

NLP Sentiment Classification on an Imbalanced Dataset

Project 2-2

Student Name: Asraa Alwattar

Student ID: 253001066

Objective

The objective of this project is to preprocess an NLP dataset, apply different feature representation methods, train multiple classification models, and evaluate their performance.

Libraries and Setup

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)
```

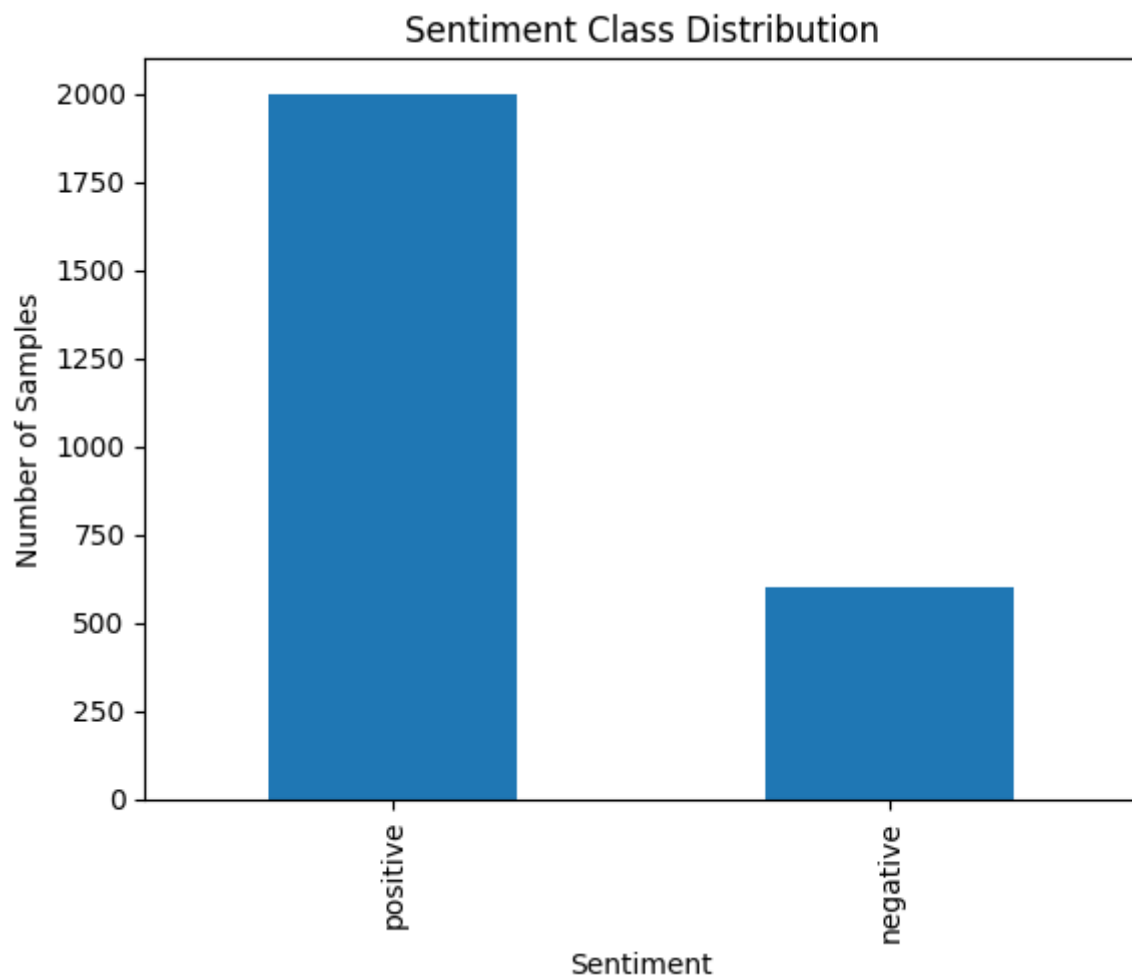
```
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
```

Dataset Loading and Exploration

```
df = pd.read_csv("/unbalanceddataset (1).csv")  
df.head()
```

	text	sentiment
0	Java Concurrency in Practice is probably the b...	positive
1	haha aww hun i bet you are more creative tha...	positive
2	_pickle lol, thank you very much Hope you`re h...	positive
3	Out for an evening on the town with jeremy. Sa...	negative
4	- just took over the #1 Most Endorsed spot on...	positive

```
df['sentiment'].value_counts().plot(kind='bar')  
  
plt.title('Sentiment Class Distribution')  
plt.xlabel('Sentiment')  
plt.ylabel('Number of Samples')  
plt.show()
```



```
print("Class Distribution BEFORE Balancing")
print("-----")
print(df['sentiment'].value_counts())

print("\nClass Distribution Percentage:")
print((df['sentiment'].value_counts(normalize=True) * 100).round(2))
```

```
Class Distribution BEFORE Balancing
-----
sentiment
positive    2000
negative     600
Name: count, dtype: int64

Class Distribution Percentage:
sentiment
positive    76.92
negative    23.08
Name: proportion, dtype: float64
```

Text Preprocessing

```
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+|www\S+', '', text)

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove numbers, punctuation and special characters
    text = re.sub(r'^a-zA-Z\s', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stop words
    tokens = [word for word in tokens if word not in stop_words]

    # Stemming
    tokens = [stemmer.stem(word) for word in tokens]

    # Reconstruct cleaned text
    clean_text = ' '.join(tokens)

    return clean_text
```

```
df['clean_text'] = df['text'].apply(preprocess_text)
```

```
df[['text', 'clean_text', 'sentiment']].head()
```

	text	clean_text	sentiment
0	Java Concurrency in Practice is probably the b...	java concurr practic probabl best java book iv...	positive
1	haha aww hun i bet you are more creative tha...	haha aww hun bet creativ	positive
2	_pickle lol, thank you very much Hope you're h...	pickl lol thank much hope your great day	positive
3	Out for an evening on the town with jeremy. Sa...	even town jeremi sad carri cant come	negative
4	- just took over the #1 Most Endorsed spot on...	took endors spot twindexxcom thank endors	positive

```
print("BEFORE PREPROCESSING:")
print(df['text'].iloc[0])

print("\nAFTER PREPROCESSING:")
print(df['clean_text'].iloc[0])
```

BEFORE PREPROCESSING:

Java Concurrency in Practice is probably the best Java book I've ever bought. There's a recipe in there for interrupting blocking IO ops

AFTER PREPROCESSING:

java concurr practic probabl best java book ive ever bought there recip interrupt block io op

```
X = df['clean_text']
y = df['sentiment']
```

Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=42,
    stratify=y
)
```

```
print("Training size:", len(X_train))
print("Testing size:", len(X_test))

print("\nTraining distribution:")
print(y_train.value_counts())

print("\nTesting distribution:")
print(y_test.value_counts())
```

Training size: 1820
Testing size: 780

Training distribution:
sentiment
positive 1400
negative 420
Name: count, dtype: int64

Testing distribution:
sentiment
positive 600
negative 180
Name: count, dtype: int64

Feature Representation

```
count_vectorizer = CountVectorizer()

X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)

print("CountVectorizer training shape:", X_train_count.shape)
```

CountVectorizer training shape: (1820, 3659)

```
tfidf_vectorizer = TfidfVectorizer()

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

print("TF-IDF training shape:", X_train_tfidf.shape)
```

TF-IDF training shape: (1820, 3659)

Model Training and Evaluation

```
def evaluate_model(model_name, model, X_train, X_test, y_train, y_test):

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(
        y_test,
        y_pred,
        pos_label='negative',
        zero_division=0
    )

    recall = recall_score(
        y_test,
        y_pred,
        pos_label='negative',
        zero_division=0
    )

    f1 = f1_score(
        y_test,
        y_pred,
        pos_label='negative',
        zero_division=0
    )

    print("=" * 60)
    print(model_name)
    print("=" * 60)

    print("Accuracy:", round(accuracy, 4))
    print("Precision (Negative):", round(precision, 4))
    print("Recall (Negative):", round(recall, 4))
    print("F1 Score (Negative):", round(f1, 4))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    return {
        'Model': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    }
```

```
results_tfidf = []
```

```

lr = LogisticRegression(max_iter=1000, random_state=42)

results_tfidf.append(
    evaluate_model(
        "Logistic Regression - TF-IDF",
        lr,
        X_train_tfidf,
        X_test_tfidf,
        y_train,
        y_test
    )
)

```

=====

Logistic Regression - TF-IDF

=====

Accuracy: 0.8244
 Precision (Negative): 0.9216
 Recall (Negative): 0.2611
 F1 Score (Negative): 0.4069

Classification Report:

	precision	recall	f1-score	support
negative	0.92	0.26	0.41	180
positive	0.82	0.99	0.90	600
accuracy			0.82	780
macro avg	0.87	0.63	0.65	780
weighted avg	0.84	0.82	0.78	780

Confusion Matrix:
 [[47 133]
 [4 596]]

```

nb = MultinomialNB()

results_tfidf.append(
    evaluate_model(
        "Naive Bayes - TF-IDF",
        nb,
        X_train_tfidf,
        X_test_tfidf,
        y_train,
        y_test
    )
)

```

=====

Naive Bayes - TF-IDF

=====

Accuracy: 0.791
 Precision (Negative): 1.0
 Recall (Negative): 0.0944
 F1 Score (Negative): 0.1726

Classification Report:

	precision	recall	f1-score	support
negative	1.00	0.09	0.17	180
positive	0.79	1.00	0.88	600
accuracy			0.79	780
macro avg	0.89	0.55	0.53	780
weighted avg	0.84	0.79	0.72	780

Confusion Matrix:
 [[17 163]
 [0 600]]

```

svm = LinearSVC(random_state=42)

results_tfidf.append(
    evaluate_model(
        "Linear SVM - TF-IDF",
        svm,
        X_train_tfidf,
        X_test_tfidf,
        y_train,
        y_test
    )
)

```

```

=====
Linear SVM - TF-IDF
=====

```

```

Accuracy: 0.8615
Precision (Negative): 0.8103
Recall (Negative): 0.5222
F1 Score (Negative): 0.6351

```

```

Classification Report:

```

	precision	recall	f1-score	support
negative	0.81	0.52	0.64	180
positive	0.87	0.96	0.91	600
accuracy			0.86	780
macro avg	0.84	0.74	0.77	780
weighted avg	0.86	0.86	0.85	780

```

Confusion Matrix:
[[ 94  86]
 [ 22 578]]

```

```

rf = RandomForestClassifier(
    n_estimators=200,
    random_state=42
)

results_tfidf.append(
    evaluate_model(
        "Random Forest - TF-IDF",
        rf,
        X_train_tfidf,
        X_test_tfidf,
        y_train,
        y_test
    )
)

```

```

=====
Random Forest - TF-IDF
=====

```

```

Accuracy: 0.8462
Precision (Negative): 0.8191
Recall (Negative): 0.4278
F1 Score (Negative): 0.562

```

```

Classification Report:

```

	precision	recall	f1-score	support
negative	0.82	0.43	0.56	180
positive	0.85	0.97	0.91	600
accuracy			0.85	780
macro avg	0.83	0.70	0.73	780
weighted avg	0.84	0.85	0.83	780

```

Confusion Matrix:
[[ 77 103]
 [ 17 583]]

```

```
results_count = []

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(random_state=42),
    'Random Forest': RandomForestClassifier(
        n_estimators=200,
        random_state=42
    )
}

for name, model in models.items():
    result = evaluate_model(
        name + " - CountVectorizer",
        model,
        X_train_count,
        X_test_count,
        y_train,
        y_test
    )

    results_count.append(result)
```

=====

Logistic Regression - CountVectorizer

=====

Accuracy: 0.8526
Precision (Negative): 0.7928
Recall (Negative): 0.4889
F1 Score (Negative): 0.6048

Classification Report:

	precision	recall	f1-score	support
negative	0.79	0.49	0.60	180
positive	0.86	0.96	0.91	600
accuracy			0.85	780
macro avg	0.83	0.73	0.76	780
weighted avg	0.85	0.85	0.84	780

Confusion Matrix:
[[88 92]
 [23 577]]

=====

Naive Bayes - CountVectorizer

=====

Accuracy: 0.8462
Precision (Negative): 0.8846
Recall (Negative): 0.3833
F1 Score (Negative): 0.5349

Classification Report:

	precision	recall	f1-score	support
negative	0.88	0.38	0.53	180
positive	0.84	0.98	0.91	600
accuracy			0.85	780
macro avg	0.86	0.68	0.72	780
weighted avg	0.85	0.85	0.82	780

Confusion Matrix:
[[69 111]
 [9 591]]

=====

Linear SVM - CountVectorizer

=====

Accuracy: 0.8564
Precision (Negative): 0.7361
Recall (Negative): 0.5889
F1 Score (Negative): 0.6543

Classification Report:

	precision	recall	f1-score	support
negative	0.74	0.59	0.65	180
positive	0.88	0.94	0.91	600
accuracy			0.86	780
macro avg	0.81	0.76	0.78	780
weighted avg	0.85	0.86	0.85	780

Confusion Matrix:
[[106 74]
 [38 562]]

=====

Random Forest - CountVectorizer

=====

Accuracy: 0.8564
Precision (Negative): 0.8617
Recall (Negative): 0.45
F1 Score (Negative): 0.5912

Classification Report:

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

negative	0.86	0.45	0.59	180
positive	0.86	0.98	0.91	600
accuracy			0.86	780
macro avg	0.86	0.71	0.75	780
weighted avg	0.86	0.86	0.84	780

Confusion Matrix:

```
[[ 81  99]
 [ 13 587]]
```

Model Comparison Before Balancing

```
results_df = pd.DataFrame(results_tfidf + results_count)
```

```
results_df
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression - TF-IDF	0.824359	0.921569	0.261111	0.406926
1	Naive Bayes - TF-IDF	0.791026	1.000000	0.094444	0.172589
2	Linear SVM - TF-IDF	0.861538	0.810345	0.522222	0.635135
3	Random Forest - TF-IDF	0.846154	0.819149	0.427778	0.562044
4	Logistic Regression - CountVectorizer	0.852564	0.792793	0.488889	0.604811
5	Naive Bayes - CountVectorizer	0.846154	0.884615	0.383333	0.534884
6	Linear SVM - CountVectorizer	0.856410	0.736111	0.588889	0.654321
7	Random Forest - CountVectorizer	0.856410	0.861702	0.450000	0.591241

```
results_df.sort_values(
    by='F1 Score',
    ascending=False
)
```

	Model	Accuracy	Precision	Recall	F1 Score
6	Linear SVM - CountVectorizer	0.856410	0.736111	0.588889	0.654321
2	Linear SVM - TF-IDF	0.861538	0.810345	0.522222	0.635135
4	Logistic Regression - CountVectorizer	0.852564	0.792793	0.488889	0.604811
7	Random Forest - CountVectorizer	0.856410	0.861702	0.450000	0.591241
3	Random Forest - TF-IDF	0.846154	0.819149	0.427778	0.562044
5	Naive Bayes - CountVectorizer	0.846154	0.884615	0.383333	0.534884
0	Logistic Regression - TF-IDF	0.824359	0.921569	0.261111	0.406926
1	Naive Bayes - TF-IDF	0.791026	1.000000	0.094444	0.172589

Handling Class Imbalance

```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
```

Random Oversampling

```
ros = RandomOverSampler(random_state=42)

X_train_ros, y_train_ros = ros.fit_resample(
    X_train_tfidf,
    y_train
)

print("BEFORE Random Oversampling:")
print(Counter(y_train))

print("\nAFTER Random Oversampling:")
print(Counter(y_train_ros))
```

BEFORE Random Oversampling:
Counter({'positive': 1400, 'negative': 420})

AFTER Random Oversampling:
Counter({'positive': 1400, 'negative': 1400})

```
import pandas as pd
import matplotlib.pyplot as plt

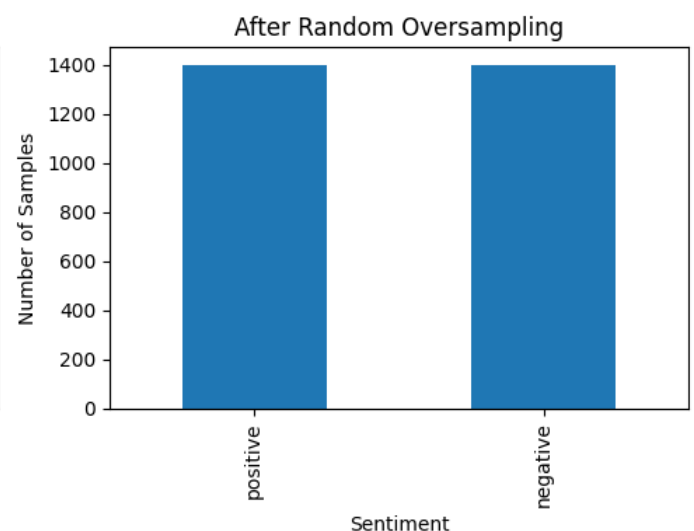
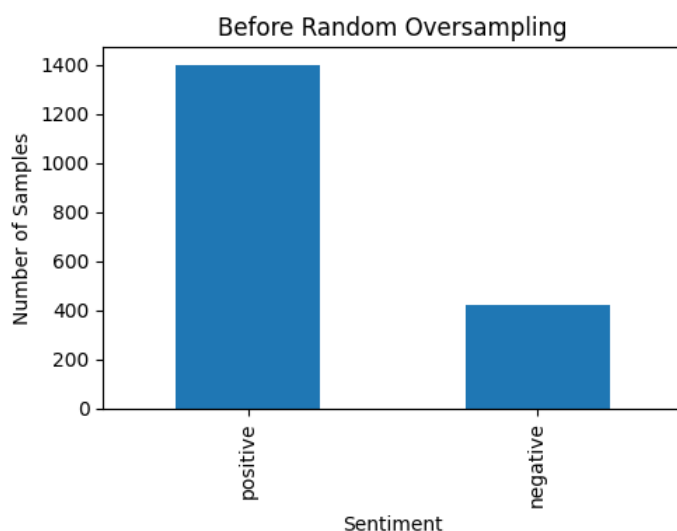
before_counts = pd.Series(y_train).value_counts()
after_counts = pd.Series(y_train_ros).value_counts()

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

before_counts.plot(kind='bar', ax=axes[0])
axes[0].set_title('Before Random Oversampling')
axes[0].set_xlabel('Sentiment')
axes[0].set_ylabel('Number of Samples')

after_counts.plot(kind='bar', ax=axes[1])
axes[1].set_title('After Random Oversampling')
axes[1].set_xlabel('Sentiment')
axes[1].set_ylabel('Number of Samples')

plt.tight_layout()
plt.show()
```



```
lr_ros = LogisticRegression(max_iter=1000, random_state=42)

ros_result = evaluate_model(
    "Logistic Regression - Random Oversampling",
    lr_ros,
    X_train_ros,
    X_test_tfidf,
    y_train_ros,
    y_test
)
```

```
=====
Logistic Regression - Random Oversampling
=====
```

```
Accuracy: 0.85
Precision (Negative): 0.674
Recall (Negative): 0.6778
F1 Score (Negative): 0.6759
```

Classification Report:

	precision	recall	f1-score	support
negative	0.67	0.68	0.68	180
positive	0.90	0.90	0.90	600
accuracy			0.85	780
macro avg	0.79	0.79	0.79	780
weighted avg	0.85	0.85	0.85	780

Confusion Matrix:

```
[[122  58]
 [ 59 541]]
```

Random Undersampling

```
rus = RandomUnderSampler(random_state=42)

X_train_rus, y_train_rus = rus.fit_resample(
    X_train_tfidf,
    y_train
)

print("BEFORE Random Undersampling:")
print(Counter(y_train))

print("\nAFTER Random Undersampling:")
print(Counter(y_train_rus))
```

```
BEFORE Random Undersampling:
Counter({'positive': 1400, 'negative': 420})
```

```
AFTER Random Undersampling:
Counter({'negative': 420, 'positive': 420})
```

```

before_counts = pd.Series(y_train).value_counts()
after_rus_counts = pd.Series(y_train_rus).value_counts()

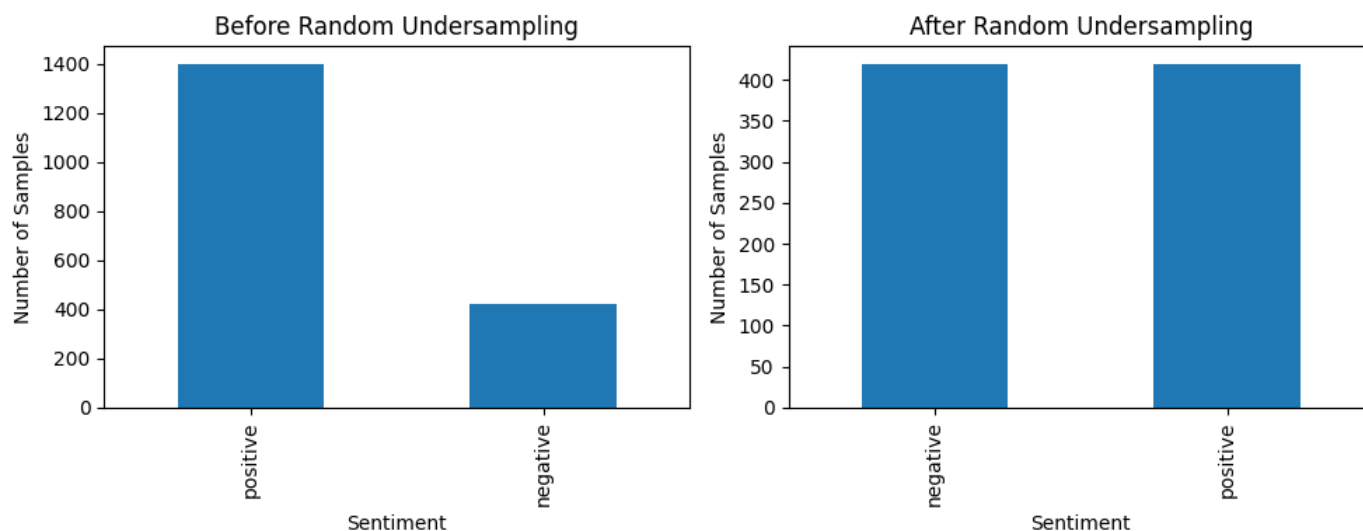
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

before_counts.plot(kind='bar', ax=axes[0])
axes[0].set_title('Before Random Undersampling')
axes[0].set_xlabel('Sentiment')
axes[0].set_ylabel('Number of Samples')

after_rus_counts.plot(kind='bar', ax=axes[1])
axes[1].set_title('After Random Undersampling')
axes[1].set_xlabel('Sentiment')
axes[1].set_ylabel('Number of Samples')

plt.tight_layout()
plt.show()

```



```

lr_rus = LogisticRegression(max_iter=1000, random_state=42)

rus_result = evaluate_model(
    "Logistic Regression - Random Undersampling",
    lr_rus,
    X_train_rus,
    X_test_tfidf,
    y_train_rus,
    y_test
)

```

=====

Logistic Regression - Random Undersampling

=====

Accuracy: 0.7795
Precision (Negative): 0.5136
Recall (Negative): 0.8389
F1 Score (Negative): 0.6371

Classification Report:

	precision	recall	f1-score	support
negative	0.51	0.84	0.64	180
positive	0.94	0.76	0.84	600
accuracy			0.78	780
macro avg	0.73	0.80	0.74	780
weighted avg	0.84	0.78	0.79	780

Confusion Matrix:

```

[[151  29]
 [143 457]]

```

SMOTE

```
smote = SMOTE(random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(
    X_train_tfidf,
    y_train
)

print("BEFORE SMOTE:")
print(Counter(y_train))

print("\nAFTER SMOTE:")
print(Counter(y_train_smote))
```

BEFORE SMOTE:
Counter({'positive': 1400, 'negative': 420})

AFTER SMOTE:
Counter({'positive': 1400, 'negative': 1400})

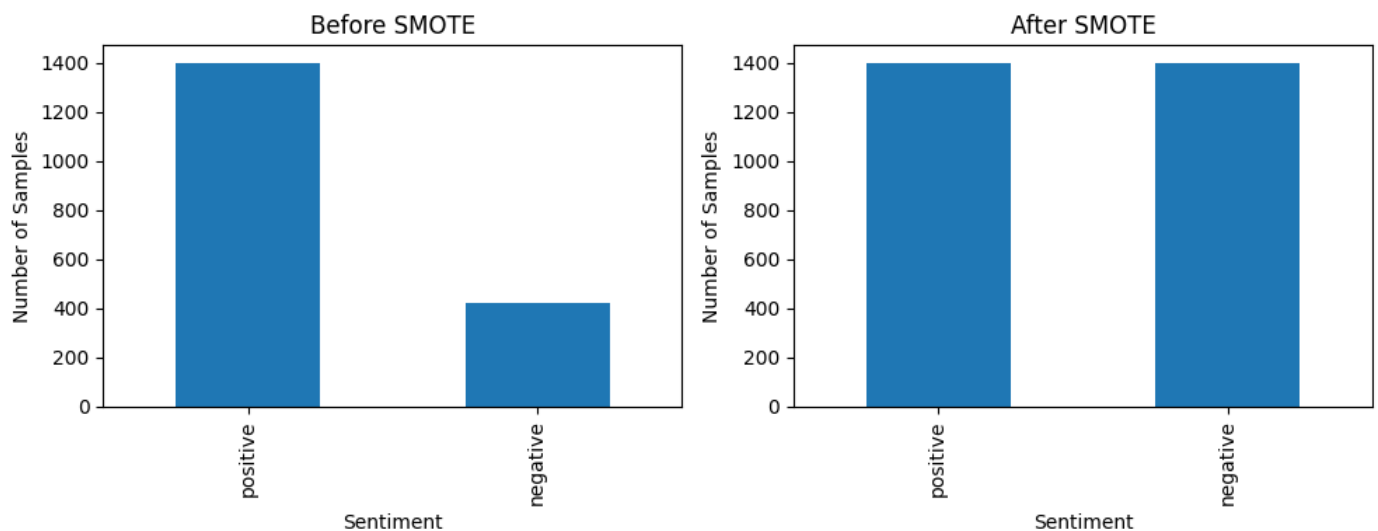
```
before_counts = pd.Series(y_train).value_counts()
after_smote_counts = pd.Series(y_train_smote).value_counts()

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

before_counts.plot(kind='bar', ax=axes[0])
axes[0].set_title('Before SMOTE')
axes[0].set_xlabel('Sentiment')
axes[0].set_ylabel('Number of Samples')

after_smote_counts.plot(kind='bar', ax=axes[1])
axes[1].set_title('After SMOTE')
axes[1].set_xlabel('Sentiment')
axes[1].set_ylabel('Number of Samples')

plt.tight_layout()
plt.show()
```



```
lr_smote = LogisticRegression(max_iter=1000, random_state=42)

smote_result = evaluate_model(
    "Logistic Regression - SMOTE",
    lr_smote,
    X_train_smote,
    X_test_tfidf,
    y_train_smote,
    y_test
)
```

```
=====
Logistic Regression - SMOTE
=====
```

```
Accuracy: 0.8436
Precision (Negative): 0.6576
Recall (Negative): 0.6722
F1 Score (Negative): 0.6648
```

```
Classification Report:
              precision    recall  f1-score   support

   negative       0.66       0.67       0.66       180
   positive       0.90       0.90       0.90       600

   accuracy                   0.84       780
  macro avg       0.78       0.78       0.78       780
 weighted avg     0.84       0.84       0.84       780
```

```
Confusion Matrix:
[[121  59]
 [ 63 537]]
```

Final Performance Comparison

```
original_lr_result = results_tfidf[0].copy()
original_lr_result['Model'] = 'Original Imbalanced'

comparison_df = pd.DataFrame([
    original_lr_result,
    ros_result,
    rus_result,
    smote_result
])

comparison_df = comparison_df[
    ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score']
]

comparison_df.round(4)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Original Imbalanced	0.8244	0.9216	0.2611	0.4069
1	Logistic Regression - Random Oversampling	0.8500	0.6740	0.6778	0.6759
2	Logistic Regression - Random Undersampling	0.7795	0.5136	0.8389	0.6371
3	Logistic Regression - SMOTE	0.8436	0.6576	0.6722	0.6648

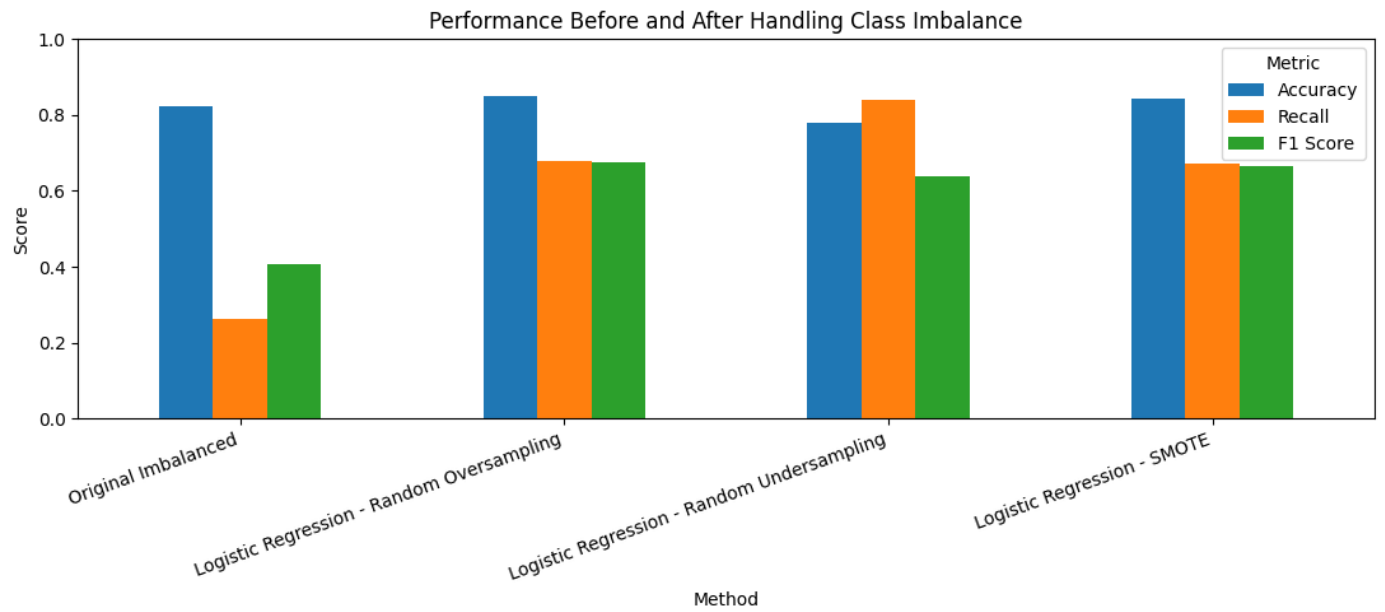
```

plot_df = comparison_df.set_index('Model')[
    ['Accuracy', 'Recall', 'F1 Score']
]

plot_df.plot(
    kind='bar',
    figsize=(11, 5)
)

plt.title('Performance Before and After Handling Class Imbalance')
plt.xlabel('Method')
plt.ylabel('Score')
plt.ylim(0, 1)
plt.xticks(rotation=20, ha='right')
plt.legend(title='Metric')
plt.tight_layout()
plt.show()

```



Confusion Matrix: Before vs After Balancing

```
from sklearn.metrics import ConfusionMatrixDisplay

# Predictions BEFORE balancing
y_pred_before = lr.predict(X_test_tfidf)

# Predictions AFTER Random Oversampling
y_pred_after = lr_ros.predict(X_test_tfidf)

# Confusion matrices
cm_before = confusion_matrix(
    y_test,
    y_pred_before,
    labels=['negative', 'positive']
)

cm_after = confusion_matrix(
    y_test,
    y_pred_after,
    labels=['negative', 'positive']
)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

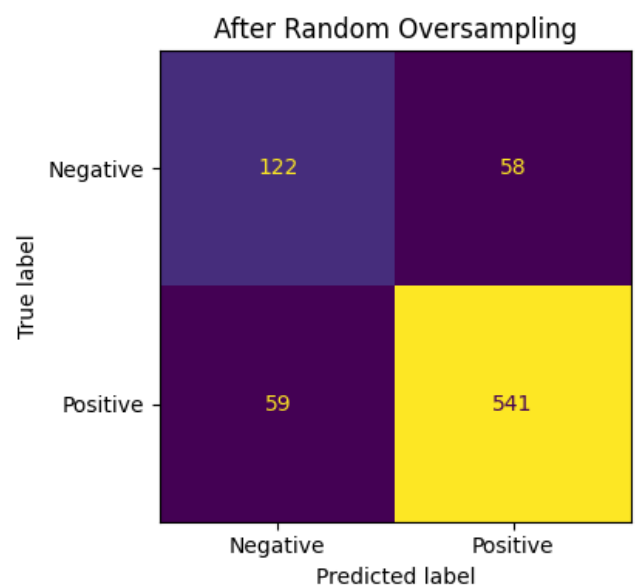
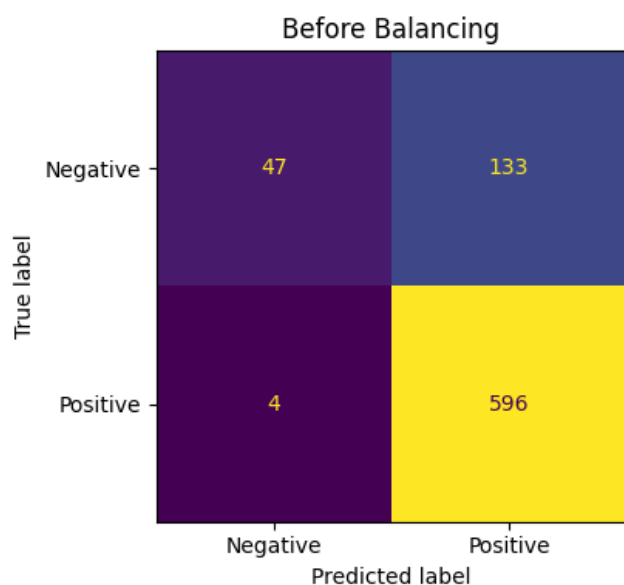
ConfusionMatrixDisplay(
    confusion_matrix=cm_before,
    display_labels=['Negative', 'Positive']
).plot(ax=axes[0], colorbar=False)

axes[0].set_title('Before Balancing')

ConfusionMatrixDisplay(
    confusion_matrix=cm_after,
    display_labels=['Negative', 'Positive']
).plot(ax=axes[1], colorbar=False)

axes[1].set_title('After Random Oversampling')

plt.tight_layout()
plt.show()
```



Conclusion

The original dataset was imbalanced, containing 2,000 positive samples and 600 negative samples.

After splitting the dataset, the training set contained 1,400 positive samples and 420 negative samples. Different text representation techniques and classification models were evaluated.

Before handling class imbalance, the models showed relatively high accuracy, but their ability to detect the minority negative class was limited. For example, Logistic Regression with TF-IDF achieved an accuracy of 82.44%, but the recall for the negative class was only 26.11%.

Three techniques were applied to handle class imbalance: Random Oversampling, Random Undersampling, and SMOTE.

Random Oversampling provided the best overall balance, achieving an accuracy of 85.00%, a recall of 67.78%, and an F1-score of 67.59% for the negative class.

Random Undersampling achieved the highest negative-class recall of 83.89%, but its overall accuracy decreased to 77.95%.

SMOTE also improved minority-class detection, achieving a recall of 67.22% and an F1-score of 66.48%.

The confusion matrix further showed that Random Oversampling increased the number of correctly classified negative samples from 47 to 122.

Therefore, Random Oversampling was selected as the most balanced approach for this dataset because it improved minority-class detection while maintaining good overall accuracy.