

```
In [2]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_s
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

import re
import nltk

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

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

Out[2]: True

```
In [3]: 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)
```

```

# Output results
metrics = {
    'Model Name': model_name,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1,
    'Classification Report': report
}
return metrics

```

```

In [4]: from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import GridSearchCV

        def train_random_forest_classifier_with_grid_search(X_train_vec, y_train,
            # 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_
                                     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('RandomForestClassifier', y_
            # Print the evaluation results
            for key, value in evaluation_results.items():
                if key == 'Classification Report':
                    print(value) # Print report separately for better readabilit
                else:
                    print(f"{key}: {value:.4f}" if isinstance(value, float) else
            # Print the best parameters found by grid search
            print("\nBest hyperparameters found by GridSearchCV:")
            print(grid_search.best_params_)

```

```

In [5]: df = pd.read_csv("unbalanceddataset.csv")
        df.head()

```

```
Out [5]: <style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }
.dataframe tbody tr th { vertical-align: top; } .dataframe thead th { text-align: right; }
</style> <thead> <th></th> <th>text</th> <th>sentiment</th> </thead> <tbody>
<th>0</th> <th>1</th> <th>2</th> <th>3</th> <th>4</th> </tbody> <tbody>
</tbody>
```

Java Concurrency in Practice is probably the b...	positive
---	----------

haha aww hun i bet you are more creative tha...	positive
---	----------

_pickle lol, thank you very much Hope you`re h...	positive
---	----------

Out for an evening on the town with jeremy. Sa...	negative
---	----------

- just took over the #1 Most Endorsed spot on...	positive
--	----------

```
In [6]: df['sentiment'].value_counts()
```

```
Out[6]: sentiment
positive      2000
negative       600
Name: count, dtype: int64
```

```
In [7]: stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    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
```

```
In [8]: df["clean_text"] = df["text"].apply(preprocess_text)
df = df.dropna()
df['clean_text']
```

```
Out[8]: 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 [9]: df.head()
```

```
Out[9]: <style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }
.dataframe tbody tr th { vertical-align: top; } .dataframe thead th { text-align: right; }
</style> <thead> <th></th> <th>text</th> <th>sentiment</th> <th>clean_text</th>
</thead> <tbody> <th>0</th> <th>1</th> <th>2</th> <th>3</th> <th>4</th>
</tbody> <tbody></tbody>
```

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

```
In [10]: X = df['clean_text']
y = df['sentiment']
```

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
In [12]: train_random_forest_classifier_with_grid_search(X_train_vec, y_train, X_t
```

Model Name:
 RandomForestClassifier
 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}

```
In [13]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
In [14]: from imblearn.over_sampling import RandomOverSampler
from collections import Counter

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

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

```
In [15]: train_random_forest_classifier_with_grid_search(X_ros, y_ros, X_test_vec,
```

Model Name:
 RandomForestClassifier
 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}

```
In [16]: from imblearn.under_sampling import RandomUnderSampler

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

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

In [17]: `train_random_forest_classifier_with_grid_search(X_res, y_res, X_test_vec,`

Model Name:

RandomForestClassifier

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}

In [18]: `from imblearn.over_sampling import SMOTE`

`smote = SMOTE(random_state=42)`

`X_smote, y_smote = smote.fit_resample(X_train_vec, y_train)`

`print("After SMOTE:", Counter(y_smote))`

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

In [19]: `train_random_forest_classifier_with_grid_search(X_smote, y_smote, X_test_`

Model Name:

RandomForestClassifier

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}