

```
!pip install nltk
```

```
Collecting nltk
  Downloading nltk-3.9.4-py3-none-any.whl.metadata (3.2 kB)
Collecting click (from nltk)
  Downloading click-8.3.2-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: joblib in C:\Users\saldwsari\AppData\Roaming\Python\Python313\site-packages (from nltk)
Collecting regex>=2021.8.3 (from nltk)
  Downloading regex-2026.4.4-cp313-cp313-win_amd64.whl.metadata (41 kB)
Requirement already satisfied: tqdm in C:\Users\saldwsari\AppData\Roaming\Python\Python313\site-packages (from nltk) (
Requirement already satisfied: colorama in C:\ProgramData\anaconda3\envs\aaaa\Lib\site-packages (from click->nltk) (0.
Downloading nltk-3.9.4-py3-none-any.whl (1.6 MB)
----- 0.0/1.6 MB ? eta -:--:--
----- 0.0/1.6 MB ? eta -:--:--
----- 0.3/1.6 MB ? eta -:--:--
----- 0.5/1.6 MB 2.0 MB/s eta 0:00:01
----- 1.3/1.6 MB 2.1 MB/s eta 0:00:01
----- 1.6/1.6 MB 2.0 MB/s 0:00:00
Downloading regex-2026.4.4-cp313-cp313-win_amd64.whl (277 kB)
Downloading click-8.3.2-py3-none-any.whl (108 kB)
Installing collected packages: regex, click, nltk

----- 1/3 [click]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 2/3 [nltk]
----- 3/3 [nltk]

Successfully installed click-8.3.2 nltk-3.9.4 regex-2026.4.4
```

```
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_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import re
import nltk
```

```
try:
    nltk.download('punkt')
    nltk.download('stopwords')
    nltk.download('wordnet')
except Exception as e:
    print(f"Error downloading NLTK data: {e}")
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\saldwsari\AppData\Roaming\nltk_data...
[nltk_data] Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\saldwsari\AppData\Roaming\nltk_data...
[nltk_data] Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\saldwsari\AppData\Roaming\nltk_data...
```

```
import nltk
print(nltk.data.path)
```

```
['C:\\Users\\saldwsari\\nltk_data', 'C:\\ProgramData\\anaconda3\\envs\\aaaa\\nltk_data', 'C:\\ProgramData\\anaconda3\\e
```

```
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
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

def train_random_forest_classifier_with_grid_search(X_train_vec, y_train, X_test_vec, y_test, evaluate_model_func):
    # 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('RandomForestClassifier', 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_)
```

```
df = pd.read_csv("unbalanceddataset (1).csv")
df.head()
df['sentiment'].value_counts()
```

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

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

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

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

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

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

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]   C:\Users\saldwsari\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt_tab.zip.
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\saldwsari\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   C:\Users\saldwsari\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

rf_classifier = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_classifier.fit(X_train_vec, y_train)

y_pred = rf_classifier.predict(X_test_vec)
print(classification_report(y_test, y_pred))

```

	precision	recall	f1-score	support
negative	0.72	0.53	0.61	116
positive	0.87	0.94	0.91	404
accuracy			0.85	520
macro avg	0.80	0.73	0.76	520
weighted avg	0.84	0.85	0.84	520

```

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)

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))

train_random_forest_classifier_with_grid_search(X_ros, y_ros, X_test_vec, y_test, evaluate_model)

```

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

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}

```

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))

train_random_forest_classifier_with_grid_search(X_res, y_res, X_test_vec, y_test, evaluate_model)

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))

```

```
train_random_forest_classifier_with_grid_search(X_smote, y_smote, X_test_vec, y_test, evaluate_model)
```

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

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}

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

Model Name:

RandomForestClassifier

Accuracy: 0.8487

Precision: 0.8421

Recall: 0.8487

F1 Score: 0.8341

	precision	recall	f1-score	support
negative	0.79	0.47	0.59	180
positive	0.86	0.96	0.91	600
accuracy			0.85	780
macro avg	0.82	0.72	0.75	780
weighted avg	0.84	0.85	0.83	780

Best hyperparameters found by GridSearchCV:

{'max_depth': None, 'n_estimators': 300}

Start coding or [generate](#) with AI.