

Project 2 - NLP Classification with Imbalanced Data

SID: 253000264

Methodology: TF-IDF feature representation + Random Forest classifier, comparing baseline performance against resampling techniques (RandomOverSampler, RandomUnderSampler, SMOTE) for handling class imbalance.

Steps:

1. Download the NLP dataset for the classification problem.
2. Apply data preprocessing.
3. Split the dataset.
4. Apply feature representation methods (TF-IDF).
5. Train and evaluate models (baseline vs. resampling techniques).

Import necessary libraries

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

import warnings
warnings.filterwarnings("ignore")

# Download NLTK resources (only first time)
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')

[nltk_data] Downloading package punkt to /Users/suad/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /Users/suad/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/suad/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/suad/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!

Out [ ]: True
```

Function to evaluate model

```
In [ ]: def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Plot confusion matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
```

```
        'Classification Report': report
    }
    return metrics
```

Function to apply Random Forest with grid search

```
In [ ]: def train_random_forest_classifier_with_grid_search(X_train_vec, y_train, X_test_vec, y_test, evaluate_model_func, experiment_name):
    # Define the Random Forest Classifier
    rf_classifier = RandomForestClassifier(random_state=42)

    # Define the hyperparameters for grid search
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
    }

    # Initialize GridSearchCV
    grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5,
                               scoring='accuracy', n_jobs=-1)

    # Fit the grid search to the data
    grid_search.fit(X_train_vec, y_train)

    # Get the best estimator from grid search
    best_rf_classifier = grid_search.best_estimator_

    # Make predictions using the best model
    y_pred = best_rf_classifier.predict(X_test_vec)

    # Evaluate the model
    evaluation_results = evaluate_model_func(experiment_name, y_test, y_pred)

    # Print the evaluation results
    for key, value in evaluation_results.items():
        if key == 'Classification Report':
            print(value) # Print report separately for better readability
        else:
            print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")

    # Print the best parameters found by grid search
    print("\nBest hyperparameters found by GridSearchCV:")
    print(grid_search.best_params_)

    return evaluation_results
```

Reading dataset

```
In [ ]: df = pd.read_csv("unbalanceddataset.csv")
df.head()
```

Out[]:

	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

```
In [ ]: df['sentiment'].value_counts()
```

```
Out[ ]: sentiment
positive    2000
negative     600
Name: count, dtype: int64
```

Preprocessing English text (Data Preprocessing step)

```
In [ ]: stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # 1. Lowercasing
    text = text.lower()
    # 2. Remove URLs
    text = re.sub(r'http\S+|www.\S+', '', text)
    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # 4. Remove Special Characters, Numbers, Punctuation
    text = re.sub(r'^a-zA-Z\s]', '', text)
    # 5. Tokenization
    tokens = word_tokenize(text)
    # 6. Remove Stop Words
    tokens = [word for word in tokens if word not in stop_words]
    # 7. Stemming
    stemmed_tokens = [stemmer.stem(word) for word in tokens]
    # 8. Final clean text reconstruction (optional)
```

```
clean_text = ' '.join(stemmed_tokens)

return clean_text

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

```
Out[ ]: 0      java concurr practic probabl best java book iv...
1              haha aww hun bet creativ
2      pickl lol thank much hope your great day
3      even town jeremi sad carri cant come
4      took endors spot twindexxcom thank endors

      ...
2595  what yall made earli night think ima bout take...
2596              agre
2597              yeah thank
2598      good morn everyon
2599      em babi start kindergarten crazi summer go
Name: clean_text, Length: 2600, dtype: str
```

```
In [ ]: df.head()
```

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

Splitting the dataset and applying feature representation (TF-IDF)

First, we study the effect of applying an ML model directly on the imbalanced dataset (baseline, no resampling).

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

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

vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

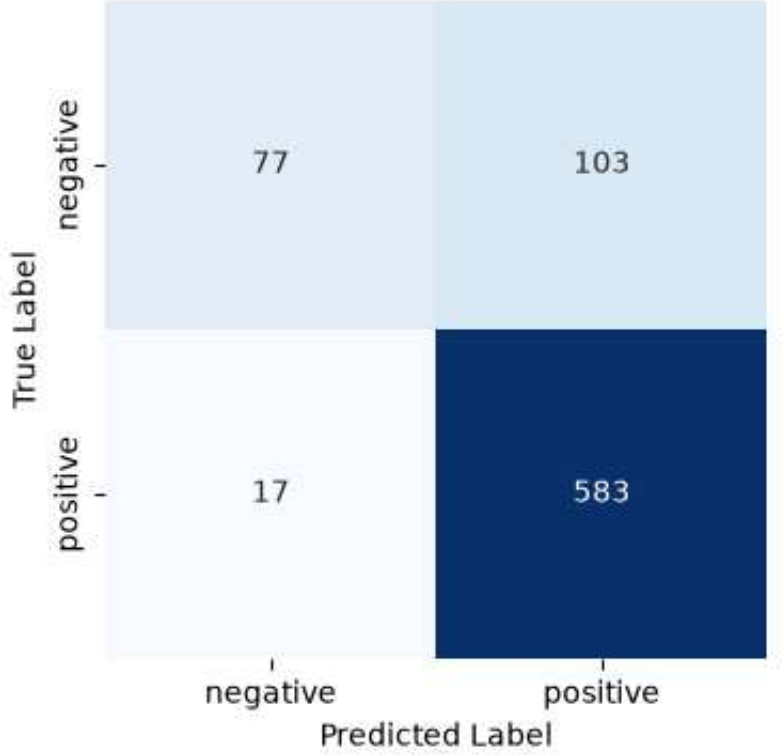
print("Train size:", X_train_vec.shape, "| Test size:", X_test_vec.shape)
```

Train size: (1820, 3657) | Test size: (780, 3657)

Baseline: Random Forest Classifier on the imbalanced dataset (no resampling)

```
In [ ]: results_baseline = train_random_forest_classifier_with_grid_search(
    X_train_vec, y_train, X_test_vec, y_test, evaluate_model,
    experiment_name='RandomForest - Imbalanced (baseline)'
)
```

Confusion Matrix for RandomForest - Imbalanced (baseline)



Model Name:
RandomForest – Imbalanced (baseline)
Accuracy: 0.8462
Precision: 0.8428
Recall: 0.8462
F1 Score: 0.8272

	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

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 200}

Resampling Techniques

- 1. **Undersampling** (Reducing Majority Class)
- 2. **Oversampling** (Increasing Minority Class)
- 3. **SMOTE** (Synthetic Minority Oversampling Technique)

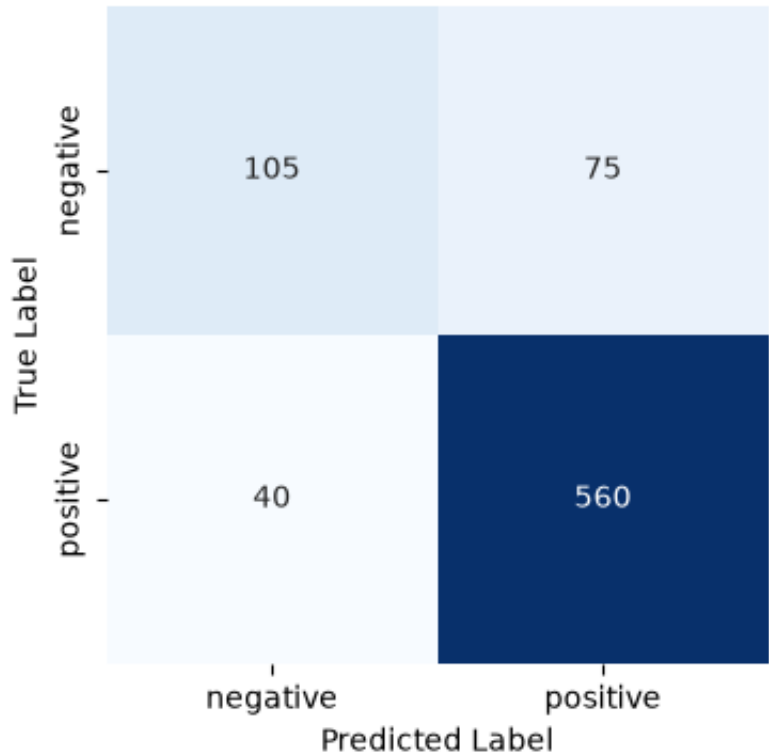
RandomOverSampler

```
In [ ]: ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train_vec, y_train)
print("After RandomOversampling:", Counter(y_ros))

results_ros = train_random_forest_classifier_with_grid_search(
    X_ros, y_ros, X_test_vec, y_test, evaluate_model,
    experiment_name='RandomForest – RandomOverSampler'
)
```

After RandomOversampling: Counter({'positive': 1400, 'negative': 1400})

Confusion Matrix for RandomForest - RandomOverSampler



Model Name:
RandomForest – RandomOverSampler
Accuracy: 0.8526
Precision: 0.8455
Recall: 0.8526
F1 Score: 0.8467

	precision	recall	f1-score	support
negative	0.72	0.58	0.65	180
positive	0.88	0.93	0.91	600
accuracy			0.85	780
macro avg	0.80	0.76	0.78	780
weighted avg	0.85	0.85	0.85	780

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 300}

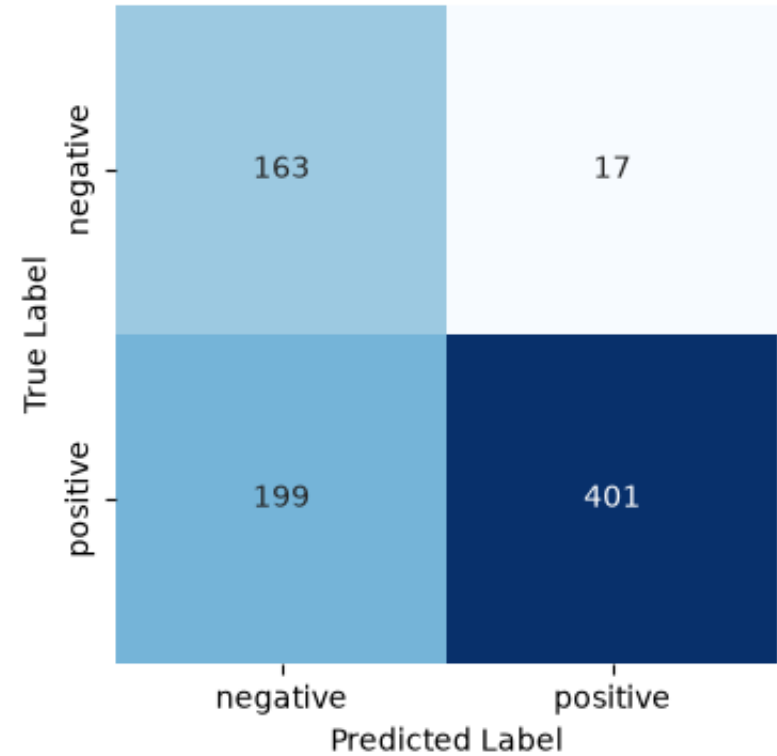
RandomUnderSampler

```
In [ ]: rus = RandomUnderSampler(random_state=42)
X_res, y_res = rus.fit_resample(X_train_vec, y_train)
print("After undersampling:", Counter(y_res))

results_rus = train_random_forest_classifier_with_grid_search(
    X_res, y_res, X_test_vec, y_test, evaluate_model,
    experiment_name='RandomForest – RandomUnderSampler'
)
```

After undersampling: Counter({'negative': 420, 'positive': 420})

Confusion Matrix for RandomForest - RandomUnderSampler



Model Name:
RandomForest - RandomUnderSampler
Accuracy: 0.7231
Precision: 0.8419
Recall: 0.7231
F1 Score: 0.7448

	precision	recall	f1-score	support
negative	0.45	0.91	0.60	180
positive	0.96	0.67	0.79	600
accuracy	0.72			780
macro avg	0.70	0.79	0.69	780
weighted avg	0.84	0.72	0.74	780

Best hyperparameters found by GridSearchCV:
{'max_depth': 20, 'n_estimators': 200}

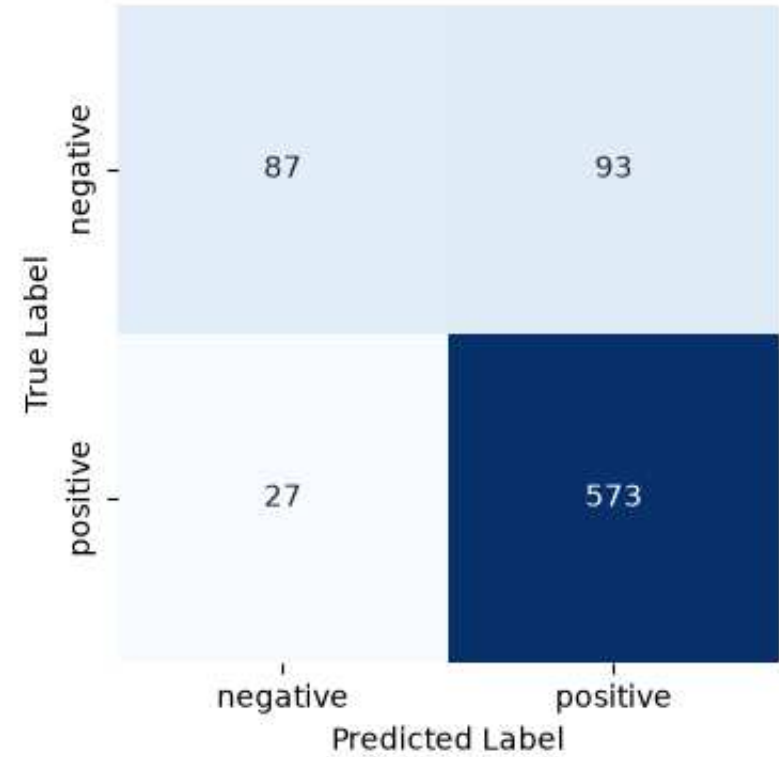
SMOTE

```
In [ ]: smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_train_vec, y_train)
print("After SMOTE:", Counter(y_smote))

results_smote = train_random_forest_classifier_with_grid_search(
    X_smote, y_smote, X_test_vec, y_test, evaluate_model,
    experiment_name='RandomForest - SMOTE'
)
```

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

Confusion Matrix for RandomForest - SMOTE



Model Name:
RandomForest - SMOTE
Accuracy: 0.8462
Precision: 0.8379
Recall: 0.8462
F1 Score: 0.8329

	precision	recall	f1-score	support
negative	0.76	0.48	0.59	180
positive	0.86	0.95	0.91	600
accuracy			0.85	780
macro avg	0.81	0.72	0.75	780
weighted avg	0.84	0.85	0.83	780

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 200}

Summary comparison of all experiments

```
In [ ]: all_results = [results_baseline, results_ros, results_rus, results_smote]

summary_df = pd.DataFrame([
    {
        'Experiment': r['Model Name'],
        'Accuracy': round(r['Accuracy'], 4),
        'Precision': round(r['Precision'], 4),
        'Recall': round(r['Recall'], 4),
        'F1 Score': round(r['F1 Score'], 4),
    }
    for r in all_results
])

summary_df
```

Out []:

	Experiment	Accuracy	Precision	Recall	F1 Score
0	RandomForest - Imbalanced (baseline)	0.8462	0.8428	0.8462	0.8272
1	RandomForest - RandomOverSampler	0.8526	0.8455	0.8526	0.8467
2	RandomForest - RandomUnderSampler	0.7231	0.8419	0.7231	0.7448
3	RandomForest - SMOTE	0.8462	0.8379	0.8462	0.8329

```
In [ ]: summary_df.set_index('Experiment')[['Accuracy', 'Precision', 'Recall', 'F1 Score']].plot(
    kind='bar', figsize=(10, 5)
)

plt.title('Effect of Resampling Techniques on Model Performance')
plt.ylabel('Score')
plt.xticks(rotation=20, ha='right')
plt.ylim(0, 1)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

