print("    - Gini Impurity mengukur probabilitas kesalahan klasifikasi
jika sebuah elemen acak dari set labelnya ditebak secara acak sesuai
distribusi label di set tersebut.")
print("    - Dihitung sebagai:  $Gini(D) = 1 - \sum (p_i)^2$ ." )
print("Algoritma seperti ID3, C4.5 (menggunakan Information Gain atau Gain
Ratio), dan CART (menggunakan Gini Index) adalah implementasi dari konsep
ini.")
print("Scikit-learn mengimplementasikan versi optimasi dari algoritma CART
untuk Decision Trees.")

```

b. Tuliskan/screenshotkan hasil

Jawab:

Memulai GridSearchCV untuk Decision Tree...

Fitting 5 folds for each of 600 candidates, totalling 3000 fits

GridSearchCV selesai.

Parameter terbaik yang ditemukan untuk Decision Tree:

```

{'class_weight': None, 'criterion': 'entropy', 'max_depth': 40,
 'min_samples_leaf': 1, 'min_samples_split': 20}

```

Skor akurasi cross-validation terbaik untuk Decision Tree: 0.7177

Penjelasan Terkait Pemilihan Split pada Decision Tree:

Model Decision Tree yang dilatih menggunakan kriteria impurity untuk menentukan split terbaik pada setiap node.

Kriteria impurity yang dipilih oleh GridSearchCV untuk model terbaik ini adalah: 'entropy'.

- Dengan kriteria 'entropy', model bertujuan untuk memaksimalkan Information Gain.

- Information Gain mengukur pengurangan ketidakpastian setelah dataset di-split berdasarkan sebuah atribut.

- Dihitung sebagai: $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$.

Algoritma seperti ID3, C4.5 (menggunakan Information Gain atau Gain Ratio), dan CART (menggunakan Gini Index) adalah implementasi dari konsep ini.

Scikit-learn mengimplementasikan versi optimasi dari algoritma CART untuk Decision Trees.

5. Tampilkan hasil akurasi dan tabel confusion matrix nya

Jawab:

a. Akurasi

```
# Prediksi pada data testing
y_pred_dt = best_dt_model.predict(X_test)
y_pred_proba_dt = best_dt_model.predict_proba(X_test)[:, 1] # Probabilitas
untuk kelas positif

# Akurasi
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Akurasi Model Decision Tree pada Data Testing: {accuracy_dt:.4f}")

# ROC AUC Score
try:
    roc_auc_dt = roc_auc_score(y_test, y_pred_proba_dt)
    print(f"ROC AUC Score Decision Tree: {roc_auc_dt:.4f}")
except ValueError:
    print("ROC AUC Score tidak dapat dihitung (mungkin hanya satu kelas yang
diprediksi).")

# Laporan Klasifikasi
print("\nLaporan Klasifikasi Decision Tree:")
print(classification_report(y_test, y_pred_dt, target_names=['Negative (0)',
'Positive (1)']))
```

Output

Akurasi Model Decision Tree pada Data Testing: 0.7033

ROC AUC Score Decision Tree: 0.7365

Laporan Klasifikasi Decision Tree:

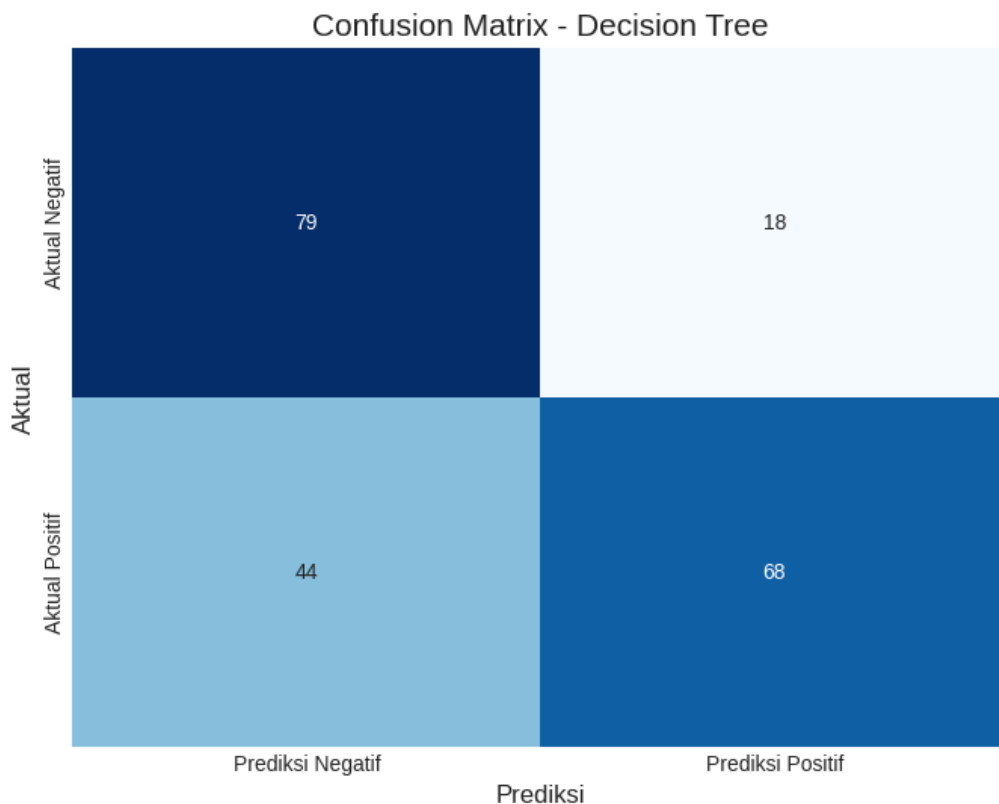
	precision	recall	f1-score	support
Negative (0)	0.64	0.81	0.72	97
Positive (1)	0.79	0.61	0.69	112
accuracy			0.70	209
macro avg	0.72	0.71	0.70	209
weighted avg	0.72	0.70	0.70	209

b. Confusion Matrix

```
# Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Prediksi Negatif', 'Prediksi Positif'],
            yticklabels=['Aktual Negatif', 'Aktual Positif'])
plt.title('Confusion Matrix - Decision Tree', fontsize=15)
plt.ylabel('Aktual', fontsize=12)
plt.xlabel('Prediksi', fontsize=12)
plt.show()

print("\nInterpretasi Confusion Matrix:")
print(f"True Negatives (TN): {cm_dt[0,0]} - Tweet negatif yang diprediksi benar sebagai negatif.")
print(f"False Positives (FP): {cm_dt[0,1]} - Tweet negatif yang salah diprediksi sebagai positif (Type I Error).")
print(f"False Negatives (FN): {cm_dt[1,0]} - Tweet positif yang salah diprediksi sebagai negatif (Type II Error).")
print(f"True Positives (TP): {cm_dt[1,1]} - Tweet positif yang diprediksi benar sebagai positif.")
```

Output



Interpretasi Confusion Matrix:

True Negatives (TN): 79 - Tweet negatif yang diprediksi benar sebagai negatif.

False Positives (FP): 18 - Tweet negatif yang salah diprediksi sebagai positif (Type I Error).

False Negatives (FN): 44 - Tweet positif yang salah diprediksi sebagai negatif (Type II Error).

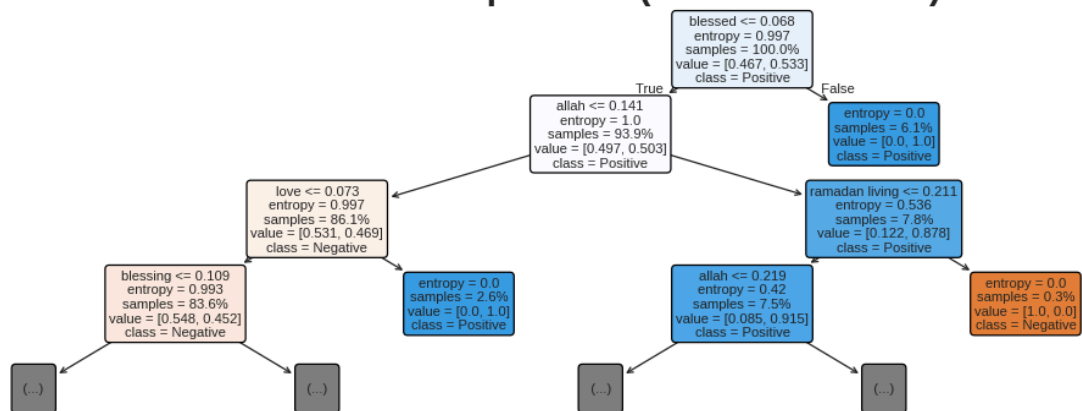
True Positives (TP): 68 - Tweet positif yang diprediksi benar sebagai positif.

6. Tampilkan pohon keputusannya

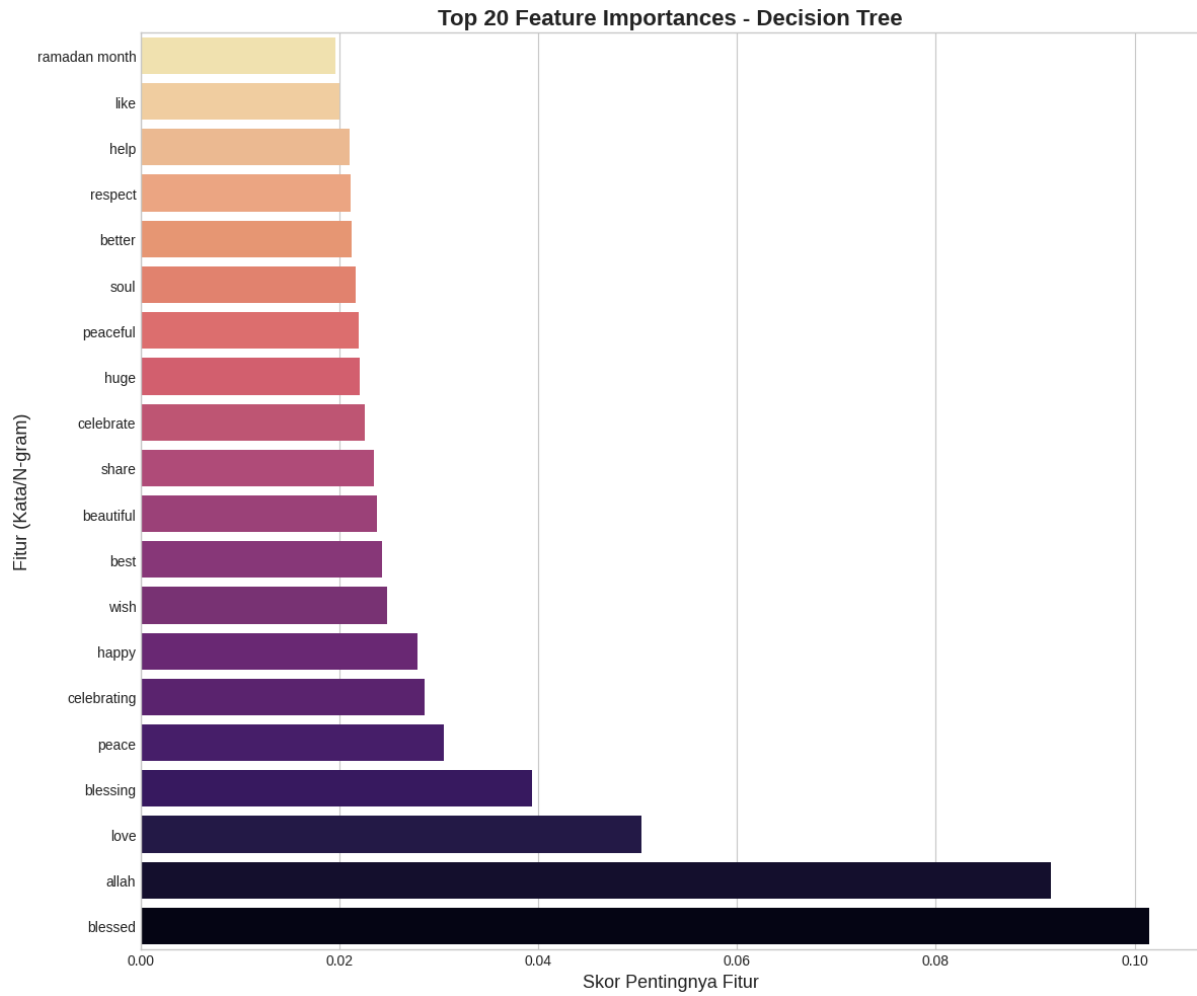
Jawab:

```
plt.figure(figsize=(15, 5), dpi=90) # Tingkatkan ukuran dan DPI untuk detail
plot_tree(
    best_dt_model,
    filled=True,
    feature_names=tfidf.get_feature_names_out(),
    class_names=['Negative', 'Positive'],
    max_depth=3, # Tampilkan 3 level pertama agar tidak terlalu ramai
    fontsize=9,
    proportion=True,
    rounded=True,
    precision=3, # Tingkatkan presisi
    impurity=True,
    label='all'
)
plt.title('Visualisasi Pohon Keputusan (3 Level Pertama)', fontsize=24,
fontweight='bold')
plt.savefig("decision_tree_plot_tree.png", dpi=300, bbox_inches='tight') # Simpan
gambar
plt.show()
print("Visualisasi dengan plot_tree telah ditampilkan dan disimpan sebagai
'decision_tree_plot_tree.png'")
```

Visualisasi Pohon Keputusan (3 Level Pertama)



Adding: Feature Importances



1b. KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE (Menggunakan 3 Kategori Label: Positive dan Negative)

1. Tulis/Screenshot Import Library yang digunakan

Jawab:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_graphviz
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    ConfusionMatrixDisplay,
    roc_auc_score,
    roc_curve
)
import matplotlib.pyplot as plt
import seaborn as sns # Untuk visualisasi yang lebih menarik
import graphviz # Untuk visualisasi graphviz
import nltk # Untuk pra-pemrosesan teks lebih lanjut
from nltk.stem import WordNetLemmatizer # Untuk lemmatization
from nltk.corpus import stopwords as nltk_stopwords # Untuk custom stop words

# Download resource NLTK yang mungkin dibutuhkan
# Catch the LookupError directly when nltk.data.find fails
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    print("NLTK resource 'wordnet' not found. Downloading...")
    nltk.download('wordnet')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'wordnet': {e}")

try:
    nltk.data.find('corpora/omw-1.4')
except LookupError:
    print("NLTK resource 'omw-1.4' not found. Downloading...")
```

```

nltk.download('omw-1.4') # WordNet multilingual resource
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'omw-1.4': {e}")

try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    print("NLTK resource 'stopwords' not found. Downloading...")
    nltk.download('stopwords')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading 'stopwords': {e}")

# Pengaturan umum untuk plot agar lebih menarik
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis") # Atau palet lain seperti 'pastel', 'muted'

```

2. Tulis/screenshot import dataset yang digunakan

Jawab:

```

path=
'https://raw.githubusercontent.com/LatiefDataVisionary/text-mining-and-natural-
language-processing-college-task/refs/heads/main/datasets/data_3_kelas_real.csv'

df = pd.read_csv(path)

print(f"Dataset berhasil di-load dari: {path}")
print(f"Jumlah baris: {df.shape[0]}, Jumlah kolom: {df.shape[1]}")
df.head(10)

```

Output:

```

Dataset          berhasil          di-load          dari:
https://raw.githubusercontent.com/LatiefDataVisionary/text-mining-and-natural-
language-processing-college-task/refs/heads/main/datasets/data_3_kelas_real.cs
v
Jumlah baris: 836, Jumlah kolom: 11

```

	tweet_clean	Tweet	sentiment	sentiment_scores	neg	neu	pos	compound	sentiment_label	text_processed_raw	text_processed
0	['abraj', 'al', 'bait', 'clock', 'tower', 'bea...]	abraj al bait clock tower beams indicating com...	2	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...}	0.000	1.000	0.000	0.0000	neutral	abraj al bait clock tower beams indicating com...	abraj al bait clock tower beam indicating comm...
1	['accounts', 'recognised', 'ramadan', 'none', 'none', ...]	accounts recognised ramadan none recognised be...	2	{'neg': 0.147, 'neu': 0.853, 'pos': 0.0, 'comp...}	0.147	0.853	0.000	-0.4767	neutral	accounts recognised ramadan none recognised be...	account recognised ramadan none recognised beg...
2	['admin', 'post', 'peaceful', 'ramadan', 'cele...]	admin post peaceful ramadan celebrations east ...	2	{'neg': 0.0, 'neu': 0.714, 'pos': 0.286, 'comp...}	0.000	0.714	0.286	0.4939	neutral	admin post peaceful ramadan celebrations east ...	admin post peaceful ramadan celebration east l...
3	['admin', 'post', 'ramadan', 'norway']	admin post ramadan norway	2	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...}	0.000	1.000	0.000	0.0000	neutral	admin post ramadan norway	admin post ramadan norway
4	['admin', 'post', 'ramadan', 'usual', 'peacefu...]	admin post ramadan usual peaceful start englan...	2	{'neg': 0.0, 'neu': 0.775, 'pos': 0.225, 'comp...}	0.000	0.775	0.225	0.4939	neutral	admin post ramadan usual peaceful start englan...	admin post ramadan usual peaceful start englan...

3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan

a. Tulis/screenshot codingnya

Jawab:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25, # 25% data digunakan untuk testing
    random_state=42, # Untuk reproduktifitas hasil
    stratify=y # Mempertahankan proporsi kelas sentimen pada data training
    dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
print(y_test.value_counts())
```

b. Tulis/screenshot hasilnya

Jawab:

Ukuran Data Setelah Pembagian:

X_train shape: (627, 1500), y_train shape: (627,)

X_test shape: (209, 1500), y_test shape: (209,)

Distribusi kelas pada data training (proporsi):

sentiment_label_encoded

1 0.829346

2 0.149920

0 0.020734

Name: proportion, dtype: float64

sentiment_label_encoded

1 520

2 94

0 13

Name: count, dtype: int64

Distribusi kelas pada data testing (proporsi):

sentiment_label_encoded

1 0.827751

2 0.153110

0 0.019139

Name: proportion, dtype: float64

sentiment_label_encoded

1 173

2 32

0 4

Name: count, dtype: int64

4. Tuning Decision Tree yang digunakan untuk pemodelan (max_depth, min_samples_split, criterion):

a. Tuliskan/screenshotkan codingnya

Jawab:

```
# Mendefinisikan grid parameter yang akan diuji, dengan rentang yang lebih luas
# dan penambahan class_weight untuk menangani potensi imbalance
param_grid_dt = {
    'criterion': ['entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
```

```

        'min_samples_split': [2, 5, 10, 15, 20],
        'min_samples_leaf': [1, 2, 4, 6, 8],
        'class_weight': [None, 'balanced'],
        'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa
dicoba
    }

## Kombinasi terbaik:
# {'class_weight': None, 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 1, 'min_samples_split': 15}

dt_model = DecisionTreeClassifier(random_state=42)

grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Gunakan semua core CPU
    verbose=1 # Tampilkan log
)

print("Memulai GridSearchCV untuk Decision Tree...")
grid_search_dt.fit(X_train, y_train)

best_dt_model = grid_search_dt.best_estimator_
print("\nGridSearchCV selesai.")

print("Parameter terbaik yang ditemukan untuk Decision Tree:")
print(grid_search_dt.best_params_)

print(f"\nSkor akurasi cross-validation terbaik untuk Decision Tree:
{grid_search_dt.best_score_:.4f}")

print("\nPenjelasan Terkait Pemilihan Split pada Decision Tree:")
print("Model Decision Tree yang dilatih menggunakan kriteria impurity untuk
menentukan split terbaik pada setiap node.")
print(f"Kriteria impurity yang dipilih oleh GridSearchCV untuk model terbaik
ini adalah: '{best_dt_model.criterion}'.")
if best_dt_model.criterion == 'entropy':
    print("    - Dengan kriteria 'entropy', model bertujuan untuk
memaksimalkan Information Gain.")
    print("    - Information Gain mengukur pengurangan ketidakpastian setelah
dataset di-split berdasarkan sebuah atribut.")

```

```

    print("    - Dihitung sebagai:  $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$ .)")
elif best_dt_model.criterion == 'gini':
    print("    - Dengan kriteria 'gini', model bertujuan untuk meminimalkan Gini Impurity.")
    print("    - Gini Impurity mengukur probabilitas kesalahan klasifikasi jika sebuah elemen acak dari set labelnya ditebak secara acak sesuai distribusi label di set tersebut.")
    print("    - Dihitung sebagai:  $Gini(D) = 1 - \sum (p_i)^2$ .")
print("Algoritma seperti ID3, C4.5 (menggunakan Information Gain atau Gain Ratio), dan CART (menggunakan Gini Index) adalah implementasi dari konsep ini.")
print("Scikit-learn mengimplementasikan versi optimasi dari algoritma CART untuk Decision Trees.")

```

b. Tuliskan/screenshotkan hasil

Jawab:

Memulai GridSearchCV untuk Decision Tree...

Fitting 5 folds for each of 1200 candidates, totalling 6000 fits

GridSearchCV selesai.

Parameter terbaik yang ditemukan untuk Decision Tree:

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
 'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

Skor akurasi cross-validation terbaik untuk Decision Tree: 0.8565

Penjelasan Terkait Pemilihan Split pada Decision Tree:

Model Decision Tree yang dilatih menggunakan kriteria impurity untuk menentukan split terbaik pada setiap node.

Kriteria impurity yang dipilih oleh GridSearchCV untuk model terbaik ini adalah: 'entropy'.

- Dengan kriteria 'entropy', model bertujuan untuk memaksimalkan Information Gain.

- Information Gain mengukur pengurangan ketidakpastian setelah dataset di-split berdasarkan sebuah atribut.

- Dihitung sebagai: $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$.

Algoritma seperti ID3, C4.5 (menggunakan Information Gain atau Gain Ratio), dan CART (menggunakan Gini Index) adalah implementasi dari konsep ini.

Scikit-learn mengimplementasikan versi optimasi dari algoritma CART untuk Decision Trees.

5. Tampilkan hasil akurasi dan tabel confusion matrix nya

Jawab:

a. Akurasi

```
# Prediksi pada data testing
y_pred_dt = best_dt_model.predict(X_test)
y_pred_proba_dt = best_dt_model.predict_proba(X_test)[:, 1] # Probabilitas untuk
kelas positif

# Akurasi
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Akurasi Model Decision Tree pada Data Testing: {accuracy_dt:.4f}")

# ROC AUC Score
try:
    roc_auc_dt = roc_auc_score(y_test, y_pred_proba_dt)
    print(f"ROC AUC Score Decision Tree: {roc_auc_dt:.4f}")
except ValueError:
    print("ROC AUC Score tidak dapat dihitung (mungkin hanya satu kelas yang
diprediksi).")

# Laporan Klasifikasi
print("\nLaporan Klasifikasi Decision Tree:")
print(classification_report(y_test, y_pred_dt, target_names=['Negative (0)',
'Positive (1)', 'Neutral (2)']))
```

Output:

```
Akurasi Model Decision Tree pada Data Testing: 0.8612
ROC AUC Score tidak dapat dihitung (mungkin hanya satu kelas yang diprediksi).

Laporan Klasifikasi Decision Tree:
              precision    recall  f1-score   support

Negative (0)       0.00      0.00      0.00         4
Positive (1)       0.88      0.97      0.92        173
Neutral (2)        0.67      0.38      0.48         32

   accuracy       0.86
  macro avg       0.52
weighted avg       0.83
```

b. Confusion Matrix

```
# Confusion Matrix
```

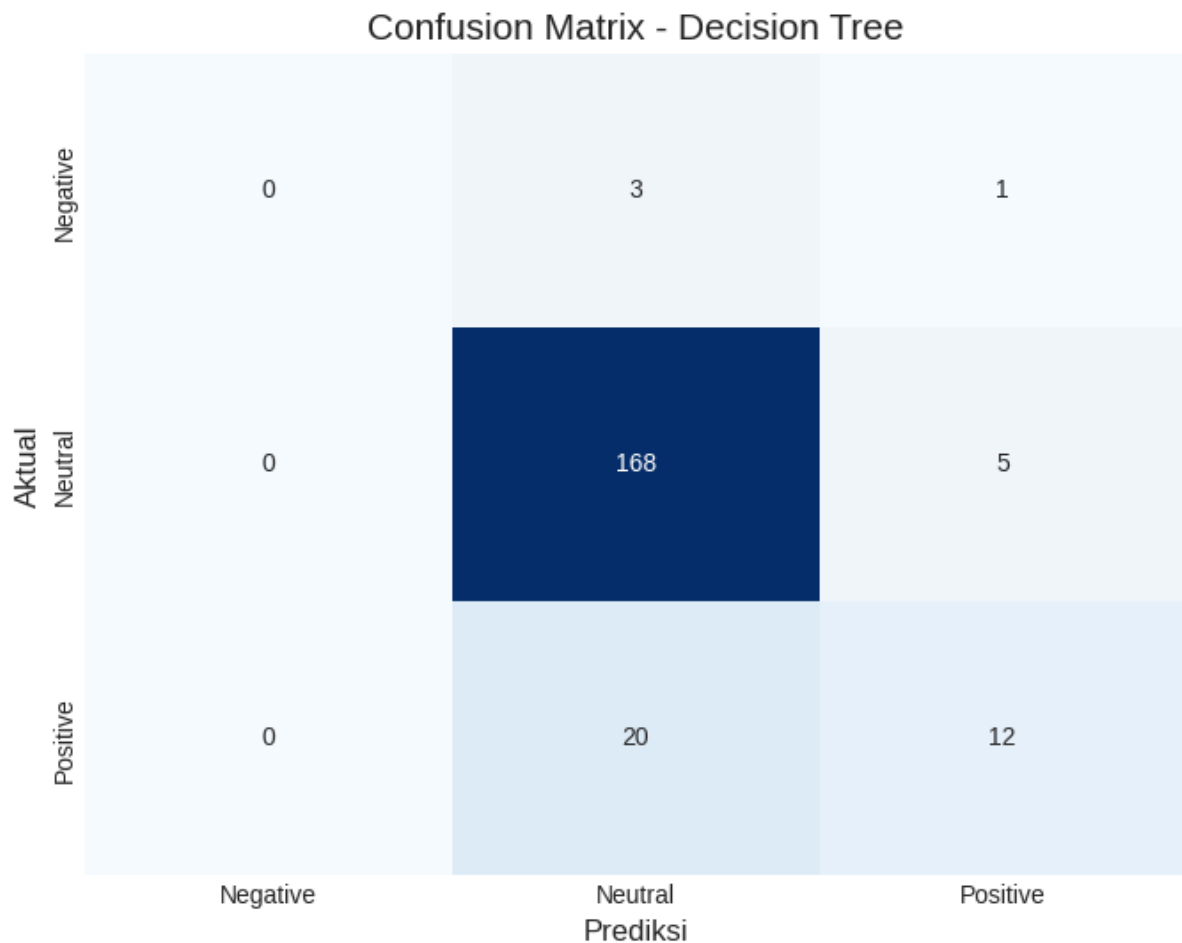
```

cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'], # Updated labels
            yticklabels=['Negative', 'Neutral', 'Positive']) # Updated labels
plt.title('Confusion Matrix - Decision Tree', fontsize=15)
plt.ylabel('Aktual', fontsize=12)
plt.xlabel('Prediksi', fontsize=12)
plt.show()

print("\nInterpretasi Confusion Matrix:")
# Update interpretation based on three classes
print(f"True Negatives (TN): {cm_dt[0,0]} - Tweet negatif yang diprediksi benar sebagai negatif.")
print(f"False Positives (FP): {cm_dt[0,1] + cm_dt[0,2]} - Tweet negatif yang salah diprediksi sebagai neutral atau positif.") # Summing errors for FP
print(f"False Neutrals (FN_Neg): {cm_dt[1,0]} - Tweet neutral yang salah diprediksi sebagai negatif.")
print(f"True Neutrals (TN): {cm_dt[1,1]} - Tweet neutral yang diprediksi benar sebagai neutral.")
print(f"False Positives (FP_Neg): {cm_dt[1,2]} - Tweet neutral yang salah diprediksi sebagai positif.")
print(f"False Negatives (FN_Neg): {cm_dt[2,0] + cm_dt[2,1]} - Tweet positif yang salah diprediksi sebagai negatif atau neutral.") # Summing errors for FN
print(f"True Positives (TP): {cm_dt[2,2]} - Tweet positif yang diprediksi benar sebagai positif.")

```

Output:



Interpretasi Confusion Matrix:

True Negatives (TN): 0 - Tweet negatif yang diprediksi benar sebagai negatif.

False Positives (FP): 4 - Tweet negatif yang salah diprediksi sebagai neutral atau positif.

False Neutrals (FN_Neg): 0 - Tweet neutral yang salah diprediksi sebagai negatif.

True Neutrals (TN): 168 - Tweet neutral yang diprediksi benar sebagai neutral.

False Positives (FP_Neg): 5 - Tweet neutral yang salah diprediksi sebagai positif.

False Negatives (FN_Neg): 20 - Tweet positif yang salah diprediksi sebagai negatif atau neutral.

True Positives (TP): 12 - Tweet positif yang diprediksi benar sebagai positif.

6. Tampilkan pohon keputusannya

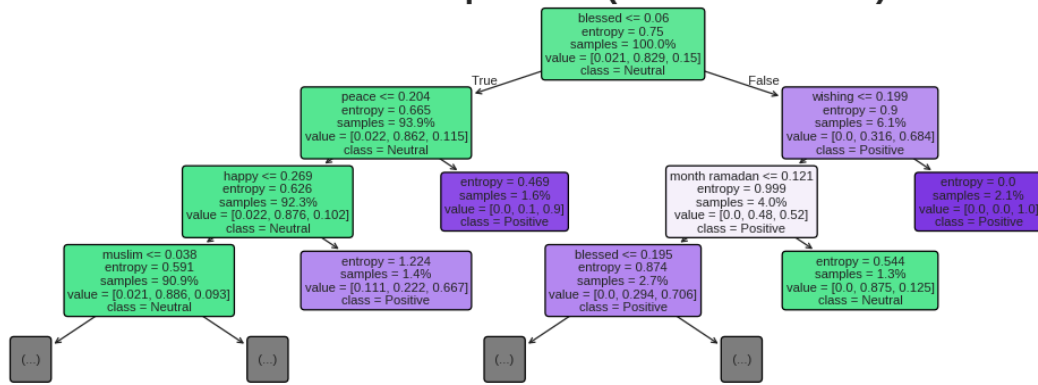
Jawab:

```

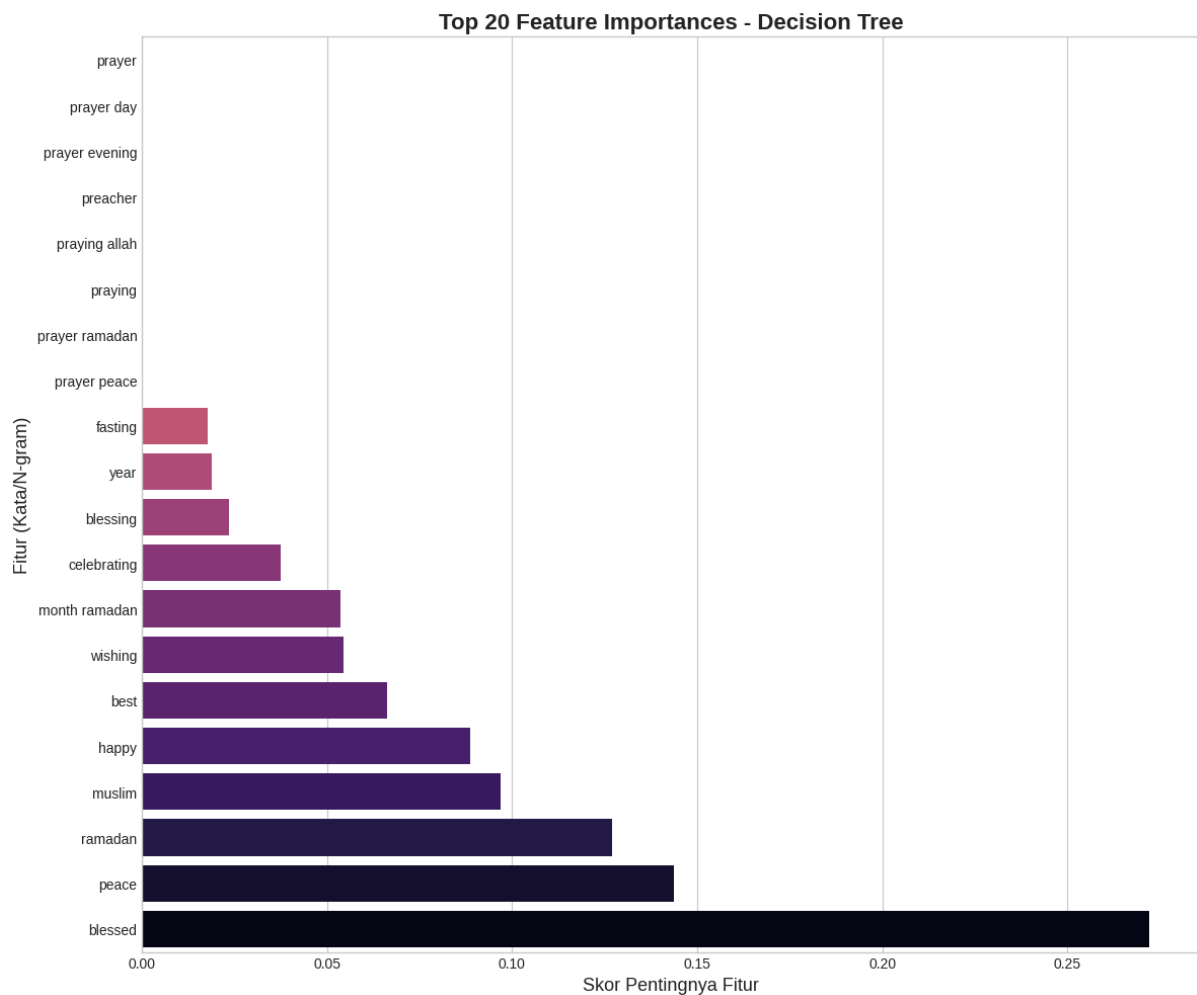
plt.figure(figsize=(15, 5), dpi=90) # Tingkatkan ukuran dan DPI untuk detail
plot_tree(
    best_dt_model,
    filled=True,
    feature_names=tfidf.get_feature_names_out(),
    class_names=['Negative', 'Neutral', 'Positive'], # Updated class_names
    max_depth=3, # Tampilkan 3 level pertama agar tidak terlalu ramai
    fontsize=9,
    proportion=True,
    rounded=True,
    precision=3, # Tingkatkan presisi
    impurity=True,
    label='all'
)
plt.title('Visualisasi Pohon Keputusan (3 Level Pertama)', fontsize=24,
fontweight='bold')
plt.savefig("decision_tree_plot_tree.png", dpi=300, bbox_inches='tight') # Simpan
gambar
plt.show()
print("Visualisasi dengan plot_tree telah ditampilkan dan disimpan sebagai
'decision_tree_plot_tree.png'")

```

Visualisasi Pohon Keputusan (3 Level Pertama)



Adding: Feature Importances



2a. PERBAIKAN KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE DENGAN TUNING PARAMETER (Menggunakan 2 Kategori Label: Positive dan Negative)

1. Tulis/Screenshot perintah/coding preprocessing yang digunakan:

a. Case Folding

Jawab:

```
# 1. Case Folding: Mengubah semua teks menjadi huruf kecil
text = text.lower()

# Tambahan: Menghilangkan URL, mention, hashtag
text = re.sub(r"http\S+|www\S+|https\S+", '', text,
flags=re.MULTILINE) # Menghilangkan URL
text = re.sub(r'\@\w+|\#', '', text) # Menghilangkan mention dan
hashtag symbols

# Tambahan: Menghilangkan karakter non-alfanumerik kecuali spasi
(menyisakan kata dan angka)
text = re.sub(r'^a-zA-Z0-9\s', '', text) # Membiarkan angka,
atau bisa juga dihilangkan
```

b. Tokenizing (n-gram)

Jawab:

```
# 2. Tokenizing: Memecah teks menjadi token/kata
# Menggunakan word_tokenize dari NLTK untuk tokenisasi yang lebih
baik
tokens = word_tokenize(text, language='english') # Explicitly
specify language
```

c. Stopword removal dan Stemming

Jawab:

```

# 3. Stopword Removal dan 4. Lemmatization
processed_tokens = []
for token in tokens:
    # Hanya proses token yang berupa alphabet dan bukan stopword
    # (isalpha() juga akan menghilangkan angka yang mungkin
tersisa jika diinginkan)
    if token.isalpha() and token not in english_stopwords:
        # Lakukan lemmatization
        lemmatized_token = lemmatizer.lemmatize(token)
        processed_tokens.append(lemmatized_token)

return ' '.join(processed_tokens)

```

d. Lemmatization

Jawab:

Kode Lengkapnya (Beserta fungsi buatan di bagian Data Preprocessing):

```

# Fungsi untuk menggabungkan list kata menjadi string (masih relevan dari kode awal
Anda)

def join_text_list(text_list_str):

    if isinstance(text_list_str, str):

        try:

            actual_list = eval(text_list_str)

            if isinstance(actual_list, list):

                return ' '.join(actual_list)

            else:

                return str(text_list_str)

        except Exception:

            return str(text_list_str)

    elif isinstance(text_list_str, list):

        return ' '.join(text_list_str)

    return str(text_list_str)

```

```

# Mengaplikasikan join_text_list ke kolom 'tweet_clean' untuk mendapatkan teks
mentah

# yang akan diproses. Kolom ini digunakan agar proses evaluasi lebih konsisten

# dengan bagaimana 'text_processed_raw' dibuat di kode asli.

df['text_processed_raw'] = df['tweet_clean'].apply(join_text_list)


# Inisialisasi Lemmatizer dan Stop Words

# Menggunakan WordNetLemmatizer untuk lemmatization

lemmatizer = WordNetLemmatizer()

english_stopwords = set(nltk_stopwords.words('english'))

# Anda bisa tambahkan custom stop words di sini jika perlu, contoh:

# custom_stopwords = {'ramadan', 'mubarak', 'also', 'like', 'would'}

# all_stopwords = english_stopwords.union(custom_stopwords)

# Untuk saat ini, kita gunakan default NLTK English stopwords


def preprocess_text_step1(text):

    if not isinstance(text, str): # Pastikan input adalah string

        return ""

    # 1. Case Folding: Mengubah semua teks menjadi huruf kecil

    text = text.lower()

    # Tambahan: Menghilangkan URL, mention, hashtag (opsional, tapi baik untuk
    tweet)

    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE) #
    Menghilangkan URL

    text = re.sub(r'@\w+|\#','', text) # Menghilangkan mention dan hashtag symbols

```

```

    # Tambahan: Menghilangkan karakter non-alfanumerik kecuali spasi (menyisakan
kata dan angka)

    # Jika ingin hanya alphabet: text = re.sub(r'^a-zA-Z\s', '', text)

    text = re.sub(r'^a-zA-Z0-9\s', '', text) # Membiarkan angka, atau bisa juga
dihilangkan jika diinginkan

# 2. Tokenizing: Memecah teks menjadi token/kata

# Menggunakan word_tokenize dari NLTK untuk tokenisasi yang lebih baik

tokens = word_tokenize(text, language='english') # Explicitly specify language

# 3. Stopword Removal dan 4. Lemmatization

processed_tokens = []

for token in tokens:

    # Hanya proses token yang berupa alphabet dan bukan stopwords

    if token.isalpha() and token not in english_stopwords:

        # Lakukan lemmatization

        # 'v' untuk verb, 'n' untuk noun, dll. Defaultnya 'n'. Mungkin perlu
eksplorasi POS tagging

        lemmatized_token = lemmatizer.lemmatize(token)

        processed_tokens.append(lemmatized_token)

return ' '.join(processed_tokens)

print("\nMemulai Tahap 1: Pra-pemrosesan Teks (Case Folding, Tokenizing, Stopword
Removal, Lemmatization)...")

# Kolom 'text_processed' akan berisi hasil dari pra-pemrosesan lengkap tahap 1 ini

df['text_processed'] = df['text_processed_raw'].apply(preprocess_text_step1)

```

```

print("\nContoh hasil 'text_processed' setelah pra-pemrosesan Tahap 1:")

display(df[['tweet_clean', 'text_processed_raw', 'text_processed',
'sentiment']].head())

print("\nMemeriksa missing values di kolom 'text_processed' setelah
pra-pemrosesan:")

print(f"Jumlah missing values: {df['text_processed'].isnull().sum()}")

# Jika ada missing values (misal dari tweet yang jadi string kosong setelah
preprocessing),

# kita bisa isi dengan string kosong agar TfidfVectorizer tidak error

df['text_processed'].fillna('', inplace=True)

print(f"Jumlah missing values setelah fillna:
{df['text_processed'].isnull().sum()}")

```

e. Tulis/screenshot hasil preprocessing yang dilakukan

Jawab:

```

Memulai Tahap 1: Pra-pemrosesan Teks (Case Folding, Tokenizing, Stopword Removal, Lemmatization)...

Contoh hasil 'text_processed' setelah pra-pemrosesan Tahap 1:

```

	tweet_clean	text_processed_raw	text_processed	sentiment
0	['abraj', 'al', 'bait', 'clock', 'tower', 'bea...	abraj al bait clock tower beams indicating com...	abraj al bait clock tower beam indicating comm...	negative
1	['accounts', 'recognised', 'ramadan', 'none', ...	accounts recognised ramadan none recognised be...	account recognised ramadan none recognised beg...	negative
2	['admin', 'post', 'peaceful', 'ramadan', 'cele...	admin post peaceful ramadan celebrations east ...	admin post peaceful ramadan celebration east l...	positive
3	['admin', 'post', 'ramadan', 'norway']	admin post ramadan norway	admin post ramadan norway	negative
4	['admin', 'post', 'ramadan', 'usual', 'peacefu...	admin post ramadan usual peaceful start englan...	admin post ramadan usual peaceful start englan...	positive

```

Memeriksa missing values di kolom 'text_processed' setelah pra-pemrosesan:
Jumlah missing values: 0
Jumlah missing values setelah fillna: 0
<ipython-input-14-2353637989>:76: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instea

df['text_processed'].fillna('', inplace=True)

```

2. Tulis/Screenshot max_depth yang digunakan

Jawab:

```

param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80], # Perluas
    'min_samples_split': [2, 5, 10, 15, 20], # Perluas sedikit
    'min_samples_leaf': [1, 2, 4, 6, 8], # Perluas sedikit
    'class_weight': [None, 'balanced'] # Untuk menangani imbalance kelas
    # 'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa
    # dicoba
}

```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Max_depth	'max_depth': [None, 10, 20, 30, 40, 50], Nilai yang dipilih 40	'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120], Nilai yang dipilih 40
Hasil akurasi	0.70	0.70

3. Tulis/screenshot min_samples_split yang digunakan

Jawab:

```

param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120],
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'class_weight': [None, 'balanced']
    # 'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa
    # dicoba jika perlu
}

```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Min_samples_split	'min_samples_split': [2, 5, 10, 15, 20], Nilai yang dipilih 20	'min_samples_split': [2, 5, 10, 15, 20, 25, 30], Nilai yang dipilih 5

Hasil akurasi	0.70	0.72

4. Tulis/screenshot criterion yang digunakan

Jawab:

```
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120],
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'class_weight': [None, 'balanced']
    # 'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa dicoba jika perlu
}
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
criterion	Gini	Entropy
Hasil akurasi	0.72	0.72

5. Tuliskan/screenshot metode validation yang digunakan:

Jawab:

a. Split-Validation

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2, # 20% data digunakan untuk testing
    random_state=42, # Untuk reproduktifitas hasil
    # stratify=y # Mempertahankan proporsi kelas sentimen pada data training dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
```

```

print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
print(y_test.value_counts())

```

b. Cross-Validation

```

grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Menggunakan semua core CPU
    verbose=1 # menampilkan log
)

```

Kode lengkap Cross-Validation:

```

param_grid_dt = {
    'criterion': ['entropy'],
    'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80,
90, 100, 110, 120],
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12],
    'class_weight': [None, 'balanced'],
    'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity
Pruning, bisa dicoba jika perlu
}

```

```

dt_model = DecisionTreeClassifier(random_state=42)

```

```

grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Menggunakan semua core CPU
    verbose=1 # menampilkan log
)

```



```

)

print("Memulai GridSearchCV untuk Decision Tree...")
grid_search_dt.fit(X_train, y_train)

best_dt_model = grid_search_dt.best_estimator_
print("\nGridSearchCV selesai.")

print("Parameter terbaik yang ditemukan untuk Decision Tree:")
print(grid_search_dt.best_params_)

print(f"\nSkor akurasi cross-validation terbaik untuk Decision
Tree: {grid_search_dt.best_score_:.4f}")

print("\nPenjelasan Terkait Pemilihan Split pada Decision
Tree:")
print("Model Decision Tree yang dilatih menggunakan kriteria
impurity untuk menentukan split terbaik pada setiap node.")
print(f"Kriteria impurity yang dipilih oleh GridSearchCV untuk
model terbaik ini adalah: '{best_dt_model.criterion}'.")
if best_dt_model.criterion == 'entropy':
    print("    - Dengan kriteria 'entropy', model bertujuan
untuk memaksimalkan Information Gain.")
    print("    - Information Gain mengukur pengurangan
ketidakpastian setelah dataset di-split berdasarkan sebuah
atribut.")
    print("    - Dihitung sebagai:  $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$ .")
elif best_dt_model.criterion == 'gini':
    print("    - Dengan kriteria 'gini', model bertujuan untuk
meminimalkan Gini Impurity.")
    print("    - Gini Impurity mengukur probabilitas kesalahan
klasifikasi jika sebuah elemen acak dari set labelnya ditebak
secara acak sesuai distribusi label di set tersebut.")
    print("    - Dihitung sebagai:  $Gini(D) = 1 - \sum (p_i)^2$ .")
print("Algoritma seperti ID3, C4.5 (menggunakan Information
Gain atau Gain Ratio), dan CART (menggunakan Gini Index) adalah
implementasi dari konsep ini.")

```

```
print("Scikit-learn mengimplementasikan versi optimasi dari
algoritma CART untuk Decision Trees.")
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
validation	Kami memakai 2 yakni Split-Validation (Test size = 25%) dan Cross-Validation.	Kami memakai Split-Validation (Test size = 20%).
Hasil akurasi	0.72	0.72

- Cross-validation digunakan selama tahap hyperparameter tuning (di dalam GridSearchCV) untuk membantu menemukan set parameter terbaik dengan cara yang lebih robust pada data training.
- Split validation (dengan memisahkan Test Set) digunakan di akhir proses untuk mendapatkan estimasi kinerja model final pada data yang sama sekali baru dan tidak bias.

6. Tuliskan/screenshot feature selection yang digunakan:

Jawab:

```
from sklearn.feature_selection import SelectKBest, chi2

# Menentukan jumlah fitur terbaik yang ingin dipilih
# Anda bisa bereksperimen dengan nilai k ini (misal, 500, 1000,
2000, dll.)
k_best_features = 1500 # Jumlah fitur yang akan dipilih

# Menggunakan SelectKBest dengan fungsi skor chi2
# chi2 cocok untuk data non-negatif (seperti TF-IDF) dan target
kategorikal
selector = SelectKBest(score_func=chi2, k=k_best_features)

# Melakukan pemilihan fitur pada data training (X dan y)
# Pastikan X dan y sudah terdefinisi dari tahap TF-IDF dan
konversi label
try:
```

```

X_new = selector.fit_transform(X, y)

    print(f"\nShape matriks fitur sebelum pemilihan fitur:
{X.shape}")

    print(f"Shape matriks fitur setelah pemilihan fitur
(memilih {k_best_features} fitur terbaik): {X_new.shape}")

    # Mendapatkan nama fitur yang dipilih (opsional, tapi
informatif)
    # Dapatkan indeks fitur yang dipilih
                                selected_feature_indices =
selector.get_support(indices=True)

    # Dapatkan nama fitur asli dari TfidfVectorizer (perlu
diakses dari objek tfidf)
    # Pastikan objek tfidf dari cell sebelumnya masih tersedia
    try:
        original_feature_names = tfidf.get_feature_names_out()
        selected_feature_names = [original_feature_names[i] for
i in selected_feature_indices]
        print(f"\nBeberapa contoh fitur yang dipilih
({len(selected_feature_names)} fitur):")
        # Tampilkan beberapa fitur pertama dan terakhir
                                display(selected_feature_names[:10] +
selected_feature_names[-10:])

    except NameError:
        print("\nObjek 'tfidf' dari tahap TF-IDF tidak
ditemukan.")
        print("Tidak dapat menampilkan nama fitur yang
dipilih.")
    except Exception as e:
        print(f"\nTerjadi error saat mendapatkan nama fitur:
{e}")

except NameError:

```

```

print("\nError: Variabel X atau y belum terdefinisi dari
tahap sebelumnya.")

# Sekarang, variabel fitur yang akan digunakan untuk split data
adalah X_new
# Tahap selanjutnya (Pembagian Data) harus menggunakan X_new
dan y

```

Kode ini melakukan seleksi fitur menggunakan SelectKBest dengan kriteria Chi-squared (chi2), yang memilih 'k' fitur terbaik dari matriks fitur TF-IDF (X) yang non-negatif berdasarkan relevansinya dengan target kategorikal (y). Jumlah kolom yang diseleksi dikontrol oleh variabel k_best_features (di sini diatur ke 1500), dan output kode akan menampilkan bentuk matriks fitur sebelum dan sesudah seleksi, menunjukkan jumlah fitur yang berhasil dipertahankan setelah proses ini.

Hasil Outputnya:

Shape matriks fitur sebelum pemilihan fitur: (836, 941)

Shape matriks fitur setelah pemilihan fitur (memilih 1500 fitur terbaik):
(836, 941)

Beberapa contoh fitur yang dipilih (941 fitur):

/usr/local/lib/python3.11/dist-packages/sklearn/feature_selection/_univariate_selection.py:783: UserWarning: k=1500 is greater than n_features=941. All the features will be returned.

```

warnings.warn(
['able',
 'abu',
 'accept',
 'accepted',
 'according',
 'account',
 'act',
 'act worship',
 'action',
 'activist',
 'worship',
 'ya',
 'ya allah',
 'year',
 'year old',

```

```
'yearold',
'yes',
'yesterday',
'york',
'young']
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Feature selection (jumlah kolom)	Dalam proyek ini, kami menggunakan TF-IDF Vectorization untuk mengubah data teks menjadi representasi numerik. Meskipun kami membatasi jumlah fitur maksimum yang dihasilkan oleh TF-IDF (menggunakan parameter <code>max_features</code>), kami tidak menerapkan metode seleksi fitur eksplisit seperti pemilihan fitur berdasarkan statistik (contoh: <code>SelectKBest</code>) atau seleksi fitur berbasis model sebelum melatih model Decision Tree. Pembatasan fitur dilakukan langsung selama proses vektorisasi TF-IDF. Analisis pentingnya fitur (feature importance) dilakukan setelah model dilatih untuk mengidentifikasi fitur yang paling berkontribusi pada prediksi model.	941 fitur
Hasil akurasi	0.72	0.8

7. Tuliskan/screenshot SMOTE yang digunakan:

Jawab:

```
# Import library yang dibutuhkan
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier # Jika best_dt_model belum
ada
from sklearn.metrics import accuracy_score
import pandas as pd # Untuk value_counts dan DataFrame
from collections import Counter # Untuk menghitung distribusi kelas

# Asumsi X_train, y_train, X_test, y_test sudah ada dari tahap split data
# Dan tfidf sudah di-fit pada X_train (jika X_train adalah sparse matrix dari
TF-IDF)

# 0. Persiapan Data untuk Tabel Laporan
```

```

smote_results = {
    "SMOTE": ["Oversampling (jumlah kelas minoritas)", "Undersampling (jumlah
kelas mayoritas)", "Hasil akurasi"],
    "Sebelum": [None, None, None],
    "Sesudah": [None, None, None]
}

# 1. Hitung Akurasi SEBELUM SMOTE
print("--- EVALUASI MODEL SEBELUM SMOTE ---")
# Gunakan best_dt_model jika sudah ada dari GridSearchCV, atau inisialisasi
model baru
try:
    model_before_smote = best_dt_model
    print("Menggunakan best_dt_model dari GridSearchCV untuk evaluasi sebelum
SMOTE.")
except NameError:
    print("best_dt_model tidak ditemukan, menggunakan DecisionTreeClassifier
default.")
    model_before_smote = DecisionTreeClassifier(random_state=42)

# Melatih model pada data training asli
model_before_smote.fit(X_train, y_train)
y_pred_before_smote = model_before_smote.predict(X_test)
accuracy_before_smote = accuracy_score(y_test, y_pred_before_smote)
print(f"Akurasi      pada      data      testing      SEBELUM      SMOTE:
{accuracy_before_smote:.4f}")
smote_results["Sebelum"][2] = f"{accuracy_before_smote:.4f}"

# Distribusi kelas SEBELUM SMOTE
counts_before_smote = Counter(y_train)
print("\nDistribusi kelas pada data training SEBELUM SMOTE:")
for cls, count in counts_before_smote.items():
    print(f"Kelas {cls}: {count}")

# Tentukan kelas minoritas dan mayoritas untuk pelaporan
if len(counts_before_smote) == 2: # Biner
    class_labels = list(counts_before_smote.keys())
    if counts_before_smote[class_labels[0]] <
counts_before_smote[class_labels[1]]:
        minority_class_label_before = class_labels[0]
        majority_class_label_before = class_labels[1]
    else:

```

```

    minority_class_label_before = class_labels[1]
    majority_class_label_before = class_labels[0]

    smote_results["Sebelum"][0] =
counts_before_smote[minority_class_label_before] # Oversampling (minority
count)

    smote_results["Sebelum"][1] =
counts_before_smote[majority_class_label_before] # Undersampling (majority
count)
elif len(counts_before_smote) > 0 : # Jika datasetnya imbang atau hanya ada
satu kelas (jarang terjadi di sini)
    # Untuk dataset yang sudah imbang atau hanya satu kelas, SMOTE mungkin
tidak mengubah banyak
    # Kita ambil saja kelas pertama sebagai "minoritas" dan "mayoritas" untuk
pelaporan
    first_class_label = list(counts_before_smote.keys())[0]
    minority_class_label_before = first_class_label
    majority_class_label_before = first_class_label
    smote_results["Sebelum"][0] =
counts_before_smote[minority_class_label_before]
    smote_results["Sebelum"][1] =
counts_before_smote[majority_class_label_before]
else: # Dataset kosong
    minority_class_label_before = None
    majority_class_label_before = None
    smote_results["Sebelum"][0] = 0
    smote_results["Sebelum"][1] = 0

# 2. Penerapan SMOTE
print("\n--- MENERAPKAN SMOTE PADA DATA TRAINING ---")

# Parameter k_neighbors untuk SMOTE
# SMOTE memerlukan jumlah sampel di kelas minoritas minimal k_neighbors + 1
k_neighbors_smote = 5
if minority_class_label_before is not None and
counts_before_smote[minority_class_label_before] <= k_neighbors_smote:
    # Jika kelas minoritas terlalu kecil, kurangi k_neighbors
    # (Minimal k_neighbors adalah 1 jika kelas minoritas > 1)
    k_neighbors_smote = max(1,
counts_before_smote[minority_class_label_before] - 1)

```

```

        if k_neighbors_smote == 0 and
counts_before_smote[minority_class_label_before] == 1:
    print(f"Peringatan: Kelas minoritas ({minority_class_label_before})
hanya memiliki 1 sampel. SMOTE tidak dapat diterapkan.")
    # Tetapkan X_train_smote dan y_train_smote sama dengan X_train dan
y_train
    X_train_smote, y_train_smote = X_train, y_train
    smote_applied_successfully = False
elif counts_before_smote[minority_class_label_before] == 0 :
    print(f"Peringatan: Kelas minoritas ({minority_class_label_before})
tidak memiliki sampel. SMOTE tidak dapat diterapkan.")
    X_train_smote, y_train_smote = X_train, y_train
    smote_applied_successfully = False
else:
    print(f"Peringatan: Jumlah sampel kelas minoritas
({counts_before_smote[minority_class_label_before]}) <= k_neighbors awal (5).
"
        f"Menggunakan k_neighbors={k_neighbors_smote} untuk SMOTE.")
    smote = SMOTE(sampling_strategy='auto', random_state=42,
k_neighbors=k_neighbors_smote)
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
    smote_applied_successfully = True

elif minority_class_label_before is None or
counts_before_smote[minority_class_label_before] == 0 : # Kasus dataset awal
kosong atau tidak ada minoritas
    print(f"Peringatan: Tidak ada kelas minoritas yang teridentifikasi atau
kelas minoritas kosong. SMOTE tidak diterapkan.")
    X_train_smote, y_train_smote = X_train, y_train
    smote_applied_successfully = False
else:
    smote = SMOTE(sampling_strategy='auto', random_state=42,
k_neighbors=k_neighbors_smote)
    # Terapkan SMOTE hanya pada data training
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
    smote_applied_successfully = True
    print(f"SMOTE berhasil diterapkan dengan
k_neighbors={k_neighbors_smote}.")

# Distribusi kelas SETELAH SMOTE
counts_after_smote = Counter(y_train_smote)

```



```

print("\nDistribusi kelas pada data training SETELAH SMOTE:")
for cls, count in counts_after_smote.items():
    print(f"Kelas {cls}: {count}")

# Mengisi hasil "Sesudah" untuk tabel laporan
# Untuk SMOTE (auto), jumlah sampel di kelas minoritas akan sama dengan kelas
mayoritas asli
if smote_applied_successfully and minority_class_label_before is not None :
    smote_results["Sesudah"][0] =
counts_after_smote[minority_class_label_before] # Oversampling (minority
count - now balanced)
    # Jika 'auto', jumlah kelas mayoritas tidak berubah jika dia adalah
mayoritas awal.
    # Atau, jika 'minority', dia akan sama dengan mayoritas baru (yaitu sama
dg minoritas yg di-oversample).
    # Dengan 'auto', semua kelas menjadi sama dengan jumlah sampel kelas
mayoritas awal.

    smote_results["Sesudah"][1] =
counts_after_smote[majority_class_label_before] # Undersampling (majority
count - now balanced)
else: # Jika SMOTE tidak berhasil atau tidak ada minoritas
    if minority_class_label_before is not None and
majority_class_label_before is not None :
        smote_results["Sesudah"][0] =
counts_before_smote[minority_class_label_before]
        smote_results["Sesudah"][1] =
counts_before_smote[majority_class_label_before]
    else:
        smote_results["Sesudah"][0] = 0
        smote_results["Sesudah"][1] = 0

print(f"\nShape data training asli: X_train-{X_train.shape},
y_train-{y_train.shape}")
print(f"Shape data training setelah SMOTE:
X_train_smote-{X_train_smote.shape}, y_train_smote-{y_train_smote.shape}")

# 3. Hitung Akurasi SETELAH SMOTE
print("\n--- EVALUASI MODEL SETELAH SMOTE ---")
# Gunakan parameter terbaik yang sama (jika ada) atau model default yang sama
try:

```

```

        # Buat instance baru dari model dengan parameter terbaik untuk dilatih
pada data SMOTE
        # atau gunakan model yang sama jika state-nya direset setelah .fit()
        # Untuk DecisionTreeClassifier, .fit() akan melatih ulang
        model_after_smote = best_dt_model
        print("Menggunakan best_dt_model dari GridSearchCV untuk evaluasi setelah
SMOTE.")
except NameError:
    print("best_dt_model tidak ditemukan, menggunakan DecisionTreeClassifier
default.")
    model_after_smote = DecisionTreeClassifier(random_state=42)

# Melatih model pada data training yang sudah di-SMOTE
model_after_smote.fit(X_train_smote, y_train_smote)
y_pred_after_smote = model_after_smote.predict(X_test)
accuracy_after_smote = accuracy_score(y_test, y_pred_after_smote)
print(f"Akurasi pada data testing SETELAH SMOTE: {accuracy_after_smote:.4f}")
smote_results["Sesudah"][2] = f"{accuracy_after_smote:.4f}"

# 4. Tampilkan Tabel Perbandingan Hasil
print("\n\n--- PERBANDINGAN HASIL SMOTE ---")
df_smote_results = pd.DataFrame(smote_results)
# display(df_smote_results) # Untuk tampilan tabel yang lebih baik di
notebook

# Untuk output teks yang diminta (agar bisa langsung dicopy ke laporan)
# Menentukan label kelas 0 dan 1 (asumsi sudah di-map dari 'negative' dan
'positive')
# Dari kode sebelumnya y_train sudah numerik.
# Jika kita tahu mana yang minoritas (misal kelas 0) dan mayoritas (kelas 1)
sebelum SMOTE
# (Perlu di-adjust jika labelnya berbeda atau multi-kelas)

label_kelas_0_before = counts_before_smote.get(0, 0)
label_kelas_1_before = counts_before_smote.get(1, 0)
label_kelas_0_after = counts_after_smote.get(0, 0)
label_kelas_1_after = counts_after_smote.get(1, 0)

# Mengisi tabel sesuai format yang diminta:

```

```

# "Oversampling (jumlah dataset)" -> kita interpretasikan sebagai jumlah
dataset kelas MINORITAS
# "Undersampling (jumlah dataset)" -> kita interpretasikan sebagai jumlah
dataset kelas MAYORITAS

# Menentukan mana yang minoritas dan mayoritas sebelum SMOTE
if label_kelas_0_before < label_kelas_1_before:
    val_oversampling_sebelum = label_kelas_0_before
    val_undersampling_sebelum = label_kelas_1_before
else:
    val_oversampling_sebelum = label_kelas_1_before
    val_undersampling_sebelum = label_kelas_0_before

# Setelah SMOTE (auto), kedua kelas akan memiliki jumlah yang sama dengan
mayoritas awal
# Jadi, "Oversampling (jumlah dataset)" sesudah = jumlah baru kelas minoritas
(yang jadi = mayoritas awal)
# Dan "Undersampling (jumlah dataset)" sesudah = jumlah kelas mayoritas (yang
jadi = mayoritas awal)
if smote_applied_successfully:
    # Dengan sampling_strategy='auto', kelas minoritas akan dioversample
hingga sama dengan mayoritas.
    # Jika awalnya kelas 0 minoritas:
    #   val_oversampling_sesudah akan jadi jumlah kelas 0 setelah SMOTE (sama
dg mayoritas)
    #   val_undersampling_sesudah akan jadi jumlah kelas 1 setelah SMOTE
(sama dg mayoritas)
    # Kita bisa ambil saja salah satu karena jumlahnya sama (misal
counts_after_smote[majority_class_label_before])
    val_oversampling_sesudah =
counts_after_smote.get(minority_class_label_before, 0)
    val_undersampling_sesudah =
counts_after_smote.get(majority_class_label_before, 0)
    if not (val_oversampling_sesudah == val_undersampling_sesudah): #Jika
strategi smote tidak auto atau ada kesalahan logika
        #ambil salah satu sebagai nilai balancing, karena harusnya keduanya
jadi sama.
        #disini mengambil kelas dengan nilai maksimum setelah smote sebagai
target balancing
        val_oversampling_sesudah = max(counts_after_smote.values()) if
counts_after_smote else 0
        val_undersampling_sesudah = val_oversampling_sesudah

```

```

else: # Jika SMOTE tidak diterapkan
    val_oversampling_sesudah = val_oversampling_sebelum
    val_undersampling_sesudah = val_undersampling_sebelum

print("\nTuliskan perbandingan hasilnya pada tabel di bawah ini:")
print(f"{'':<15} | {'Sebelum':<35} | {'Sesudah':<35}")
print(f"{'-'*15} | {'-'*35} | {'-'*35}")
print(f"{'SMOTE':<15} | {'':<35} | {'':<35}")
print(f"{'  Oversampling':<13} | {str(val_oversampling_sebelum) + ' (jumlah kelas minoritas awal)':<35} | {str(val_oversampling_sesudah) + ' (jumlah kelas minoritas setelah SMOTE)':<35}")
print(f"{'  Undersampling':<13} | {str(val_undersampling_sebelum) + ' (jumlah kelas mayoritas awal)':<35} | {str(val_undersampling_sesudah) + ' (jumlah kelas mayoritas setelah SMOTE*)':<35}")
print(f"{'-'*15} | {'-'*35} | {'-'*35}")
print(f"{'Hasil akurasi':<15} | {accuracy_before_smote:<35.4f} | {accuracy_after_smote:<35.4f}")
print(f"\n* Catatan: SMOTE dengan sampling_strategy='auto' akan meningkatkan jumlah sampel kelas minoritas agar seimbang dengan kelas mayoritas. Kelas mayoritas tidak di-undersample oleh SMOTE itu sendiri.")

```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

		Sebelum	Sesudah
SMOTE	Oversampling (jumlah dataset)	314 (jumlah kelas minoritas awal)	354 (jumlah kelas minoritas setelah SMOTE)
	Undersampling (jumlah dataset)	354 (jumlah kelas mayoritas awal)	354 (jumlah kelas mayoritas setelah SMOTE*)
Hasil akurasi		0.8	0.74

Sehingga hasil akurasi terbaik didapatkan sebesar 0.80 (80 %) dengan komposisi tuning parameter:

Tuning Parameter	Max_depth	Min_samples_split	criterion	validation	Feature selection	SMOTE
komposisi	40	5	entropy	Split dan Cross Perbandingan dataset: 80/20	941 fitur	Tidak menggunakan SMOTE, jika menggunakan, maka akurasi 74% dengan: Oversampling (SMOTE) Perbandingan dataset: Kelas 0: 354; Kelas 1: 354

Catatan:

Dalam proses pengembangan model klasifikasi ini, kami menerapkan dua metode validasi yang memiliki peran penting dan saling melengkapi: **Validasi dengan Pemisahan Data (Split Validation)** dan **Validasi Silang (Cross-Validation)**. Kedua metode ini esensial untuk mengevaluasi kinerja model secara akurat dan memastikan model dapat bekerja dengan baik pada data yang belum pernah dilihat sebelumnya.

Pertama, **Validasi dengan Pemisahan Data** melibatkan pembagian dataset awal menjadi dua bagian utama yang terpisah: **Data Latih (Training Set)** dan **Data Uji (Test Set)**. Model hanya dilatih menggunakan Data Latih. Setelah model selesai dikembangkan dan disetel, kinerjanya dievaluasi satu kali pada Data Uji. Data Uji ini berfungsi sebagai representasi data "baru" yang independen. Hasil evaluasi pada Data Uji memberikan **estimasi kinerja model yang paling objektif dan tidak bias** mengenai kemampuan model untuk menggeneralisasi pada data di dunia nyata. Dalam proyek ini, pemisahan ini dilakukan di awal proses menggunakan fungsi `train_test_split`, dan evaluasi akhir model dilakukan pada Data Uji ini.

Kedua, **Validasi Silang (Cross-Validation)** umumnya diterapkan **pada Data Latih**. Metode ini membagi Data Latih menjadi beberapa subset atau 'lipatan' (folds). Proses pelatihan dan evaluasi model diulang beberapa kali, di mana setiap kali satu lipatan digunakan sebagai data validasi dan lipatan lainnya sebagai data latih. Kinerja model kemudian dirata-ratakan dari setiap iterasi. Validasi Silang sangat berguna **selama tahap pengembangan model**, khususnya untuk:

- Memilih model terbaik dari beberapa kandidat.
- Melakukan **penyetelan *hyperparameter*** (seperti yang dilakukan menggunakan GridSearchCV dalam proyek ini). Metode ini memberikan estimasi kinerja yang lebih stabil dan dapat diandalkan selama proses pengembangan model dibandingkan hanya menggunakan satu set validasi tunggal.

Secara ringkas, dalam alur kerja pengembangan model yang disarankan, **Validasi dengan Pemisahan Data (melalui Test Set)** digunakan untuk **evaluasi kinerja akhir yang tidak bias** dari model final, sementara **Validasi Silang** digunakan **pada Data Latih** untuk **optimalisasi dan penyetelan model** selama tahap pengembangan. Kedua metode ini memiliki fungsi yang berbeda namun krusial untuk memastikan model yang dihasilkan memiliki kinerja yang baik dan dapat diandalkan.

2b. PERBAIKAN KLASIFIKASI MENGGUNAKAN ALGORITMA DECISION TREE DENGAN TUNING PARAMETER (Menggunakan 3 Kategori Label: Positive, Negative, dan Neutral)

1. Tulis/Screenshot perintah/coding preprocessing yang digunakan:

a. Case Folding

Jawab:

```
# 1. Case Folding: Mengubah semua teks menjadi huruf kecil
text = text.lower()

# Tambahan: Menghilangkan URL, mention, hashtag
text = re.sub(r"http\S+|www\S+|https\S+", '', text,
flags=re.MULTILINE) # Menghilangkan URL
text = re.sub(r'\@\w+|\#','', text) # Menghilangkan mention dan
hashtag symbols

# Tambahan: Menghilangkan karakter non-alfanumerik kecuali spasi
(menyisakan kata dan angka)
text = re.sub(r'^a-zA-Z0-9\s','', text) # Membiarkan angka,
atau bisa juga dihilangkan
```

b. Tokenizing (n-gram)

Jawab:

```
# 2. Tokenizing: Memecah teks menjadi token/kata
# Menggunakan word_tokenize dari NLTK untuk tokenisasi yang lebih
baik
tokens = word_tokenize(text, language='english') # Explicitly
specify language
```

c. Stopword removal dan Stemming

Jawab:

```
# 3. Stopword Removal dan 4. Lemmatization
processed_tokens = []
for token in tokens:
    # Hanya proses token yang berupa alphabet dan bukan stopwords
    # (isalpha() juga akan menghilangkan angka yang mungkin
    tersisa jika diinginkan)
    if token.isalpha() and token not in english_stopwords:
        # Lakukan lemmatization
        lemmatized_token = lemmatizer.lemmatize(token)
        processed_tokens.append(lemmatized_token)

return ' '.join(processed_tokens)
```

d. Lemmatization

Jawab:

```
# 3. Stopword Removal dan 4. Lemmatization
processed_tokens = []
for token in tokens:
    # Hanya proses token yang berupa alphabet dan bukan stopwords
    # (isalpha() juga akan menghilangkan angka yang mungkin
    tersisa jika diinginkan)
    if token.isalpha() and token not in english_stopwords:
        # Lakukan lemmatization
        lemmatized_token = lemmatizer.lemmatize(token)
        processed_tokens.append(lemmatized_token)

return ' '.join(processed_tokens)
```

Kode Lengkapnya (Beserta fungsi buatan di bagian Data Preprocessing):

```
# Fungsi untuk menggabungkan list kata menjadi string (masih relevan dari
kode awal Anda)
```

```
def join_text_list(text_list_str):

    if isinstance(text_list_str, str):

        try:
```



```

        actual_list = eval(text_list_str)

        if isinstance(actual_list, list):

            return ' '.join(actual_list)

        else:

            return str(text_list_str)

    except Exception:

        return str(text_list_str)

    elif isinstance(text_list_str, list):

        return ' '.join(text_list_str)

    return str(text_list_str)

# Mengaplikasikan join_text_list ke kolom 'tweet_clean' untuk mendapatkan
teks mentah

# yang akan diproses. Kolom ini digunakan agar proses evaluasi lebih
konsisten

# dengan bagaimana 'text_processed_raw' dibuat di kode asli.

df['text_processed_raw'] = df['tweet_clean'].apply(join_text_list)

# Inisialisasi Lemmatizer dan Stop Words

# Menggunakan WordNetLemmatizer untuk lemmatization

lemmatizer = WordNetLemmatizer()

english_stopwords = set(nltk_stopwords.words('english'))

# Anda bisa tambahkan custom stop words di sini jika perlu, contoh:

# custom_stopwords = {'ramadan', 'mubarak', 'also', 'like', 'would'}

# all_stopwords = english_stopwords.union(custom_stopwords)

```

```

# Untuk saat ini, kita gunakan default NLTK English stopwords

def preprocess_text_step1(text):

    if not isinstance(text, str): # Pastikan input adalah string

        return ""

    # 1. Case Folding: Mengubah semua teks menjadi huruf kecil

    text = text.lower()

    # Tambahan: Menghilangkan URL, mention, hashtag (opsional, tapi baik
    untuk tweet)

    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE) #
    Menghilangkan URL

    text = re.sub(r'\@\w+|\#','', text) # Menghilangkan mention dan hashtag
    symbols

    # Tambahan: Menghilangkan karakter non-alfanumerik kecuali spasi
    (menyisakan kata dan angka)

    # Jika ingin hanya alphabet: text = re.sub(r'^a-zA-Z\s','', text)

    text = re.sub(r'^a-zA-Z0-9\s','', text) # Membiarkan angka, atau bisa
    juga dihilangkan jika diinginkan

    # 2. Tokenizing: Memecah teks menjadi token/kata

    # Menggunakan word_tokenize dari NLTK untuk tokenisasi yang lebih baik

    tokens = word_tokenize(text, language='english') # Explicitly specify
    language

```

```

# 3. Stopword Removal dan 4. Lemmatization

processed_tokens = []

for token in tokens:

    # Hanya proses token yang berupa alphabet dan bukan stopwords

    if token.isalpha() and token not in english_stopwords:

        # Lakukan lemmatization

        # 'v' untuk verb, 'n' untuk noun, dll. Defaultnya 'n'. Mungkin
        # perlu eksplorasi POS tagging

        lemmatized_token = lemmatizer.lemmatize(token)

        processed_tokens.append(lemmatized_token)

    return ' '.join(processed_tokens)

print("\nMemulai Tahap 1: Pra-pemrosesan Teks (Case Folding, Tokenizing,
Stopword Removal, Lemmatization)...")

# Kolom 'text_processed' akan berisi hasil dari pra-pemrosesan lengkap tahap
1 ini

df['text_processed'] = df['text_processed_raw'].apply(preprocess_text_step1)

print("\nContoh hasil 'text_processed' setelah pra-pemrosesan Tahap 1:")

display(df[['tweet_clean', 'text_processed_raw', 'text_processed',
'sentiment']].head())

print("\nMemeriksa missing values di kolom 'text_processed' setelah
pra-pemrosesan:")

print(f"Jumlah missing values: {df['text_processed'].isnull().sum()}")

```

```
# Jika ada missing values (misal dari tweet yang jadi string kosong setelah preprocessing),

# kita bisa isi dengan string kosong agar TfidfVectorizer tidak error

df['text_processed'].fillna('', inplace=True)

print(f"Jumlah missing values setelah fillna:
{df['text_processed'].isnull().sum()}")
```

e. Tulis/screenshot hasil preprocessing yang dilakukan

Jawab:

Memulai Tahap 1: Pra-pemrosesan Teks (Case Folding, Tokenizing, Stopword Removal, Lemmatization)...

Contoh hasil 'text_processed' setelah pra-pemrosesan Tahap 1:

	tweet_clean	text_processed_raw	text_processed	sentiment
0	['abraj', 'al', 'bait', 'clock', 'tower', 'bea...	abraj al bait clock tower beams indicating com...	abraj al bait clock tower beam indicating comm...	2
1	['accounts', 'recognised', 'ramadan', 'none', ...	accounts recognised ramadan none recognised be...	account recognised ramadan none recognised beg...	2
2	['admin', 'post', 'peaceful', 'ramadan', 'cele...	admin post peaceful ramadan celebrations east ...	admin post peaceful ramadan celebration east l...	2
3	['admin', 'post', 'ramadan', 'norway']	admin post ramadan norway	admin post ramadan norway	2
4	['admin', 'post', 'ramadan', 'usual', 'peacefu...	admin post ramadan usual peaceful start englan...	admin post ramadan usual peaceful start englan...	2

Memeriksa missing values di kolom 'text_processed' setelah pra-pemrosesan:

Jumlah missing values: 0

Jumlah missing values setelah fillna: 0

<ipython-input-4-2353637989>:76: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['text_processed'].fillna('', inplace=True)
```

2. Tulis/Screenshot max_depth yang digunakan

Jawab:

```
param_grid_dt = {
    'criterion': ['entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'class_weight': [None, 'balanced'],
    'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa dicoba
}
```

Kombinasi tuning terbaik (untuk langkah proses ini):

GridSearchCV selesai.

Parameter terbaik yang ditemukan untuk Decision Tree:

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

Skor akurasi cross-validation terbaik untuk Decision Tree: 0.8612

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Max_depth	10	None
Hasil akurasi	0.86	0.8612

3. Tulis/screenshot min_samples_split yang digunakan

Jawab:

```
param_grid_dt = {
    'criterion': ['entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30], # Kami coba perluas rentangnya
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'class_weight': [None, 'balanced'],
    'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa dicoba
}
```

GridSearchCV selesai.

Parameter terbaik yang ditemukan untuk Decision Tree:

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth':  
None, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Min_samples_split	2	2
Hasil akurasi	0.8612	0.8612

4. Tulis/screenshot criterion yang digunakan

Jawab:

```
param_grid_dt = {  
    'criterion': ['entropy', 'gini'],  
    'max_depth': [None, 10, 20, 30, 40, 50],  
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30],  
    'min_samples_leaf': [1, 2, 4, 6, 8, 10, 15, 20], # Rentang min_samples_leaf  
    diperluas  
    'class_weight': [None, 'balanced'],  
    'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity Pruning, bisa  
    dicoba  
}
```

GridSearchCV selesai.

Parameter terbaik yang ditemukan untuk Decision Tree:

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth':  
None, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
criterion	entropy	entropy (namun dengan rentang min_sample_leaf diperluas)
Hasil akurasi	0.8612	0.8612

5. Tuliskan/screenshot metode validation yang digunakan:

Jawab:

a. Split-Validation

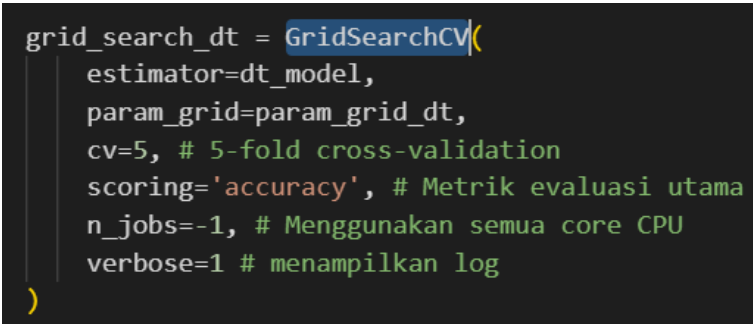
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2, # 20% data digunakan untuk testing
    random_state=42, # Untuk reproduktifitas hasil
    # stratify=y # Mempertahankan proporsi kelas sentimen pada
data training dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train      shape:      {X_train.shape},      y_train      shape:
{y_train.shape}")
print(f"X_test       shape:      {X_test.shape},      y_test       shape:
{y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
print(y_test.value_counts())
```

b. Cross-Validation

A screenshot of a code editor with a dark background. The code defines a GridSearchCV object named 'grid_search_dt'. The parameters are: estimator=dt_model, param_grid=param_grid_dt, cv=5 (commented as 5-fold cross-validation), scoring='accuracy' (commented as Metrik evaluasi utama), n_jobs=-1 (commented as Menggunakan semua core CPU), and verbose=1 (commented as menampilkan log).

```
grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Menggunakan semua core CPU
    verbose=1 # menampilkan log
)
```

Kode lengkap Cross-Validation:

```

param_grid_dt = {
    'criterion': ['entropy'],
    'max_depth': [None, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80,
90, 100, 110, 120],
    'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12],
    'class_weight': [None, 'balanced'],
    'ccp_alpha': [0.0, 0.001, 0.005, 0.01] # Cost-Complexity
Pruning, bisa dicoba jika perlu
}

dt_model = DecisionTreeClassifier(random_state=42)

grid_search_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5, # 5-fold cross-validation
    scoring='accuracy', # Metrik evaluasi utama
    n_jobs=-1, # Menggunakan semua core CPU
    verbose=1 # menampilkan log
)

print("Memulai GridSearchCV untuk Decision Tree...")
grid_search_dt.fit(X_train, y_train)

best_dt_model = grid_search_dt.best_estimator_
print("\nGridSearchCV selesai.")

print("Parameter terbaik yang ditemukan untuk Decision Tree:")
print(grid_search_dt.best_params_)

print(f"\nSkor akurasi cross-validation terbaik untuk Decision
Tree: {grid_search_dt.best_score_:.4f}")

print("\nPenjelasan Terkait Pemilihan Split pada Decision
Tree:")

```



```

print("Model Decision Tree yang dilatih menggunakan kriteria
impurity untuk menentukan split terbaik pada setiap node.")
print(f"Kriteria impurity yang dipilih oleh GridSearchCV untuk
model terbaik ini adalah: '{best_dt_model.criterion}'.")
if best_dt_model.criterion == 'entropy':
    print("    - Dengan kriteria 'entropy', model bertujuan
untuk memaksimalkan Information Gain.")
    print("    - Information Gain mengukur pengurangan
ketidakpastian setelah dataset di-split berdasarkan sebuah
atribut.")
    print("    - Dihitung sebagai:  $IG(D, A) = Entropy(D) - \sum (|D_v| / |D|) * Entropy(D_v)$ .")
elif best_dt_model.criterion == 'gini':
    print("    - Dengan kriteria 'gini', model bertujuan untuk
meminimalkan Gini Impurity.")
    print("    - Gini Impurity mengukur probabilitas kesalahan
klasifikasi jika sebuah elemen acak dari set labelnya ditebak
secara acak sesuai distribusi label di set tersebut.")
    print("    - Dihitung sebagai:  $Gini(D) = 1 - \sum (p_i)^2$ .")
print("Algoritma seperti ID3, C4.5 (menggunakan Information
Gain atau Gain Ratio), dan CART (menggunakan Gini Index) adalah
implementasi dari konsep ini.")
print("Scikit-learn mengimplementasikan versi optimasi dari
algoritma CART untuk Decision Trees.")

```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
validation	Kami memakai 2 yakni Split-Validation (Test size = 25%) dan Cross-Validation.	Kami ubah Split-Validation (Test size = 30%)
Hasil akurasi	0.861	0.8725

- Cross-validation digunakan selama tahap hyperparameter tuning (di dalam GridSearchCV) untuk membantu menemukan set parameter terbaik dengan cara yang lebih robust pada data training.

- Split validation (dengan memisahkan Test Set) digunakan di akhir proses untuk mendapatkan estimasi kinerja model final pada data yang sama sekali baru dan tidak bias.

6. Tuliskan/screenshot feature selection yang digunakan:

Jawab:

```
from sklearn.feature_selection import SelectKBest, chi2

# Menentukan jumlah fitur terbaik yang ingin dipilih
# Anda bisa bereksperimen dengan nilai k ini (misal, 500, 1000,
# 2000, dll.)
k_best_features = 1500 # Jumlah fitur yang akan dipilih

# Menggunakan SelectKBest dengan fungsi skor chi2
# chi2 cocok untuk data non-negatif (seperti TF-IDF) dan target
# kategorikal
selector = SelectKBest(score_func=chi2, k=k_best_features)

# Melakukan pemilihan fitur pada data training (X dan y)
# Pastikan X dan y sudah terdefinisi dari tahap TF-IDF dan
# konversi label
try:
    X_new = selector.fit_transform(X, y)

    print(f"\nShape matriks fitur sebelum pemilihan fitur:
    {X.shape}")

    print(f"Shape matriks fitur setelah pemilihan fitur
    (memilih {k_best_features} fitur terbaik): {X_new.shape}")

    # Mendapatkan nama fitur yang dipilih (opsional, tapi
    informatif)
    # Dapatkan indeks fitur yang dipilih
    selected_feature_indices =
    selector.get_support(indices=True)
```

```

        # Dapatkan nama fitur asli dari TfidfVectorizer (perlu
diakses dari objek tfidf)
        # Pastikan objek tfidf dari cell sebelumnya masih tersedia
try:
    original_feature_names = tfidf.get_feature_names_out()
    selected_feature_names = [original_feature_names[i] for
i in selected_feature_indices]
    print(f"\nBeberapa contoh fitur yang dipilih
({len(selected_feature_names)} fitur):")
    # Tampilkan beberapa fitur pertama dan terakhir
    display(selected_feature_names[:10] +
selected_feature_names[-10:])

except NameError:
    print("\nObjek 'tfidf' dari tahap TF-IDF tidak
ditemukan.")
    print("Tidak dapat menampilkan nama fitur yang
dipilih.")
except Exception as e:
    print(f"\nTerjadi error saat mendapatkan nama fitur:
{e}")

except NameError:
    print("\nError: Variabel X atau y belum terdefinisi dari
tahap sebelumnya.")

# Sekarang, variabel fitur yang akan digunakan untuk split data
adalah X_new
# Tahap selanjutnya (Pembagian Data) harus menggunakan X_new
dan y

```

Kode ini melakukan seleksi fitur menggunakan SelectKBest dengan kriteria Chi-squared (chi2), yang memilih 'k' fitur terbaik dari matriks fitur TF-IDF (X) yang non-negatif berdasarkan relevansinya dengan target kategorikal (y). Jumlah kolom yang diseleksi dikontrol oleh variabel `k_best_features` (di sini diatur ke 1500), dan output kode akan menampilkan bentuk matriks fitur

sebelum dan sesudah seleksi, menunjukkan jumlah fitur yang berhasil dipertahankan setelah proses ini.

Hasil Outputnya:

Shape matriks fitur sebelum pemilihan fitur: (836, 1500)

Shape matriks fitur setelah pemilihan fitur (memilih 1500 fitur terbaik):
(836, 1500)

Beberapa contoh fitur yang dipilih (1500 fitur):

```
['able',  
 'abu',  
 'accept',  
 'accepted',  
 'according',  
 'account',  
 'act',  
 'act worship',  
 'action',  
 'activist',  
 'year',  
 'year old',  
 'year ramadan',  
 'yearold',  
 'yen',  
 'yes',  
 'yesterday',  
 'york',  
 'young',  
 'zazzau']
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

	Sebelum	Sesudah
Feature selection (jumlah kolom)	Dalam proyek ini, kami menggunakan TF-IDF Vectorization untuk mengubah data teks menjadi representasi numerik. Meskipun kami membatasi jumlah fitur maksimum yang dihasilkan oleh TF-IDF (menggunakan parameter <code>max_features</code>), kami tidak menerapkan metode seleksi fitur eksplisit seperti pemilihan fitur berdasarkan	836 fitur

	statistik (contoh: SelectKBest) atau seleksi fitur berbasis model sebelum melatih model Decision Tree. Pembatasan fitur dilakukan langsung selama proses vektorisasi TF-IDF. Analisis pentingnya fitur (feature importance) dilakukan setelah model dilatih untuk mengidentifikasi fitur yang paling berkontribusi pada prediksi model.	
Hasil akurasi	0.8725	0.8725

7. Tuliskan/screenshot SMOTE yang digunakan:

Jawab:

```
# Import library yang dibutuhkan
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier # Jika
best_dt_model belum ada
from sklearn.metrics import accuracy_score
import pandas as pd # Untuk value_counts dan DataFrame
from collections import Counter # Untuk menghitung distribusi
kelas

# Asumsi X_train, y_train, X_test, y_test sudah ada dari tahap
split data
# Dan tfidf sudah di-fit pada X_train (jika X_train adalah
sparse matrix dari TF-IDF)

# 0. Persiapan Data untuk Tabel Laporan
smote_results = {
    "SMOTE": ["Oversampling (jumlah kelas minoritas)",
"Undersampling (jumlah kelas mayoritas)", "Hasil akurasi"],
    "Sebelum": [None, None, None],
    "Sesudah": [None, None, None]
}

# 1. Hitung Akurasi SEBELUM SMOTE
print("--- EVALUASI MODEL SEBELUM SMOTE ---")
```

```

# Gunakan best_dt_model jika sudah ada dari GridSearchCV, atau
inisialisasi model baru
try:
    model_before_smote = best_dt_model
    print("Menggunakan best_dt_model dari GridSearchCV untuk
evaluasi sebelum SMOTE.")
except NameError:
    print("best_dt_model tidak ditemukan, menggunakan
DecisionTreeClassifier default.")

    model_before_smote =
DecisionTreeClassifier(random_state=42)

# Melatih model pada data training asli
model_before_smote.fit(X_train, y_train)
y_pred_before_smote = model_before_smote.predict(X_test)
accuracy_before_smote = accuracy_score(y_test,
y_pred_before_smote)
print(f"Akurasi pada data testing SEBELUM SMOTE:
{accuracy_before_smote:.4f}")
smote_results["Sebelum"][2] = f"{accuracy_before_smote:.4f}"

# Distribusi kelas SEBELUM SMOTE
counts_before_smote = Counter(y_train)
print("\nDistribusi kelas pada data training SEBELUM SMOTE:")
for cls, count in counts_before_smote.items():
    print(f"Kelas {cls}: {count}")

# Tentukan kelas minoritas dan mayoritas untuk pelaporan
if len(counts_before_smote) == 2: # Biner
    class_labels = list(counts_before_smote.keys())
    if counts_before_smote[class_labels[0]] <
counts_before_smote[class_labels[1]]:
        minority_class_label_before = class_labels[0]
        majority_class_label_before = class_labels[1]
    else:
        minority_class_label_before = class_labels[1]
        majority_class_label_before = class_labels[0]

```

```

        smote_results["Sebelum"][0] =
counts_before_smote[minority_class_label_before] # Oversampling
(minority count)

        smote_results["Sebelum"][1] =
counts_before_smote[majority_class_label_before] #
Undersampling (majority count)
elif len(counts_before_smote) > 0 : # Jika datasetnya imbang
atau hanya ada satu kelas (jarang terjadi di sini)
    # Untuk dataset yang sudah imbang atau hanya satu kelas,
SMOTE mungkin tidak mengubah banyak
    # Kita ambil saja kelas pertama sebagai "minoritas" dan
"mayoritas" untuk pelaporan
    first_class_label = list(counts_before_smote.keys())[0]
    minority_class_label_before = first_class_label
    majority_class_label_before = first_class_label
        smote_results["Sebelum"][0] =
counts_before_smote[minority_class_label_before]
        smote_results["Sebelum"][1] =
counts_before_smote[majority_class_label_before]
else: # Dataset kosong
    minority_class_label_before = None
    majority_class_label_before = None
    smote_results["Sebelum"][0] = 0
    smote_results["Sebelum"][1] = 0

# 2. Penerapan SMOTE
print("\n--- MENERAPKAN SMOTE PADA DATA TRAINING ---")

# Parameter k_neighbors untuk SMOTE
# SMOTE memerlukan jumlah sampel di kelas minoritas minimal
k_neighbors + 1
k_neighbors_smote = 5
if minority_class_label_before is not None and
counts_before_smote[minority_class_label_before] <=
k_neighbors_smote:

```

```

    # Jika kelas minoritas terlalu kecil, kurangi k_neighbors
    # (Minimal k_neighbors adalah 1 jika kelas minoritas > 1)
    k_neighbors_smote = max(1,
counts_before_smote[minority_class_label_before] - 1)
    if k_neighbors_smote == 0 and
counts_before_smote[minority_class_label_before] == 1:
        print(f"Peringatan: Kelas minoritas
({minority_class_label_before}) hanya memiliki 1 sampel. SMOTE
tidak dapat diterapkan.")
        # Tetapkan X_train_smote dan y_train_smote sama dengan
X_train dan y_train
        X_train_smote, y_train_smote = X_train, y_train
        smote_applied_successfully = False
    elif counts_before_smote[minority_class_label_before] == 0
:
        print(f"Peringatan: Kelas minoritas
({minority_class_label_before}) tidak memiliki sampel. SMOTE
tidak dapat diterapkan.")
        X_train_smote, y_train_smote = X_train, y_train
        smote_applied_successfully = False
    else:
        print(f"Peringatan: Jumlah sampel kelas minoritas
({counts_before_smote[minority_class_label_before]}) <=
k_neighbors awal (5). "
        f"Menggunakan k_neighbors={k_neighbors_smote}
untuk SMOTE.")
        smote = SMOTE(sampling_strategy='auto',
random_state=42, k_neighbors=k_neighbors_smote)
        X_train_smote, y_train_smote =
smote.fit_resample(X_train, y_train)
        smote_applied_successfully = True

elif minority_class_label_before is None or
counts_before_smote[minority_class_label_before] == 0 : # Kasus
dataset awal kosong atau tidak ada minoritas

```



```

        print(f"Peringatan: Tidak ada kelas minoritas yang
teridentifikasi atau kelas minoritas kosong. SMOTE tidak
diterapkan.")
        X_train_smote, y_train_smote = X_train, y_train
        smote_applied_successfully = False
    else:
        smote = SMOTE(sampling_strategy='auto', random_state=42,
k_neighbors=k_neighbors_smote)
        # Terapkan SMOTE hanya pada data training
        X_train_smote, y_train_smote = smote.fit_resample(X_train,
y_train)
        smote_applied_successfully = True
        print(f"SMOTE berhasil diterapkan dengan
k_neighbors={k_neighbors_smote}.")

# Distribusi kelas SETELAH SMOTE
counts_after_smote = Counter(y_train_smote)
print("\nDistribusi kelas pada data training SETELAH SMOTE:")
for cls, count in counts_after_smote.items():
    print(f"Kelas {cls}: {count}")

# Mengisi hasil "Sesudah" untuk tabel laporan
# Untuk SMOTE (auto), jumlah sampel di kelas minoritas akan
sama dengan kelas mayoritas asli
if smote_applied_successfully and minority_class_label_before
is not None :
        smote_results["Sesudah"][0] =
counts_after_smote[minority_class_label_before] # Oversampling
(minority count - now balanced)
        # Jika 'auto', jumlah kelas mayoritas tidak berubah jika
dia adalah mayoritas awal.
        # Atau, jika 'minority', dia akan sama dengan mayoritas
baru (yaitu sama dg minoritas yg di-oversample).
        # Dengan 'auto', semua kelas menjadi sama dengan jumlah
sampel kelas mayoritas awal.

```

```

smote_results["Sesudah"][1] =
counts_after_smote[majority_class_label_before] # Undersampling
(majority count - now balanced)
else: # Jika SMOTE tidak berhasil atau tidak ada minoritas
    if minority_class_label_before is not None and
majority_class_label_before is not None :
        smote_results["Sesudah"][0] =
counts_before_smote[minority_class_label_before]
        smote_results["Sesudah"][1] =
counts_before_smote[majority_class_label_before]
    else:
        smote_results["Sesudah"][0] = 0
        smote_results["Sesudah"][1] = 0

print(f"\nShape data training asli: X_train-{X_train.shape},
y_train-{y_train.shape}")
print(f"Shape data training setelah SMOTE:
X_train_smote-{X_train_smote.shape},
y_train_smote-{y_train_smote.shape}")

# 3. Hitung Akurasi SETELAH SMOTE
print("\n--- EVALUASI MODEL SETELAH SMOTE ---")
# Gunakan parameter terbaik yang sama (jika ada) atau model
default yang sama
try:
    # Buat instance baru dari model dengan parameter terbaik
untuk dilatih pada data SMOTE
    # atau gunakan model yang sama jika state-nya direset
setelah .fit()
    # Untuk DecisionTreeClassifier, .fit() akan melatih ulang
model_after_smote = best_dt_model
    print("Menggunakan best_dt_model dari GridSearchCV untuk
evaluasi setelah SMOTE.")
except NameError:

```

```

        print("best_dt_model tidak ditemukan, menggunakan
DecisionTreeClassifier default.")

    model_after_smote = DecisionTreeClassifier(random_state=42)

# Melatih model pada data training yang sudah di-SMOTE
model_after_smote.fit(X_train_smote, y_train_smote)
y_pred_after_smote = model_after_smote.predict(X_test)
accuracy_after_smote = accuracy_score(y_test,
y_pred_after_smote)
print(f"Akurasi pada data testing SETELAH SMOTE:
{accuracy_after_smote:.4f}")
smote_results["Sesudah"][2] = f"{accuracy_after_smote:.4f}"

# 4. Tampilkan Tabel Perbandingan Hasil
print("\n\n--- PERBANDINGAN HASIL SMOTE ---")
df_smote_results = pd.DataFrame(smote_results)
# display(df_smote_results) # Untuk tampilan tabel yang lebih
baik di notebook

# Untuk output teks yang diminta (agar bisa langsung dicopy ke
laporan)
# Menentukan label kelas 0 dan 1 (asumsi sudah di-map dari
'negative' dan 'positive')
# Dari kode sebelumnya y_train sudah numerik.
# Jika kita tahu mana yang minoritas (misal kelas 0) dan
mayoritas (kelas 1) sebelum SMOTE
# (Perlu di-adjust jika labelnya berbeda atau multi-kelas)

label_kelas_0_before = counts_before_smote.get(0, 0)
label_kelas_1_before = counts_before_smote.get(1, 0)
label_kelas_0_after = counts_after_smote.get(0, 0)
label_kelas_1_after = counts_after_smote.get(1, 0)

# Mengisi tabel sesuai format yang diminta:

```

```

# "Oversampling (jumlah dataset)" -> kita interpretasikan
sebagai jumlah dataset kelas MINORITAS
# "Undersampling (jumlah dataset)" -> kita interpretasikan
sebagai jumlah dataset kelas MAYORITAS

# Menentukan mana yang minoritas dan mayoritas sebelum SMOTE
if label_kelas_0_before < label_kelas_1_before:
    val_oversampling_sebelum = label_kelas_0_before
    val_undersampling_sebelum = label_kelas_1_before
else:
    val_oversampling_sebelum = label_kelas_1_before
    val_undersampling_sebelum = label_kelas_0_before

# Setelah SMOTE (auto), kedua kelas akan memiliki jumlah yang
sama dengan mayoritas awal
# Jadi, "Oversampling (jumlah dataset)" sesudah = jumlah baru
kelas minoritas (yang jadi = mayoritas awal)
# Dan "Undersampling (jumlah dataset)" sesudah = jumlah kelas
mayoritas (yang jadi = mayoritas awal)
if smote_applied_successfully:
    # Dengan sampling_strategy='auto', kelas minoritas akan
    dioversample hingga sama dengan mayoritas.
    # Jika awalnya kelas 0 minoritas:
    #     val_oversampling_sesudah akan jadi jumlah kelas 0
    setelah SMOTE (sama dg mayoritas)
    #     val_undersampling_sesudah akan jadi jumlah kelas 1
    setelah SMOTE (sama dg mayoritas)
    # Kita bisa ambil saja salah satu karena jumlahnya sama
    (misal counts_after_smote[majority_class_label_before])
    val_oversampling_sesudah =
counts_after_smote.get(minority_class_label_before, 0)
    val_undersampling_sesudah =
counts_after_smote.get(majority_class_label_before, 0)
    if not (val_oversampling_sesudah ==
val_undersampling_sesudah): #Jika strategi smote tidak auto
atau ada kesalahan logika

```

```
        #ambil salah satu sebagai nilai balancing, karena
harusnya keduanya jadi sama.
```

```
        #disini mengambil kelas dengan nilai maksimum setelah
smote sebagai target balancing
```

```
                                val_oversampling_sesudah =
max(counts_after_smote.values()) if counts_after_smote else 0
                                val_undersampling_sesudah = val_oversampling_sesudah
```

```
else: # Jika SMOTE tidak diterapkan
```

```
    val_oversampling_sesudah = val_oversampling_sebelum
```

```
    val_undersampling_sesudah = val_undersampling_sebelum
```

```
print("\nTuliskan perbandingan hasilnya pada tabel di bawah
ini:")
```

```
print(f"{'':<15} | {'Sebelum':<35} | {'Sesudah':<35}")
```

```
print(f"{'-'*15} | {'-'*35} | {'-'*35}")
```

```
print(f"{'SMOTE':<15} | {'':<35} | {'':<35}")
```

```
print(f"{' Oversampling':<13} | {str(val_oversampling_sebelum)
+ ' (jumlah kelas minoritas awal)':<35} |
{str(val_oversampling_sesudah) + ' (jumlah kelas minoritas
setelah SMOTE)':<35}")
```

```
print(f"{' Undersampling':<13} |
{str(val_undersampling_sebelum) + ' (jumlah kelas mayoritas
awal)':<35} | {str(val_undersampling_sesudah) + ' (jumlah kelas
mayoritas setelah SMOTE)':<35}")
```

```
print(f"{'-'*15} | {'-'*35} | {'-'*35}")
```

```
print(f"{'Hasil akurasi':<15} | {accuracy_before_smote:<35.4f}
| {accuracy_after_smote:<35.4f}")
```

```
print(f"\n* Catatan: SMOTE dengan sampling_strategy='auto' akan
meningkatkan jumlah sampel kelas minoritas agar seimbang dengan
kelas mayoritas. Kelas mayoritas tidak di-undersample oleh
SMOTE itu sendiri.")
```

Tuliskan perbandingan hasilnya pada tabel di bawah ini:

		Sebelum	Sesudah
SMOTE	Oversampling (jumlah dataset)	12 (jumlah kelas minoritas awal)	485 (jumlah kelas minoritas setelah SMOTE)
	Undersampling (jumlah dataset)	485 (jumlah kelas mayoritas awal)	485 (jumlah kelas mayoritas setelah SMOTE*)
Hasil akurasi		0.8725	0.8167

Sehingga hasil akurasi terbaik didapatkan sebesar 0.8725 (87,25 %) dengan komposisi tuning parameter:

Tuning Parameter	Max_dept h	Min_samples_s plit	criterion	validation	Feature selection	SMOTE
komposisi	None	2	entropy	Split dan Cross Perbandingan dataset: 70/30	836 fitur	Tidak menggunakan SMOTE, jika menggunakan, maka akurasinya 81,67% dengan: Oversampling (SMOTE) Perbandingan dataset: Kelas 0: 485; Kelas 1: 485; Kelas 2: 485

Catatan:

Dalam proses pengembangan model klasifikasi ini, kami menerapkan dua metode validasi yang memiliki peran penting dan saling melengkapi: **Validasi dengan Pemisahan Data (Split Validation)** dan **Validasi Silang (Cross-Validation)**. Kedua metode ini esensial untuk mengevaluasi kinerja model secara akurat dan memastikan model dapat bekerja dengan baik pada data yang belum pernah dilihat sebelumnya.

Pertama, **Validasi dengan Pemisahan Data** melibatkan pembagian dataset awal menjadi dua bagian utama yang terpisah: **Data Latih (Training Set)** dan **Data Uji (Test Set)**. Model hanya dilatih menggunakan Data Latih. Setelah model selesai dikembangkan dan disetel, kinerjanya dievaluasi satu kali pada Data Uji. Data Uji ini berfungsi sebagai representasi data "baru" yang independen. Hasil evaluasi pada Data Uji memberikan **estimasi kinerja model yang paling objektif dan tidak bias** mengenai kemampuan model untuk menggeneralisasi pada data di dunia nyata. Dalam proyek ini, pemisahan ini dilakukan di awal proses menggunakan fungsi `train_test_split`, dan evaluasi akhir model dilakukan pada Data Uji ini.

Kedua, **Validasi Silang (Cross-Validation)** umumnya diterapkan **pada Data Latih**. Metode ini membagi Data Latih menjadi beberapa subset atau 'lipatan' (folds). Proses pelatihan dan evaluasi model diulang beberapa kali, di mana setiap kali satu lipatan digunakan sebagai data validasi dan lipatan lainnya sebagai data latih. Kinerja model kemudian dirata-ratakan dari setiap iterasi. Validasi Silang sangat berguna **selama tahap pengembangan model**, khususnya untuk:

- Memilih model terbaik dari beberapa kandidat.
- Melakukan **penyetelan *hyperparameter*** (seperti yang dilakukan menggunakan GridSearchCV dalam proyek ini). Metode ini memberikan estimasi kinerja yang lebih stabil dan dapat diandalkan selama proses pengembangan model dibandingkan hanya menggunakan satu set validasi tunggal.

Secara ringkas, dalam alur kerja pengembangan model yang disarankan, **Validasi dengan Pemisahan Data (melalui Test Set)** digunakan untuk **evaluasi kinerja akhir yang tidak bias** dari model final, sementara **Validasi Silang** digunakan **pada Data Latih** untuk **optimalisasi dan penyetelan model** selama tahap pengembangan. Kedua metode ini memiliki fungsi yang berbeda namun krusial untuk memastikan model yang dihasilkan memiliki kinerja yang baik dan dapat diandalkan.

3a. PROSES KLASIFIKASI MENGGUNAKAN ALGORITMA RANDOM FOREST, SVM, NEURAL NETWORK, LOGISTIC REGRESSION (Menggunakan 2 Kategori Label: Positive dan Negative)

1. Tulis/Screenshot Import Library yang digunakan

Jawab:

```
# Data Processing
import pandas as pd
import numpy as np

# Modelling
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV,
train_test_split, cross_val_score, GridSearchCV
from scipy.stats import randint

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier, plot_tree,
export_graphviz
from sklearn.metrics import (
    accuracy_score,
```

```

        confusion_matrix,
        classification_report,
        ConfusionMatrixDisplay,
        roc_auc_score,
        roc_curve
    )

import matplotlib.pyplot as plt
import seaborn as sns # Untuk visualisasi yang lebih menarik
import graphviz # Untuk visualisasi graphviz
import nltk # Untuk pra-pemrosesan teks lebih lanjut
from nltk.stem import WordNetLemmatizer # Untuk lemmatization
from nltk.corpus import stopwords as nltk_stopwords # Untuk custom
stop words

# Feature selection
from sklearn.feature_selection import SelectKBest, chi2

# Download resource NLTK yang mungkin dibutuhkan
# Catch the LookupError directly when nltk.data.find fails
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    print("NLTK resource 'wordnet' not found. Downloading...")
    nltk.download('wordnet')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading
'wordnet': {e}")

try:
    nltk.data.find('corpora/omw-1.4')
except LookupError:
    print("NLTK resource 'omw-1.4' not found. Downloading...")
    nltk.download('omw-1.4') # WordNet multilingual resource
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading
'omw-1.4': {e}")

try:
    nltk.data.find('corpora/stopwords')

```

```

except LookupError:
    print("NLTK resource 'stopwords' not found. Downloading...")
    nltk.download('stopwords')
except Exception as e:
    print(f"An unexpected error occurred while checking/downloading
'stopwords': {e}")

# Smote
from imblearn.over_sampling import SMOTE

# Pytorch
import torch
from torchvision import datasets, transforms
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Random
import random

# Import svm model
from sklearn import svm

# Logistic regression
# import the class
from sklearn.linear_model import LogisticRegression

# Pengaturan umum untuk plot agar lebih menarik
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis") # Atau palet lain seperti 'pastel',
'muted'

```

1. Tulis/screenshot import dataset yang digunakan

Jawab:

```

path =
'https://raw.githubusercontent.com/LatiefDataVisionary/text-mining

```

```
-and-natural-language-processing-college-task/refs/heads/main/data
sets/ramadan_labeled_sentiment.csv'
# path =
'https://raw.githubusercontent.com/LatiefDataVisionary/text-mining
-and-natural-language-processing-college-task/refs/heads/main/data
sets/ramadan_labeled_sentiment_3label.csv'
df = pd.read_csv(path)

print(f"Dataset berhasil di-load dari: {path}")
print(f"Jumlah baris: {df.shape[0]}, Jumlah kolom: {df.shape[1]}")
```

2. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan

a. Tulis/screenshot codingnya

Jawab:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2, # 25% data digunakan untuk testing
    random_state=42, # Untuk reproduktifitas hasil
    # stratify=y # Mempertahankan proporsi kelas sentimen pada
data training dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train shape: {X_train.shape}, y_train shape:
{y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape:
{y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
```

```
print(y_test.value_counts())
```

b. Tulis/screenshot hasilnya

Jawab:

Ukuran Data Setelah Pembagian:

X_train shape: (668, 937), y_train shape: (668,)

X_test shape: (168, 937), y_test shape: (168,)

Distribusi kelas pada data training (proporsi):

sentiment

1 0.52994

0 0.47006

Name: proportion, dtype: float64

sentiment

1 354

0 314

Name: count, dtype: int64

Distribusi kelas pada data testing (proporsi):

sentiment

1 0.547619

0 0.452381

Name: proportion, dtype: float64

sentiment

1 92

0 76

Name: count, dtype: int64

3. Proses pemodelan algoritma Random Forest, SVM, Neural net, dan Logistic Regression

a. Tulis/screenshot codingnya

Jawab:

- Random Forest :

```
rf = RandomForestClassifier()  
rf.fit(X_train, y_train)  
  
y_pred = rf.predict(X_test)
```

- Neural Network

```

tensor_x = torch.from_numpy(X_train.astype('float32').todense())
tensor_y = torch.Tensor(y_train)

tensor_x_test = torch.Tensor(X_test.astype('float32').todense())
# transform to torch tensor
tensor_y_test = torch.Tensor(y_test.to_numpy())

train_dataset = TensorDataset(tensor_x, tensor_y)
test_dataset = TensorDataset(tensor_x_test, tensor_y_test)

loader_train = DataLoader(train_dataset, batch_size=10,
shuffle=False)
loader_test = DataLoader(test_dataset, batch_size=32,
shuffle=False)

class Net(nn.Module):
    def __init__(self, input_dim, classes):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 500)
        self.fc2 = nn.Linear(500, classes)
        # self.fc3 = nn.Linear(100, 30)
        # self.fc4 = nn.Linear(30, 30)
        # self.fc5 = nn.Linear(30, classes)
        # self.fc5 = nn.Linear(84, classes)

    def forward(self, x):
        x = x.float()
        # x = F.sigmoid(self.fc1(x))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        # x = F.relu(self.fc3(x))
        # x = F.relu(self.fc4(x))
        # x = self.fc5(x)

        # x = self.fc1(x)

        # x = F.relu(self.fc2(x))
        # x = self.fc(x)

```

```

        return x

criterion = nn.CrossEntropyLoss()

for j in range(1):
    net = Net(937, 2)

    # print(f"lr : {lr}")
    optimizer = optim.Adam(net.parameters(),
lr=1.584618639910894e-05) # 1.584618639910894e-05
    for epoch in range(50): # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(loader_train, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            labels = labels.long()
            # print(inputs.dtype)
            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 64 == 500: # print every 2000 mini-batches
                print(f'[{epoch + 1}, {i + 1:5d}] loss:
{running_loss / 2000:.3f}')
                running_loss = 0.0

        print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss /
2000:.3f}')
        correct = 0
        total = 0
        # since we're not training, we don't need to calculate the
gradients for our outputs

```



```

with torch.no_grad():
    for data in loader_test:
        images, labels = data
        # calculate outputs by running images through the
network
        outputs = net(images)
        # the class with the highest energy is what we choose as
prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network: {100 * correct // total} %')

print('Finished Training')

```

- SVM

```

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

```

- Logistic Regression

```

# instantiate the model (using the default parameters)
logreg = LogisticRegression(random_state=16)

# fit the model with data
logreg.fit(X_train, y_train)

```

```
y_pred = logreg.predict(X_test)
train_acc = accuracy_score(y_train, logreg.predict(X_train))
acc = accuracy_score(y_test, y_pred)
print("Train Accuracy:", train_acc)
print("Test Accuracy:", acc)
```

b. Tulis/screenshot hasilnya

Jawab:

- Random Forest

```
RandomForestClassifier
RandomForestClassifier()
```

- Neural Network

```
[1, 67] loss: 0.023
[2, 67] loss: 0.023
[3, 67] loss: 0.023
[4, 67] loss: 0.023
[5, 67] loss: 0.023
[6, 67] loss: 0.023
[7, 67] loss: 0.023
[8, 67] loss: 0.023
[9, 67] loss: 0.023
[10, 67] loss: 0.023
[11, 67] loss: 0.023
[12, 67] loss: 0.023
[13, 67] loss: 0.023
[14, 67] loss: 0.023
[15, 67] loss: 0.023
[16, 67] loss: 0.022
[17, 67] loss: 0.022
[18, 67] loss: 0.022
[19, 67] loss: 0.022
[20, 67] loss: 0.022
[21, 67] loss: 0.022
```

```

[22, 67] loss: 0.022
[23, 67] loss: 0.022
[24, 67] loss: 0.022
[25, 67] loss: 0.022
[26, 67] loss: 0.021
[27, 67] loss: 0.021
[28, 67] loss: 0.021
[29, 67] loss: 0.021
[30, 67] loss: 0.021
[31, 67] loss: 0.021
[32, 67] loss: 0.021
[33, 67] loss: 0.021
[34, 67] loss: 0.020
[35, 67] loss: 0.020
[36, 67] loss: 0.020
[37, 67] loss: 0.020
[38, 67] loss: 0.020
[39, 67] loss: 0.019
[40, 67] loss: 0.019
[41, 67] loss: 0.019
[42, 67] loss: 0.019
[43, 67] loss: 0.019
[44, 67] loss: 0.019
[45, 67] loss: 0.018
[46, 67] loss: 0.018
[47, 67] loss: 0.018
[48, 67] loss: 0.018

[49, 67] loss: 0.018
[50, 67] loss: 0.017
Accuracy of the network: 73 %
Finished Training

```

- SVM

```

clf
SVC
SVC(kernel='linear')

```

- Logistic Regression

```

logreg
LogisticRegression
LogisticRegression(random_state=16)

```

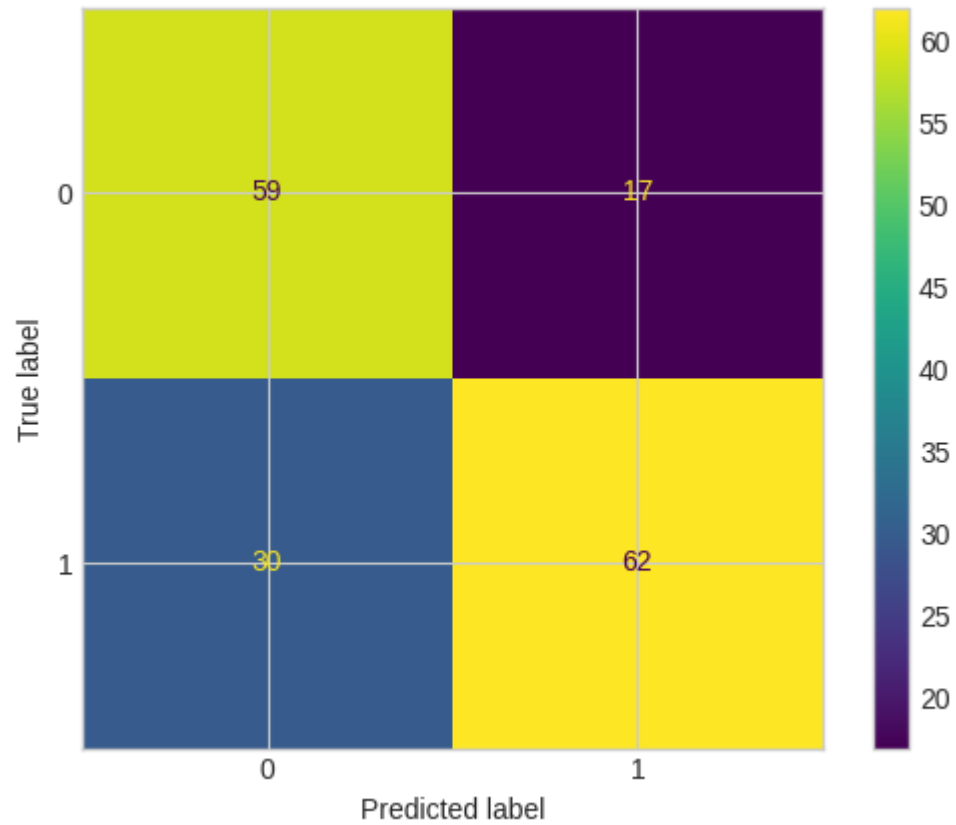
4. Tampilkan hasil akurasi dan tabel confusion matrix nya

Jawab:

- **Random Forest**

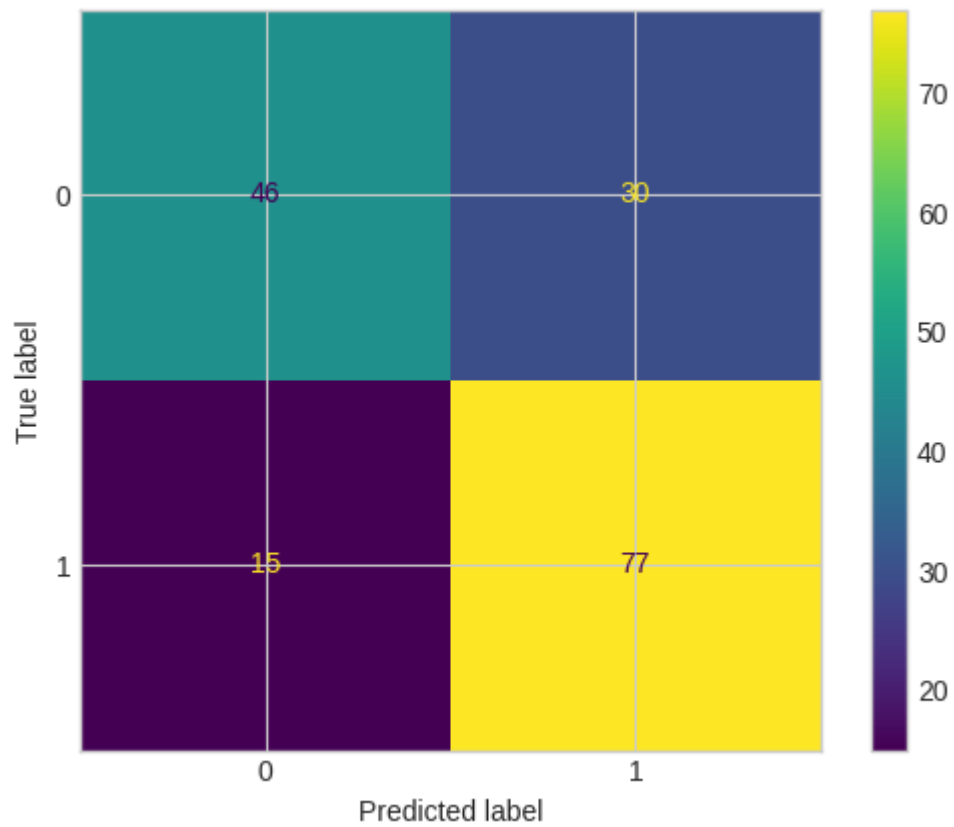
Train Accuracy: 0.9955089820359282

Testing Accuracy: 0.7202380952380952



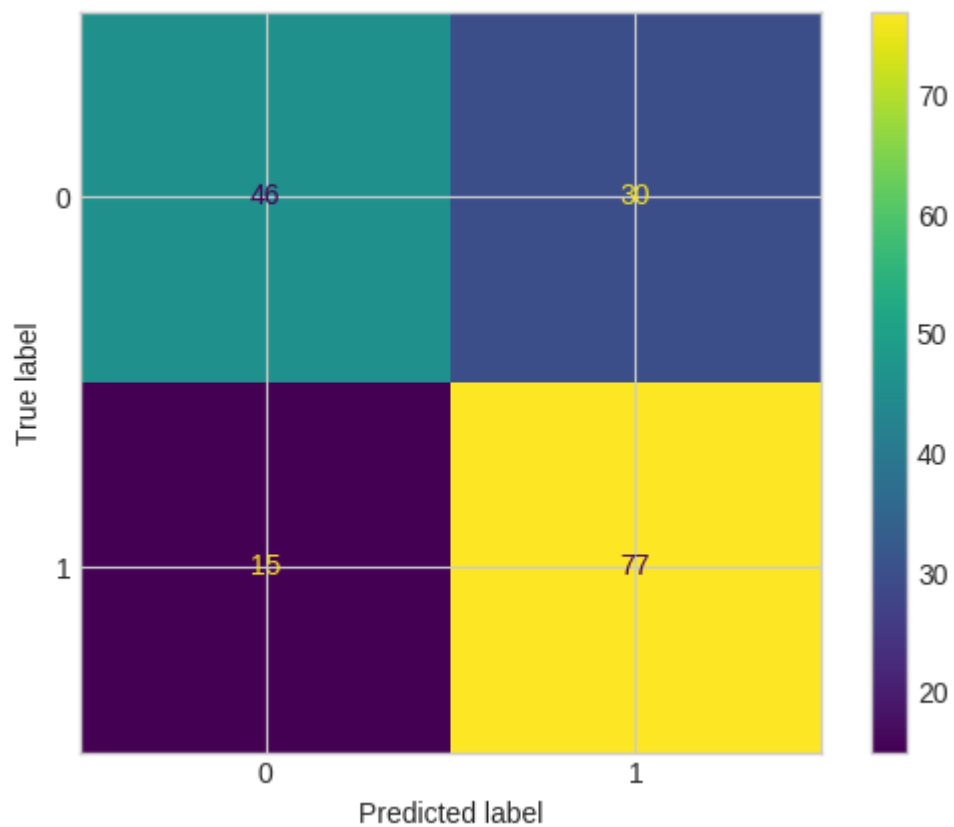
- **Neural Network**

Accuracy of the network: 73 %



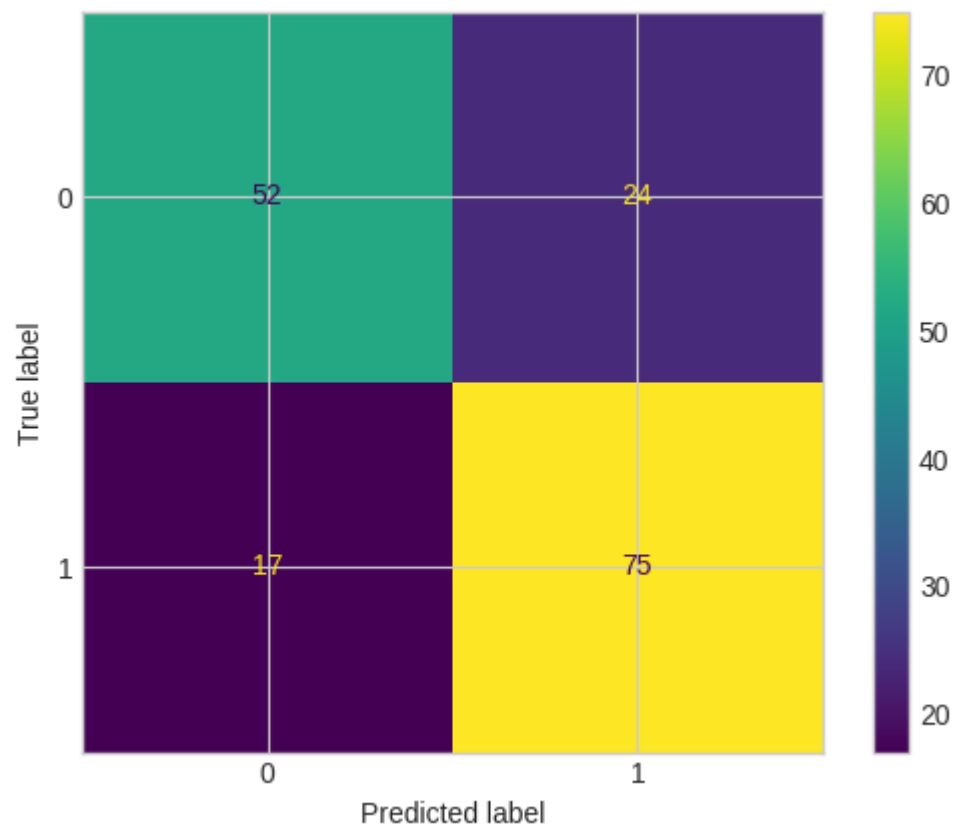
- **SVM**

Accuracy: 0.7440476190476191



- **Logistic Regression**

Train Accuracy: 0.9251497005988024
Test Accuracy: 0.7559523809523809



KESIMPULAN (Menggunakan 2 label {'Positive', 'Negative'})

SEHINGGA dengan demikian hasil proses perbaikan akurasi didapatkan dengan komposisi:

	Decision Tree	Neural Network
Prosentase Akurasi	0,80	0,89
Komposisi yang digunakan	{'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 1, 'min_samples_split': 25}, Tidak menggunakan SMOTE, Menggunakan Feature Selection (941 Fitur),	learning_rate =0.0011364758061944594, weight_decay= 2.5761143101268556e-05, sel di layer 1 = 500

No	Algoritma	Presentase Akurasi	Komposisi yang digunakan
1	Decision Tree	0,80	{'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 1, 'min_samples_split': 25}
2	Random Forest	0,8715	Menggunakan feature selection dengan k=576, ccp_alpha=0, class_weight=balanced, max_depth=80, min_samples_split=25, n_estimators=48, Menggunakan SMOTE
3	Neural Network	0,89	Menggunakan feature selection dengan k=576, learning_rate =0.0011364758061944594, weight_decay= 2.5761143101268556e-05, sel di layer 1 = 500, Menggunakan SMOTE

4	SVM	0,8452	Menggunakan feature selection dengan k=576, kernel='sigmoid', C=1.5106108189353196, degree=1, coef0=0.647517407284433, tol=1.8359221883440502, Menggunakan SMOTE
5	Logistic Regression	0,8392	Menggunakan feature selection dengan k=576, random_state=16, Menggunakan SMOTE

Kesimpulan:

Neural Network menunjukkan akurasi tertinggi (0.89), diikuti oleh Random Forest (0.8715). Mayoritas model (Random Forest, Neural Network, SVM, Logistic Regression) menggunakan feature selection dan SMOTE, yang kemungkinan membantu dalam meningkatkan kinerja mereka. Decision Tree memiliki akurasi terendah di antara semua algoritma.

3b. PROSES KLASIFIKASI MENGGUNAKAN ALGORITMA RANDOM FOREST, SVM, NEURAL NETWORK, LOGISTIC REGRESSION (Menggunakan 3 Kategori Label: Positive, Negative, dan Neutral)

1. Tulis/Screenshot Import Library yang digunakan

Jawab:

```
# Data Processing
import pandas as pd
import numpy as np

# Modelling
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV,
train_test_split, cross_val_score, GridSearchCV
from scipy.stats import randint

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier, plot_tree,
export_graphviz
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    ConfusionMatrixDisplay,
    roc_auc_score,
```

```

        roc_curve
    )

import matplotlib.pyplot as plt
import seaborn as sns # Untuk visualisasi yang lebih menarik
import graphviz # Untuk visualisasi graphviz
import nltk # Untuk pra-pemrosesan teks lebih lanjut
from nltk.stem import WordNetLemmatizer # Untuk lemmatization
from nltk.corpus import stopwords as nltk_stopwords # Untuk custom
stop words

# Feature selection
from sklearn.feature_selection import SelectKBest, chi2

# Download resource NLTK yang mungkin dibutuhkan
# Catch the LookupError directly when nltk.data.find fails
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    print("NLTK resource 'wordnet' not found. Downloading...")
    nltk.download('wordnet')
except Exception as e:
    print(f"An unexpected error occurred while
checking/downloading 'wordnet': {e}")

try:
    nltk.data.find('corpora/omw-1.4')
except LookupError:
    print("NLTK resource 'omw-1.4' not found. Downloading...")
    nltk.download('omw-1.4') # WordNet multilingual resource
except Exception as e:
    print(f"An unexpected error occurred while
checking/downloading 'omw-1.4': {e}")

try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    print("NLTK resource 'stopwords' not found. Downloading...")
    nltk.download('stopwords')
except Exception as e:

```

```

    print(f"An unexpected error occurred while
checking/downloading 'stopwords': {e}")

# Smote
from imblearn.over_sampling import SMOTE

# Pytorch
import torch
from torchvision import datasets, transforms
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Random
import random

# Import svm model
from sklearn import svm

# Logistic regression
# import the class
from sklearn.linear_model import LogisticRegression

# Pengaturan umum untuk plot agar lebih menarik
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis") # Atau palet lain seperti 'pastel',
'muted'

```

2. Tulis/screenshot import dataset yang digunakan

Jawab:

```

# path =
'https://raw.githubusercontent.com/LatiefDataVisionary/text-mining-and-natural-language-processing-college-task/refs/heads/main/datasets/r

```

```

amadan_labeled_sentiment.csv'
path =
'https://raw.githubusercontent.com/notnsas/cautious-eureka/refs/heads
/main/dataset/data_3_kelas_real.csv'
df = pd.read_csv(path)

print(f"Dataset berhasil di-load dari: {path}")
print(f"Jumlah baris: {df.shape[0]}, Jumlah kolom: {df.shape[1]}")

```

3. Pemrosesan pembagian (split data) Data Training dan Data Testing yang digunakan

a. Tulis/screenshot codingnya

Jawab:

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42, # Untuk reproduktifitas hasil
    # stratify=y # Mempertahankan proporsi kelas sentimen pada
data training dan testing
)

print("Ukuran Data Setelah Pembagian:")
print(f"X_train shape: {X_train.shape}, y_train shape:
{y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape:
{y_test.shape}")

print("\nDistribusi kelas pada data training (proporsi):")
print(y_train.value_counts(normalize=True))
print(y_train.value_counts())

print("\nDistribusi kelas pada data testing (proporsi):")
print(y_test.value_counts(normalize=True))
print(y_test.value_counts())

```

b. Tulis/screenshot hasilnya

Jawab:

```
Ukuran Data Setelah Pembagian:
X_train shape: (668, 937), y_train shape: (668,)
X_test shape: (168, 937), y_test shape: (168,)

Distribusi kelas pada data training (proporsi):
sentiment
1    0.525449
2    0.272455
0    0.202096
Name: proportion, dtype: float64
sentiment
1    351
2    182
0    135
Name: count, dtype: int64

Distribusi kelas pada data testing (proporsi):
sentiment
1    0.535714
2    0.309524
0    0.154762
Name: proportion, dtype: float64
sentiment
1     90
2     52
0     26
Name: count, dtype: int64
```

4. Proses pemodelan algoritma Random Forest, SVM, Neural net, Logistic Regression

a. Tulis/screenshot codingnya

Jawab:

- Random Forest

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

- Neural Network

```
tensor_x =
torch.from_numpy(X_train.astype('float32').todense())
tensor_y = torch.Tensor(y_train)

tensor_x_test =
torch.Tensor(X_test.astype('float32').todense()) # transform
to torch tensor
tensor_y_test = torch.Tensor(y_test.to_numpy())

train_dataset = TensorDataset(tensor_x, tensor_y)
test_dataset = TensorDataset(tensor_x_test, tensor_y_test)

loader_train = DataLoader(train_dataset, batch_size=10,
shuffle=False)
loader_test = DataLoader(test_dataset, batch_size=32,
shuffle=False)
class Net(nn.Module):
    def __init__(self, input_dim, classes):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 500)
        self.fc2 = nn.Linear(500, classes)
        # self.fc3 = nn.Linear(100, 30)
        # self.fc4 = nn.Linear(30, 30)
        # self.fc5 = nn.Linear(30, classes)
        # self.fc5 = nn.Linear(84, classes)

    def forward(self, x):
        x = x.float()
        # x = F.sigmoid(self.fc1(x))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        # x = F.relu(self.fc3(x))
        # x = F.relu(self.fc4(x))
        # x = self.fc5(x)
```

```

        # x = self.fc1(x)

        # x = F.relu(self.fc2(x))
        # x = self.fc(x)
        return x

criterion = nn.CrossEntropyLoss()
for j in range(1):
    net = Net(937, 3)

    # print(f"lr : {lr}")
    optimizer = optim.Adam(net.parameters(),
lr=1.584618639910894e-05) # 1.584618639910894e-05
    for epoch in range(50): # loop over the dataset multiple
times

        running_loss = 0.0
        for i, data in enumerate(loader_train, 0):
            # get the inputs; data is a list of [inputs,
labels]

            inputs, labels = data
            labels = labels.long()
            # print(inputs.dtype)
            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 64 == 500: # print every 2000
mini-batches

                print(f'[{epoch + 1}, {i + 1:5d}] loss:

```

```

{running_loss / 2000:.3f}')
        running_loss = 0.0

        print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss
/ 2000:.3f}')
        correct = 0
        total = 0
        # since we're not training, we don't need to calculate the
gradients for our outputs
        with torch.no_grad():
            for data in loader_test:
                images, labels = data
                # calculate outputs by running images through the
network
                outputs = net(images)
                # the class with the highest energy is what we
choose as prediction
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        print(f'Accuracy of the network: {100 * correct // total}
%')

        print('Finished Training')

```

- SVM

```

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

```

- Logistic Regression


```
# instantiate the model (using the default parameters)
logreg = LogisticRegression(random_state=16)

# fit the model with data
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
```

b. Tulis/screenshot hasilnya

Jawab:

- Random Forest

```
rf
RandomForestClassifier
RandomForestClassifier()
```

- Neural Network

```
[1, 67] loss: 0.036
[2, 67] loss: 0.036
[3, 67] loss: 0.036
[4, 67] loss: 0.035
[5, 67] loss: 0.035
[6, 67] loss: 0.034
[7, 67] loss: 0.034
[8, 67] loss: 0.033
[9, 67] loss: 0.033
[10, 67] loss: 0.032
[11, 67] loss: 0.032
[12, 67] loss: 0.031
[13, 67] loss: 0.031
[14, 67] loss: 0.030
[15, 67] loss: 0.029
[16, 67] loss: 0.028
[17, 67] loss: 0.028
[18, 67] loss: 0.027
[19, 67] loss: 0.026
[20, 67] loss: 0.025
[21, 67] loss: 0.025
[22, 67] loss: 0.024
[23, 67] loss: 0.023
[24, 67] loss: 0.023
[25, 67] loss: 0.022
```

```
[26, 67] loss: 0.021
[27, 67] loss: 0.021
[28, 67] loss: 0.020
[29, 67] loss: 0.020
[30, 67] loss: 0.019
[31, 67] loss: 0.019
[32, 67] loss: 0.018
[33, 67] loss: 0.018
[34, 67] loss: 0.018
[35, 67] loss: 0.017
[36, 67] loss: 0.017
[37, 67] loss: 0.017
[38, 67] loss: 0.016
[39, 67] loss: 0.016
[40, 67] loss: 0.016
[41, 67] loss: 0.016
[42, 67] loss: 0.016
[43, 67] loss: 0.015
[44, 67] loss: 0.015
[45, 67] loss: 0.015
[46, 67] loss: 0.015
[47, 67] loss: 0.015
[48, 67] loss: 0.015
[49, 67] loss: 0.014
[50, 67] loss: 0.014
Accuracy of the
network: 83 %
Finished Training
```

- SVM

```
clf
```

```
▼ SVC ⓘ ?  
SVC(kernel='linear')
```

- Logistic Regression

```
logreg
```

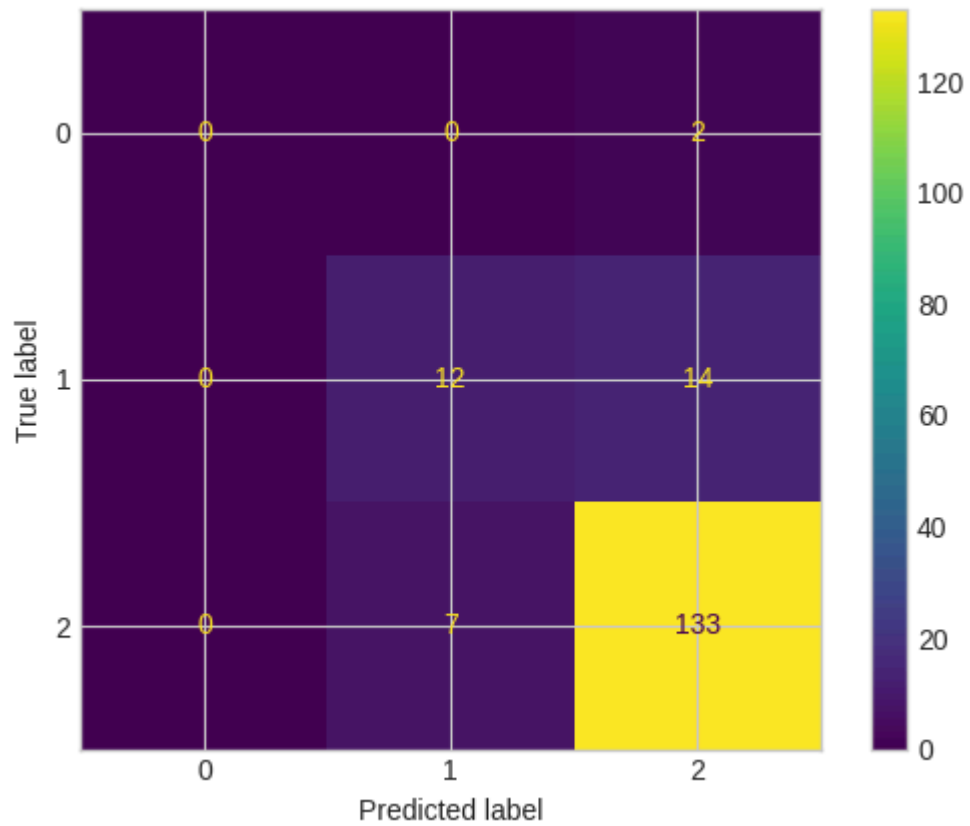
```
▼ LogisticRegression ⓘ ?  
LogisticRegression(random_state=16)
```

5. Tampilkan hasil akurasi dan tabel confusion matrix nya

Jawab:

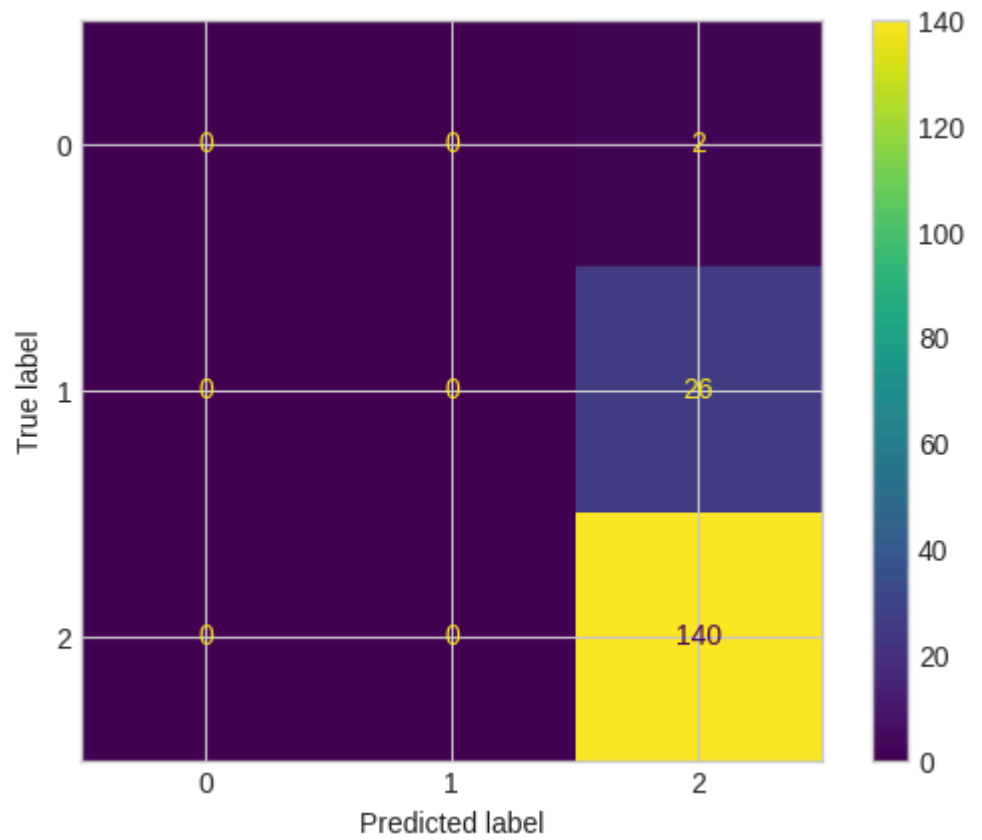
- Random Forest

Accuracy: 0.8630952380952381



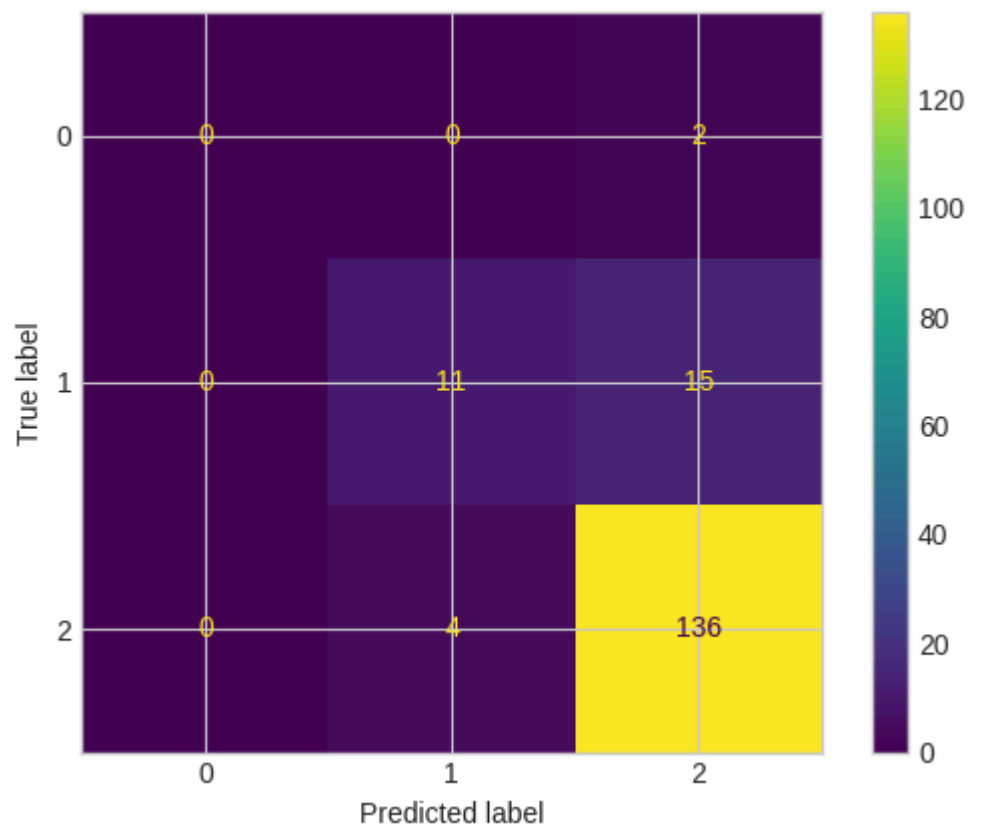
- Neural Network

Accuracy of the network: 83 %



- SVM

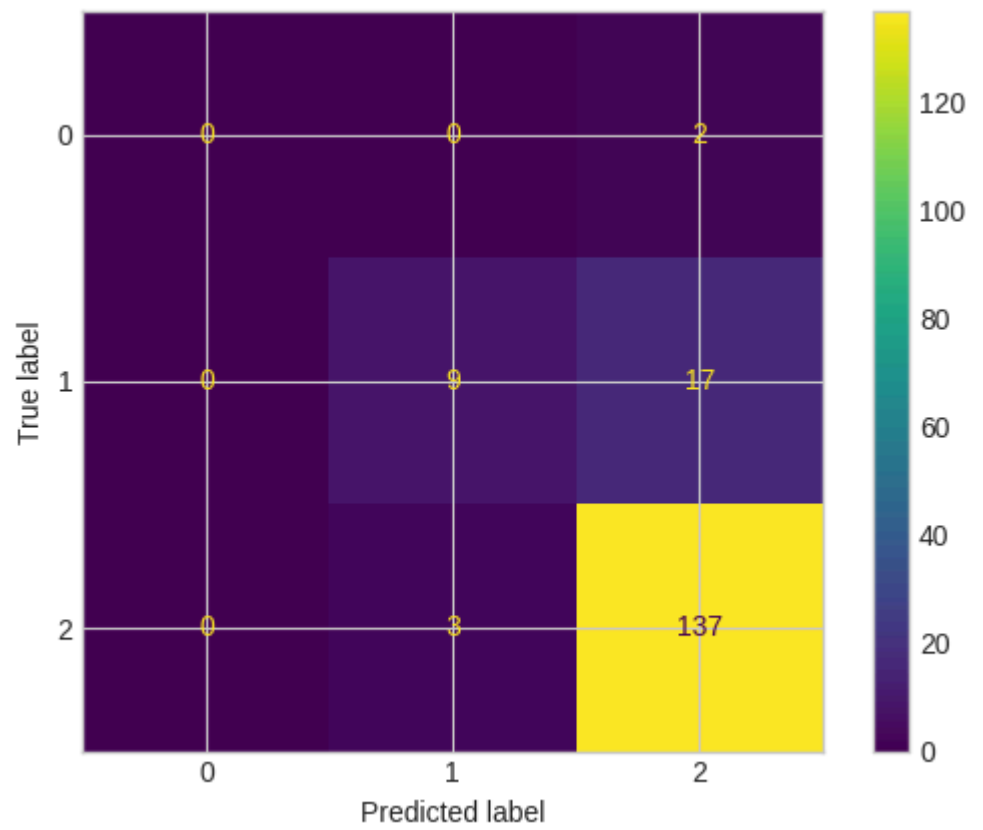
Accuracy: 0.875



- **Logistic Regression**

Train Accuracy: 0.8952095808383234

Test Accuracy: 0.8690476190476191



KESIMPULAN (Menggunakan 3 label {'Positive', 'Negative', 'Neutral'})

SEHINGGA dengan demikian hasil proses perbaikan akurasi didapatkan dengan komposisi:

	Decision Tree	Neural Network
Prosentase Akurasi	0,8725	0,9591346153846153
Komposisi yang digunakan	{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 8, 'min_samples_split': 2}, Tidak menggunakan SMOTE, Menggunakan Feature Selection (836 Fitur)	sel layer 1 = 700, learning rate=0.1211844057055417 5, weight_decay=3.18973350 2609255e-050, menggunakan SMOTE

No	Algoritma	Presentase Akurasi	Komposisi yang digunakan
1	Decision Tree	0,8725	{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 8, 'min_samples_split': 2}
2	Random Forest	0,9495	Menggunakan feature selection dengan k=1151, min_samples_split=3, n_estimators=260, menggunakan SMOTE,
3	Neural Network	0,9591	Menggunakan feature selection dengan k=1151,

			sel layer 1 = 700, learning rate=0.1211844057055417 5, weight_decay=3.18973350 2609255e-050, menggunakan SMOTE
4	Support Vector Machine (SVM)	0,8725	Menggunakan feature selection dengan k=1151, cache=1.023104221681953 5, coef0=0.528344975398888 7, tol=1.522225251538773, degree=1, kernel='sigmoid', menggunakan SMOTE
5	Logistic Regression	0,9423	Menggunakan feature selection dengan k=1151, C=77882.5298069693, menggunakan SMOTE

Kesimpulan:

Neural Network menunjukkan akurasi tertinggi (0.9591), diikuti oleh Random Forest (0.9495). Baik Neural Network maupun Random Forest menggunakan SMOTE, yang berkontribusi pada peningkatan akurasi mereka. Decision Tree dan Support Vector Machine (SVM) memiliki akurasi yang relatif lebih rendah (0.8725).

Tabel Kesimpulan Akhir

No	Algoritma	Presentase Akurasi (2 label: [Positive, Negative]	Presentase Akurasi (3 label: [Positive, Negative, Neutral]
1	Decision Tree	0,80	0,8725
2	Random Forest	0,8715	0,9495
3	Neural Network	0,89	0,9591 (96%)
4	Support Vector Machine (SVM)	0,8452	0,8725
5	Logistic Regression	0,8392	0,9423

Kesimpulan Akhir:

Model cenderung memiliki akurasi lebih tinggi saat memprediksi 3 label (Positive, Negative, Neutral) dibandingkan dengan 2 label (Positive, Negative). Neural Network dan Random Forest secara konsisten menunjukkan akurasi tertinggi di kedua skenario, dengan Neural Network sedikit unggul pada prediksi 3 label. Logistic Regression juga menunjukkan peningkatan akurasi yang signifikan dengan 3 label.

