

```

from __future__ import annotations

import re
from pathlib import Path
from typing import Dict, List

import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    precision_score,
    recall_score,
)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

from pathlib import Path

DATA_PATH = Path("/Users/afnannasseer/Downloads/unbalanceddataset.csv")
RESULTS_PATH = Path("/Users/afnannasseer/Downloads/nlp_model_results.csv")
PREDICTIONS_PATH =
Path("/Users/afnannasseer/Downloads/nlp_test_predictions.csv")
RANDOM_STATE = 42
TEST_SIZE = 0.2

def clean_text(text: str) -> str:
    if pd.isna(text):
        return ""

    text = str(text).lower()

    text = re.sub(r"http\S+|www\.\S+", " URL ", text)
    text = re.sub(r"@w+", " USER ", text)

    text = re.sub(r"#(\w+)", r"\1", text)

    text = text.replace("`", "'")

```

```

text = re.sub(r"^[a-z0-9\s'!?.,]", " ", text)

text = re.sub(r"\s+", " ", text).strip()

return text

def preprocess_dataframe(df: pd.DataFrame) -> pd.DataFrame:
    required_columns = {"text", "sentiment"}
    missing = required_columns - set(df.columns)

    if missing:
        raise ValueError(f"الأعمدة المطلوبة غير موجودة: {missing}")

    df = df.copy()
    df = df.dropna(subset=["text", "sentiment"])
    df["clean_text"] = df["text"].astype(str).apply(clean_text)
    df = df[df["clean_text"].str.len() > 0].reset_index(drop=True)

    return df

def evaluate_predictions(y_true: np.ndarray, y_pred: np.ndarray) -> Dict[str, float]:
    return {
        "accuracy": accuracy_score(y_true, y_pred),
        "precision_macro": precision_score(y_true, y_pred, average="macro",
zero_division=0),
        "recall_macro": recall_score(y_true, y_pred, average="macro",
zero_division=0),
        "f1_macro": f1_score(y_true, y_pred, average="macro",
zero_division=0),
        "precision_weighted": precision_score(y_true, y_pred,
average="weighted", zero_division=0),
        "recall_weighted": recall_score(y_true, y_pred, average="weighted",
zero_division=0),
        "f1_weighted": f1_score(y_true, y_pred, average="weighted",
zero_division=0),
    }

def print_model_report(model_name: str, y_true: np.ndarray, y_pred:
np.ndarray, label_encoder: LabelEncoder) -> None:
    print("\n" + "=" * 80)
    print(f"Model: {model_name}")
    print("=" * 80)

    print("Confusion Matrix:")
    print(confusion_matrix(y_true, y_pred))

    print("\nClassification Report:")

```

```

print(
    classification_report(
        y_true,
        y_pred,
        target_names=label_encoder.classes_,
        zero_division=0
    )
)

def build_pipelines() -> Dict[str, Pipeline]:
    pipelines = {
        # Bag of Words + Naive Bayes
        "bow_unigram_multinomial_nb": Pipeline([
            ("vectorizer", CountVectorizer(ngram_range=(1, 1), min_df=2)),
            ("classifier", MultinomialNB())
        ]),

        # TF-IDF Word Level + Logistic Regression
        "tfidf_word_logistic_regression": Pipeline([
            ("vectorizer", TfidfVectorizer(ngram_range=(1, 2), min_df=2,
max_df=0.95)),
            ("classifier", LogisticRegression(max_iter=2000,
class_weight="balanced"))
        ]),

        # TF-IDF Word Level + Linear SVM
        "tfidf_word_linear_svm": Pipeline([
            ("vectorizer", TfidfVectorizer(ngram_range=(1, 2), min_df=2,
max_df=0.95)),
            ("classifier", LinearSVC(class_weight="balanced"))
        ]),

        # TF-IDF Character Level + Linear SVM
        "tfidf_char_linear_svm": Pipeline([
            ("vectorizer", TfidfVectorizer(analyzer="char_wb",
ngram_range=(3, 5), min_df=2)),
            ("classifier", LinearSVC(class_weight="balanced"))
        ]),
    }

    return pipelines

def main():

    df = pd.read_csv(DATA_PATH)

    print("شكل البيانات الأصلي:", df.shape)
    print("أسماء الأعمدة:", list(df.columns))

```

```

df = preprocess_dataframe(df)

print("\nشكل البيانات بعد التنظيف:", df.shape)
print("\nتوزيع الكلاسات:")
print(df["sentiment"].value_counts())

X = df["clean_text"]
y_raw = df["sentiment"]

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y_raw)

X_train, X_test, y_train, y_test, original_text_train,
original_text_test = train_test_split(
    X,
    y,
    df["text"],
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE,
    stratify=y
)

print("\nحجم بيانات التدريب:", len(X_train))
print("\nحجم بيانات الاختبار:", len(X_test))

# بناء وتدريب النماذج
pipelines = build_pipelines()

results: List[Dict[str, float]] = []
predictions_df = pd.DataFrame({
    "original_text": original_text_test.reset_index(drop=True)
})

for model_name, pipeline in pipelines.items():
    pipeline.fit(X_train, y_train)

    y_pred = pipeline.predict(X_test)

    metrics = evaluate_predictions(y_test, y_pred)
    metrics["model"] = model_name
    results.append(metrics)

    print_model_report(model_name, y_test, y_pred, label_encoder)

    predictions_df[model_name] = label_encoder.inverse_transform(y_pred)

predictions_df["true_label"] = label_encoder.inverse_transform(y_test)

```

```
results_df = pd.DataFrame(results).sort_values(by="f1_weighted",
ascending=False)

print("\n" + "=" * 80)
print("Final Ranked Results")
print("=" * 80)
print(results_df.to_string(index=False))

results_df.to_csv(RESULTS_PATH, index=False)
predictions_df.to_csv(PREDICTIONS_PATH, index=False)

print(f"\nتم حفظ النتائج في: {RESULTS_PATH}")
print(f"تم حفظ توقعات الاختبار في: {PREDICTIONS_PATH}")

if __name__ == "__main__":
    main()
```

Output

afnannasseer@alMacBook-Air-alkhas-bafnan ~ % /usr/local/bin/python3
/Users/afnannasseer/Downloads/nlp_classification_pipeline.py

شكل البيانات الأصلي: (٢, ٢٦٠٠)
أسماء الأعمدة: ['text', 'sentiment']

شكل البيانات بعد التنظيف: (٣, ٢٥٩٩)

توزيع الكلاسات:

sentiment

positive 2000

negative 599

Name: count, dtype: int64

حجم بيانات التدريب: ٢٠٧٩

حجم بيانات الاختبار: ٥٢٠

```
=====
=====
Model: bow_unigram_multinomial_nb
=====
=====
:Confusion Matrix
[[ ٦٤  ٥٦ ]
 [٣٨٠  ٢٠ ]
```

:Classification Report

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

negative	0.74	0.47	0.57	120
----------	------	------	------	-----

positive	0.86	0.95	0.90	400
----------	------	------	------	-----

accuracy			0.84	520
----------	--	--	------	-----

macro avg	0.80	0.71	0.74	520
-----------	------	------	------	-----

weighted avg	0.83	0.84	0.82	520
--------------	------	------	------	-----

```
=====
=====
Model: tfidf_word_logistic_regression
=====
=====
:Confusion Matrix
[[ ٣٨  ٨٢ ]
```

[[337 63]

:Classification Report

precision recall f1-score support

negative 0.57 0.68 0.62 120

positive 0.90 0.84 0.87 400

accuracy 0.81 520

macro avg 0.73 0.76 0.74 520

weighted avg 0.82 0.81 0.81 520

=====
=====

Model: tfidf_word_linear_svm

=====
=====

:Confusion Matrix

[40 70]

[[308 42]

:Classification Report

precision recall f1-score support

negative 0.64 0.62 0.63 120

positive 0.89 0.90 0.89 400

accuracy 0.83 520

macro avg 0.76 0.76 0.76 520

weighted avg 0.83 0.83 0.83 520

=====
=====

Model: tfidf_char_linear_svm

=====
=====

:Confusion Matrix

[39 81]

[[361 39]

:Classification Report

precision recall f1-score support

negative 0.68 0.68 0.68 120

positive 0.90 0.90 0.90 400

```

=====
Final Ranked Results
=====
=====
accuracy precision_macro recall_macro f1_macro precision_weighted
recall_weighted f1_weighted model
.,.80.... .,788870. .,788870. .,788870. .,80....
tfidf_char_linear_svm .,80.... .,80....
.,831266 .,762284 .,76.... .,764682 .,832692
tfidf_word_linear_svm .,831946 .,832692
.,828391 .,730901 .,7.8333 .,796349 .,838462
bow_unigram_multinomial_nb .,82404. .,838462
.,821786 .,744273 .,762917 .,732.92 .,8.05769
tfidf_word_logistic_regression .,811798 .,8.05769

```

تم حفظ توقعات الاختبار في `/Users/afnannasseer/Downloads/nlp_test_predictions.csv`: