

```

In [1]: import os
import re
import pandas as pd
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import (
    ENGLISH_STOP_WORDS,
    CountVectorizer,
    TfidfVectorizer,
)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
)
from imblearn.over_sampling import RandomOverSampler

RANDOM_STATE = 42

def locate_dataset():
    """Locate the CSV in Colab, Jupyter, or the current directory."""
    candidates = [
        "/content/unbalanceddataset.csv",
        "/mnt/data/unbalanceddataset.csv",
        "unbalanceddataset.csv",
    ]
    for path in candidates:
        if os.path.exists(path):
            return path
    raise FileNotFoundError(
        "Upload unbalanceddataset.csv to the notebook/session directory."
    )

stemmer = PorterStemmer()
stop_words = set(ENGLISH_STOP_WORDS)

def preprocess_text(text):
    """Clean English text using the steps demonstrated in the NLP lecture
    text = str(text).lower()
    text = re.sub(r"http\S+|www\.\S+", " ", text)      # URLs
    text = re.sub(r"<.*?>", " ", text)                # HTML
    text = re.sub(r"@w+|#\w+", " ", text)             # mentions/hashtags
    text = re.sub(r"^[a-z\s]", " ", text)              # punctuation/number
    tokens = re.findall(r"\b[a-z]{2,}\b", text)         # tokenization
    tokens = [
        stemmer.stem(token)
        for token in tokens
        if token not in stop_words                      # stop-word removal
    ]

```

```

return " ".join(tokens)

def evaluate_model(model_name, representation, y_true, y_pred):
    """Calculate the evaluation measures used in the lectures."""
    return {
        "Feature Representation": representation,
        "Model": model_name,
        "Accuracy": accuracy_score(y_true, y_pred),
        "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
        ),
        # Macro F1 is important for an imbalanced dataset because it give
        # equal importance to each class.
        "F1 (macro)": f1_score(
            y_true, y_pred, average="macro", zero_division=0
        ),
    }

def main():
    dataset_path = locate_dataset()
    df = pd.read_csv(dataset_path)

    print("Dataset shape:", df.shape)
    print("\nClass distribution before preprocessing:")
    print(df["sentiment"].value_counts())

    # Remove missing values and duplicate records.
    df = df.dropna(subset=["text", "sentiment"]).drop_duplicates().copy()

    # Apply preprocessing.
    df["clean_text"] = df["text"].apply(preprocess_text)
    df = df[df["clean_text"].str.strip().ne("")]

    # Stratification preserves the class distribution in both subsets.
    X_train, X_test, y_train, y_test = train_test_split(
        df["clean_text"],
        df["sentiment"],
        test_size=0.20,
        random_state=RANDOM_STATE,
        stratify=df["sentiment"],
    )

    print("\nTraining class distribution:")
    print(y_train.value_counts())
    print("\nTesting class distribution:")
    print(y_test.value_counts())

    representations = {
        "Bag of Words": CountVectorizer(
            ngram_range=(1, 1),
            min_df=2,
        ),
    }

```

```

        "TF-IDF (1-2 grams)": TfidfVectorizer(
            ngram_range=(1, 2),
            min_df=2,
            max_features=15000,
        ),
    }

    # Class weights compensate for the minority class in applicable models
    models = {
        "Logistic Regression": LogisticRegression(
            max_iter=2000,
            class_weight="balanced",
            random_state=RANDOM_STATE,
        ),
        "Multinomial Naive Bayes": MultinomialNB(),
        "Random Forest": RandomForestClassifier(
            n_estimators=200,
            class_weight="balanced",
            random_state=RANDOM_STATE,
            n_jobs=-1,
        ),
    }

    results = []

    for representation_name, vectorizer in representations.items():
        X_train_vec = vectorizer.fit_transform(X_train)
        X_test_vec = vectorizer.transform(X_test)

        for model_name, model in models.items():
            model.fit(X_train_vec, y_train)
            y_pred = model.predict(X_test_vec)

            results.append(
                evaluate_model(
                    model_name,
                    representation_name,
                    y_test,
                    y_pred,
                )
            )

            print("\n" + "=" * 72)
            print(f"{representation_name} + {model_name}")
            print(classification_report(y_test, y_pred, zero_division=0))
            print(
                "Confusion matrix [negative, positive]:\n",
                confusion_matrix(
                    y_test,
                    y_pred,
                    labels=["negative", "positive"],
                ),
            )

    # Lecture 9 method: oversample only the training data, never the test
    tfidf_ros = TfidfVectorizer(
        ngram_range=(1, 2),
        min_df=2,
        max_features=15000,
    )

```

```

X_train_tfidf = tfidf_ros.fit_transform(X_train)
X_test_tfidf = tfidf_ros.transform(X_test)

oversampler = RandomOverSampler(random_state=RANDOM_STATE)
X_train_ros, y_train_ros = oversampler.fit_resample(
    X_train_tfidf,
    y_train,
)

ros_model = LogisticRegression(
    max_iter=2000,
    random_state=RANDOM_STATE,
)
ros_model.fit(X_train_ros, y_train_ros)
ros_pred = ros_model.predict(X_test_tfidf)

results.append(
    evaluate_model(
        "Logistic Regression",
        "TF-IDF (1-2 grams) + Random Oversampling",
        y_test,
        ros_pred,
    )
)

print("\n" + "=" * 72)
print("TF-IDF + Logistic Regression + Random Oversampling")
print("Resampled training distribution:")
print(pd.Series(y_train_ros).value_counts())
print(classification_report(y_test, ros_pred, zero_division=0))
print(
    "Confusion matrix [negative, positive]:\n",
    confusion_matrix(
        y_test,
        ros_pred,
        labels=["negative", "positive"],
    ),
)

results_df = pd.DataFrame(results).sort_values(
    ["F1 (macro)", "Accuracy"],
    ascending=False,
)

print("\nFinal model comparison:")
print(results_df.round(4).to_string(index=False))
results_df.to_csv("nlp_model_results.csv", index=False)

best = results_df.iloc[0]
print("\nBest model based on Macro F1:")
print(best.to_string())

if __name__ == "__main__":
    main()

```

Dataset shape: (2600, 2)

Class distribution before preprocessing:

sentiment

positive 2000

negative 600

Name: count, dtype: int64

Training class distribution:

sentiment

positive 1596

negative 479

Name: count, dtype: int64

Testing class distribution:

sentiment

positive 399

negative 120

Name: count, dtype: int64

=====

Bag of Words + Logistic Regression

	precision	recall	f1-score	support
negative	0.62	0.79	0.69	120
positive	0.93	0.85	0.89	399
accuracy			0.84	519
macro avg	0.77	0.82	0.79	519
weighted avg	0.86	0.84	0.84	519

Confusion matrix [negative, positive]:

[[95 25]

[59 340]]

=====

Bag of Words + Multinomial Naive Bayes

	precision	recall	f1-score	support
negative	0.76	0.54	0.63	120
positive	0.87	0.95	0.91	399
accuracy			0.86	519
macro avg	0.82	0.75	0.77	519
weighted avg	0.85	0.86	0.85	519

Confusion matrix [negative, positive]:

[[65 55]

[20 379]]

=====

Bag of Words + Random Forest

	precision	recall	f1-score	support
negative	0.67	0.70	0.69	120
positive	0.91	0.90	0.90	399
accuracy			0.85	519
macro avg	0.79	0.80	0.79	519
weighted avg	0.85	0.85	0.85	519

Confusion matrix [negative, positive]:

```
[[ 84 36]
 [ 41 358]]
```

=====

TF-IDF (1-2 grams) + Logistic Regression

	precision	recall	f1-score	support
negative	0.62	0.79	0.70	120
positive	0.93	0.86	0.89	399
accuracy			0.84	519
macro avg	0.78	0.82	0.80	519
weighted avg	0.86	0.84	0.85	519

Confusion matrix [negative, positive]:

```
[[ 95 25]
 [ 57 342]]
```

=====

TF-IDF (1-2 grams) + Multinomial Naive Bayes

	precision	recall	f1-score	support
negative	0.96	0.22	0.35	120
positive	0.81	1.00	0.89	399
accuracy			0.82	519
macro avg	0.89	0.61	0.62	519
weighted avg	0.84	0.82	0.77	519

Confusion matrix [negative, positive]:

```
[[ 26 94]
 [  1 398]]
```

=====

TF-IDF (1-2 grams) + Random Forest

	precision	recall	f1-score	support
negative	0.68	0.68	0.68	120
positive	0.90	0.90	0.90	399
accuracy			0.85	519
macro avg	0.79	0.79	0.79	519
weighted avg	0.85	0.85	0.85	519

Confusion matrix [negative, positive]:

```
[[ 81 39]
 [ 38 361]]
```

=====

TF-IDF + Logistic Regression + Random Oversampling

Resampled training distribution:

sentiment

positive 1596

negative 1596

Name: count, dtype: int64

	precision	recall	f1-score	support
negative	0.65	0.74	0.70	120

positive	0.92	0.88	0.90	399
accuracy			0.85	519
macro avg	0.79	0.81	0.80	519
weighted avg	0.86	0.85	0.85	519

Confusion matrix [negative, positive]:

```
[[ 89  31]
 [ 47 352]]
```

Final model comparison:

	Feature Representation	Model	Accuracy
Precision (weighted)	Recall (weighted)	F1 (weighted)	F1 (macro)
TF-IDF (1-2 grams) + Random Oversampling		Logistic Regression	0.8497
0.8579	0.8497	0.8529	0.7978
	TF-IDF (1-2 grams)	Logistic Regression	0.8420
0.8609	0.8420	0.8480	0.7957
	Bag of Words	Random Forest	0.8516
0.8539	0.8516	0.8527	0.7943
	Bag of Words	Logistic Regression	0.8382
0.8588	0.8382	0.8446	0.7917
	TF-IDF (1-2 grams)	Random Forest	0.8516
0.8512	0.8516	0.8514	0.7907
	Bag of Words	Multinomial Naive Bayes	0.8555
0.8482	0.8555	0.8462	0.7721
	TF-IDF (1-2 grams)	Multinomial Naive Bayes	0.8170
0.8446	0.8170	0.7686	0.6236

Best model based on Macro F1:

Feature Representation	TF-IDF (1-2 grams) + Random Oversampling
Model	Logistic Regression
Accuracy	0.849711
Precision (weighted)	0.85787
Recall (weighted)	0.849711
F1 (weighted)	0.85287
F1 (macro)	0.797784

In []: